Universität Bielefeld

**Forschungsbericht der
Technischen Fakultät
Abteilung Informationstechnik**

Divide-and-Conquer Multiple Sequence Alignment

Jens Stoye

Report 97-02

Universität Bielefeld • Postfach 10 01 31 • 33501 Bielefeld • FRG

Impressum: Herausgeber:
Robert Giegerich, Alois Knoll, Peter Ladkin,
Helge Ritter, Gerhard Sagerer, Ipke Wachsmuth

Technische Fakultät der Universität Bielefeld,
Abteilung Informationstechnik, Postfach 10 01 31,
33501 Bielefeld, FRG

ISSN 0946-7831

Divide-and-Conquer Multiple Sequence Alignment

Zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften
der Universität Bielefeld
vorgelegte
Dissertation

von
Jens Stoye

Bielefeld, im Januar 1997

Contents

Chapter 1

Introduction

On the molecular level, the machinery which is responsible for the reproduction of life and the connected evolutionary diversification can well be described as string processing. The primary structures are sequential data which are manipulated by a complex metabolic system. An enormous effort of research is devoted to the elucidation of the underlying biochemical interactions. Starting with the discovery of the molecular structure of genetic information by J. D. Watson and F. H. C. Crick more than forty years ago, data output from biological laboratories, mostly in form of nucleic acid and protein sequences, and – with a continuously decreasing delay – databases storing those data grow so rapidly that there is a strong need for efficient as well as effective algorithms to handle these data and to extract and analyze the contained information.

To give an impression of the involved quantities, Figure 1.1 displays the proliferation of one of the largest genomic databases over the last fifteen years. A clear exponential growth can be observed, and the current trend of doubling every twenty-two months does not seem to be likely to change within the near future. Compared with the information content still to be uncovered, this is still a fairly small amount. For example, the human genome alone consists of twenty-three pairs of chromosomes with typically a hundred million nucleotides each. In order to sift through, classify, and understand the raw data, a general technique has been well established among molecular biologists: Seeking for similarities, biosequences are compared with each other systematically, hoping that what is true for one sequence – either physically or functionally – might be true for its analogue(s) as well. Consequently, methods for automatic sequence comparison are indispensable. The present thesis contributes to this important field of research.

As is well-known, biology is not the only and was not the first application of string comparison techniques. Other major applications are found in computer science where string correction/string editing and the comparison of computer files are important tasks. Error detection and error correction are crucial in coding theory and data transmission. Speech processing, bird song and handwriting analysis are examples of comparison of continuous data streams. Even geological strata have been

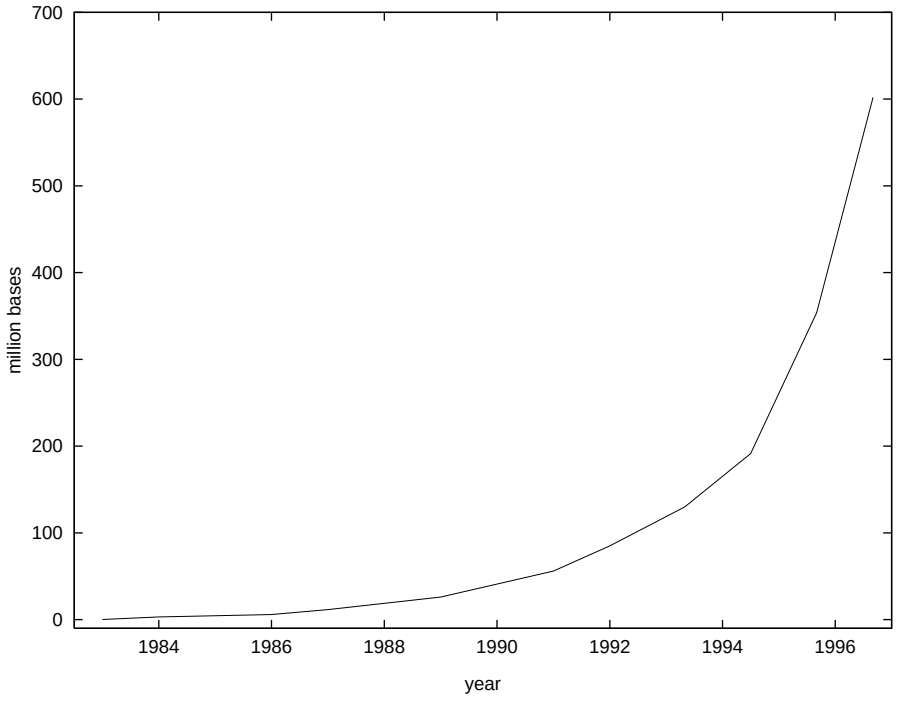

Figure 1.1: The amount of DNA sequence data in GenBank [226, 219, 29, 36, 37, 25, 26, 27, 28].

---

analyzed by methods very similar to those studied here. Kruskal [119] gives a survey of these and further applications. Yet, the focus of the present work will be on string comparison as a tool for the molecular biologist.

1.1 Preliminaries

In this section, we introduce our terminology, and we discuss some questions which have led to the development of a new algorithm, the *Divide-and-Conquer Alignment* algorithm (DCA). This section cannot be a self-contained introduction to computational biology. For this purpose, the reader might use any one of the relevant textbooks (e.g. [193, 3, 48, 34]).

The most prominent biological sequences are *proteins* and *nucleic acid* sequences. Proteins are polymers made up of amino acids connected linearly by peptide bonds, that is they are *polypeptides*. They play an important role as enzymes in the metabolism of the cell. A protein sequence is usually a few hundred units long. On our level of abstraction, proteins can be viewed as sequences of letters drawn from the alphabet of the twenty amino acids occurring in living matter. Table 1.1 lists the full names, the three letter acronyms and the one letter code.

Nucleic acids are also polymers made up of small molecules called nucleotides

Alanine Ala A	Arginine Arg R	Asparagine Asn N	Aspartic acid Asp D	Cysteine Cys C
Glutamine Gln Q	Glutamic acid Glu E	Glycine Gly G	Histidine His H	Isoleucine Ile I
Leucine Leu L	Lysine Lys K	Methionine Met M	Phenylalanine Phe F	Proline Pro P
Serine Ser S	Threonine Thr T	Tryptophan Trp W	Tyrosine Tyr Y	Valine Val V

Table 1.1: The twenty common amino acids.

---

which can be distinguished by the four bases they contain: *adenine* (A), *cytosine* (C), *guanine* (G), and either *thymine* (T) or *uracil* (U) for *deoxyribonucleic acids* (DNA) or *ribonucleic acids* (RNA), respectively. A and G are *purines*, C and T are *pyrimidines*. Nucleic acids contain typically from tens to thousands of units (for RNA) or millions of units (for DNA).

Given one or several such sequences, many questions arising in molecular biology can be reformulated as string processing problems:

- The inference of physical mapping and sequence assembly, i.e. the reconstruction of a DNA molecule from the nucleic acid sequences of fragments within it [189, 76, 106, 113];
- Molecular modelling, i.e. the determination of the three-dimensional structure of a protein from its sequence of amino acids [32, 88, 151, 231, 23, 118];
- The assessment of structure–function correlations, mostly by conclusions from structurally similar or historically related molecules [53];
- The reconstruction of phylogenies, i.e. the inference of the evolutionary history of some species from their associated sequences [66, 110, 55];
- The exploration of statistical geometry in sequence space [57];
- The database search for sequences similar to a given sequence [49, 129, 10, 154];
- The prediction of RNA secondary (and tertiary) structure, i.e. the way in which different segments of the sequence connect with each other [202, 230, 56, 131].

Most of these applications require the comparison of sequences. An early approach to compare sequences of the same length is the computation of their Hamming Distance

[93]. Yet, it is often necessary to compare sequences of different length. An *alignment* of the involved sequences is then the method of choice. It can be defined for pairs of sequences as well as for whole families of related sequences. In this introduction, it suffices to view a sequence alignment as a way to specify which parts of the given sequences correspond to each other. The aim of a sequence alignment is to provide the molecular biologist with a simple way of obtaining information about similarities and differences between sequences.

Unfortunately, often more than one reasonable alignment of a family of sequences exists. In general, it is not easy to decide which one is closest to biological truth. At least for homologous molecules which stem from the same ancestor, there exists a *correct* alignment which best reflects the evolutionary history of the molecule. However, it is often very difficult – if not impossible – to read this true history from today's data. To find out the correct alignment of protein or nucleic acid sequences is *the alignment problem in molecular biology.*

For amino acid sequences coding for proteins, it has been argued that the best approximation of the correct alignment is obtained by the superposition of the three-dimensional protein structures [110, 134, 32, 126]. However, the three-dimensional structure is known for only a few hundred proteins. Without knowledge of structure, the sequence of letters from the amino acid alphabet is the only information that can be used. To this end, quantifying criteria – so-called *score functions* – are defined which measure the reasonability of an alignment just in terms of the arrangement of the letters. The development of methods for computing alignments which are *optimal* with regard to such a score or – at least – close to the optimum is *the alignment problem in computational biology.*

Statistical advantages of *multiple* as opposed to *pairwise* sequence alignment have long been recognized [69], especially when similarities between component pairs suggest homology but are not strong enough on their own to be regarded as significant. When sensitive statements about multiple sequences are required, a multiple alignment can give more information than can be readily obtained from pairwise comparison [87, 206]. For instance, the results produced by methods of reconstructing phylogenetic relationships from sequences (e.g. maximum likelihood [63, 164] and maximum parsimony methods [62, 47], spectral analysis [190], or the split decomposition method [18, 54], see also [55, 195]) depend highly on the choice of the underlying alignment.

Due to the high computational complexity required for the *exact* solution of multiple alignment problems, faster methods for the computation of *approximate* – though not necessarily optimal – solutions are used. A well established technique is the (order-sensitive) *iterated pairwise alignment.* However, it is well-known that the order in iterated sequence alignments can bias the topology of a reconstructed phylogenetic tree, and thus order-independent, *simultaneous* alignments are to be preferred [122, 123, 98]. Consequently, there is a strong need for procedures computing simultaneous rather than iterative multiple sequence alignments.

In the last decades, various simultaneous multiple alignment algorithms have been developed (see e.g. [166, 225, 142, 81, 39, 11, 114, 90]). Unfortunately, almost all of these methods either exhibit a prohibitive computational complexity or yield biologically unplausible results. Current algorithms which try to optimize one of the standard score functions are limited to half a dozen of sequences. With the algorithm DCA developed in this thesis, we improve this situation. By slicing the sequences at appropriate positions in a divide-and-conquer manner, DCA allows to compute high quality – but not necessarily optimal – simultaneous alignments of up to fourteen related sequences in a rather short time requiring only moderate computer memory. Surprisingly, to our knowledge, the divide-and-conquer principle, a well established method in algorithmic computer science (cf. [2, 45]), has never been applied before systematically in such a simple way within this context.

1.2 Overview

In Chapter 2, a formal introduction to the multiple sequence alignment problem is given. We present definitions of pairwise and multiple sequence alignments followed by a discussion of the most commonly used alignment score functions. The multiple sequence alignment problem is formulated on this basis.

In Chapter 3, we recall the standard method for solving the alignment problem. Time and space limitations of this approach will be discussed which make it necessary to accept fast but in general suboptimal solutions. The standard principles of computing such heuristic alignments are briefly summarized.

Chapter 4 is devoted to the presentation of the basic divide-and-conquer alignment algorithm which – in contrast to previous heuristic methods – computes simultaneous multiple sequence alignments optimizing a well-defined alignment score. Time and space complexity of the algorithm are analyzed, and several suggestions for variations and further uses of the algorithm are discussed.

In Chapter 5, we present improvements of the basic algorithm which allow an enormous increase of efficiency. Using branch-and-bound techniques, the search for appropriate slicing positions of the sequences is accelerated so that more than a dozen of sequences can be aligned simultaneously by our method.

Chapter 6 gives a short description of the computer program `DCA` which is part of this thesis.

The applicability of our algorithm is illustrated in Chapters 7 and 8. We establish the high quality of the computed alignments in mathematical as well as in biological terms and validate the theoretical time and space analyses. Using well established benchmark problems, alignments produced by our algorithm are also compared to those computed with other methods.

Chapter 9 concludes the thesis by recalling its main results.

1.3 Acknowledgements

Parts of this dissertation thesis have been published in advance [203, 192, 157], and two further papers are forthcoming [35, 158].

The work was carried out whilst being a doctoral fellow at the Center for Interdisciplinary Research on Structure Formation (FSPM) at the University of Bielefeld, Germany, and member of the "Graduiertenkolleg Strukturbildungsprozesse", both of which have been a perfect work environment due to the interdisciplinary nature of these institutions, which made possible many instructive discussions with colleagues and friends.

I would like to thank my advisor, Prof. Dr. Robert Giegerich, for bringing me into contact with the sequence analysis aspects of molecular biology and for maintaining my enthusiasm. I would also like to thank Prof. Dr. Andreas Dress for introducing me to the particular subject of this thesis as well as for his helpful comments and the fruitful discussions during the development of the algorithm and its many variants. My special thank goes to Dr. Sören Perrey who worked out parts of joint publications and who suggested several improvements regarding an earlier version of the manuscript. He also provided me with the example in Section 8.4. Finally I am much obliged to Dr. Vincent Moulton for improving the English of the thesis.

Chapter 2

Analysis of Differences: Sequence Alignment

2.1 Introduction

Before we define alignments formally in Section 2.2, we take a closer look at the events which created – for the most part millions of years ago – the diversity we find in today's nature, and which we aim to detect with our computational methods. In particular, we summarize the principal kinds of mutations which occur on the molecular level during replication and translation of the genetic material. Ignoring any kind of chemical and biophysical properties of the represented molecules, we describe mutations solely as changes in abstract sequences.

We need to introduce some terminology:

Notation 2.1
To allow a broad range of application, we consider any finite set of letters $\mathcal{A}$ as the *alphabet*. A string $\mathbf{s}$ of letters from $\mathcal{A}$ is called *sequence* over $\mathcal{A}$, $|\mathbf{s}|$ denotes the length of $\mathbf{s}$, i.e. the number of letters that $\mathbf{s}$ consists of. The symbol s_i refers to the ith letter (or *site*) of $\mathbf{s}$. The empty sequence of length zero is denoted by ε.

The size of $\mathcal{A}$ is four in case of DNA or RNA sequences, or twenty in case of proteins. Throughout this thesis, any single letter of $\mathcal{A}$ is written in italics font (a, b, c, $s_1, s_2, \ldots$) while sequences over $\mathcal{A}$ are denoted by bold font characters ($\mathbf{s}, \mathbf{t}, \mathbf{s}_1, \mathbf{s}_2, \ldots$). Examples are indicated by capital typewriter font letters (e.g. $\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}, \mathtt{TATAA}$).

Sequences are concatenated by the "++" operator: $\mathbf{s}{+}{+}\mathbf{t}$, or simply by juxtaposition: $\mathbf{st}$. The *reverse* of sequence $\mathbf{s} = s_1 \ldots s_n$ is denoted by $\mathbf{s}^{-1} := s_n \ldots s_1$. In any partition of $\mathbf{s} = \mathbf{uvw}$ into (possibly empty) strings $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{w}$, $\mathbf{u}$ is called a *prefix*, $\mathbf{v}$ is called a *subsequence*[1], and $\mathbf{w}$ is called a *suffix* of $\mathbf{s}$. For all i with $0 \leq i \leq |\mathbf{s}|$, $\mathbf{s}^i$ denotes the prefix of $\mathbf{s}$ consisting of the first i letters of $\mathbf{s}$, and ${}^i\mathbf{s}$ denotes the

[1]Note that – in contrast to some other authors – we define subsequences as consisting of *consecutive* letters of the supersequence.

corresponding suffix. So we have $\mathbf{s}^i{+\!\!+}^i\mathbf{s} = \mathbf{s}$. We also say that $\mathbf{s}$ is *cut* at *slicing position* i. □

The just defined formalism facilitates a closer look at mutations as they are observed in biological sequences, so-called *accepted mutations*. Dayhoff *et al.* [49] distinguish two principal kinds: *large changes* and *point mutations*.

Large changes in genetic sequences are believed to be caused by unequal crossing-over of the chromosomes. These large-scale rearrangements can include entire genes. The subsequences $\mathbf{s}$ and $\mathbf{t}$ in the following list of the most important types of large changes can be several thousands or millions of nucleotides long.

Inversions: A subsequence is reversed, $\mathbf{usv} \to \mathbf{us^{-1}v}$.

Translocations: Subsequences from different chromosomes are exchanged, $\mathbf{usv}$ and $\mathbf{wtx} \to \mathbf{utv}$ and $\mathbf{wsx}$. In most cases, the exchanged subsequences are terminal, i.e. $\mathbf{u}$ or $\mathbf{v}$ (and $\mathbf{w}$ or $\mathbf{x}$, respectively) are empty in which case $\mathbf{s}$ and $\mathbf{t}$ are prefixes or suffixes of the chromosome sequence.

Deletions: A subsequence is deleted, $\mathbf{usv} \to \mathbf{uv}$.

Duplications: A subsequence is duplicated, $\mathbf{usvw} \to \mathbf{usvsw}$.

Transpositions: Two subsequences on the same chromosome are exchanged, $\mathbf{usvtw} \to \mathbf{utvsw}$.

Point mutations are local changes of one or a few consecutive nucleotides during the copying process of DNA. Occurring within coding regions, they may be observed as changes of a single or a few amino acids in the translated protein. In the following formal description of the most important types of point mutations, the subsequences $\mathbf{a}$ and $\mathbf{b}$ are usually very short (e.g. one to five letters in case of amino acids [153]).

Insertions: A single letter or a small number of consecutive letters is inserted, $\mathbf{uv} \to \mathbf{uav}$.

Deletions: A single letter or a small number of consecutive letters is deleted, $\mathbf{uav} \to \mathbf{uv}$.

Substitutions: A short subsequence is substituted by a different sequence, $\mathbf{uav} \to \mathbf{ubv}$, $\mathbf{a} \neq \mathbf{b}$.

In other than biological applications of string comparison, further local "mutations" are considered:

Swaps: Two single neighboring letters are exchanged, $\mathbf{u}ab\mathbf{v} \to \mathbf{u}ba\mathbf{v}$. Swaps play an important role in spelling correction [130, 213].

***Compression* and *expansion*:** In speech recognition [108] and other applications [120], where continuous input streams are compared, it is often necessary to re-scale the incoming data. Scaling operations of this kind are also referred to as *time-* (or *space-*) *warps*.

In the study of genomic sequences, large changes and point mutations are contemplated in different situations. In the search for the correct order of sequence fragments on a chromosome, the so-called *sequence assembly problem*, and in the comparison of the gene pool of different species, large changes have to be considered. In contrast, point mutations play the most important role in the study of single proteins or other comparatively small regions of the genome (sequences of some hundreds up to 100,000 letters in length). Such sequences are often compared for common overall (also called *global*) or local similarities. Global and local sequence comparison can be handled by alignments. The divide-and-conquer algorithm developed in this thesis facilitates global sequence comparison by speeding up the search for global sequence alignments.

2.2 Global Sequence Alignment

A standard method in computational molecular biology for presenting the result of sequence comparison is an *alignment*, which we formalize in this section. *Multiple* sequence alignments are a natural generalization of *pairwise* sequence alignments which we introduce first.

2.2.1 Pairwise Sequence Alignment

Assume that we are given two sequences $\mathbf{s}$ and $\mathbf{t}$ which are known to be globally related. In general $\mathbf{s}$ and $\mathbf{t}$ are of different length. For better comparibility, the lengths of the sequences are equalized: *Blanks*, denoted by dashes "$-$", are inserted into or at either end of $\mathbf{s}$ and $\mathbf{t}$ such that the two resulting sequences $\mathbf{s}^*$ and $\mathbf{t}^*$, respectively, are of the same length N. Apart from the length equalization, the most important aim of inserting blanks is the following: By selecting the location of blanks carefully, regions from $\mathbf{s}$ may be located in $\mathbf{s}^*$ at the same position where similar regions of $\mathbf{t}$ are located in $\mathbf{t}^*$. By writing the sequences $\mathbf{s}^*$ and $\mathbf{t}^*$ above each other, similarities and differences are then easier to observe.

Definition 2.2 (Pairwise Sequence Alignment)
An *alignment* of two sequences $\mathbf{s}$ and $\mathbf{t}$ over $\mathcal{A}$ is a matrix

$$A = \begin{pmatrix} s_1^* & s_2^* & \dots & s_N^* \\ t_1^* & t_2^* & \dots & t_N^* \end{pmatrix}$$

with two rows $\mathbf{s}^* := s_1^* s_2^* \dots s_N^*$, $\mathbf{t}^* := t_1^* t_2^* \dots t_N^*$ and N columns ($\max\{|\mathbf{s}|, |\mathbf{t}|\} \leq N \leq |\mathbf{s}| + |\mathbf{t}|$) where

- $s_j^*, t_j^* \in \mathcal{A} \cup \{-\}$ for all $1 \leq j \leq N$,
- the rows $\mathbf{s}^*$ and $\mathbf{t}^*$ reproduce the sequences $\mathbf{s}$ and $\mathbf{t}$, respectively, upon elimination of blanks, and
- A has no column consisting of two blanks.

□

Note that, in view of the last condition, the number of alignments of two finite sequences $\mathbf{s}$ and $\mathbf{t}$ is finite. To be more precise, the number $f(m,n)$ of alignments of two sequences with m, respectively n letters, can be computed recursively by [221]:

$$f(m,n) \;=\; f(m-1,n) + f(m-1,n-1) + f(m,n-1) \tag{2.1}$$

with initializations $f(0,n) = f(m,0) = 1$ for all $n, m \geq 0$. The correctness of this formula can be checked easily by considering all possible endings of the alignment: If s_m is aligned with a blank, then there exist $f(m-1,n)$ alignments of the earlier parts of the sequences. If s_m and t_n are aligned, $f(m-1,n-1)$ alignments remain. If t_n is aligned with a blank, then $f(m,n-1)$ alignments result. Note that many equivalent alignments are counted, e.g. $\begin{pmatrix} \mathtt{A} & - \\ - & \mathtt{T} \end{pmatrix}$ and $\begin{pmatrix} - & \mathtt{A} \\ \mathtt{T} & - \end{pmatrix}$.

Laquer [124] gives an explicit formula for the values $f(m,n)$:

$$f(m,n) \;=\; \sum_{k=max\{0,m-n\}}^{m} \binom{n+k}{m} \binom{m}{k}.$$

Summing over $N := n + k$, this can be rewritten as

$$\begin{aligned} f(m,n) \;&=\; \sum_{N=max\{n,m\}}^{m+n} \binom{N}{m} \binom{m}{N-n} \\ &=\; \sum_{N=max\{n,m\}}^{m+n} \frac{N!}{(N-m)!\,(N-n)!\,(m+n-N)!} \end{aligned} \tag{2.2}$$

which can also be derived directly: The positions of an alignment of length N, $\max\{m,n\} \leq N \leq m+n$, have either

(i) a gap in the first sequence ($N-m$ positions), or

(ii) a gap in the second sequence ($N-n$ positions), or

(iii) two letters aligned to each other (the remaining $m+n-N$ positions).

Equation 2.2 gives exactly the number of three-way permutations of these three disjoint sets, summed over all N.

Using Stirling's Formula, $f(n,n)$ can be approximated by [124, 221]

$$f(n,n) \approx (1+\sqrt{2})^{2n+1} n^{-1/2}. \tag{2.3}$$

Notation 2.3
Two identical letters above each other in an alignment form a *match* and two distinct ones form a *mismatch* or *substitution*. A blank in one sequence aligned with a letter a in the other can be viewed as an *insertion* of a into the first sequence or as a *deletion* of a from the second sequence. Following Kruskal [119], we use the term *indel* to denote the event of a deletion or an insertion. □

The letters of the original sequence $\mathbf{s}$ and of the "padded" sequence $\mathbf{s}^*$ in an alignment are connected by the following maps $\alpha_{\mathbf{s}^*}$ and $\sigma_{\mathbf{s}^*}$:

Definition 2.4
Assume a sequence $\mathbf{s}$ over $\mathcal{A}$ and a corresponding aligned sequence $\mathbf{s}^*$ of length $N \geq |\mathbf{s}|$, i.e. assume that $\mathbf{s}^*$ reproduces $\mathbf{s}$ upon elimination of the blanks. For $j \in \{1,\ldots,N\}$, let $\sigma_{\mathbf{s}^*}(j)$ be the number of letters in $\mathbf{s}^*$ before position j which are not blanks plus one, i.e.

$$\sigma_{\mathbf{s}^*}(j) := \#\{k \in \{1,\ldots,j-1\} | s^*_k \neq -\} + 1.$$

Clearly, $\sigma_{\mathbf{s}^*}$ is monotonously increasing and the assumed relationship between $\mathbf{s}$ and $\mathbf{s}^*$ implies $\{1,2,\ldots,|\mathbf{s}|\} \subseteq \{\sigma_{\mathbf{s}^*}(1), \sigma_{\mathbf{s}^*}(2), \ldots, \sigma_{\mathbf{s}^*}(N)\} \subseteq \{1,2,\ldots,|\mathbf{s}|+1\}$ with $\sigma_{\mathbf{s}^*}(N) = |\mathbf{s}|+1$ if and only if $s^*_N = -$.

Hence, for $i \in \{1,\ldots,|\mathbf{s}|\}$, we define $\alpha_{\mathbf{s}^*}(i)$ to be the largest j with $\sigma_{\mathbf{s}^*}(j) = i$, i.e.

$$\alpha_{\mathbf{s}^*}(i) := \max_{1 \leq j \leq N} \{j | \sigma_{\mathbf{s}^*}(j) = i\}.$$

Clearly, we have $\sigma_{\mathbf{s}^*}(k) = i$ if and only if $\alpha_{\mathbf{s}^*}(i-1) < k \leq \alpha_{\mathbf{s}^*}(i)$ for $i = 1,\ldots,|\mathbf{s}|$ (with $\alpha_{\mathbf{s}^*}(0) := 0$, for convenience) and we have $s^*_k = s_i$, if $k = \alpha_{\mathbf{s}^*}(i)$ for some $i \in \{1,\ldots,|\mathbf{s}|\}$, and $s^*_k = -$, otherwise. □

Note that, apart from alignments, other equivalent or very similar data structures presenting the result of global sequence comparison are discussed in the literature, denoted as *edit scripts*, *traces*, or *listings* [79, 214, 51, 130, 119, 204, 143, 140]. In biological applications, the term *alignment* and the above definitions have become generally accepted.

2.2.2 Multiple Sequence Alignment

The concept of global pairwise alignment can be extended to the comparison of more than two sequences in a straightforward way.

Definition 2.5 (Multiple Sequence Alignment)
Consider a family[2] $\langle \mathbf{s}_1, \ldots, \mathbf{s}_k \rangle$ of k sequences over $\mathcal{A}$. A *multiple alignment* of $\langle \mathbf{s}_1, \ldots, \mathbf{s}_k \rangle$ is given by a $k \times N$ matrix

$$A = \begin{pmatrix} s^*_{1,1} & s^*_{1,2} & \cdots & s^*_{1,N} \\ \vdots & \vdots & & \vdots \\ s^*_{k,1} & s^*_{k,2} & \cdots & s^*_{k,N} \end{pmatrix}$$

for some N, $\max\{|\mathbf{s}_1|, \ldots, |\mathbf{s}_k|\} \leq N \leq \sum_{i=1}^{k} |\mathbf{s}_i|$, where

- $s^*_{i,j} \in \mathcal{A} \cup \{-\}$ for all $1 \leq i \leq k$, $1 \leq j \leq N$,
- for each $i = 1, \ldots, k$, the row $\mathbf{s}^*_i := s^*_{i,1} s^*_{i,2} \ldots s^*_{i,N}$ reproduces the sequence $\mathbf{s}_i$ upon ignoring all of its blanks, and
- A does not contain any column consisting of blanks only.

The set of all alignments of $S = \langle \mathbf{s}_1, \ldots, \mathbf{s}_k \rangle$ is denoted by $\mathbb{A}(S)$. □

Similar to sequences, alignments with the same number of rows may be concatenated by the $+\!\!+$ operator.

Definition 2.6 (Projection)
Consider a family of sequences S and an alignment $A \in \mathbb{A}(S)$. Given a subfamily $S' \subseteq S$, the alignment obtained by extracting from A all rows corresponding to the sequences in S' – where the columns consisting of blanks only are removed – is called the *projection* of A on S', denoted by $\pi_{S'}(A)$. □

Definition 2.7 (Combination)
Consider a family of sequences S and subfamilies $S_1, \ldots, S_n \subseteq S$. Given alignments $A_1 \in \mathbb{A}(S_1), \ldots, A_n \in \mathbb{A}(S_n)$, an alignment $A \in \mathbb{A}(S)$ is called a *combination* of the $A_1, \ldots, A_n$, if $\pi_{S_i}(A) = A_i$ for all $i \in \{1, \ldots, n\}$. □

Definition 2.8 (Compatibility of Alignments)
Consider, as above, a family of sequences S and subfamilies $S_1, \ldots, S_n \subseteq S$. Alignments $A_1 \in \mathbb{A}(S_1), \ldots, A_n \in \mathbb{A}(S_n)$ are *compatible* if there exists a combination $A \in \mathbb{A}(S)$ of the $A_1, \ldots, A_n$ such that $\pi_{S_i}(A) = A_i$ for all $i \in \{1, \ldots, n\}$. □

[2] Note the use of angle brackets $\langle \ldots \rangle$ to designate a *family* of sequences instead of the usual set notation with braces. Sequence families in our context are *lists* of sequences which – in contrast to sets – are ordered and may contain more than one identical element. To denote subfamilies, we use the standard symbol $\subseteq$ from set notation.

Lemma 2.9

Let $\langle \mathbf{s}_1, \ldots, \mathbf{s}_k \rangle$ be a family of sequences and $A_{i,j} \in \mathbb{A}(\langle \mathbf{s}_i, \mathbf{s}_j \rangle)$ be pairwise alignments for all (i,j), $1 \leq i < j \leq k$. If for every three sequences $\mathbf{s}_p$, $\mathbf{s}_q$, $\mathbf{s}_r$, the alignments $A_{p,q}$, $A_{p,r}$, and $A_{q,r}$ are compatible, then all of the alignments $A_{i,j}$ are also *simultaneously* compatible. □

Proof

We show by induction on k how to construct a combination A of the $A_{i,j}$:

For $k = 3$, the assertion trivially holds.

Assume $k > 3$. Let $A_{k-1} \in \mathbb{A}(\langle \mathbf{s}_1, \ldots, \mathbf{s}_{k-1} \rangle)$ be a combination of the alignments $A_{i,j}$, $1 \leq i < j \leq k-1$ (which exists due to the induction hypothesis). A new alignment $A_k \in \mathbb{A}(\langle \mathbf{s}_1, \ldots, \mathbf{s}_k \rangle)$ is constructed from A_{k-1} by adding a kth row, filled with the letters of $\mathbf{s}_k$ in a left-to-right order by the following procedure: Let a be a letter of $\mathbf{s}_k$. If in one of the pairwise alignments $A_{i,k}$, $1 \leq i < k$, a is aligned with a letter b of sequence $\mathbf{s}_i$, we place a in the same column where b occurs in A_{k-1} (due to the triplewise compatibility, no conflicts can arise here). If a is not aligned with a letter in any of the pairwise alignments $A_{i,k}$, a new column is inserted in the alignment filled with blanks in all rows except the last one where the a is placed. Finally, those positions of the new row where no letter of $\mathbf{s}_k$ is placed are filled with blanks.

Obviously, by this process, the projections on $\langle \mathbf{s}_i, \mathbf{s}_j \rangle$ with $i < j < k$ remain unchanged. Since also all projections on $\langle \mathbf{s}_i, \mathbf{s}_k \rangle$, $i < k$, are (compatibly) contained in A_k, we obtain a combination of all the $A_{i,j}$, $1 \leq i < j \leq k$. □

2.3 Alignment Scores

In many contexts it is important to assess the "quality" of an alignment, i.e. to determine if a given alignment is near to the one that would be produced by a biologist solving the problem "manually". As there seems to be no universally applicable way of quantifying the biological correctness of an alignment, usually (real-valued) functions are defined on the set of alignments, so-called *score functions*. Designing score functions in a way that reasonable alignments are scored extremely (high or low), the biological alignment problem is translated into an optimization problem on the set of alignments. The difficulty of quantifying the correctness of alignments still remains in form of finding adequate score functions.

2.3.1 Single Letter Substitutions

All commonly used definitions of alignment scores are based on the comparison of pairs of individual letters. Such a letter-to-letter comparison is based on the simplifying assumption that the nucleotides/amino acids at different sites of a sequence

have evolved independently. Therefore, only a (usually symmetric) *substitution score function* is needed which quantifies, for each pair of letters $(a, b) \in \mathcal{A}^2$, the preference of aligning letter a with letter b, represented by an $|\mathcal{A}| \times |\mathcal{A}|$ *substitution matrix* (see e.g. [68, 50, 86, 99]).

In general, there are two natural ways of defining substitution score functions:

(a) in form of a *similarity score* $s : \mathcal{A}^2 \to \mathbb{R}$ where letters (a, b) with similar properties are scored with high (positive) values and different letters are scored with low (negative) values, or

(b) in form of a *distance score* $d : \mathcal{A}^2 \to \mathbb{R}_{\geq 0}$ where pairs of similar letters are scored with small values while the alignment of different letters is penalized by a high distance score.

The simplest possible distance score is the *unit cost function*: Non-identical letters are scored with the value 1, identities are scored with 0. The unit cost function is member of a broad class of distance score functions with interesting and useful properties, the *metric functions*: $d : \mathcal{A}^2 \to \mathbb{R}_{\geq 0}$ is said to be a *metric* on $\mathcal{A}$ if, in addition to symmetry, the *zero property* $d(a, b) = 0 \Leftrightarrow a = b$ and the *triangle inequality* $d(a, b) \leq d(a, c) + d(c, b)$ hold for all $a, b, c \in \mathcal{A}$. In case d is a metric on $\mathcal{A}$, Sellers [179] and Waterman *et al.* [225] showed that for global sequence alignments, the metric properties carry over to the set of sequences over $\mathcal{A}$.

On the other hand in the context of local alignments, *similarity scores* with positive and negative values have been shown to be superior to distance scores [185]: In a similarity score, the value zero has a special meaning since it separates the positively correlated pairs of letters from the negatively correlated pairs. Thus, a series of positively scored substitutions in an alignment indicates highly similar regions of the involved sequences.

For global sequence alignment it has been shown that under certain conditions both distance and similarity score are equivalent [184] and it is easy to derive the corresponding similarity score function from a given distance score function and vice versa [186, 228].

Throughout this thesis, we will restrict the discussion of alignment scores and algorithms to distances. This choice is arbitrary, and with little changes in some definitions it is possible to apply the procedures developed in the following chapters also to a similarity score function. Furthermore, the algorithms described in this thesis do not require any of the metric properties except symmetry. Thus, the score matrices commonly used in biological applications which are non-metric (e.g. the PAM [50] and BLOSUM [99] series of amino acid substitution matrices) are applicable.

The effort made for the invention of biologically reasonable score functions differs significantly for nucleic acids and for amino acids. For nucleic acids, often rather simple score functions are used[3]. Beside the unit cost function mentioned above, score functions are largely established which distinguish only between transitions

[3]This trend may change in the near future: Very recently, Agarwal and States [1] published a nucleic acid substitution matrix based on an advanced statistical model.

(exchanges of two purines or of two pyrimidines) and transversions (all other combinations) [168, 116]. For amino acid sequences, more elaborated score matrices are used. They reflect differences in the genetic code [68, 71], chemical properties of the amino acids [92, 86, 137, 139], secondary structural propensities of amino acids in proteins [127, 150, 138], or – among which are the most popular ones – they are derived from empirical data by counting "true" matches and mismatches in databases of structural alignments and by computing the most reasonable substitution probabilities which might have led to the observed exchanges [50, 162, 7, 111, 99, 24, 156]. See [212] for a recent comparison of more than 40 different amino acid substitution matrices.

2.3.2 Pairwise Alignment Score

As in the discussion of alignments, we begin with the case of two sequences $\mathbf{s}$ and $\mathbf{t}$. Based on letter-to-letter distances, there is a natural approach of defining a *pairwise alignment score*, motivated by the following consideration: Sequences $\mathbf{s}$ and $\mathbf{t}$ are supposed to have a low overall distance if it is possible to find an alignment such that at many positions the paired letters have low distances. In the case of aligning proteins this can be read as: If the chains of amino acids in both proteins contain similar residues over their whole length, the proteins are supposed to have similar main-chain folding patterns and likely related functions [13]. Of course, this is an idealistic view and the existence of false negatives cannot be excluded. Procedures which weight different positions of a biological sequence according to their mutability or importance have been suggested [64, 152, 98, 174, 175].

Formally speaking, we extend the distance score function d to a function

$$d^* : (\mathcal{A} \cup \{-\})^2 \to \mathbb{R}_{\geq 0}$$

which is identical to d for all pairs of letters, i.e. $d^*(a,b) := d(a,b)$ for all $a, b \in \mathcal{A}$. Additionally, aligning a letter from $\mathcal{A}$ with a blank is penalized by a high value, usually a constant $d^*(a,-) = d^*(-,a) := \gamma$ for all $a \in \mathcal{A}$. Hence, a consecutive string of l blanks receives the value $l\,\gamma$. That is the reason for calling this way of scoring blanks also *additive* or *homogeneous gap costs*.

The alignment score of A is then defined as the sum of the d^*-values over all sites of the alignment:

Definition 2.10 (Alignment Score)

Assume sequences $\mathbf{s}$ and $\mathbf{t}$ over $\mathcal{A}$. Given an alignment $A = \begin{pmatrix} \mathbf{s}^* \\ \mathbf{t}^* \end{pmatrix} \in \mathrm{A}(\langle \mathbf{s}, \mathbf{t} \rangle)$ of length N and an extended distance score function $d^* : (\mathcal{A} \cup \{-\})^2 \to \mathbb{R}_{\geq 0}$, we define the *alignment score* of A with respect to d^* by

$$w_{d^*}(A) \;:=\; \sum_{1 \leq i \leq N} d^*(s_i^*, t_i^*).$$

□

Note that w_{d^*} is strongly additive: For all pairwise alignments A and B,

$$w_{d^*}(A{+\!\!+}B) \quad = \quad w_{d^*}(A) + w_{d^*}(B).$$

As described in the introduction to this chapter, there is a strong biological motivation to score *gaps*, i.e. runs of consecutive blanks in an aligned sequence, other than as the sum of the corresponding number of single letter indels since they may stem from a single mutational event. Thus, instead of summing the values $d^*(a_i, -)$ along the letters a_i of a gap, often a more sophisticated model in form of a length-dependent *gap (penalty) function* is considered [72, 80, 119, 218, 197, 81, 9, 145, 136, 146, 74, 82, 41, 221]. Sometimes, the gap function also depends on the letters that are deleted or inserted [225, 121, 90] or on additional information, e.g. known secondary structure of one or both sequences [228, 125, 20, 183, 200, 173].

In the following, we restrict our discussion to gap penalty functions $g(l)$ where l is the length of the gap. Here, the score of an alignment $A \in \mathbb{A}(\langle \mathbf{s}, \mathbf{t} \rangle)$ is computed as follows:

Definition 2.11 (Alignment Score with Length-Dependent Gap-Costs)
Let $\mathbf{s}$ and $\mathbf{t}$ be sequences over $\mathcal{A}$. Given an alignment $A = \begin{pmatrix} \mathbf{s}^* \\ \mathbf{t}^* \end{pmatrix} \in \mathbb{A}(\langle \mathbf{s}, \mathbf{t} \rangle)$ of length N, a distance score function $d : \mathcal{A}^2 \to \mathbb{R}_{\geq 0}$, and a length-dependent gap function $g(l) : \mathbb{N}_{\geq 1} \to \mathbb{R}_{\geq 0}$, we define the *alignment score* of A with respect to d and g by

$$w_{d,g}(A) \quad := \quad \sum_{\substack{1 \leq i \leq N \\ s_i^* \neq - \neq t_i^*}} d(s_i^*, t_i^*) \; + \; \sum_{l>0} g(l) \cdot \#\text{gaps of length } l \text{ contained in } A.$$

□

In some contexts we do not need identify the distance function d and gap penalty function g, and omit the indices.
Of course, for the biological reasons mentioned above, the penalty for a gap should never exceed the sum of the penalties given to its parts, i.e. $g(l)$ should be *subadditive*:

$$\begin{aligned} g(l_1 + l_2 + \ldots + l_k) \quad &\leq \quad g(l_1) + g(l_2) + \ldots + g(l_k) \\ \iff \quad g(l+m) \quad &\leq \quad g(l) + g(m) \end{aligned}$$

(the equivalence can be easily shown by induction on k).

From the subadditivity of the gap function, the subadditivity of the corresponding alignment score can be derived:

Lemma 2.12 (Subadditivity of Alignment Score)
Given a distance score function d and subadditive gap penalties $g(l)$, the alignment score $w_{d,g}$ is subadditive upon concatenation of alignments,

$$w_{d,g}(A{+\!\!+}B) \quad \leq \quad w_{d,g}(A) + w_{d,g}(B) \quad \text{for all pairwise alignments } A \text{ and } B.$$

□

Proof
Assume $A = \begin{pmatrix} s_1^* & \cdots & s_N^* \\ t_1^* & \cdots & t_N^* \end{pmatrix} \in \mathbb{A}(\langle \mathbf{s}, \mathbf{t} \rangle)$ and $B = \begin{pmatrix} u_1^* & \cdots & u_M^* \\ v_1^* & \cdots & v_M^* \end{pmatrix} \in \mathbb{A}(\langle \mathbf{u}, \mathbf{v} \rangle)$. We consider three cases:

1. $\mathbf{s}^*$ ends with a gap of length, say, $l \geq 1$, and $\mathbf{u}^*$ begins with a gap of length $m \geq 1$. In this case,

$$\begin{aligned} w_{d,g}(A{+\!\!+}B) &= w_{d,g}\begin{pmatrix} s_1^* & \cdots & s_{N-l}^* \\ t_1^* & \cdots & t_{N-l}^* \end{pmatrix} + g(l+m) + w_{d,g}\begin{pmatrix} u_{m+1}^* & \cdots & u_M^* \\ v_{m+1}^* & \cdots & v_M^* \end{pmatrix} \\ &\leq w_{d,g}\begin{pmatrix} s_1^* & \cdots & s_{N-l}^* \\ t_1^* & \cdots & t_{N-l}^* \end{pmatrix} + g(l) + g(m) + w_{d,g}\begin{pmatrix} u_{m+1}^* & \cdots & u_M^* \\ v_{m+1}^* & \cdots & v_M^* \end{pmatrix} \\ &= w_{d,g}(A) + w_{d,g}(B). \end{aligned}$$

2. In case $\mathbf{t}^*$ ends with a gap and $\mathbf{v}^*$ starts with a gap, the proposition holds by symmetric logic.

3. In all other cases, the score $w_{d,g}(A{+\!\!+}B)$ is simply the sum of $w_{d,g}(A)$ and $w_{d,g}(B)$ since all gap lengths remain unchanged upon concatenation of A and B.

□

In the literature, some special subadditive gap functions $g(l)$ are discussed. Most prominent are (*affine*) *linear functions* of the form $g(l) = \alpha + \beta l$ with $\alpha \neq 0$ (see e.g. [80, 119, 72, 9, 82, 145, 58, 41, 221]) and *concave functions*, i.e. $g(l)$ which fulfill the *quadrangle inequality* $g(l+m) - g(l) \geq g(l+m+n) - g(l+n)$ for all $l, m, n \geq 1$ (see [218, 136, 74, 59, 41]). Subadditivity of these functions can be shown in a straightforward way [218, 205].

In our description of the divide-and-conquer alignment algorithm, we will restrict the alignment scores to functions $w_{d,g}$ with linear gap costs $g(l) = \alpha + \beta l$ which are the most commonly used in biological applications and for which efficient alignment algorithms exist [80, 136]. Other gap functions are applicable with a lack of speed.

Of further importance in the discussion of biologically reasonable gap penalty functions is how to score *terminal gaps*, i.e. gaps at the beginning or at the end of an alignment. To make the compensation of differences in the length of the sequences free of charge, those "necessary gaps" should be scored different from internal gaps [161, 180, 72] which is also known as *free shift* of the sequences. In score functions based on similarity scores, this is easily achieved by scoring terminal gaps with zero. While Fitch and Smith [72] claim that "distance algorithms cannot be defined for unweighted end gaps", we adopt the method of Gupta *et al.* [90]: In terminal gaps, only the length-dependent part βl of the gap is scored, while the fixed portion α is omitted.

However, to keep the exposition simpler, we do not discuss free shift when introducing the alignment algorithms. In the implementation which we describe in Chapter 6, we have incorporated the free shift of the sequences.

2.3.3 Multiple Sequence Alignment Score

Historically, several approaches of generalizing scores to alignments of more than two sequences have been proposed. The first approaches were *tree alignment methods* [166, 167, 11, 95]. A pre-given (mostly unrooted) phylogenetic tree is considered. The leaves of the tree represent the given sequences to be aligned while inner nodes of the phylogenetic tree correspond to *ancestor* sequences which are usually unknown. Defining the *length* of an edge in such a tree as the pairwise alignment score of the sequences connected by the edge, the *tree alignment score* is simply the sum of all edge lengths in the tree.

In *generalized tree alignment methods* [167, 105, 95, 96, 91, 211], the topology of the tree is not given in advance. The tree and the alignment are both to be calculated.

The following approach, suggested by G. Gonnet [77] is motivated by tree alignment. Assuming that the sequences are related by a tree structure, it is generally easier to find a cyclic order of the sequences than finding the correct phylogenetic tree. The sum of the pairwise projection costs of sequences which are adjacent in a minimum cost cycle defines the *cyclic alignment score*.

In the last decade, the so-called *sum-of-pairs* (SP) score, defined as the sum of the scores of *all* pairwise projections of a multiple alignment, has received large attention (cf. [142, 81, 15, 39, 11, 177, 159, 91, 16, 215]). Sometimes, the projection costs are additionally weighted according to sequence-dependent (non-negative) weight factors [8, 6, 90]. The weights, in the following denoted by $\alpha_{p,q}$, reflect e.g. phylogenetic relationship of the sequences. For the moment, assume the weights as given. A more detailed discussion follows in Section 2.5.

Definition 2.13 (Weighted SP Multiple Sequence Alignment Score)
Assume an alignment $A \in \mathbb{A}(\langle \mathbf{s}_1, \ldots, \mathbf{s}_k \rangle)$, a pairwise alignment score function w, and weight factors $\alpha_{p,q} \in \mathbb{R}_{\geq 0}$ for all (p,q), $1 \leq p < q \leq k$. The *weighted sum-of-pairs multiple sequence alignment score* of A is given by

$$w(A) \quad := \sum_{1 \leq p < q \leq k} \alpha_{p,q}\, w(\pi_{\langle \mathbf{s}_p, \mathbf{s}_q \rangle}(A)).$$

□

Obviously, subadditivity of the alignment score upon concatenation of pairwise alignments (Lemma 2.12) carries over to weighted SP multiple alignment score.

Note that the cyclic alignment score mentioned above, for a given cyclic order of the sequences, is just a special case of the weighted SP score with the weight factors $\alpha_{p,q}$ set to 1 for sequences $\mathbf{s}_p$ and $\mathbf{s}_q$ which are neighbors in the cycle and 0 otherwise. But, of course, a direct computation of the cyclic alignment score can be more efficient than proceeding this way since only as many pairwise scores have to be considered as there are sequences involved, instead of a quadratic number of pairwise projections in the general case.

In contrast to the approaches discussed so far, in a large number of heuristic alignment procedures (e.g. [109, 20, 66, 101, 201, 133, 223, 199, 42, 22, 163, 21, 39, 129, 155, 10]) much more complicated objective functions are considered. Some base on sophisticated quality models including additional knowledge about the sequences involved and about estimated as well as pre-given phylogenetic relationships, others use less transparent heuristics to speed up the computation. But the lack of a simple well-defined score which is optimized or approximated does not necessarily imply biologically unreasonable alignments.

In principle, the divide-and-conquer alignment method can be applied to any of the introduced alignment scores. In the further discussion, we focus on the weighted SP score. With this approach the least amount of additional knowledge about the sequences is required although methods based on evolutionary trees might be closer to biological intuition [11].

2.4 The Problem

With the definitions of the previous section, a formalization of the multiple sequence alignment problem is straightforward. It is based on the largely accepted *parsimony principle* [38, 71, 70, 169]: Of all possible mutational paths for one sequence to change into another, we expect the shortest one, i.e. the one with the least amount of evolution, to approximate the true history best. Note that the parsimony principle can only underestimate, but never overestimate the true evolutionary distance.

Multiple Sequence Alignment Problem
Given a family of sequences $S = \langle \mathbf{s}_1, \ldots, \mathbf{s}_k \rangle$ and a distance-based alignment score function $w : \mathbb{A}(S) \to \mathbb{R}_{\geq 0}$, find an alignment $A \in \mathbb{A}(S)$ that minimizes $w(A)$. □

Such an alignment is called an *optimal alignment* of $S = \langle \mathbf{s}_1, \ldots, \mathbf{s}_k \rangle$ with respect to w, its score is referred to as the *optimal alignment score* of S, short

$$w_{opt}(S) \quad := \quad \min\{w(A) | A \in \mathbb{A}(S)\}.$$

Note that optimal alignments are not necessarily unique.

By simple reasoning, the subadditivity property discussed above for the concatenation of pre-given alignments (see Lemma 2.12 and the remark in Section 2.3.3) carries over to optimal alignment scores:

Lemma 2.14 (Subadditivity of Optimal Alignment Score)
Assume sequence families $\langle \mathbf{s}_1, \ldots, \mathbf{s}_k \rangle$ and $\langle \mathbf{t}_1, \ldots, \mathbf{t}_k \rangle$. Given a score function $w_{d,g}$ with subadditive gap penalty function $g(l)$, the following holds:

$$w_{opt}(\langle \mathbf{s}_1 +\!\!+ \mathbf{t}_1, \ldots, \mathbf{s}_k +\!\!+ \mathbf{t}_k \rangle) \quad \leq \quad w_{opt}(\langle \mathbf{s}_1, \ldots, \mathbf{s}_k \rangle) + w_{opt}(\langle \mathbf{t}_1, \ldots, \mathbf{t}_k \rangle).$$

□

Proof

Let A and B be optimal alignments of $\langle \mathbf{s}_1, \ldots, \mathbf{s}_k \rangle$ and $\langle \mathbf{t}_1, \ldots, \mathbf{t}_k \rangle$, respectively. Since the optimal alignment score is the minimum over all possible alignment scores of the sequences under consideration,

$$\begin{aligned} w_{opt}(\langle \mathbf{s}_1 +\!\!+ \mathbf{t}_1, \ldots, \mathbf{s}_k +\!\!+ \mathbf{t}_k \rangle) &\leq w_{d,g}(A +\!\!+ B) \\ &\leq w_{d,g}(A) + w_{d,g}(B) \\ &= w_{opt}(\langle \mathbf{s}_1, \ldots, \mathbf{s}_k \rangle) + w_{opt}(\langle \mathbf{t}_1, \ldots, \mathbf{t}_k \rangle). \end{aligned}$$

□

In some cases, solving the multiple sequence alignment problem is trivial when optimal pairwise alignment are given:

Lemma 2.15 (Combination of Optimal Alignments)

Assume a family of sequences $\langle \mathbf{s}_1, \ldots, \mathbf{s}_k \rangle$ and triplewise compatible pairwise alignments $A_{i,j} \in \mathbb{A}(\langle \mathbf{s}_i, \mathbf{s}_j \rangle)$ for all (i,j), $1 \leq i < j \leq k$. If each $A_{i,j}$ is an *optimal* alignment of $\mathbf{s}_i$ and $\mathbf{s}_j$, then each combination A of the $A_{i,j}$, $1 \leq i < j \leq k$, is a (weighted) SP-optimal *multiple* alignment of $\langle \mathbf{s}_1, \ldots, \mathbf{s}_k \rangle$. □

Proof

Let A be a combination of the $A_{i,j}$ (which exists due to Lemma 2.9), and let A_{opt} be an optimal multiple alignment of $\langle \mathbf{s}_1, \ldots, \mathbf{s}_k \rangle$. Then, by definition,

$$\sum_{1 \leq p < q \leq k} \alpha_{p,q}\, w(\pi_{\langle \mathbf{s}_p, \mathbf{s}_q \rangle}(A_{opt})) \;=\; w(A_{opt}) \;\leq\; w(A) \;=\; \sum_{1 \leq p < q \leq k} \alpha_{p,q}\, w_{opt}(\langle \mathbf{s}_p, \mathbf{s}_q \rangle).$$

On the other hand, for all (p,q), $1 \leq p < q \leq k$,

$$w(\pi_{\langle \mathbf{s}_p, \mathbf{s}_q \rangle}(A_{opt})) \;\geq\; w_{opt}(\langle \mathbf{s}_p, \mathbf{s}_q \rangle).$$

Together, this implies

$$w(\pi_{\langle \mathbf{s}_p, \mathbf{s}_q \rangle}(A_{opt})) \;=\; w_{opt}(\langle \mathbf{s}_p, \mathbf{s}_q \rangle)$$

for all (p,q), $1 \leq p < q \leq k$, since all summands are non-negative. Thus,

$$w(A_{opt}) \;=\; w(A),$$

i.e. A is also an optimal alignment of $\langle \mathbf{s}_1, \ldots, \mathbf{s}_k \rangle$. □

It is well-known that the multiple sequence alignment problem is equivalent to the problem of finding a shortest path in a directed graph with $\prod_{1 \leq i \leq k}(|\mathbf{s}_i| + 1)$ nodes, a single source vertex and a single sink vertex, the so-called *alignment graph* (see e.g. [121, 144]). This analogy will not be considered in further detail in this thesis, although several insights from graph theory have been successfully adapted to alignment problems.

Finally, we again have to emphasize that with the formalization of the alignment problem only a mathematical framework for the quality assessment of alignments is given. But an alignment which is optimal with respect to an arbitrary score function $w_{d,g}$ does not guarantee to be biologically meaningful. The choice of substitution function d and gap penalty function g is very important. Furthermore, there does not seem to exist a single score function $w_{d,g}$ which, applied to any set of biological sequences, would result in an alignment that perfectly matches the biologically correct one. Thus, the choice of adequate alignment parameters is and will probably remain for a long time to be a difficult problem in computational molecular biology [67, 78, 221]. Indeed, it has been shown that alignments optimized with respect to additional criteria such as structural models can be closer to the truth than alignments based on simple numerical score functions as we have defined here [87, 88].

2.5 Sequence Weighting

How to choose weight factors in a multiple sequence alignment appropriate for a given task is an important and much studied scientific topic (cf. [206, 182, 8, 6, 209]). In some contexts, weights attached to individual sequences are used to avoid overweighting redundant information which can arise e.g. from some identical or very similar sequences in the data set. These problems come up when *profiles* (probability vectors representing the relative frequencies of occurrence of each letter in an alignment site) are aligned [206, 182, 201, 85], or when edge lengths in a pre-given evolutionary tree are determined [8]. In the weighted SP alignment score, weights are not required for individual sequences but for pairs of sequences. Such weights are often computed from pairwise distances of the sequences in an estimated evolutionary tree [8, 6]. Higher weights are given to the more similar pairs, as having them aligned optimally should be more important than aligning two fairly unrelated sequences optimally at the expense of worsening a good alignment of closely related sequences. But in contrast to these methods, we do explicitly not want to use a tree as a basis for our weights because such weights might bias the alignment in the direction of a certain phylogeny. So, we decided to develop much simpler weights computed from the pairwise distances of the sequences, only.

In [203], the following weights are suggested:

$$\alpha_{p,q} \quad := \quad \frac{minscore}{w_{opt}(\langle \mathbf{s}_p, \mathbf{s}_q \rangle)}$$

where $minscore := \min_{1 \le i < j \le k} \{w_{opt}(\langle \mathbf{s}_i, \mathbf{s}_j \rangle)\}$. This formula gives rise to two remarks:

1. Assume two identical sequences $\mathbf{s}_p$ and $\mathbf{s}_q$ and a metric score function w such that $minscore = w_{opt}(\langle \mathbf{s}_p, \mathbf{s}_q \rangle) = 0$. In this case, $\alpha_{p,q}$ is undefined.

2. An alignment of short sequences usually has a lower absolute score than the alignment of long sequences simply because the corresponding alignments also

differ in length and the global alignment score increases linearly with the length of an alignment [44, 221]. Thus, in the above weighting scheme, pairs of short sequences in general receive higher weight factors than pairs of long sequences.

Due to the second remark, we use *length-normalized scores* as introduced by Feng *et al.* [67]:

$$\overline{w}_{opt}(\langle \mathbf{s}_p, \mathbf{s}_q \rangle) \quad := \quad \frac{w_{opt}(\langle \mathbf{s}_p, \mathbf{s}_q \rangle)}{\min\{|\mathbf{s}_p|, |\mathbf{s}_q|\}}$$

rather than absolute alignment scores. The division by the length of the *shorter* sequence follows from the same rationale as not to penalize dangling ends of longer sequences in an alignment. A more sophisticated normalization procedure of alignment scores is discussed by Johnson and Doolittle [109], but it matches the philosophy of our approach to use this rather simple technique.

In conclusion, we propose the following formula for the computation of pairwise weight factors for the SP alignment with the divide-and-conquer method:

$$\alpha_{p,q} \quad := \quad \begin{cases} 1 & \text{if } \mathit{maxscore} = 0 \\ 1 - \lambda \cdot \frac{\overline{w}_{opt}(\langle \mathbf{s}_p, \mathbf{s}_q \rangle) - \mathit{minscore}}{\mathit{maxscore}} & \text{otherwise} \end{cases}$$

where $\mathit{minscore} := \min_{1 \le i < j \le k}\{\overline{w}_{opt}(\langle \mathbf{s}_i, \mathbf{s}_j \rangle)\}$, $\mathit{maxscore} := \max_{1 \le i < j \le k}\{\overline{w}_{opt}(\langle \mathbf{s}_i, \mathbf{s}_j \rangle)\}$, and λ is a (user-specifiable) parameter, the *weight intensity*. The weight intensity may be set to any value between $\lambda = 1$ (maximum weighting: smallest weights for (p, q) with $\overline{w}_{opt}(\mathbf{s}_p, \mathbf{s}_q) = \mathit{maxscore}$, highest weights $\alpha_{p,q} = 1$ for (p, q) with $\overline{w}_{opt}(\mathbf{s}_p, \mathbf{s}_q) = \mathit{minscore}$) and $\lambda = 0$ (no weighting: $\alpha_{p,q} = 1$ for all $1 \le p < q \le k$).

Chapter 3

Calculating Distances: Dynamic Programming

In this chapter, we discuss the standard method to compute optimal solutions of the multiple sequence alignment problem: *dynamic programming*. Due to the large number of alignments for reasonably long sequences (see Equations 2.1 – 2.3), the naive way of enumerating all alignments and then to choose the one with the smallest score cannot be the method of choice. Historically, the use of dynamic programming algorithms for sequence comparison has several independent sources. Among the most frequently cited are Needleman and Wunsch [149] for biological applications and Wagner and Fischer [214] in the computer science field. See [119] for details about the invention process.

While the algorithms given in the literature differ slightly, the general concept is always the same. We begin with describing the simplest, most basic pairwise algorithm, followed by extensions for affine gap penalty functions and for more than two sequences. Finally, we present a famous and elegant branch-and-bound approach which speeds up the computation in many cases.

3.1 Optimal Alignment of Two Sequences

We consider the problem of aligning two sequences $\mathbf{s}$ and $\mathbf{t}$. The *standard dynamic programming* algorithm for sequence alignment proceeds in two stages. The first stage finds the optimal alignment score, and an optional second stage can be used to find an optimal alignment if desired. In the following, we will focus on the first stage and describe the second stage only briefly because this stage does not affect the divide-and-conquer procedure.

The essential data structure used by dynamic programming algorithms is the *distance matrix*. In the context of sequence alignment, it contains the optimal alignment scores of all prefixes of $\mathbf{s}$ with all prefixes of $\mathbf{t}$:

Definition 3.1 (Distance Matrix)
The *distance matrix* of sequences $\mathbf{s}$ and $\mathbf{t}$ is defined by

$$D_{\mathbf{s},\mathbf{t}} \quad := \quad (D_{\mathbf{s},\mathbf{t}}[i,j])_{0 \le i \le |\mathbf{s}|, 0 \le j \le |\mathbf{t}|}$$

where

$$D_{\mathbf{s},\mathbf{t}}[i,j] \quad := \quad w_{opt}(\langle \mathbf{s}^i, \mathbf{t}^j \rangle)$$

for all (i,j), $0 \le i \le |\mathbf{s}|$, $0 \le j \le |\mathbf{t}|$. □

In the simple case of a homogeneous gap function $g(l) = \gamma\, l$, the distance matrix can be computed by the well-known dynamic programming recurrence

$$\begin{array}{lcl} D_{\mathbf{s},\mathbf{t}}[0,0] & := & 0 \\ D_{\mathbf{s},\mathbf{t}}[i,0] & := & D_{\mathbf{s},\mathbf{t}}[i-1,0] + \gamma \\ D_{\mathbf{s},\mathbf{t}}[0,j] & := & D_{\mathbf{s},\mathbf{t}}[0,j-1] + \gamma \\ D_{\mathbf{s},\mathbf{t}}[i,j] & := & \min \left\{ \begin{array}{c} D_{\mathbf{s},\mathbf{t}}[i-1,j-1] + d(s_i,t_j), \\ D_{\mathbf{s},\mathbf{t}}[i-1,j] + \gamma, \\ D_{\mathbf{s},\mathbf{t}}[i,j-1] + \gamma \end{array} \right\} \end{array} \tag{3.1}$$

for all i, $1 \le i \le |\mathbf{s}|$, and j, $1 \le j \le |\mathbf{t}|$.

The initializations in the upper three lines are obvious. The correctness of the recurrence in the fourth line can easily be verified by considering the last column of an optimal alignment A of $\mathbf{s}^i$ and $\mathbf{t}^j$:

- Either A ends with a substitution of s_i by t_j, $A = \begin{pmatrix} \cdots & s_i \\ \cdots & t_j \end{pmatrix}$. In this case, $w_{opt}(\langle \mathbf{s}^i, \mathbf{t}^j \rangle)$ is the score $D_{\mathbf{s},\mathbf{t}}[i-1,j-1]$ of an optimal alignment of $\mathbf{s}^{i-1}$ and $\mathbf{t}^{j-1}$ plus the cost $d(s_i,t_j)$ for substituting s_i by t_j.

- Assume now A ends with a deletion of s_i, i.e. $A = \begin{pmatrix} \cdots & s_i \\ \cdots & - \end{pmatrix}$. In this case, the score of an optimal alignment of $\mathbf{s}^i$ and $\mathbf{t}^j$ is the optimal alignment score $D_{\mathbf{s},\mathbf{t}}[i-1,j]$ of the prefixes $\mathbf{s}^{i-1}$ and $\mathbf{t}^j$ plus the cost γ for aligning s_i with the blank.

- The third case $A = \begin{pmatrix} \cdots & - \\ \cdots & t_j \end{pmatrix}$ is symmetric to the second one.

Thus, the optimal alignment score $w_{opt}(\langle \mathbf{s}^i, \mathbf{t}^j \rangle)$ can be obtained by computing the three-way minimum given in (3.1).

A straightforward computation order is to proceed column by column (or row by row) but any other order consistent with the data dependencies of (3.1) is possible as well. Every entry in $D_{\mathbf{s},\mathbf{t}}$ is computed in constant time leading to an $\mathcal{O}(|\mathbf{s}| \cdot |\mathbf{t}|)$ time complexity to compute $w_{opt}(\langle \mathbf{s}, \mathbf{t} \rangle) = D_{\mathbf{s},\mathbf{t}}[|\mathbf{s}|, |\mathbf{t}|]$.

Once the distance matrix $D_{\mathbf{s},\mathbf{t}}$ is computed, the second stage of the alignment algorithm is then rather simple: An optimal alignment is recovered by tracing back

from the cell $D_{\mathbf{s},\mathbf{t}}[|\mathbf{s}|,|\mathbf{t}|]$ through the matrix along the entries which yielded the three-way minima until cell $D_{\mathbf{s},\mathbf{t}}[0,0]$ is reached. Therefore it may be useful to save back-pointers at each cell of the distance matrix but given the matrix $D_{\mathbf{s},\mathbf{t}}$, a re-computation of the minimizations along the way back is possible as well. In both cases, storing the whole matrix of size $\mathcal{O}(|\mathbf{s}| \cdot |\mathbf{t}|)$ is necessary.

If one is interested only in the optimal alignment score of $\mathbf{s}$ and $\mathbf{t}$, $\mathcal{O}(\min\{|\mathbf{s}|,|\mathbf{t}|\})$ space suffices since for computing the jth column of $D_{\mathbf{s},\mathbf{t}}$, only entries of the $(j-1)$th column are required. Based on multiple computation of $D_{\mathbf{s},\mathbf{t}}$ using such a score-only algorithm in a divide-and-conquer manner, Hirschberg [104] showed that it is also possible to keep the amount of memory required for the computation of an optimal alignment at $\mathcal{O}(|\mathbf{s}|+|\mathbf{t}|)$ by the drawback of doubling the computation time. This is optimal space since it is required even by the sequences $\mathbf{s}$ and $\mathbf{t}$ alone.

Note that the definition of the distance matrix $D_{\mathbf{s},\mathbf{t}}$ based on the prefixes of $\mathbf{s}$ and $\mathbf{t}$, respectively, is somewhat arbitrary. In a symmetric way, the calculation of an optimal alignment is also possible beginning at the right end of the sequences, using a *reverse distance matrix*

$$D^r_{\mathbf{s},\mathbf{t}} \;:=\; (D^r_{\mathbf{s},\mathbf{t}}[i,j])_{0\le i\le|\mathbf{s}|,0\le j\le|\mathbf{t}|} \quad \text{where} \quad D^r_{\mathbf{s},\mathbf{t}}[i,j] \;:=\; w_{opt}(\langle {}^i\mathbf{s}, {}^j\mathbf{t}\rangle).$$

For alignment scores which are invariant about reverting both sequences – as all those discussed above –, the following holds:

$$D^r_{\mathbf{s},\mathbf{t}}[i,j] \;=\; D_{\mathbf{s}^{-1},\mathbf{t}^{-1}}[|\mathbf{s}|-i,|\mathbf{t}|-j].$$

So it is obvious that $D^r_{\mathbf{s},\mathbf{t}}$ can also be computed in $\mathcal{O}(|\mathbf{s}| \cdot |\mathbf{t}|)$ time. For symmetry reasons, the distance matrix $D_{\mathbf{s},\mathbf{t}}$ is also called *forward* distance matrix, denoted by $D^f_{\mathbf{s},\mathbf{t}}$.

3.2 General Gap Functions

The standard dynamic programming procedure for sequence alignment can be generalized to subadditive gap penalties $g(l)$. First approaches by Waterman *et al.* [225] have led to the following generalization of (3.1) with time complexity $\mathcal{O}(|\mathbf{s}|\cdot|\mathbf{t}|\cdot(|\mathbf{s}|+|\mathbf{t}|))$:

$$\begin{array}{lcl} D_{\mathbf{s},\mathbf{t}}[0,0] & := & 0 \\ D_{\mathbf{s},\mathbf{t}}[i,0] & := & g(i) \\ D_{\mathbf{s},\mathbf{t}}[0,j] & := & g(j) \\ D_{\mathbf{s},\mathbf{t}}[i,j] & := & \min\left\{\begin{array}{l} D_{\mathbf{s},\mathbf{t}}[i-1,j-1]+d(s_i,t_j), \\ \min\limits_{1\le l\le i}\{D_{\mathbf{s},\mathbf{t}}[i-l,j]+g(l)\}, \\ \min\limits_{1\le l\le j}\{D_{\mathbf{s},\mathbf{t}}[i,j-l]+g(l)\} \end{array}\right\} \end{array} \tag{3.2}$$

for all i, $1 \le i \le |\mathbf{s}|$, and j, $1 \le j \le |\mathbf{t}|$.

For some special gap penalty functions, more efficient algorithms have been devised. For alignment scores $w_{d,g}$ with a linear gap function $g(l) := \alpha + \beta\, l$, Gotoh [80] and other authors [9, 145] were able to reduce the required time to $\mathcal{O}(|\mathbf{s}| \cdot |\mathbf{t}|)$ as for the homogeneous case: Due to the fact that the penalties for the gaps of various length all change by the same additive constant β when the algorithm proceeds to cell $D_{\mathbf{s},\mathbf{t}}[i,j]$ from $D_{\mathbf{s},\mathbf{t}}[i-1,j]$ or from $D_{\mathbf{s},\mathbf{t}}[i,j-1]$, *history matrices* $V_{\mathbf{s},\mathbf{t}} := (V_{\mathbf{s},\mathbf{t}}[i,j])_{0\le i\le|\mathbf{s}|,0\le j\le|\mathbf{t}|}$ and $H_{\mathbf{s},\mathbf{t}} := (H_{\mathbf{s},\mathbf{t}}[i,j])_{0\le i\le|\mathbf{s}|,0\le j\le|\mathbf{t}|}$ are introduced which indicate the score of the best alignments ending with a gap in either sequence:

$$\begin{aligned} V_{\mathbf{s},\mathbf{t}}[i,j] &:= \min\{w(A) | A \in \mathbb{A}(\langle \mathbf{s}^i, \mathbf{t}^j \rangle), A = \begin{pmatrix} \cdots & s_i \\ \cdots & - \end{pmatrix}\}, \\ H_{\mathbf{s},\mathbf{t}}[i,j] &:= \min\{w(A) | A \in \mathbb{A}(\langle \mathbf{s}^i, \mathbf{t}^j \rangle), A = \begin{pmatrix} \cdots & - \\ \cdots & t_j \end{pmatrix}\} \end{aligned}$$

for all (i,j), $0 \le i \le |\mathbf{s}|$, $0 \le j \le |\mathbf{t}|$. Defining the minimum of the empty set as $+\infty$, note that $V_{\mathbf{s},\mathbf{t}}[0,j] = +\infty$ for all $j \ge 0$ and that $H_{\mathbf{s},\mathbf{t}}[i,0] = +\infty$ for all $i \ge 0$.

Using $V_{\mathbf{s},\mathbf{t}}$ and $H_{\mathbf{s},\mathbf{t}}$, (3.2) can be rewritten for affine gap functions $g(l) = \alpha + \beta\, l$:

$$\begin{aligned} D_{\mathbf{s},\mathbf{t}}[0,0] &:= 0 \\ V_{\mathbf{s},\mathbf{t}}[0,0] = H_{\mathbf{s},\mathbf{t}}[0,0] &:= +\infty \\ D_{\mathbf{s},\mathbf{t}}[i,0] = V_{\mathbf{s},\mathbf{t}}[i,0] &:= g(i) \\ H_{\mathbf{s},\mathbf{t}}[i,0] &:= +\infty \\ D_{\mathbf{s},\mathbf{t}}[0,j] = H_{\mathbf{s},\mathbf{t}}[0,j] &:= g(j) \\ V_{\mathbf{s},\mathbf{t}}[0,j] &:= +\infty \\ V_{\mathbf{s},\mathbf{t}}[i,j] &:= \min\{D_{\mathbf{s},\mathbf{t}}[i-1,j] + \alpha, V_{\mathbf{s},\mathbf{t}}[i-1,j]\} + \beta \\ H_{\mathbf{s},\mathbf{t}}[i,j] &:= \min\{D_{\mathbf{s},\mathbf{t}}[i,j-1] + \alpha, H_{\mathbf{s},\mathbf{t}}[i,j-1]\} + \beta \\ D_{\mathbf{s},\mathbf{t}}[i,j] &:= \min \left\{ \begin{matrix} D_{\mathbf{s},\mathbf{t}}[i-1,j-1] + d(s_i,t_j), \\ V_{\mathbf{s},\mathbf{t}}[i,j],\ H_{\mathbf{s},\mathbf{t}}[i,j] \end{matrix} \right\} \end{aligned} \tag{3.3}$$

for all i, $1 \le i \le |\mathbf{s}|$, and j, $1 \le j \le |\mathbf{t}|$. The proof of the equivalence of both recurrences has been shown several times (see e.g. [80, 218, 205, 221]) and is omitted here.

Though, the initializations have not always been devised very carefully. In several early publications of this algorithm [80, 197, 218] and even in a very recent one [221][1], the following initializations are suggested:

$$\begin{aligned} H_{\mathbf{s},\mathbf{t}}[i,0] &:= g(i) \qquad \text{for all } i,\ 0 \le i \le |\mathbf{s}|, \\ V_{\mathbf{s},\mathbf{t}}[0,j] &:= g(j) \qquad \text{for all } j,\ 0 \le j \le |\mathbf{t}|. \end{aligned} \tag{3.4}$$

The following example shows that this might lead to errorneous alignment scores.

[1] Upon our notification, the mistake will be corrected in the next edition of this textbook [216].

Example 3.2
Let $\mathcal{A} := \{\mathtt{A}, \mathtt{T}\}$, $d(\mathtt{A}, \mathtt{T}) := 16$, and $g(l) := 5 + 4\,l$. Consider the trivial sequences $\mathbf{s} := \mathtt{A}$ and $\mathbf{t} := \mathtt{T}$. With the initializations given in (3.4), the optimal alignment score is computed as follows:

$$\begin{aligned}
&D_{\mathbf{s},\mathbf{t}}[0,0] = 0, V_{\mathbf{s},\mathbf{t}}[0,0] = H_{\mathbf{s},\mathbf{t}}[0,0] = g(0) = 5;\\
&D_{\mathbf{s},\mathbf{t}}[1,0] = V_{\mathbf{s},\mathbf{t}}[1,0] = H_{\mathbf{s},\mathbf{t}}[1,0] = g(1) = 9;\\
&D_{\mathbf{s},\mathbf{t}}[0,1] = H_{\mathbf{s},\mathbf{t}}[0,1] = V_{\mathbf{s},\mathbf{t}}[0,1] = g(1) = 9;\\
&V_{\mathbf{s},\mathbf{t}}[1,1] = \min\{D_{\mathbf{s},\mathbf{t}}[0,1] + 5, V_{\mathbf{s},\mathbf{t}}[0,1]\} + 4 = 13,\\
&H_{\mathbf{s},\mathbf{t}}[1,1] = \min\{D_{\mathbf{s},\mathbf{t}}[1,0] + 5, V_{\mathbf{s},\mathbf{t}}[1,0]\} + 4 = 13,\\
&D_{\mathbf{s},\mathbf{t}}[1,1] = \min\{D_{\mathbf{s},\mathbf{t}}[0,0] + 16, V_{\mathbf{s},\mathbf{t}}[1,1], H_{\mathbf{s},\mathbf{t}}[1,1]\} = 13.
\end{aligned}$$

Thus, $w_{opt}(\langle \mathbf{s}, \mathbf{t}\rangle) = D_{\mathbf{s},\mathbf{t}}[1,1] = 13$ with the corresponding optimal alignments $\begin{pmatrix} \mathtt{A} & - \\ - & \mathtt{T} \end{pmatrix}$ and $\begin{pmatrix} - & \mathtt{A} \\ \mathtt{T} & - \end{pmatrix}$. In fact, these alignments have score 18. The correct optimal alignment is $\begin{pmatrix} \mathtt{A} \\ \mathtt{T} \end{pmatrix}$ with score 16. □

In contrast to the publications mentioned above, Altschul and Erickson [9] give the correct initializations for $V_{\mathbf{s},\mathbf{t}}$ and $H_{\mathbf{s},\mathbf{t}}$.

In analogy to $D^r_{\mathbf{s},\mathbf{t}}$, we define *reverse history matrices*

$$V^r_{\mathbf{s},\mathbf{t}}[i,j] \;:=\; \min\{w(A) | A \in \mathbb{A}(\langle {}^i\mathbf{s}, {}^j\mathbf{t}\rangle), A = \begin{pmatrix} s_{i+1} & \cdots \\ - & \cdots \end{pmatrix}\}$$

and

$$H^r_{\mathbf{s},\mathbf{t}}[i,j] \;:=\; \min\{w(A) | A \in \mathbb{A}(\langle {}^i\mathbf{s}, {}^j\mathbf{t}\rangle), A = \begin{pmatrix} - & \cdots \\ t_{j+1} & \cdots \end{pmatrix}\}.$$

To emphasize the symmetry, the *forward* history matrices $V_{\mathbf{s},\mathbf{t}}$ and $H_{\mathbf{s},\mathbf{t}}$ are also denoted by $V^f_{\mathbf{s},\mathbf{t}}$ and $H^f_{\mathbf{s},\mathbf{t}}$, respectively.

Adopting the space-saving technique of Hirschberg [104] to the history matrices $H_{\mathbf{s},\mathbf{t}}$ and $V_{\mathbf{s},\mathbf{t}}$, Myers and Miller [145] showed that also the computation of an optimal alignment with affine gap costs is possible in linear space.

3.3 Optimal Alignment of More Than Two Sequences

Dynamic programming can be generalized to multiple alignment of a family of k sequences $\langle \mathbf{s}_1, \ldots, \mathbf{s}_k\rangle$ [166, 225]. A k-dimensional *distance lattice* $D_{\mathbf{s}_1,\ldots,\mathbf{s}_k}$ is used instead of the distance matrix $D_{\mathbf{s},\mathbf{t}}$, taking $\mathcal{O}(n^k)$ space where n is the length of the longest sequence under consideration. Straightforward generalizations of (3.1) and (3.2) lead to algorithms with time complexity $\mathcal{O}(2^k n^k)$ for homogeneous gap costs

and $\mathcal{O}(2^k n^k kn)$ for arbitrary subadditive gap costs, respectively. The generalization of the quadratic time algorithm for linear gap costs is not that simple. Gotoh [81] devised an algorithm for the optimal alignment of three sequences which is cubic in space and time. In the general case, efficient algorithms are rather complicated. The number of "histories" one has to consider at each point of the lattice increases enormously with the number of sequences. Altschul [5] showed that the growth is of the order $\mathcal{O}([k/e\ln(2)]^k\sqrt{k})$.

To avoid this problem, he suggested a slightly different scoring scheme for linear gap penalties in a multiple alignment called *quasi natural gap costs* [5]. Whether a blank in one row of the multiple alignment when aligned with a letter in another row is the opening of a new or the elongation of an existing gap is decided based on the entries in the previous column of the alignment, only. Thus, no histories have to be saved. The number of cases where an alignment optimal with respect to quasi natural gap costs differs from an optimal alignment in the "natural" (sum-of-pairs) sense can be shown to be marginal [5].

Anyway, space and time usage of standard dynamic programming for sequence alignment grow exponentially with the number of sequences. Less time consuming variants and a number of speed-ups for the optimal alignment of more than two sequences have therefore been proposed (e.g. [73, 142, 81, 39, 188, 141, 4, 107, 90]). In some of these approaches, the search for the optimal alignment is reorganized such that most promising regions along the main diagonal of the k-dimensional search space are scanned first and the search is stopped when it is clear that no alignment exists which scores better than the best one found so far. Other methods determine in advance parts of the distance lattice which provably cannot contain an optimal alignment. In general, the closer the sequences are related, the better perform such procedures. (This observation seems to be valid for most, even for "manual" alignment methods [198].) But despite all the efforts made on the development of faster optimal alignment procedures, dealing with more than, say, five sequences of length 1000 or twelve sequences of length 20 within reasonable time (some seconds or minutes) seems virtually impossible. However, as we shall see, this limited potential meets exactly the requirements of the divide-and-conquer approach.

3.4 The Approach of Carrillo and Lipman

In this section, we briefly describe the approach of Carrillo and Lipman [39] as an illustrative example of a method which allows to reduce the search space of multi-dimensional dynamic programmic applied to optimal sum-of-pairs sequence alignment.

In the first stage, a heuristic alignment of the sequences $\langle \mathbf{s}_1, \ldots, \mathbf{s}_k \rangle$ is computed. Several procedures for fast heuristic SP alignments are available, see e.g. [187, 109, 220, 39]. In the second stage, the projections of this alignment on the two-dimensional surfaces of the k-dimensional lattice together with the pairwise optimal

alignment scores are used to compute upper bounds for the projections of the optimal multiple alignment. These bounds allow to determine regions in the k-dimensional lattice which do not have to be considered when searching for the optimal multiple alignment.

To be more precise, let $\widehat{D}$ be the score of the heuristic multiple alignment and let

$$w_{opt}(\langle \mathtt{s}_1, \ldots, \mathtt{s}_k \rangle) \quad = \quad \sum_{1 \le i < j \le k} \widetilde{D}_{\mathtt{s}_i, \mathtt{s}_j}$$

be the (yet unknown) optimal alignment score with pairwise projection costs $\widetilde{D}_{\mathtt{s}_i, \mathtt{s}_j}$. (For shortness of the exposition, we set all pairwise sequence weights to 1 in this discussion.) The score of the heuristic multiple alignment $\widehat{D}$ cannot be smaller than the optimal alignment score:

$$\widehat{D} \quad \ge \quad w_{opt}(\langle \mathtt{s}_1, \ldots, \mathtt{s}_k \rangle),$$

and the cost $\widetilde{D}_{\mathtt{s}_i, \mathtt{s}_j}$ of a projection of the optimal alignment is never smaller than the corresponding optimal pairwise alignment score:

$$\widetilde{D}_{\mathtt{s}_i, \mathtt{s}_j} \quad \ge \quad w_{opt}(\langle \mathtt{s}_i, \mathtt{s}_j \rangle).$$

Hence, for each pair of sequences $(\mathtt{s}_p, \mathtt{s}_q)$, $1 \le p < q \le k$, an upper bound for the projection cost $\widetilde{D}_{\mathtt{s}_p, \mathtt{s}_q}$, based only on the cost of the heuristic alignment and of the optimal pairwise alignments can easily be computed:

$$\widehat{D} \quad \ge \quad w_{opt}(\langle \mathtt{s}_1, \ldots, \mathtt{s}_k \rangle) \quad = \quad \sum_{1 \le i < j \le k} \widetilde{D}_{\mathtt{s}_i, \mathtt{s}_j},$$

thus

$$\widetilde{D}_{\mathtt{s}_p, \mathtt{s}_q} \quad \le \quad \widehat{D} \; - \sum_{\substack{1 \le i < j \le k \\ (i,j) \ne (p,q)}} \widetilde{D}_{\mathtt{s}_i, \mathtt{s}_j} \quad \le \quad \widehat{D} \; - \sum_{\substack{1 \le i < j \le k \\ (i,j) \ne (p,q)}} w_{opt}(\langle \mathtt{s}_i, \mathtt{s}_j \rangle).$$

By this upper bound, for each pair $(\mathtt{s}_p, \mathtt{s}_q)$, a region in the corresponding two-dimensional surface of the distance lattice can be defined such that the optimal alignment path must lie inside the intersection of the k-dimensional lattice with all these regions.

This procedure with quasi natural gap costs has been implemented in the computer program `MSA` [128, 90]. Unfortunately, the approach of Carrillo and Lipman alone does not speed up the standard dynamic programming procedure sufficiently to make it applicable to a much broader range of sequence families. Thus, several looser heuristics have been implemented in `MSA`, relaxing from the restriction that the optimal alignment necessarily lies within the remaining part of the k-dimensional lattice. Hence, `MSA` does not always find an optimal alignment of the given sequences [90].

3.5 NP-Completeness of Multiple Sequence Alignment

Without going much into detail, we report in this section an important result of Wang and Jiang [215].

Theorem 3.3
Multiple sequence alignment with respect to the sum-of-pairs score is NP-complete. □

The proof which shows the reducibility of the *shortest common supersequence problem* [75] to multiple sequence alignment can be found in [215].

With the NP-completeness of multiple sequence alignment, any attempt of developing a fast algorithm to compute optimal multiple sequence alignments is expected to fail. Consequently, there is a strong need for heuristic algorithms producing near-optimal alignments, and an abundance of procedures have been developed. For reviews and comparisons see [13, 40, 160, 135]. Existing approaches fall in general in one of the follwing two classes.

- The *iterated pairwise* or *progresssive alignment methods* follow the rationale of tree alignments described in Section 2.3.3: Guided by the branching order of a pre-given (mostly unrooted) tree whose leaves represent the sequences to be aligned – or by any other data structure representing a hierarchical clustering of the sequences –, in the first stage the two closest related sequences are aligned pairwise optimally. The result is stored in form of a so-called *weighted average sequence* [224] or *profile* [89] which permits further alignment with other individual sequences or profiles/average sequences. In the next stages, this procedure is iterated for the next closest related pair and so on. Several variations of this basic strategy have been developed [167, 224, 105, 17, 20, 66, 89, 46, 196, 102, 194, 206, 199, 97, 30, 208, 40, 183, 83, 103, 84, 201, 211, 85].

- Algorithms that fall in the second class, *fragment-based methods*, follow the strategy of assembling pairwise or multiple *local* alignments to achieve a global multiple alignment [132, 228, 187, 15, 109, 206, 223, 207, 12, 208, 52, 177, 22, 229, 163, 147, 140]. After a consistency check, the local alignments (also denoted as *fragments*, *segments*, or *diagonals*) define fixed regions or *anchors* of the intended global alignment. Only the remaining subsequences between the anchors have to be aligned optimally. Unfortunately, the feasibility of this approach depends largely on the relatedness of the sequences. For closely related sequences many significant fragments can be found, and the optimal alignment of the intermediate segments is no challenge. But for feebly related sequences, often only a small number of consistent local alignments can be found. Then, the usual fragment-based alignment methods take rather long running times due to the long subsequences which have to be aligned optimally.

The divide-and-conquer approach to multiple sequence alignment developed in this thesis can be seen as flowing from the same concept as the fragment-based methods but being more general than previous procedures. Systematically, anchor points are fixed in all sequences, no matter if there are obvious local similarities between the sequences or not. Hence, a considerable speed-up compared to multiple sequence alignment by standard dynamic programming can be guaranteed.

Chapter 4

Basic Divide-and-Conquer Alignment

4.1 Introduction

The general idea of the *divide-and-conquer multiple sequence alignment* (DCA) is rather simple. Suppose that we are given a family $S = \langle \mathbf{s}_1, \ldots, \mathbf{s}_k \rangle$ of sequences. First, we cut each sequence at a suitable position near to its midpoint, say, we cut sequence $\mathbf{s}_1$ at position c_1, $\mathbf{s}_2$ at position c_2, and so on. This way, we obtain two new families of shorter sequences, one family consisting of the prefixes $\langle \mathbf{s}_1^{c_1}, \ldots, \mathbf{s}_k^{c_k} \rangle$ and one of the suffixes $\langle {}^{c_1}\mathbf{s}_1, \ldots, {}^{c_k}\mathbf{s}_k \rangle$. If we could align these two new families of sequences optimally, we could then concatenate our resulting alignments to obtain an alignment $A \in \mathbb{A}(S)$ of the original sequences. On the other hand, if it takes still too much time to align these two new families optimally, we could apply our procedure to the prefix family and to the suffix family in a recursive manner, thus reducing the original multiple alignment problem to more and more alignment problems involving shorter and shorter sequences, until we reach (sub)sequences short enough (e.g. shorter than a threshold L) so that they can be aligned optimally. Thus, we reduce the problem of aligning k sequences of length at most n to the problem of aligning about $\frac{n}{L}$ families of short (sub)sequences of maximal length L. For a schematic representation of the divide-and-conquer method for three sequences, see Figure 4.1.

Figure 4.2 gives an impression about the reduction of search space achieved by this approach: Suppose each of the three sequences is represented by a set of parallel edges of the large cube. The size of the corresponding alignment problem is proportional to the volume of the cube. By slicing the sequences, we reduce the large problem to several small alignment problems, represented by the "chain" of boxes along the diagonal of the cube. The remaining search space is then the sum of the volumes of these small boxes.

The main difficulty arising with this approach is how to find *suitable* slicing positions such that the resulting total alignment is optimal or – at least – very

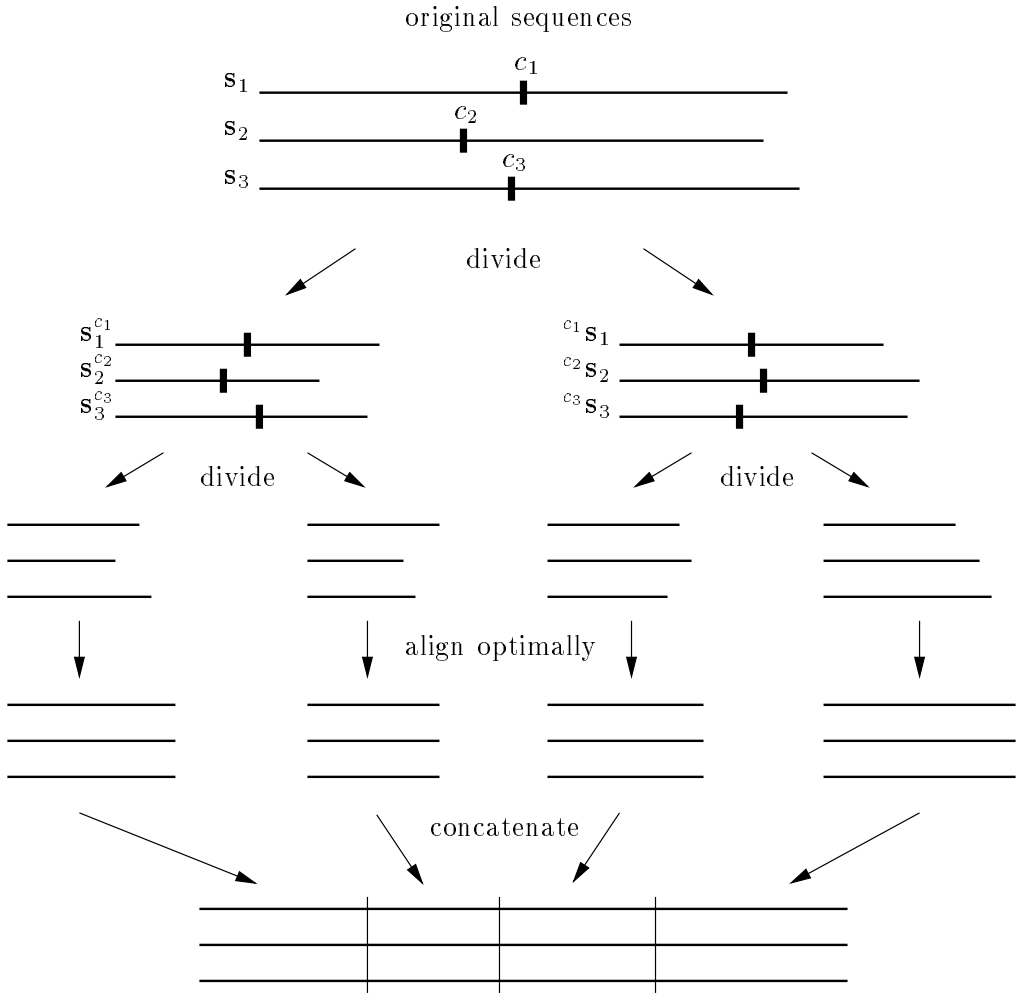

Figure 4.1: Schematic representation of the divide-and-conquer method.

similar to an optimal alignment of the original sequences.

Definition 4.1 (Optimal Slicing Positions)
A family of slicing positions $\langle c_1, \ldots, c_k \rangle$ of sequences $\langle \mathbf{s}_1, \ldots, \mathbf{s}_k \rangle$ is called *optimal* if there is an alignment A of the family of prefixes $\langle \mathbf{s}_1^{c_1}, \ldots, \mathbf{s}_k^{c_k} \rangle$ and there is an alignment B of the family of suffixes $\langle {}^{c_1}\mathbf{s}_1, \ldots, {}^{c_k}\mathbf{s}_k \rangle$ such that the concatenation $A{+\!+}B$ is an optimal alignment of the original sequences $\langle \mathbf{s}_1, \ldots, \mathbf{s}_k \rangle$. □

Two optimal families of slicing positions can be given trivially: $\langle 0, \ldots, 0 \rangle$ and $\langle |\mathbf{s}_1|, \ldots, |\mathbf{s}_k| \rangle$. But obviously, slicing the sequences this way does not contribute in any way to a solution of the alignment problem. Instead, symmetry reasons suggest slicing positions near to the midpoint of the sequences since this way the largest reduction of search space can be expected (cf. Figure 4.2).

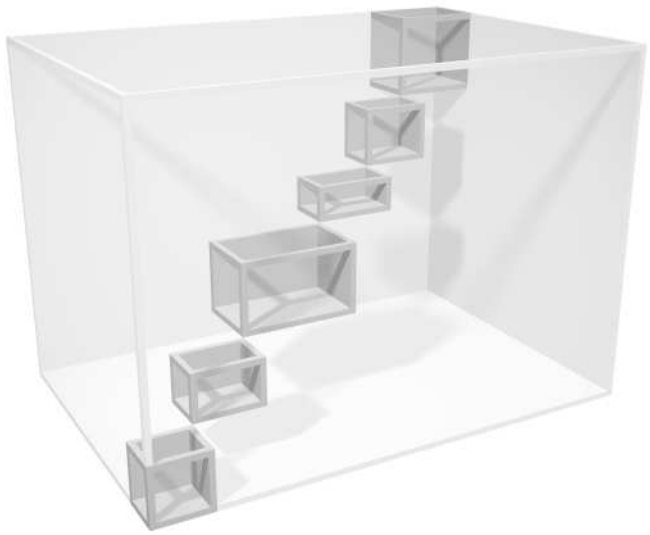

Figure 4.2: The reduction of search space.

A simple observation about the existence of optimal families of slicing positions is the following.

Lemma 4.2
Suppose that we are given a family of sequences $\langle \mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_k \rangle$ and a slicing position of one of the sequences, say position $\hat{c}_1$ of sequence $\mathbf{s}_1$, $0 \leq \hat{c}_1 \leq |\mathbf{s}_1|$. Then there always exist positions $c_2, \ldots, c_k$ such that $\langle \hat{c}_1, c_2, \ldots, c_k \rangle$ is an optimal family of slicing positions of $\langle \mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_k \rangle$. □

Proof
Let A be an optimal alignment of $S = \langle \mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_k \rangle$. Recall that $z := \alpha_{\mathbf{s}_1^*}(\hat{c}_1)$ is the index of the column in A in which the $\hat{c}_1$th letter of $\mathbf{s}_1$ occurs. Since A is an optimal alignment, $\hat{c}_1$ together with the positions $c_j := \sigma_{\mathbf{s}_j^*}(z)$ for $j = 2, \ldots, k$ builds an optimal family $\langle \hat{c}_1, c_2, \ldots, c_k \rangle$ of slicing positions of S. □

Without a pre-given optimal alignment, the computation of an optimal family of slicing positions is not at all that simple. In fact, due to the NP-completeness of optimal multiple sequence alignment, we cannot expect that the computation of optimal slicing positions requires less time than the computation of an optimal alignment itself. Consequently, to achieve a fast divide-and-conquer alignment algorithm, the search for *optimal* families of slicing positions cannot be the method of choice.

What we intend to find instead is a way to compute good though not necessarily optimal families of slicing positions $\langle c_1, c_2, \ldots, c_k \rangle$. Here we have two main goals: The

computation should be faster than the $\mathcal{O}(2^k n^k)$ time of dynamic programming and the required space should be less than exponential in the number of sequences. Note the importance of the second condition since space, not time, is often the limiting factor when computing simultaneous multiple alignments of long sequences. Indeed, we will present a way of computing high quality slicing positions using the space required for pairwise alignments, only.

4.2 Additional-Cost Matrix of Two Sequences

As described above, the computation of an optimal multiple alignment from compatible optimal pairwise alignments is very simple (see Lemma 2.15). However, in general, optimal pairwise alignments are not compatible. Consequently, a compromise has to be made when constructing a multiple alignment from the optimal pairwise alignments. Some of the alignments have to be "broken", and the sequences are shifted against each other. With the following definition, we introduce a measure which quantifies the deviation from being an optimal pairwise alignment:

Definition 4.3 (Pairwise Additional Cost)
Given two sequences $\mathbf{s}$ and $\mathbf{t}$ and an alignment score function w, we define for each pair (i,j) of possible slicing positions of $\mathbf{s}$ and $\mathbf{t}$, $0 \leq i \leq |\mathbf{s}|$, $0 \leq j \leq |\mathbf{t}|$, the *pairwise additional cost* with respect to w by

$$C_{\mathbf{s},\mathbf{t}}[i,j] \;:=\; \min\left\{w(A{+\!\!+}B) | A \in \mathbb{A}(\langle \mathbf{s}^i, \mathbf{t}^j\rangle), B \in \mathbb{A}(\langle {}^i\mathbf{s}, {}^j\mathbf{t}\rangle)\right\} - w_{opt}(\langle \mathbf{s}, \mathbf{t}\rangle).$$

The matrix

$$C_{\mathbf{s},\mathbf{t}} \;:=\; (C_{\mathbf{s},\mathbf{t}}[i,j])_{0\leq i\leq|\mathbf{s}|,0\leq j\leq|\mathbf{t}|}$$

is called the *additional-cost matrix* of $\mathbf{s}$ and $\mathbf{t}$ with respect to w. □

Figure 4.3[1] illustrates the definition: An optimal alignment path with cost $w_{opt}(\langle \mathbf{s}, \mathbf{t}\rangle)$ is represented by the white boxes. The shaded boxes show a best alignment path passing through vertex (i,j). The additional cost is simply the "length difference" of these two paths.

Obviously, the additional cost is always greater than or equal to zero. Moreover, in general the additional-cost matrices have a particularly regular form (see Figure 4.4). If we regard the scores in each position of the matrices as a "height function", then the matrices have a valley which runs along or near to the main diagonal. The lowest elevation is zero height which is attained along the paths of optimal alignments (see the white boxes in Figure 4.4). From the zero height paths to the top and bottom,

[1] The unusual way of denoting the columns of $C_{\mathbf{s},\mathbf{t}}$ by the first index (i) and the rows by the second index (j) in Figures 4.3 – 4.6 and 5.1 – 5.9 is by reason of consistency with earlier publications [192, 157].

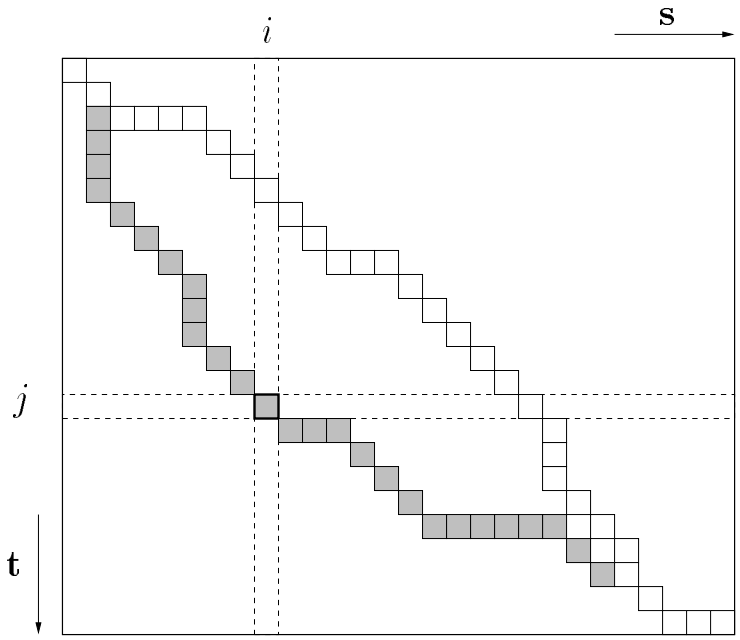

Figure 4.3: Definition of $C_{\mathbf{s},\mathbf{t}}$. White boxes denote an optimal alignment path, shaded boxes denote a best alignment path through vertex (i, j).

values in each column are almost always monotonically increasing (the darker shaded regions in Figure 4.4).

Notation 4.4

For all $i \in \{0, \ldots, |\mathbf{s}|\}$, we denote the ith column of additional-cost matrix $C_{\mathbf{s},\mathbf{t}}$ by

$$Col^i_{\mathbf{s},\mathbf{t}} \;:=\; \left(Col^i_{\mathbf{s},\mathbf{t}}[j]\right)_{0 \le j \le |\mathbf{t}|} \qquad \text{where} \qquad Col^i_{\mathbf{s},\mathbf{t}}[j] \;:=\; C_{\mathbf{s},\mathbf{t}}[i,j].$$

For all $j \in \{0, \ldots, |\mathbf{t}|\}$, we denote the jth row of $C_{\mathbf{s},\mathbf{t}}$ by

$$Row^j_{\mathbf{s},\mathbf{t}} \;:=\; \left(Row^j_{\mathbf{s},\mathbf{t}}[i]\right)_{0 \le i \le |\mathbf{s}|} \qquad \text{where} \qquad Row^j_{\mathbf{s},\mathbf{t}}[i] \;:=\; C_{\mathbf{s},\mathbf{t}}[i,j].$$

□

Lemma 4.5

For each $i \in \{0, \ldots, |\mathbf{s}|\}$, there exists at least one index $j \in \{0, \ldots, |\mathbf{t}|\}$ such that $Col^i_{\mathbf{s},\mathbf{t}}[j] = 0$. □

Proof

This follows immediately from Lemma 4.2 with $k = 2$. □

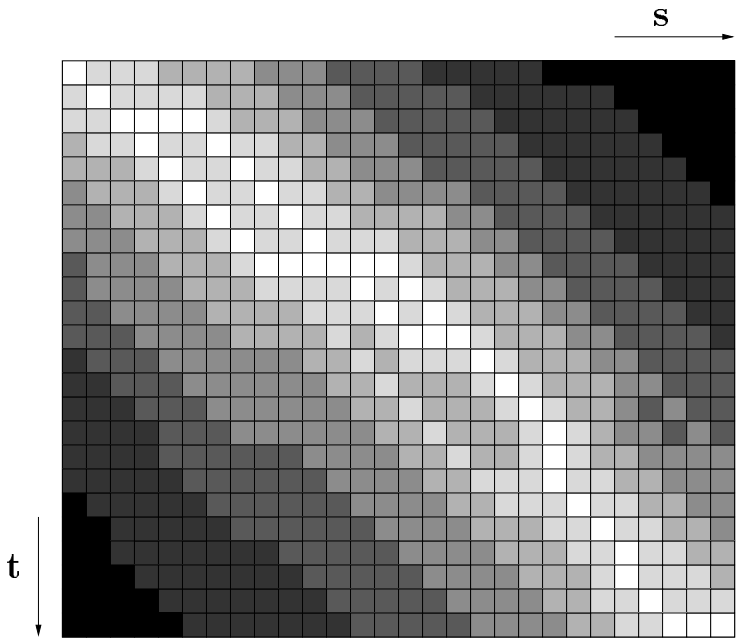

Figure 4.4: Typical form of an additional-cost matrix. Light boxes denote low values, dark boxes denote high values.

Of course, an analogous assertion holds for the rows of an additional-cost matrix.

Note that in case of an additive alignment score function w, the above definition of the pairwise additional cost is equivalent to that given in [203] and [192]:

$$\begin{aligned} C_{\mathbf{s},\mathbf{t}}[i,j] &:= \min\{w(A{+\!\!+}B) | A \in \mathbb{A}(\langle \mathbf{s}^i, \mathbf{t}^j \rangle), B \in \mathbb{A}(\langle {}^i\mathbf{s}, {}^j\mathbf{t} \rangle)\} - w_{opt}(\langle \mathbf{s}, \mathbf{t} \rangle) \\ &= \min\{w(A) + w(B) | A \in \mathbb{A}(\langle \mathbf{s}^i, \mathbf{t}^j \rangle), B \in \mathbb{A}(\langle {}^i\mathbf{s}, {}^j\mathbf{t} \rangle)\} - w_{opt}(\langle \mathbf{s}, \mathbf{t} \rangle) \\ &= \min\{w(A) | A \in \mathbb{A}(\langle \mathbf{s}^i, \mathbf{t}^j \rangle)\} + \min\{w(B) | B \in \mathbb{A}(\langle {}^i\mathbf{s}, {}^j\mathbf{t} \rangle)\} - w_{opt}(\langle \mathbf{s}, \mathbf{t} \rangle) \\ &= w_{opt}(\langle \mathbf{s}^i, \mathbf{t}^j \rangle) + w_{opt}(\langle {}^i\mathbf{s}, {}^j\mathbf{t} \rangle) - w_{opt}(\mathbf{s}, \mathbf{t}). \end{aligned}$$

Hence, for additive alignment scores, additional-cost matrices are easily computed from the forward and reverse distance matrices introduced in Section 3.1:

$$C_{\mathbf{s},\mathbf{t}}[i,j] = D^f_{\mathbf{s},\mathbf{t}}[i,j] + D^r_{\mathbf{s},\mathbf{t}}[i,j] - w_{opt}(\langle \mathbf{s}, \mathbf{t} \rangle)$$

with

$$w_{opt}(\langle \mathbf{s}, \mathbf{t} \rangle) = D^f_{\mathbf{s},\mathbf{t}}[|\mathbf{s}|, |\mathbf{t}|] = D^r_{\mathbf{s},\mathbf{t}}[0,0].$$

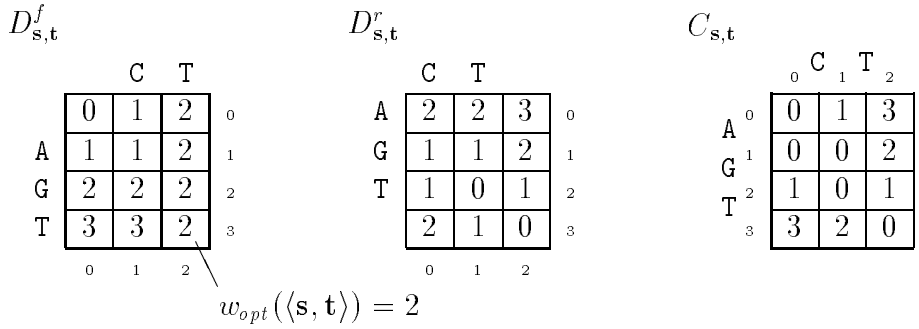

Figure 4.5: Forward and reverse distance matrix for $\mathbf{s} = \mathtt{CT}$ and $\mathbf{t} = \mathtt{AGT}$ with respect to the unit cost function. The additional-cost matrix $C_{\mathbf{s},\mathbf{t}}$ is simply the sum of both matrices minus the constant value $w_{opt}(\langle \mathbf{s}, \mathbf{t} \rangle) = 2$.

Example 4.6

Let $\mathcal{A} = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$, and consider the unit cost function $d^*(x, y) = 1 - \delta_{xy}$ for all $x, y \in \mathcal{A} \cup \{-\}$. The forward and reverse distance matrices as well as the additional-cost matrix for the sequences $\mathbf{s} = \mathtt{CT}$ and $\mathbf{t} = \mathtt{AGT}$ are shown in Figure 4.5. □

We now consider the more general case of affine gap penalties $g(l) = \alpha + \beta l$. Due to the possibility of gaps "running through" a slicing position, the score of the concatenated alignment can be different from the sum of the prefix and suffix alignment scores. As above, consider alignments $A \in \mathbb{A}(\langle \mathbf{s}^i, \mathbf{t}^j \rangle)$ and $B \in \mathbb{A}(\langle {}^i\mathbf{s}, {}^j\mathbf{t} \rangle)$. We distinguish three cases:

1. A gap in $\mathbf{t}$ is running through the cut: $A = \begin{pmatrix} \cdots & s_i \\ \cdots & - \end{pmatrix}$, $B = \begin{pmatrix} s_{i+1} & \cdots \\ - & \cdots \end{pmatrix}$;

2. A gap in $\mathbf{s}$ is running through the cut: $A = \begin{pmatrix} \cdots & - \\ \cdots & t_j \end{pmatrix}$, $B = \begin{pmatrix} - & \cdots \\ t_{j+1} & \cdots \end{pmatrix}$;

3. None of the above.

In the first case, the scores of the alignments A and B can be obtained easily from the history matrices introduced in Section 3.2:

$$w_{opt}(A) \;=\; V^f_{\mathbf{s},\mathbf{t}}[i,j] \qquad \text{and} \qquad w_{opt}(B) \;=\; V^r_{\mathbf{s},\mathbf{t}}[i,j].$$

To obtain the score of the concatenation $A{+\!+}B$, the gap open penalty α has to be subtracted from the sum of these values because – due to the concatenation – the terminal gap in A merges with the initial gap in B. The second case can be treated similarly, replacing $V_{\mathbf{s},\mathbf{t}}$ by $H_{\mathbf{s},\mathbf{t}}$. The third case where no gap is running through the cut is identical to the case of additive gap costs discussed above. Together, we take

the minimum of the three possibilities:

$$C_{\mathbf{s},\mathbf{t}}[i,j] \;=\; \min\left\{\begin{array}{c} V^f_{\mathbf{s},\mathbf{t}}[i,j] + V^r_{\mathbf{s},\mathbf{t}}[i,j] - \alpha, \\ H^f_{\mathbf{s},\mathbf{t}}[i,j] + H^r_{\mathbf{s},\mathbf{t}}[i,j] - \alpha, \\ D^f_{\mathbf{s},\mathbf{t}}[i,j] + D^r_{\mathbf{s},\mathbf{t}}[i,j] \end{array}\right\} - w_{opt}(\langle \mathbf{s},\mathbf{t}\rangle).$$

Thus, the calculation of additional-cost matrices for affine gap penalties is possible in time $\mathcal{O}(|\mathbf{s}|\cdot|\mathbf{t}|)$ as for the homogeneous case. Obviously, beneath the matrices $D^f_{\mathbf{s},\mathbf{t}}$, $D^r_{\mathbf{s},\mathbf{t}}$, $V^f_{\mathbf{s},\mathbf{t}}$, $V^r_{\mathbf{s},\mathbf{t}}$, $H^f_{\mathbf{s},\mathbf{t}}$, $H^r_{\mathbf{s},\mathbf{t}}$, and $C_{\mathbf{s},\mathbf{t}}$, no additional space is required, such that the space usage is also in $\mathcal{O}(|\mathbf{s}|\cdot|\mathbf{t}|)$. Furthermore, note that no alignments but only scores are computed. So, when only one or just a few columns or rows of the matrix $C_{\mathbf{s},\mathbf{t}}$ are required, the memory usage can be reduced accordingly.

In different contexts, matrices similar to additional-cost matrices have been considered: for computing longest similar substrings [180, 181, 60], near optimal alignments [217, 222, 148], and in the context of *dot plots* to visualize locally similar regions of sequences [33, 207]. But in all these approaches, the entries of the matrices correspond to pairs of *letters*, as opposed to pairs of *slicing positions* (between the letters) considered in our case.

4.3 *C*-Optimal Families of Slicing Positions

We now return to the original problem of finding a family of k slicing positions for the sequences $\langle \mathbf{s}_1,\ldots,\mathbf{s}_k\rangle$. In analogy to the weighted SP alignment score, we define the *multiple additional cost* imposed by forcing the multiple alignment path of the sequences through the particular vertex $(c_1,\ldots,c_k)$ in the k-dimensional hypercube associated with the corresponding alignment problem. To this end, we use a weighted sum of additional costs over all pairwise projections (c_p,c_q).

Definition 4.7 (Weighted SP Multiple Additional Cost)
Suppose that we are given a family of k sequences $\langle \mathbf{s}_1,\ldots,\mathbf{s}_k\rangle$ and pairwise weight factors $\alpha_{p,q}$, $1\le p<q\le k$. The *weighted SP multiple additional cost* of a family of slicing positions $\langle c_1,\ldots,c_k\rangle$ is defined as

$$C(\langle c_1,\ldots,c_k\rangle) \;:=\; \sum_{1\le p<q\le k} \alpha_{p,q}\; C_{\mathbf{s}_p,\mathbf{s}_q}[c_p,c_q].$$

□

Definition 4.8 (*C*-Optimal Families of Slicing Positions)
Suppose that, in addition, we are given a slicing position of one of the sequences, say position $\hat{c}_1$ of sequence $\mathbf{s}_1$. A family of slicing positions $\langle c_2,\ldots,c_k\rangle$ of $\mathbf{s}_2,\ldots,\mathbf{s}_k$ such that the multiple additional cost $C(\langle \hat{c}_1,c_2,\ldots,c_k\rangle)$ is minimal is called *C-optimal* with respect to $\hat{c}_1$. The corresponding minimal additional cost is denoted by

$$C_{opt} \;=\; C_{opt}(\hat{c}_1) \;:=\; \min\left\{C(\langle \hat{c}_1,c_2,\ldots,c_k\rangle) \,|\, 0\le c_i\le |\mathbf{s}_i| \text{ for all } i\in\{2,\ldots,k\}\right\}.$$

□

In general, we observed that for closely related sequences the values C_{opt} are often rather small, sometimes even zero which means that all pairwise projections of the multiple cut are compatible with corresponding optimal pairwise alignments (but not necessarily that the family of slicing positions is optimal). For less related sequences, the minimal additional cost C_{opt} often has comparatively high values.

We now employ the heuristic which is the basis of our divide-and-conquer alignment procedure:

> *Cutting the sequences at C-optimal slicing positions, the concatenation of an optimal multiple alignment of the prefix sequences and an optimal multiple alignment of the corresponding suffix sequences gives a very good, if not optimal multiple alignment of the original sequences.*

Note that this is not a provable theorem but only a plausible statement motivated by the reasoning presented above. The high quality of the alignments which we obtained using this principle and are going to present in Chapters 7 and 8 justifies the heuristic. Nevertheless, a C-optimal family of slicing positions does not guarantee an optimal alignment, even if the multiple additional cost is zero. To clarify the notion of optimal and C-optimal slicing positions, as well as to prove this claim, we extend the example of Section 4.2:

Example 4.6 (cont.)
Consider sequences $\mathtt{s}_1 = \mathtt{CT}$, $\mathtt{s}_2 = \mathtt{AGT}$, $\mathtt{s}_3 = \mathtt{G}$. The corresponding pairwise additional-cost matrices are shown in Figure 4.6. We set all pairwise sequence weights $\alpha_{1,2} = \alpha_{1,3} = \alpha_{2,3}$ to 1 and assume $\hat{c}_1 := 1$ fixed. It is not difficult to verify that the family $\langle c_2, c_3 \rangle = \langle 1, 0 \rangle$ is C-optimal with respect to $\hat{c}_1$ since

$$C(\langle 1,1,0 \rangle) \;=\; C_{\mathtt{s}_1,\mathtt{s}_2}[1,1] + C_{\mathtt{s}_1,\mathtt{s}_3}[1,0] + C_{\mathtt{s}_2,\mathtt{s}_2}[1,0] \;=\; 0$$

(see the bold boxes in Figure 4.6). The corresponding prefix and suffix families are $\left\langle \begin{array}{lcl} \mathtt{s}_1^1 & = & \mathtt{C} \\ \mathtt{s}_2^1 & = & \mathtt{A} \\ \mathtt{s}_3^0 & = & \varepsilon \end{array} \right\rangle$ and $\left\langle \begin{array}{lcl} {}^1\mathtt{s}_1 & = & \mathtt{T} \\ {}^1\mathtt{s}_2 & = & \mathtt{GT} \\ {}^0\mathtt{s}_3 & = & \mathtt{G} \end{array} \right\rangle$, respectively. Given this cut, all best possible alignments have cost 7 (e.g. $\begin{pmatrix} \mathtt{C} & - & \mathtt{T} \\ \mathtt{A} & \mathtt{G} & \mathtt{T} \\ - & \mathtt{G} & - \end{pmatrix}$). The (only) optimal alignment $\begin{pmatrix} - & \mathtt{C} & \mathtt{T} \\ \mathtt{A} & \mathtt{G} & \mathtt{T} \\ - & \mathtt{G} & - \end{pmatrix}$ with cost 6 is not compatible with this cut. Thus, $\langle 1,1,0 \rangle$ is not an optimal family of slicing positions of $\langle \mathtt{s}_1, \mathtt{s}_2, \mathtt{s}_3 \rangle$.

Note that also another C-optimal family of slicing positions $\langle c_2, c_3 \rangle$ exists which indeed leads to an optimal cut: $\langle c_2, c_3 \rangle = \langle 2, 1 \rangle$. Slicing the sequences this way, the optimal alignment can be obtained:

$$\left\langle \begin{array}{lcl} \mathtt{s}_1^1 & = & \mathtt{C} \\ \mathtt{s}_2^2 & = & \mathtt{AG} \\ \mathtt{s}_3^1 & = & \mathtt{G} \end{array} \right\rangle, \left\langle \begin{array}{lcl} {}^1\mathtt{s}_1 & = & \mathtt{T} \\ {}^2\mathtt{s}_2 & = & \mathtt{T} \\ {}^1\mathtt{s}_3 & = & \varepsilon \end{array} \right\rangle \quad \rightarrow \quad \begin{pmatrix} - & \mathtt{C} \\ \mathtt{A} & \mathtt{G} \\ - & \mathtt{G} \end{pmatrix} +\!\!+ \begin{pmatrix} \mathtt{T} \\ \mathtt{T} \\ - \end{pmatrix} \quad = \quad \begin{pmatrix} - & \mathtt{C} & \mathtt{T} \\ \mathtt{A} & \mathtt{G} & \mathtt{T} \\ - & \mathtt{G} & - \end{pmatrix}.$$

□

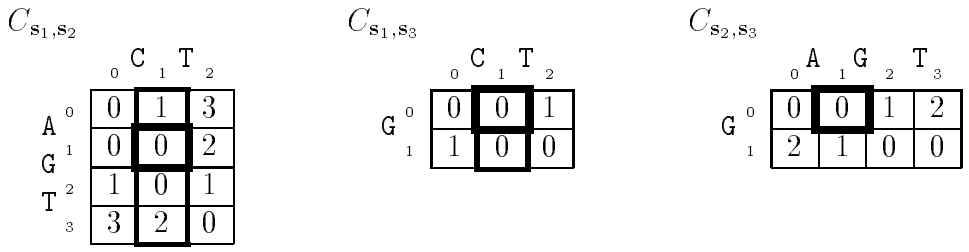

Figure 4.6: Pairwise additional-cost matrices for $\mathtt{s}_1 = \mathtt{CT}$, $\mathtt{s}_2 = \mathtt{AGT}$, $\mathtt{s}_3 = \mathtt{G}$.

The reason for this discrepancy is that a *C-optimal* family of slicing positions is optimized on the basis of independent pairwise alignments of the sequences whereas an *optimal* family of slicing positions is minimal with respect to the multiple alignment. In contrast to Example 4.6, there may be no C-optimal family of slicing positions which is an optimal cut.

4.4 The Algorithm

Now we are prepared to present a formal description of the basic divide-and-conquer multiple sequence alignment algorithm. As subfunctions, we use

- a function *MSA* which computes an optimal multiple sequence alignment of a family of sequences, e.g. by standard dynamic programming, and
- a function *C-opt* which takes as arguments a family of sequences $\langle \mathtt{s}_1, \mathtt{s}_2, \ldots, \mathtt{s}_k \rangle$ as well as a slicing position $\hat{c}_1$ of $\mathtt{s}_1$ and returns a corresponding C-optimal family of slicing positions $\langle c_2, \ldots, c_k \rangle$.

As motivated above, we fix $\hat{c}_1$ in the middle of $\mathtt{s}_1$. The recursion of *DCA* is continued as long as any one of the (sub)sequences is longer than a threshold L, which is introduced as a parameter of *DCA*. As soon as all of the (sub)sequences are of length L or shorter, they are aligned optimally.

Function

$DCA(\langle \mathtt{s}_1, \mathtt{s}_2, \ldots, \mathtt{s}_k \rangle, L)$

$:= MSA(\langle \mathtt{s}_1, \mathtt{s}_2, \ldots, \mathtt{s}_k \rangle)$, if $\max\{|\mathtt{s}_1|, |\mathtt{s}_2|, \ldots, |\mathtt{s}_k|\} \leq L$

$:= DCA(\langle \mathtt{s}_1^{\hat{c}_1}, \mathtt{s}_2^{c_2}, \ldots, \mathtt{s}_k^{c_k} \rangle, L) +\!\!+ DCA(\langle {}^{\hat{c}_1}\mathtt{s}_1, {}^{c_2}\mathtt{s}_2, \ldots, {}^{c_k}\mathtt{s}_k \rangle, L)$, otherwise

where $\hat{c}_1 := \left\lceil \frac{|\mathtt{s}_1|}{2} \right\rceil$

$\langle c_2, \ldots, c_k \rangle := C\text{-}opt(\langle \mathtt{s}_1, \mathtt{s}_2, \ldots, \mathtt{s}_k \rangle, \hat{c}_1)$

□

```
2.1  C_min := +∞
2.2  for c_2 := 0, ..., |s_2| do
         for c_3 := 0, ..., |s_3| do
             ...
                 for c_k := 0, ..., |s_k| do
                     if C(⟨ĉ_1, c_2, c_3, ..., c_k⟩) < C_min then
                         C_min := C(⟨ĉ_1, c_2, c_3, ..., c_k⟩)
                         c_min := ⟨c_2, c_3, ..., c_k⟩
2.3  return c_min
```

Figure 4.7: Naive implementation of Step 2 of function *C-opt*.

Note that choosing the parameter L rather small might result in a sequence $\mathbf{s}_1$ which is empty or consists of one letter only such that no proper slicing of $\mathbf{s}_1$ is possible. The worst case scenario is then the following: All slicing positions c_i, $2 \leq i \leq k$, of the corresponding C-optimal cut lie before the first or after the last letter of "their" sequence such that none of the sequences gets shorter from one division step to the next. Consequently, the recursion never stops. To avoid this, we re-order the sequences with each call of *DCA* so that always the longest of the sequences under consideration is chosen as $\mathbf{s}_1$.

The problem of computing optimal sequence alignments (*MSA*) is well understood (see the discussion in Chapter 3), and highly developed solutions are available which work well for short and/or highly similar sequences. So we focus our further work on an efficient implementation of *C-opt* which also needs a sophisticated treatment as well. Various realizations are discussed below and in Chapter 5.

In a straightforward implementation of *C-opt*, $C(\langle \hat{c}_1, c_2, \ldots, c_k \rangle)$ is computed for each family $\langle c_2, \ldots, c_k \rangle$, and that cut minimizing this value is returned:

Function

C-opt$(\langle \mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_k \rangle, \hat{c}_1)$

1. Calculate and save additional-cost matrices $C_{\mathbf{s}_p, \mathbf{s}_q}$ for all (p, q), $1 \leq p < q \leq k$.
2. For all $(c_2, \ldots c_k)$, $c_i \in \{0, \ldots, |\mathbf{s}_i|\}$, compute $C(\langle \hat{c}_1, c_2, \ldots, c_k \rangle)$, and return that family $\langle c_2, \ldots, c_k \rangle$ which minimizes this value.

□

For a more detailed description of Step 2, see Figure 4.7.

A single call of this implementation of *C-opt* with sequences of length at most n has time complexity $\mathcal{O}(k^2 n^2 + n^{k-1})$. The term $k^2 n^2$ comes from calculating the

$C_{\mathbf{s}_p,\mathbf{s}_q}$, while the term n^{k-1} comes from searching the C-optimal cut. The space requirement is $\mathcal{O}(k^2n^2)$ for the $\binom{k}{2}$ additional-cost matrices $C_{\mathbf{s}_p,\mathbf{s}_q}$. (Note that of the matrices $C_{\mathbf{s}_1,\mathbf{s}_q}$ with fixed slicing position $\hat{c}_1$ only the column $Col^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_q}$ has to be saved. Nevertheless, the required space still grows quadratically with the number of sequences.)

The time complexity for the entire call $DCA(\langle \mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_k \rangle, L)$ can be estimated as follows: We assume all sequences $\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_k$ to be of the same length $n = L \cdot 2^D$ and all slicing positions being exactly at the midpoint of the (sub)sequences. Thus, in each dividing level $d = 0, \ldots, D-1$, *C-opt* is called 2^d times with sequence length $n_d := \frac{n}{2^d}$. Finally, $\frac{n}{L}$ families with sequences of length L each remain to be aligned optimally. Thus, the overall time complexity T_{DCA} is given by

$$\begin{aligned} T_{DCA} &= \sum_{d=0}^{D-1} 2^d \left(k^2 n_d^2 + n_d^{k-1}\right) + \frac{n}{L}\left(2^k L^k\right) \\ &= k^2 n^2 \sum_{d=0}^{D-1} \left(\frac{1}{2}\right)^d + n^{k-1} \sum_{d=0}^{D-1} \left(\frac{1}{2^{k-2}}\right)^d + 2^k n L^{k-1} \\ &= 2k^2 n^2 \left(1 - \frac{1}{2^D}\right) + n^{k-1} \frac{1 - \left(\frac{1}{2^{k-2}}\right)^D}{1 - \frac{1}{2^{k-1}}} + 2^k n L^{k-1} \end{aligned}$$

which is in $\mathcal{O}(k^2n^2 + n^{k-1} + 2^k n L^{k-1})$ for $D > 0$ and $k > 1$. For the extrema of L, we obtain the expected results:

1. In case $L = n$ (or, equivalently, $D = 0$), the first and second summands disappear. What remains is the time required for complete dynamic programming of the original sequences $T_{DCA} = 2^k n^k$.

2. For the shortest possible stop size $L = 1$ ($D = \log_2 n$), the first and second summands remain in $\mathcal{O}(k^2n^2)$ and $\mathcal{O}(n^{k-1})$, respectively, while the third summand reduces to $2^k n$. Thus, almost all time is consumed in the dividing phase (at least for $n > 2$).

The space requirement S_{DCA} of the overall procedure is

$$\begin{aligned} S_{DCA} &= \max_{0 \le d < D} \{k n_d + \binom{k-1}{2} n_d^2\} + L^k \\ &= kn + \binom{k-1}{2} n^2 + L^k \end{aligned}$$

which is in $\mathcal{O}(k^2n^2 + L^k)$. The $\mathcal{O}(n^k)$ space usage of standard dynamic programming is reduced to the space for the additional-cost matrices plus the space required for aligning the remaining (sub)sequences of length at most L. Thus, by choosing L appropriately small, the essential space usage is quadratic in n even without using a

2.1 $C_{min} := +\infty$
2.2 for $c_2 := \left\lceil \frac{|s_2|}{2} \right\rceil, \left\lceil \frac{|s_2|}{2} \right\rceil - 1, \left\lceil \frac{|s_2|}{2} \right\rceil + 1, \ldots$ do
if $C(\langle \hat{c}_1, c_2 \rangle) < C_{min}$ then
for $c_3 := \left\lceil \frac{|s_3|}{2} \right\rceil, \left\lceil \frac{|s_3|}{2} \right\rceil - 1, \left\lceil \frac{|s_3|}{2} \right\rceil + 1, \ldots$ do
if $C(\langle \hat{c}_1, c_2, c_3 \rangle) < C_{min}$ then
$\ddots$
for $c_k := \left\lceil \frac{|s_k|}{2} \right\rceil, \left\lceil \frac{|s_k|}{2} \right\rceil - 1, \left\lceil \frac{|s_k|}{2} \right\rceil + 1, \ldots$ do
if $C(\langle \hat{c}_1, c_2, c_3, \ldots, c_k \rangle) < C_{min}$ then
$C_{min} := C(\langle \hat{c}_1, c_2, c_3, \ldots, c_k \rangle)$
$c_{min} := \langle c_2, c_3, \ldots, c_k \rangle$
2.3 return c_{min}

Figure 4.8: Improved version of Step 2 of function *C-opt*.

more sophisticated implementation of *C-opt*. This is one of the main advantages of our algorithm!

Nevertheless, the time required by *DCA* is still exponential in the number of sequences, only reduced by a factor of n. Actually, when taking care about dependencies in the search space, the average running time will fall considerably below this worst case complexity. A simple and effective branch-and-bound approach is the following: Since the multiple additional cost $C(\hat{c}_1, c_2, \ldots, c_k)$ is a sum of $\binom{k}{2}$ non-negative numbers, whenever a partial sum is larger than the minimum found so far, denoted by C_{min}, the search with the current slicing positions can be stopped. Additionally, to obtain early a low value for C_{min}, we start the search in the middle of each sequence. Proceeding in this way, we also get the midmost slicing sites if there is more than one C-optimal family of slicing positions. Due to being nearest to the symmetric case, this might also result in better alignments. The improved algorithm is shown in Figure 4.8.

Although with these improvements the worst case running time is still exponential in k, the algorithm applied to up to eight sequences computes excellent scoring alignments very quickly (see Chapter 7 for exact running times).

4.5 Variations of the Basic Algorithm

Many alignment procedures, especially those presented in the biological literature, combine well-known alignment strategies with additional features such as biochemical, structural, or functional knowledge about the sequences (see for instance [22, 200, 173]). Indeed, it turned out that most of these approaches (at least those which have been published) generate (in biological terms) better alignments than the "pure" algorithms [19, 88, 200]. Proceeding similarly, DCA can certainly be refined or com-

bined with other heuristics to obtain alignments still nearer to the biologically true alignment rather than the weighted SP optimum we pursue.

The spirit of this work, instead, is merely of theoretical nature. We are mainly interested in the principles underlying the optimization problem caused by the search for C-optimal slicing positions. For this purpose, we developed several solutions for a reduction of the practical time complexity of DCA which we present in Chapter 5. However, some variations regarding the quality of the alignments obtained with DCA have been worked out. These are briefly presented in the remainder of this chapter.

4.5.1 Stopping Criteria

In our implementation of DCA, the recursion in the dividing phase is continued until *all* (sub)sequences under consideration are shorter than the parameter L. The reason for this choice is rather simple: The time consumed by the computation of the optimal alignments can be delimited by choosing L appropriately small. A similar effect could be obtained by stopping the recursion whenever the product $\prod_{i=2}^{k}(|\mathbf{s}_i| + 1)$ is smaller than a certain threshold. Since this value is proportional to the search space of standard dynamic programming, even a better estimate of the time required for the optimal alignment of the (sub)sequences may be obtained. However, both approaches require a re-ordering of the sequences to ensure successful termination of the recursion phase. Another solution to this problem is to stop the recursion whenever the *shortest* (sub)sequence under consideration is shorter than L (see [203]). But in this case nothing can be predicted about the time required for the optimal (sub)sequence alignments: Consider a sequence family consisting of one short sequence (shorter than L) and $k-1$ arbitrarily long sequences.

Another stopping criterion which has been suggested is the *recursion depth* [203]: Depending on the original sequence length, a threshold D for the maximal number of recursive calls of *DCA* is chosen such that after D recursions there remain 2^D families of (sub)sequences which are to be aligned optimally. This way we obtain a balanced recursion tree, allowing a simple theoretical analysis of the dividing phase. But properties of the sequences are utterly ignored and an estimation of the running time of the remaining optimal alignments is not possible in this way.

Of course, much more elaborated sequence properties than simply their length can be taken into account. One could estimate the homology of the (sub)sequences and stop the recursion when the similarity is lower than a given threshold. This way, bad alignments resulting from suboptimal slicing positions may be avoided. However, time considerations justify also the opposite argumentation: The closer the sequences are related, the earlier the division phase can be stopped because standard procedures for optimal alignment usually proceed much faster for closely related sequences.

Beneath the more complicated control structure required, these approaches raise the difficult question:

Is it possible to infer reliable statements about similarity of multiple sequences from pairwise comparisons, only?

In general, sophisticated approaches like those outlined above seem to be more useful if the relatedness of the sequences under consideration differs considerably along their length. In the general study of the divide-and-conquer alignment method to which this work is devoted, sequence length as stopping criterion completely suffices.

4.5.2 Windowing

To improve the quality of the alignments obtained with DCA, a *windowing approach* has been proposed (see [192]): In some alignments, gaps in the proximity of division sites are not distributed optimally. To correct the alignment in these cases, inside a window of width W (which can be chosen depending on the threshold L), placed across each slicing site of the alignment that is obtained with the standard procedure, the subsequences are re-aligned optimally. Improvements (in terms of alignment score) resulting from this approach are presented in Section 7.2.2.

4.5.3 Relaxing $\hat{c}_1$

As another variation of the basic algorithm, the constraint of fixing the slicing position $\hat{c}_1$ exactly at the midpoint of $\mathbf{s}_1$ may be relaxed (see [203]). One might obtain better alignments (i.e. alignments with lower distance score) if the sequences are cut at positions $\langle \check{c}_1, c_2, \ldots, c_k \rangle$ which minimize $C(\langle \check{c}_1, c_2, \ldots, c_2 \rangle)$ where all positions in a region of size $2\Delta + 1$, $\Delta \geq 0$, around the midpoint of $\mathbf{s}_1$ are considered: $\check{c} \in \{\left\lceil \frac{|\mathbf{s}_1|}{2} \right\rceil - \Delta, \ldots, \left\lceil \frac{|\mathbf{s}_1|}{2} \right\rceil + \Delta\}$.

On the other hand, any $\hat{c}_1 \in \{0, \ldots, |\mathbf{s}_1|\}$ permits optimal slicing positions (see Lemma 4.2). If the C-optimal slicing positions obtained with $\hat{c}_1 = \left\lceil \frac{|\mathbf{s}_1|}{2} \right\rceil$ are suboptimal, will that be different for the other positions $\check{c}_1$ around $\hat{c}_1$? Furthermore it is not totally clear if it makes sense to compare the absolute additional-cost values at different sites of $\mathbf{s}_1$. In principle, better (i.e. lower) absolute multiple additional costs at positions $\check{c}_1$ next to the midpoint of $\mathbf{s}_1$ in an early stage of the recursion phase can also lead into only a local optimum, resulting in an overall worse alignment.

Results concerning the quality of alignments depending on the position of $\hat{c}_1$ are shown in Section 7.2.3.

4.5.4 Rapid Simultaneous Three-Way Alignment

In 1994, Vingron and v. Haeseler [210, 211] developed an algorithm which computes simultaneously a multiple sequence alignment and a phylogenetic tree. While theoretically well-founded, this generalized tree alignment procedure was rather impractical because of its long running time due to a large number of (cubic time) three-way alignments that are performed to determine an optimal tree topology.

As described above, the time used by the alignment of three sequences with DCA grows essentially quadratic with the sequence length. Indeed, by incorporation of DCA, the performance of the method could be increased enormously. Thus, one of the best performing generalized tree alignment methods currently available was obtained [31].

4.5.5 Combination with Fragment-Based Methods

As outlined in Section 3.5, the divide-and-conquer alignment procedure is somehow related to fragment-based multiple alignment methods using a more systematic way of computing the anchor positions. On the other hand, if there are obvious anchors obtainable from local alignments, the effort to compute C-optimal slicing positions can be circumvented. A combination of both approaches may be a suitable solution: Anchors are computed as in one of the standard fragment-based methods (e.g. [208, 177, 140]) asking for high similarity scores if necessary, and C-optimal slicing positions are computed between them if – due to the strong requirements for fragment similarity – the intermediate regions are too large. Then, the remaining (short) subsequences in between are aligned optimally as in the standard algorithms.

Note that an algorithm proceeding this way is expected to run faster than DCA: The search for conserved regions including the consistency check can be performed in time $\mathcal{O}(n^3k^4)$ [208] and thus is polynomial in the number of sequences and not exponential as the search for C-optimal slicing positions.

Chapter 5

Reducing Computation Time

In Section 4.4, we gave a straightforward implementation of the function *C-opt* which, for a given family of sequences $\langle \mathtt{s}_1, \mathtt{s}_2, \ldots, \mathtt{s}_k \rangle$, computes a C-optimal family of slicing positions $\langle c_2, \ldots, c_k \rangle$ with respect to a fixed $\hat{c}_1$. The running time was $\mathcal{O}(k^2 n^2 + n^{k-1})$ with a space consumption of $\mathcal{O}(k^2 n^2)$. Although for three sequences, this is a remarkable improvement compared to the cubic space and time complexity of standard dynamic programming, computing time still grows exponentially with the number of sequences. In this chapter, we discuss approaches for efficient search strategies based on branch-and-bound techniques leading to significant reductions in running time for more than three sequences. This way, we obtain a fast procedure for the simultaneous alignment of more than a dozen of related sequences of the length of an average protein.

5.1 Basic Approach

Let us first recall the problem. Suppose, as above, that we have given a family of k sequences $\langle \mathtt{s}_1, \mathtt{s}_2, \ldots, \mathtt{s}_k \rangle$ and pairwise sequence weights $\alpha_{p,q}$ for all (p, q), $1 \leq p < q \leq k$. Further, we fix a slicing position $\hat{c}_1$ of sequence $\mathtt{s}_1$. Our goal is to find a family $\langle c_2, \ldots, c_k \rangle$ of slicing positions which is C-optimal with respect to $\hat{c}_1$, i.e. it minimizes the multiple additional cost

$$C(\langle \hat{c}_1, c_2, \ldots, c_k \rangle) \;=\; \sum_{2 \leq q \leq k} \alpha_{1,q}\; Col^{\hat{c}_1}_{\mathtt{s}_1, \mathtt{s}_q}[c_q] \;+\; \sum_{2 \leq p < q \leq k} \alpha_{p,q}\; C_{\mathtt{s}_p, \mathtt{s}_q}[c_p, c_q].$$

Figure 5.1 gives a schematic representation of the additional-cost matrices involved in this search.

Our general approach to speed up the search is the following one: We precalculate an upper bound for $C_{opt}(\hat{c}_1)$, denoted by $\widehat{C}$, before storing the matrices $C_{\mathtt{s}_p, \mathtt{s}_q}$. This estimate allows us to reduce the search space enormously by a branch-and-bound technique comparable to that discussed at the end of Section 4.4: A family of slicing positions $\langle c_2, \ldots, c_k \rangle$ can be excluded whenever one of the summands or a partial sum of $C(\langle \hat{c}_1, c_2, \ldots, c_k \rangle)$ is larger than the upper bound $\widehat{C}$. In particular, for fixed $\hat{c}_1$, any

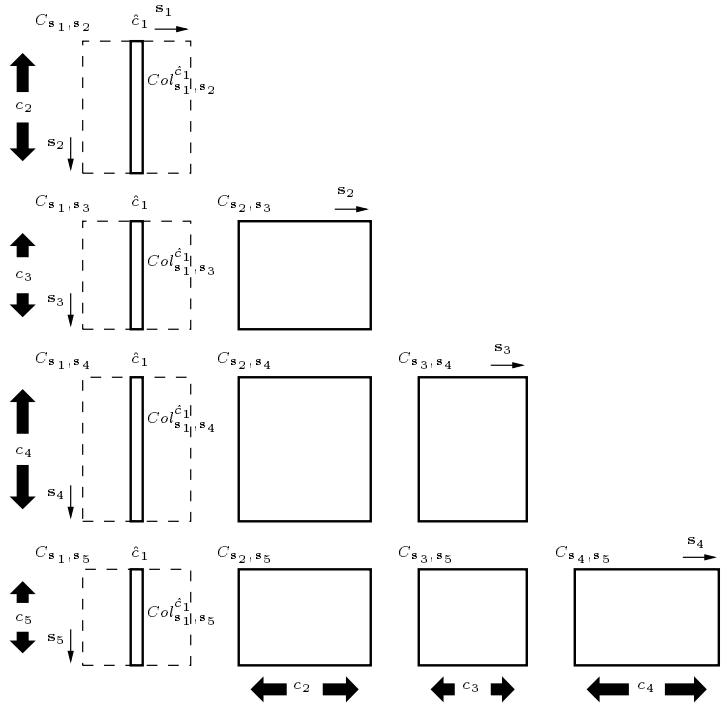

Figure 5.1: Schematic representation of the search space for a C-optimal family of slicing positions $\langle c_2, c_3, c_4, c_5 \rangle$ with respect to a given $\hat{c}_1$.

c_q, $2 \leq q \leq k$ with $\alpha_{1,q} \cdot Col^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_q}[c_q] \geq \widehat{C}$ can never lead to a sum $C(\langle \hat{c}_1, c_2, \ldots, c_k \rangle)$ which is smaller than $\widehat{C}$. Hence, for given $\widehat{C}$, we can determine large regions in our search space which cannot contain the minimum:

Definition 5.1 (Lower and Upper Bounds, Relevant Parts)
Given an upper bound $\widehat{C}$ for the optimal multiple additional cost C_{opt}, we define *lower* and *upper bounds* l_q and u_q, respectively, for all $q \in \{2, \ldots, k\}$ such that

$$\alpha_{1,q} \cdot Col^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_q}[j] \;\; \geq \;\; \widehat{C} \quad \text{for all } j < l_q \text{ and for all } j > u_q.$$

The intermediate segment with indices $l_q, \ldots, u_q$ forms the *relevant part* of column $Col^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_q}$ with respect to $\widehat{C}$. The relevant part of the matrix $C_{\mathbf{s}_p,\mathbf{s}_q}$ is the matrix

$$(C_{\mathbf{s}_p,\mathbf{s}_q}[i,j])_{l_p \leq i \leq u_p, l_q \leq j \leq u_q}.$$

□

In Figure 5.2, we show the relevant parts of the columns and matrices for a given $\widehat{C}$ in a schematic way.

Various ways of computing an upper bound $\widehat{C}$ are discussed in Section 5.3. Given such an estimate, the memory required for storing the relevant parts of the columns

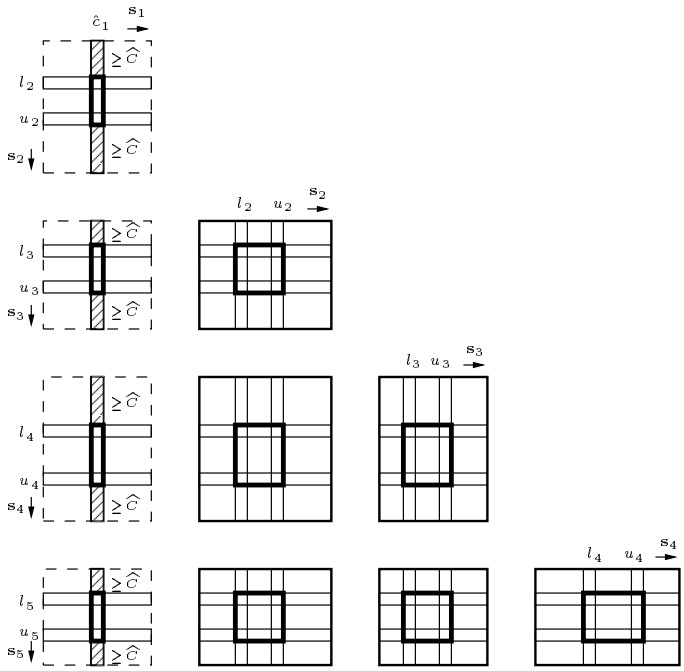

Figure 5.2: Relevant parts of the columns and additional-cost matrices for a given $\widehat{C}$.

and matrices suffices for computing C_{opt}. Yet, the actual time and space requirements depend strongly on the quality of the estimate $\widehat{C}$. Let $r := \max_{2 \leq q \leq n}\{u_q - l_q + 1\}$ be the maximal size of the relevant parts of the columns $Col^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_q}$. Then, the larger the upper bound $\widehat{C}$ is, the larger is the relative length of the relevant parts, i.e. the ratio $\frac{r}{n}$. For a very high $\widehat{C}$, r can even be as large as n. But for more realistic upper bounds, the effect is enormous: We have measured quotients $\frac{r}{n}$ of less than $\frac{1}{100}$ for small k or large n yielding memory savings for the matrices of four orders of magnitude in these cases. See Section 7.3 for detailed quantitative results.

Note that even the best possible upper bound, C_{opt} itself, does not imply $u_q = l_q$ for all $q \in \{2, \ldots, k\}$. The search space still grows exponentially with k even for $\widehat{C} = C_{opt}$. The asymptotic time complexity is $\mathcal{O}(k^2n^2 + r^{k-1})$. The space requirement is $\mathcal{O}(n + k^2r^2)$ if the upper bound $\widehat{C}$ is given. (Note that the time and space of computing $\widehat{C}$ has to be added to these values.)

At this stage of the description, it also becomes apparent that DCA – like almost all multiple sequence alignment algorithms – will perform faster and require less space for closely related than for feebly related sequences: The size of the relevant search space is highly correlated not only with the quality of the upper bound $\widehat{C}$ but also with the absolute value of the minimum C_{opt}. As noted above, this value is much lower for similar sequences.

Also another effect can be noticed: The more sequences are involved, the less ef-

fective is our approach. In the definition of the upper and lower bounds, the estimate $\widehat{C}$, which is the sum of $\binom{k}{2}$ pairwise additional costs and hence grows quadratically with k, is compared to a single entry in coumn $Col^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_q}$ whose value does not depend on the number of sequences. In Section 7.3 where different ways of computing an upper bound $\widehat{C}$ are compared, we also show that for nine sequences the relative length $\frac{r}{n}$ is five times as large as for three sequences.

On the other hand, the longer the sequences are, the smaller is the relative length of the relevant parts: The maximal values at the top and at the bottom of the columns grow proportionally with the length of the sequences while the regions near the middle of the columns are hardly affected by the sequence length. Also the estimate $\widehat{C}$ keeps rather constant upon elongation of the sequences. Measurements confirming these assertions are shown in Section 7.3 as well.

Finally note that the shape of the additional-cost matrices and thus the computation time and memory usage of DCA are also influenced by the score function and the gap costs used. In particular, gap functions which allow many relatively long indels can widen the "relevant valley" along the main diagonal of the additional-cost matrices considerably.

In consequence, we give a refined version of the function *C-opt*. The problem is divided into two subproblems:

- Given a family $S = \langle \mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_k \rangle$ of sequences and a slicing position $\hat{c}_1$ of $\mathbf{s}_1$, function *calc-Chat* computes an estimated value $\widehat{C}$ ("*C-hat*") which is always larger than or equal to $C_{opt}(\hat{c}_1)$.

- Function *calc-Copt* takes this estimate as an argument (in addition to the sequences and $\hat{c}_1$) and then computes a family $\langle c_2, \ldots, c_k \rangle$ of slicing positions which is C-optimal with respect to $\hat{c}_1$.

Function

$$\textit{C-opt}(S, \hat{c}_1) \;:=\; \textit{calc-Copt}(S, \hat{c}_1, \textit{calc-Chat}(S, \hat{c}_1))$$

□

5.2 Analogy to the Approach of Carrillo and Lipman

Before we describe explicit realizations of the subfunctions *calc-Chat* and *calc-Copt* in Sections 5.3 and 5.4, respectively, we show in this section that there is a strong analogy between our approach to reduce the search space in the additional-cost matrices and the approach of Carrillo and Lipman for optimal sequence alignment which

we described in Section 3.4. For the benefit of brevity, we set again all weights $\alpha_{p,q}$ to 1.

Assume, as above, a family of sequences $\langle \mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_k \rangle$ and a fixed slicing position $\hat{c}_1$ of sequence $\mathbf{s}_1$. Let $\widehat{C}$ be a precalculated upper bound for the minimal multiple additional cost:

$$\widehat{C} \;\geq\; C_{opt}(\hat{c}_1) \;=\; \sum_{1 \leq i < j \leq k} C_{\mathbf{s}_i,\mathbf{s}_j}[\tilde{c}_i, \tilde{c}_j] \;=:\; \sum_{1 \leq i < j \leq k} \widetilde{C}_{\mathbf{s}_i,\mathbf{s}_j}$$

where $\langle \tilde{c}_2, \ldots, \tilde{c}_k \rangle$ is a (yet unknown) C-optimal family of slicing positions with respect to $\hat{c}_1$ and the $\widetilde{C}_{\mathbf{s}_i,\mathbf{s}_j}$ are the additional costs of the projections of these optimal slicing positions on the two-dimensional surfaces of the $(k-1)$-dimensional "hyperbox" spanned by the $\langle c_2, \ldots, c_k \rangle$. Similar to the approach of Carrillo and Lipman, we can now give upper bounds for these projection costs:

$$\widetilde{C}_{\mathbf{s}_p,\mathbf{s}_q} \;\leq\; \widehat{C} \;-\; \sum_{\substack{1 \leq i < j \leq k \\ (i,j) \neq (p,q)}} \widetilde{C}_{\mathbf{s}_i,\mathbf{s}_j}.$$

But in contrast to distance matrices where the pairwise optimal distances $w_{opt}(\langle \mathbf{s}_i, \mathbf{s}_j \rangle)$ are lower bounds for the values $\widetilde{D}_{\mathbf{s}_i,\mathbf{s}_j}$, in our case the only possible lower bound for the values $\widetilde{C}_{\mathbf{s}_i,\mathbf{s}_j}$ is 0: All additional-cost values are non-negative. Thus,

$$\widetilde{C}_{\mathbf{s}_i,\mathbf{s}_j} \;\geq\; 0 \qquad \Longrightarrow \qquad \widetilde{C}_{\mathbf{s}_p,\mathbf{s}_q} \;\leq\; \widehat{C},$$

i.e. a value larger than $\widehat{C}$ cannot be part of the minimal additional cost C_{opt}. This statement is not surprising and was already given in the previous section (see Definition 5.1).

However, by the analogy, the relatedness of the limitations of both approaches becomes apparent. In the previous section, we remarked a dependence of the quotient $\frac{r}{n}$ on the number k of sequences. For the same reason, the method of Carrillo and Lipman performs less effectively for increasing k: In both approaches, the reduction of search space is based on the comparison of a single matrix entry ($\widetilde{D}_{\mathbf{s}_p,\mathbf{s}_q}$ respectively $\widetilde{C}_{\mathbf{s}_p,\mathbf{s}_q}$) with a value which is the sum of $\binom{k}{2}$ such numbers ($\widehat{D}$ respectively $\widehat{C}$).

5.3 Computing an Upper Bound: *calc-Chat*

A straightforward method of computing an upper bound $\widehat{C}$ for $C_{opt}(\hat{c}_1)$ is the *calc-Chat-firstRowZero* procedure. It is based on the fact that in any of the columns $Col^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_q}$, $2 \leq q \leq k$, there is at least one index $c_q \in \{0, \ldots, |\mathbf{s}_q|\}$ with $Col^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_q}[c_q] = 0$ (see Lemma 4.5). From all these indices, we choose that one which is nearest to the

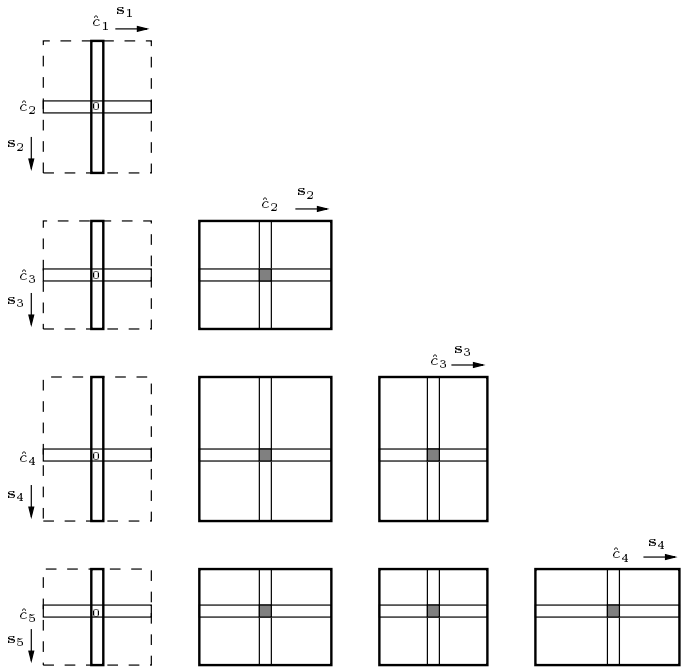

Figure 5.3: Schematic representation of *calc-Chat-firstRowZero*. The estimate $\widehat{C}$ is the sum of the values in the shaded boxes.

midpoint between 0 and $|\mathbf{s}_q|$, denoted[1]

$$\hat{c}_q \;:=\; \underset{0\le c_q\le|\mathbf{s}_q|}{\text{midmost}}\{c_q|Col^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_q}[c_q]=0\}.$$

Then, the upper bound $\widehat{C}$ is computed from the corresponding cells $C_{\mathbf{s}_p,\mathbf{s}_q}[\hat{c}_p,\hat{c}_q]$ in the additional-cost matrices (see Figure 5.3):

Function
calc-Chat-firstRowZero$(\langle\mathbf{s}_1,\mathbf{s}_2,\ldots,\mathbf{s}_k\rangle,\hat{c}_1) \;:=\; C(\langle\hat{c}_1,\hat{c}_2,\ldots,\hat{c}_k\rangle)$

where $\hat{c}_q \;:=\; \underset{0\le c_q\le|\mathbf{s}_q|}{\text{midmost}}\{c_q|Col^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_q}[c_q]=0\}$ for all $q=2,\ldots,k$.

□

Note that in the computation of $C(\langle\hat{c}_1,\hat{c}_2,\ldots,\hat{c}_k\rangle)$, we do not have to consider entries from the columns $Col^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_p}$ since they are zero by definition. Thus, in this case,

$$C(\langle\hat{c}_1,\hat{c}_2,\ldots,\hat{c}_k\rangle) \;=\; \sum_{2\le p<q\le k}\alpha_{p,q}\;C_{\mathbf{s}_p,\mathbf{s}_q}[\hat{c}_p,\hat{c}_q].$$

[1] We use the notation $\underset{0\le i\le n}{\text{midmost}}\{i|p(i)\}$ to denote that i of those indices fulfilling the predicate $p(i)$ which is nearest to the midpoint of the given range, i.e. $\hat{\imath}=\underset{0\le i\le n}{\text{midmost}}\{i|p(i)\}$ if and only if $|\frac{n}{2}-\tilde{\imath}|\ge|\frac{n}{2}-\hat{\imath}|$ for all $\tilde{\imath}\in\{i|p(i)\}_{0\le i\le n}$. (In case two indices with the same distance from the midpoint exist, we define the lower one to be the "midmost".)

The procedure requires $\mathcal{O}(k^2n^2)$ time to compute $\widehat{C}$: It computes

(i) the columns $Col^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_q}$ in $\mathcal{O}(kn^2)$ time,

(ii) the indices $\hat{c}_q$ in $\mathcal{O}(kn)$ time,

(iii) the corresponding entries $C_{\mathbf{s}_p,\mathbf{s}_q}[\hat{c}_p, \hat{c}_q]$ in $\mathcal{O}(k^2n^2)$ time, and

(iv) the sum $C(\langle \hat{c}_1, \hat{c}_2, \ldots, \hat{c}_k \rangle)$ in $\mathcal{O}(k^2)$ time.

The space can be reduced to $\mathcal{O}(n+k)$ when all additional-cost matrices are computed one after the other using the $\mathcal{O}(n)$ space method outlined in Section 4.2. (The indices $\hat{c}_q$, $1 \leq q \leq k$, take the additional $\mathcal{O}(k)$ space.) But proceeding this way, the columns $Col^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_q}$ have to be computed a second time when the lower and upper bounds of the relevant regions are to be determined. Since the later search phase requires – at least in the average case – a considerably larger amount of space than this stage of the algorithm, we prefer the faster variant where the columns $Col^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_q}$ are saved, using $\mathcal{O}(kn)$ space.

5.3.1 Some $\mathcal{O}(k^2n^2)$-Time Methods

In this and the following sections, we propose natural generalizations of the *calc-Chat-firstRowZero* function. Some of them lead to provably better, that is, smaller estimates $\widehat{C}$. If the calculation of these estimates is fast enough, we obtain faster versions of *DCA*.

calc-Chat-SCmin

In the *calc-Chat-firstRowZero* procedure, each index $\hat{c}_q$, $2 \leq q \leq k$, is chosen depending only on $\hat{c}_1$. Because we focus on the sum-of-pairs score, each slicing position c_q in a family with minimal additional cost $C_{opt}(\hat{c}_1)$ depends on all other indices c_p, $p \neq q$.

In the following variant denoted *calc-Chat-SCmin* (short for "**S**um of corresponding **C**olumns") of the *calc-Chat-firstRowZero* procedure, when fixing an index c_q, we consider also the slicing positions $\hat{c}_2, \ldots, \hat{c}_{q-1}$ computed for the previous sequences and minimize over the sum of the entries in the corresponding columns of the additional-cost matrices (see Figure 5.4).

Function

$$\textit{calc-Chat-SCmin}(\langle \mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_k \rangle, \hat{c}_1) \;:=\; C(\langle \hat{c}_1, \hat{c}_2, \ldots, \hat{c}_k \rangle)$$
$$\text{where } \hat{c}_q \;:=\; \underset{0 \leq c_q \leq |\mathbf{s}_q|}{\text{midmost}} \{ c_q \mid \sum_{p=1}^{q-1} \alpha_{p,q}\; Col^{\hat{c}_p}_{\mathbf{s}_p,\mathbf{s}_q}[c_q] \text{ is minimal} \} \quad \text{for all } q = 2, \ldots, k.$$

□

Due to the greedy strategy, *calc-Chat-SCmin* is sensitive to the whole input order of the sequences while *calc-Chat-firstRowZero* depends only on the choice of $\mathbf{s}_1$. The

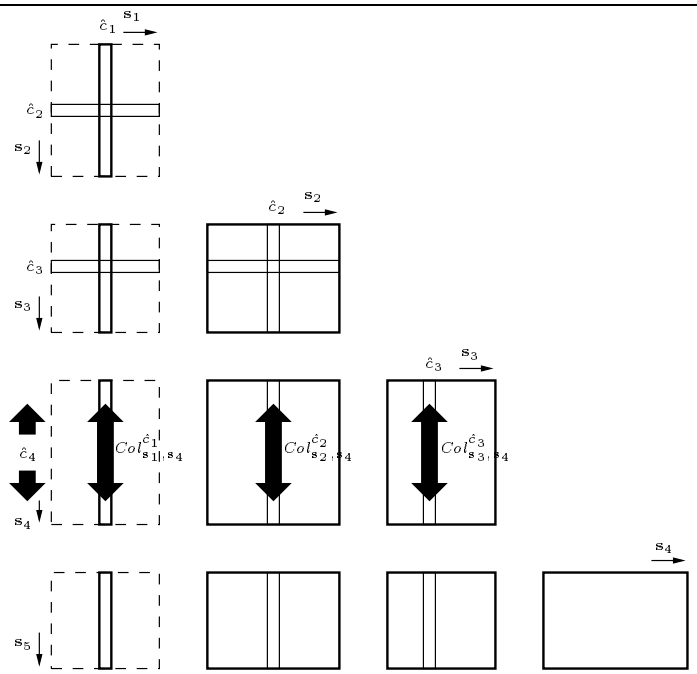

Figure 5.4: Schematic representation of *calc-Chat-SCmin*. The large arrows show the columns which are involved in the minimization process for $\hat{c}_4$.

time complexity $\mathcal{O}(k^2n^2)$ is the same as that of *calc-Chat-firstRowZero*. The space usage is $\mathcal{O}(kn)$ since the minimization is performed in up to $k-1$ columns simultaneously.

calc-Chat-SCRmin

Another variant of the *calc-Chat-firstRowZero* procedure is obtained by taking into account the sum not only of the columns $Col^{\hat{c}_p}_{\mathbf{s}_p,\mathbf{s}_q}$ for $p < q$ but also of the rows $Row^{\bar{c}_p}_{\mathbf{s}_q,\mathbf{s}_p}$ for $p > q$ where the positions $\bar{c}_p$ are computed previously in a *calc-Chat-firstRowZero*-like step (see Figure 5.5).

Function

$$\textit{calc-Chat-SCRmin}(\langle \mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_k \rangle, \hat{c}_1) \;:=\; C(\langle \hat{c}_1, \hat{c}_2, \ldots, \hat{c}_k \rangle)$$

$$\text{where } \bar{c}_p \;:=\; \underset{0 \le c_p \le |\mathbf{s}_p|}{\text{midmost}}\{c_p \mid Col^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_p}[c_p] = 0\} \quad \text{for all } p = 2, \ldots, k$$

$$\hat{c}_q \;:=\; \underset{0 \le c_q \le |\mathbf{s}_q|}{\text{midmost}}\{c_q \mid \sum_{p=1}^{q-1} \alpha_{p,q}\; Col^{\hat{c}_p}_{\mathbf{s}_p,\mathbf{s}_q}[c_q] \;+\; \sum_{p=q+1}^{k} \alpha_{q,p}\; Row^{\bar{c}_p}_{\mathbf{s}_q,\mathbf{s}_p}[c_q] \text{ is minimal}\}$$

$$\text{for all } q = 2, \ldots, k.$$

□

The name *calc-Chat-SCRmin* of this variant is short for "**S**um of corresponding **C**olumns and **R**ows". Function *calc-Chat-SCRmin* has the same asymptotic time complexity $\mathcal{O}(k^2n^2)$ as the other procedures. The required space is doubled but still in $\mathcal{O}(kn)$. As for *calc-Chat-SCmin*, the result depends on the order in which we process our sequences $\langle \mathbf{s}_1, \ldots, \mathbf{s}_k \rangle$. But this algorithm leads to a provably better (i.e. smaller) upper bound $\widehat{C}$:

Lemma 5.2

$$\textit{calc-Chat-SCRmin}(S, \hat{c}_1) \quad \leq \quad \textit{calc-Chat-firstRowZero}(S, \hat{c}_1).$$

□

Proof

To keep the exposition shorter, we set all sequence weights $\alpha_{p,q}$ to 1. Assume $k > 2$, $\kappa \in \{2, \ldots, k\}$. By definition of $C(\langle c_1, \ldots, c_k \rangle)$, we have

$$\begin{aligned}
&C(\langle \hat{c}_1, \ldots, \hat{c}_{\kappa-1}, \bar{c}_\kappa, \ldots, \bar{c}_k \rangle) \\
&= \sum_{1 \leq p < q \leq \kappa-1} C_{\mathbf{s}_p, \mathbf{s}_q}[\hat{c}_p, \hat{c}_q] \; + \sum_{p=1}^{\kappa-1} \sum_{q=\kappa}^{k} C_{\mathbf{s}_p, \mathbf{s}_q}[\hat{c}_p, \bar{c}_q] \; + \sum_{\kappa \leq p < q \leq k} C_{\mathbf{s}_p, \mathbf{s}_q}[\bar{c}_p, \bar{c}_q] \\
&= \sum_{1 \leq p < q \leq \kappa-1} C_{\mathbf{s}_p, \mathbf{s}_q}[\hat{c}_p, \hat{c}_q] \; + \sum_{p=1}^{\kappa-1} C_{\mathbf{s}_p, \mathbf{s}_\kappa}[\hat{c}_p, \bar{c}_\kappa] \; + \sum_{p=1}^{\kappa-1} \sum_{q=\kappa+1}^{k} C_{\mathbf{s}_p, \mathbf{s}_q}[\hat{c}_p, \bar{c}_q] \\
&\quad + \sum_{q=\kappa+1}^{k} C_{\mathbf{s}_\kappa, \mathbf{s}_q}[\bar{c}_\kappa, \bar{c}_q] \; + \sum_{\kappa+1 \leq p < q \leq k} C_{\mathbf{s}_p, \mathbf{s}_q}[\bar{c}_p, \bar{c}_q] \\
&\geq \sum_{1 \leq p < q \leq \kappa-1} C_{\mathbf{s}_p, \mathbf{s}_q}[\hat{c}_p, \hat{c}_q] \; + \sum_{p=1}^{\kappa-1} \sum_{q=\kappa+1}^{k} C_{\mathbf{s}_p, \mathbf{s}_q}[\hat{c}_p, \bar{c}_q] \; + \sum_{\kappa+1 \leq p < q \leq k} C_{\mathbf{s}_p, \mathbf{s}_q}[\bar{c}_p, \bar{c}_q] \\
&\quad + \min_{0 \leq c_\kappa \leq |\mathbf{s}_\kappa|} \left\{ \sum_{p=1}^{\kappa-1} C_{\mathbf{s}_p, \mathbf{s}_\kappa}[\hat{c}_p, c_\kappa] \; + \sum_{q=\kappa+1}^{k} C_{\mathbf{s}_\kappa, \mathbf{s}_q}[c_\kappa, \bar{c}_q] \right\} \\
&= \sum_{1 \leq p < q \leq \kappa-1} C_{\mathbf{s}_p, \mathbf{s}_q}[\hat{c}_p, \hat{c}_q] \; + \sum_{p=1}^{\kappa-1} \sum_{q=\kappa+1}^{k} C_{\mathbf{s}_p, \mathbf{s}_q}[\hat{c}_p, \bar{c}_q] \; + \sum_{\kappa+1 \leq p < q \leq k} C_{\mathbf{s}_p, \mathbf{s}_q}[\bar{c}_p, \bar{c}_q] \\
&\quad + \sum_{p=1}^{\kappa-1} C_{\mathbf{s}_p, \mathbf{s}_\kappa}[\hat{c}_p, \hat{c}_\kappa] \; + \sum_{q=\kappa+1}^{k} C_{\mathbf{s}_\kappa, \mathbf{s}_q}[\hat{c}_\kappa, \bar{c}_q] \\
&= C(\langle \hat{c}_1, \ldots, \hat{c}_\kappa, \bar{c}_{\kappa+1}, \ldots, \bar{c}_k \rangle)
\end{aligned}$$

Applied for $\kappa = 2, \ldots, k$, the assertion of the lemma follows.

□

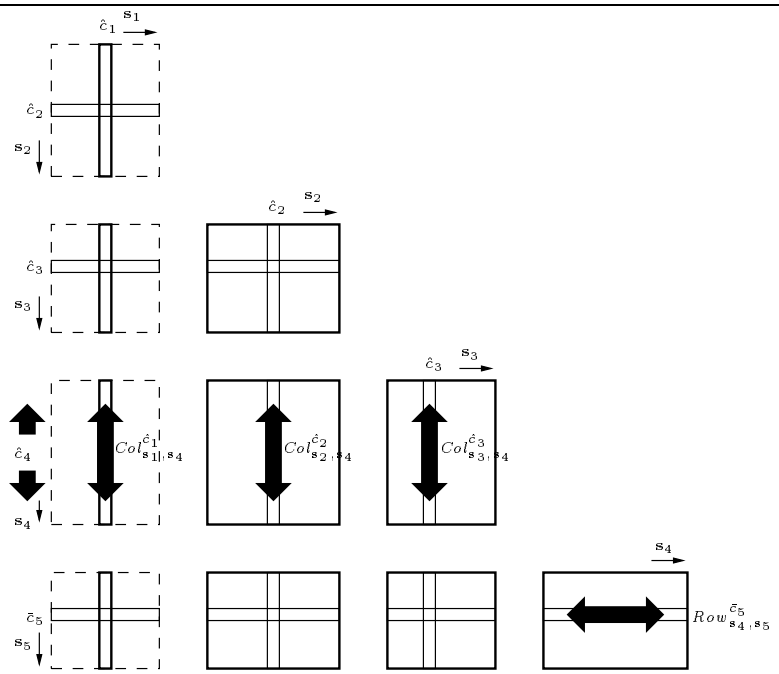

Figure 5.5: Schematic representation of *calc-Chat-SCRmin*. The large arrows show the columns and the row that are involved in the minimization process for $\hat{c}_4$.

Note that the formal descriptions of the functions *calc-Chat-...* in this section are still simplifications of the final implementations. For example, after the determination of the $\bar{c}_p$ in *calc-Chat-SCRmin*, already lower and upper bounds are computed, and the search for the $\hat{c}_q$ is restricted to the relevant parts of the columns and rows with respect to these bounds. Also, while computing the $\hat{c}_q$, the bounds can be updated on-the-fly for each smaller upper bound $\widehat{C}$. (This is also done in the later search for the minimum *calc-Copt.*) However, in order not to overcomplicate the formal presentation, we maintain the strict distinction between the computation of $\widehat{C}$ and the minimization stage.

calc-Chat-SCRoi

To avoid the dependence on the sequence order, we also developed a procedure which minimizes over columns and rows but whose result – like that of *calc-Chat-firstRowZero* – depends only on the choice of $\mathbf{s}_1$. This can be achieved by using the indices $\bar{c}_p$ not only to determine the rows but also the columns where the minimization takes place (see Figure 5.6):

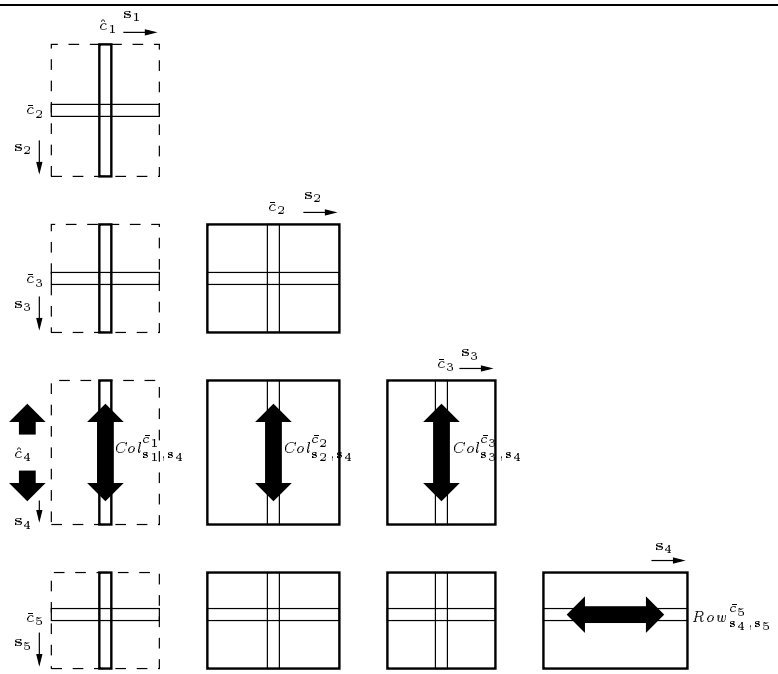

Figure 5.6: Schematic representation of *calc-Chat-SCRoi*. The large arrows show the columns and the row that are involved in the minimization process for $\hat{c}_4$. Note the indices $\bar{c}_2$ and $\bar{c}_3$ as opposed to $\hat{c}_2$ and $\hat{c}_3$, respectively, in Figure 5.5.

Function

$$\textit{calc-Chat-SCRoi}(\langle \mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_k \rangle, \hat{c}_1) \;:=\; C(\langle \hat{c}_1, \hat{c}_2, \ldots, \hat{c}_k \rangle)$$

where

$$\begin{aligned}
\bar{c}_1 &:= \hat{c}_1 \\
\bar{c}_p &:= \underset{0 \le c_p \le |\mathbf{s}_p|}{\text{midmost}} \{ c_p \,|\, Col^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_p}[c_p] = 0 \} \quad \text{for all } p = 2, \ldots, k \\
\hat{c}_q &:= \underset{0 \le c_q \le |\mathbf{s}_q|}{\text{midmost}} \{ c_q \,|\, \sum_{p=1}^{q-1} \alpha_{p,q} \; Col^{\bar{c}_p}_{\mathbf{s}_p,\mathbf{s}_q}[c_q] \;+\; \sum_{p=q+1}^{k} \alpha_{q,p} \; Row^{\bar{c}_p}_{\mathbf{s}_q,\mathbf{s}_p}[c_q] \text{ is minimal} \} \\
&\qquad\qquad \text{for all } q = 2, \ldots, k.
\end{aligned}$$

□

Obviously, *calc-Chat-SCRoi* ("**o**rder **i**ndependent") has the same time and space complexity as *calc-Chat-SCRmin*.

Note that the upper bounds computed with *calc-Chat-SCRoi* may be worse than those computed with *calc-Chat-SCRmin* since we do not use a greedy strategy which leads quickly to small values $\widehat{C}$. (In fact, we cannot even guarantee that the upper bounds are better than those computed with *calc-Chat-firstRowZero*.) However, due to the independence of the sequence order, the corresponding positions $\langle \hat{c}_2, \ldots, \hat{c}_k \rangle$ might be less susceptible of running into local minima of the search space.

In the following sections, we present extensions of the above procedures. Analogously to Lemma 5.2, it can be shown that some of them lead to further improvements of the estimate $\widehat{C}$.

5.3.2 Iterative Methods

A simple extension of *calc-Chat-SCRmin* is applying the minimization step iteratively: Given a number $I \geq 0$ of iterations, in each run $i = 1, \ldots, I$ we calculate the minimizing indices $\hat{c}_q^i$ for all $q = 2, \ldots, k$ by using the indices $\hat{c}_q^{i-1}$ obtained in run $i-1$, as we use the indices $\bar{c}_p$ for calculating the $\hat{c}_q$ in *calc-Chat-SCRmin*. Clearly, as starting points we put $\hat{c}_p^0 := \bar{c}_p$ for all $p = 2, \ldots, k$, so that the first run of the iterated procedure is *calc-Chat-SCRmin* itself.

Function

$$\textit{calc-Chat-itSCRmin}(\langle \mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_k \rangle, \hat{c}_1, I) \;:=\; C(\langle \hat{c}_1^I, \hat{c}_2^I, \ldots, \hat{c}_k^I \rangle)$$

where

$$\hat{c}_1^i := \hat{c}_1 \quad \text{for all } i = 0, \ldots, I$$

$$\hat{c}_p^0 := \underset{0 \leq c_p \leq |\mathbf{s}_p|}{\text{midmost}} \{ c_p \,|\, Col^{\hat{c}_1}_{\mathbf{s}_1, \mathbf{s}_p}[c_p] = 0 \} \quad \text{for all } p = 2, \ldots, k$$

$$\hat{c}_q^i := \underset{0 \leq c_q \leq |\mathbf{s}_q|}{\text{midmost}} \{ c_q \,|\, \sum_{p=1}^{q-1} \alpha_{p,q} \; Col^{\hat{c}_p^i}_{\mathbf{s}_p, \mathbf{s}_q}[c_q] \;+\; \sum_{p=q+1}^{k} \alpha_{q,p} \; Row^{\hat{c}_p^{i-1}}_{\mathbf{s}_q, \mathbf{s}_p}[c_q] \text{ is minimal} \}$$

for all $q = 2, \ldots, k$ and for all $i = 1, \ldots, I$.

□

The following lemma summarizes the properties of the upper bounds computed with this procedure.

Lemma 5.3

$$\text{(i)} \quad \textit{calc-Chat-itSCRmin}(S, \hat{c}_1, 0) \;=\; \textit{calc-Chat-firstRowZero}(S, \hat{c}_1)$$

$$\text{(ii)} \quad \textit{calc-Chat-itSCRmin}(S, \hat{c}_1, 1) \;=\; \textit{calc-Chat-SCRmin}(S, \hat{c}_1)$$

$$\text{(iii)} \quad \textit{calc-Chat-itSCRmin}(S, \hat{c}_1, i) \;\leq\; \textit{calc-Chat-itSCRmin}(S, \hat{c}_1, i-1)$$

for all $i > 0$.

□

Proof

(i) and (ii) are trivial. The proof of (iii) is analogous to that of Lemma 5.2: Each iteration i improves the estimate calculable from the slicing positions $\langle \hat{c}_1^i, \ldots, \hat{c}_k^i \rangle$ to a value smaller than or equal to $C(\langle \hat{c}_1^{i-1}, \ldots, \hat{c}_k^{i-1} \rangle)$. □

Because of the monotonously decreasing values of the upper bounds with increasing i, one can easily alter the stopping criterion slightly. The iteration should be carried out as long as further improvement is obtained. We observed that proceeding this way, the iteration stops mostly after two or three iterations. Note that with this alteration we have a *hill climbing* algorithm (in fact, our hill is a hollow) which in each iteration chooses the direction of steepest descent until a (local) minimum is reached. To avoid running into bad local minima, more sophisticated heuristic optimization strategies also seem to be applicable, e.g. Metropolis or Monte Carlo methods.

Besides the alternate stopping criterion, *calc-Chat-itSCRmin* needs computational time of the order $\mathcal{O}(Ik^2n^2)$ for calculating the indices $\langle \hat{c}_1^I, \hat{c}_2^I, \ldots, \hat{c}_k^I \rangle$ and is very useful in order to obtain near-to-optimal estimates $\widehat{C}$ (see the results shown in Section 7.3). To reduce the dependence of the upper bounds calculated with *calc-Chat-SCRmin* on the order of the sequences $\mathbf{s}_2, \ldots, \mathbf{s}_k$, one can also change this order from one iteration to the next.

Based on the variant *calc-Chat-SCRoi*, a similar iterative procedure (denoted *calc-Chat-itSCRoi*) is possible which is completely independent of the order of the sequences $\mathbf{s}_2, \ldots, \mathbf{s}_k$. But – as for *calc-Chat-SCRoi* – it cannot be guaranteed that the estimates obtained this way are always lower than or equal to those computed with *calc-Chat-firstRowZero*. However, as we shall see in Section 7.3, the estimates computed with *calc-Chat-itSCRoi* are almost as good as those computed with *calc-Chat-itSCRmin*.

5.3.3 Polynomial-Time Methods

Up to this stage we only have tried to minimize $C(\langle \hat{c}_1, c_2, \ldots, c_k \rangle)$ by varying one index at a time and keeping all the others fixed. In principle, one can carry out the minimization of sums of corresponding columns and rows over s indices *simultaneously* ($s \in \{2, \ldots, k-1\}$) which also reduces the probability of running into bad local minima. This leads to algorithms with a computational time complexity polynomial in n with highest exponent s (and quadratic in k). The exhaustive search for the minimum $C_{opt}(\hat{c}_1)$ described in Section 4.4 is identical to this procedure with $s = k - 1$.

However, we do not work out this approach explicitly since it is similar to a preprocessing stage in the calculation of the minimum C_{opt} which we describe in Section 5.4.2. For arbitrary $s \in \{2, \ldots, k-1\}$, the procedure (denoted *siMinC*) is described thoroughly in [157] where also the proof can be found that the upper bounds computed by this procedure are always smaller than or equal to the estimates computed by *calc-Chat-firstRowZero*.

5.4 Computing a C-Optimal Family of Slicing Positions: *calc-Copt*

We now consider the problem *calc-Copt*: Given a family of sequences $\langle \mathbf{s}_1, \ldots, \mathbf{s}_k \rangle$, a slicing position $\hat{c}_1$ of $\mathbf{s}_1$, and an upper bound $\widehat{C}$, compute a family of slicing positions $\langle c_2, \ldots, c_k \rangle$ which is C-optimal with respect to $\hat{c}_1$.

First of all, one might wonder whether there is a possibility of computing a C-optimal family of slicing positions "directly", e.g. by some kind of steepest descent approach. While such greedy strategies proved to be very useful to obtain near-optimal estimates $\widehat{C}$, we do not see a way to generally avoid local minima. So, all methods described in the following explore the search space more or less exhaustively.

The simplest way of calculating a C-optimal family of slicing positions $\langle c_2, \ldots, c_k \rangle$ with respect to a given $\hat{c}_1$ is similar to the implementation of *C-opt* presented in Section 4.4 (Figures 4.7 and 4.8). The exhaustive search can easily be adapted to the situation where an upper bound $\widehat{C}$ for C_{opt} is given: First, the bounds l_p and u_p are computed. Then, within the relevant part of the search space, the minimum C_{opt} and corresponding slicing positions $\langle c_2, \ldots, c_k \rangle$ are computed by enumeration of all possible combinations of the indices c_i, $2 \leq i \leq k$. Again, the variation of later indices can be skipped if a partial sum is larger than the minimum found so far. As described in Section 5.1, this leads to an algorithm with computational time complexity $\mathcal{O}(r^{k-1})$ for the search phase which is often much smaller than $\mathcal{O}(n^{k-1})$ but is still exponential in the number of sequences k. Yet, for up to twelve related sequences of average length 250, fairly moderate running times were achieved (below forty seconds). To speed up the procedure for more sequences, we have developed methods which allow further reduction of the search space and which we describe in the following sections.

5.4.1 Monotony Bounds

Motivated by the regular shape of the matrices $C_{\mathbf{s}_p,\mathbf{s}_q}$, we developed a procedure which utilizes the observation that the entries in a column of an additional-cost matrix from top to bottom start with rather high values, decrease almost monotonically for a long time, then reach a minimum (where the paths of the optimal alignments with value zero cross the column), and then increase almost monotonically to a high value again.

Definition 5.4 (Monotony Bounds)
For each column $Col^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_q}$, $q \in \{2, \ldots, k\}$, we define the lower and the upper *monotony bound* $Lcol^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_q}$ and $Ucol^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_q}$ by the formulae:

$$\begin{aligned} Lcol^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_q} &:= \min_{l_q < j < u_q} \{j | Col^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_q}[j] > Col^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_q}[j-1]; u_q\}, \\ Ucol^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_q} &:= \max_{l_q < j < u_q} \{j | Col^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_q}[j] > Col^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_q}[j+1]; l_q\}. \end{aligned}$$

In other words, $j_L := Lcol^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_q}$ is the largest $j \in \{l_q + 1, \ldots, u_q\}$ such that the sequence $Col^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_q}[j']$ $(l_q \leq j' < j)$ is monotonously decreasing and $j_U := Ucol^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_q}$ is the smallest $j \in \{l_q, \ldots, u_q - 1\}$ such that the sequence $Col^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_q}[j']$ $(j < j' \leq u_q)$ is monotonously increasing.

Similarly, for each column $Col^{i}_{\mathbf{s}_p,\mathbf{s}_q}$ in the relevant part $l_p \leq i \leq u_p$ of the additional-cost matrix $C_{\mathbf{s}_p,\mathbf{s}_q}$, $2 \leq p < q \leq k$, we define lower and upper *monotony bounds* $L_{\mathbf{s}_p,\mathbf{s}_q}[i]$ and $U_{\mathbf{s}_p,\mathbf{s}_q}[i]$, given by the formulae:

$$\begin{aligned} L_{\mathbf{s}_p,\mathbf{s}_q}[i] &:= \min_{l_q < j < u_q} \{j | Col^{i}_{\mathbf{s}_p,\mathbf{s}_q}[j] > Col^{i}_{\mathbf{s}_p,\mathbf{s}_q}[j-1]; u_q\}, \\ U_{\mathbf{s}_p,\mathbf{s}_q}[i] &:= \max_{l_q < j < u_q} \{j | Col^{i}_{\mathbf{s}_p,\mathbf{s}_q}[j] > Col^{i}_{\mathbf{s}_p,\mathbf{s}_q}[j+1]; l_q\}. \end{aligned}$$

□

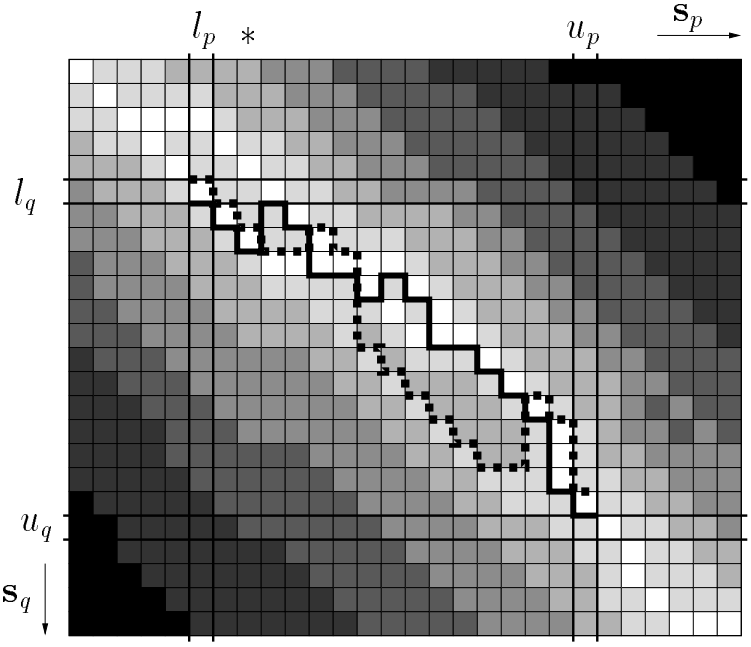

Figure 5.7: Additional-cost matrix $C_{\mathbf{s}_p,\mathbf{s}_q}$ and its relevant part with monotony bounds $L_{\mathbf{s}_p,\mathbf{s}_q}$ (solid line) and $U_{\mathbf{s}_p,\mathbf{s}_q}$ (dotted line). In the column indicated with an asterisk, the lower monotony bound does not correspond to the monotonicity of the complete column but only to that of the relevant part.

Both $L_{\mathbf{s}_p,\mathbf{s}_q}$, indicated by the solid line, and $U_{\mathbf{s}_p,\mathbf{s}_q}$, indicated by the dotted line, are pictured in Figure 5.7, superimposed on the additional-cost matrix of Figure 4.4. (The lower and upper bounds l_p, l_q and u_p, u_q, respectively, are chosen arbitrarily.)

Note that the monotony bounds are defined only for the relevant part of the additional-cost matrix. So they do not necessarily indicate the monotonicity of the complete column (see e.g. the column indicated with an asterisk in Figure 5.7).

Pre-computed monotony bounds allow in many cases to omit fairly large regions of the relevant search space: Consider the case of determining slicing position c_q of sequence $\mathbf{s}_q$, $2 \leq q \leq k$, where positions $\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_{q-1}$ have been already fixed. If, for an index $\hat{c}_q$ strictly less than $Lcol^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_q}$, the sum

$$C(\langle \hat{c}_1, \ldots, \hat{c}_{q-1} \rangle) \; + \; \alpha_{1,q} \; Col^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_q}[\hat{c}_q]$$

is larger than $\widehat{C}$ or if for any p, $2 \leq p < q$, with $\hat{c}_q$ strictly less than $L_{\mathbf{s}_p,\mathbf{s}_q}[\hat{c}_p]$, the sum

$$C(\langle \hat{c}_1, \ldots, \hat{c}_{q-1} \rangle) \; + \; \alpha_{p,q} \; C_{\mathbf{s}_p,\mathbf{s}_q}[\hat{c}_p, \hat{c}_q]$$

is larger than $\widehat{C}$, then all of the remaining entries in the column above $\hat{c}_q$ lead to a sum larger than $\widehat{C}$ as well, and thus the iteration over the indices $c_q < \hat{c}_q$ can be skipped. Figure 5.8 shows the latter case for $q = 4$ and $p = 2$.

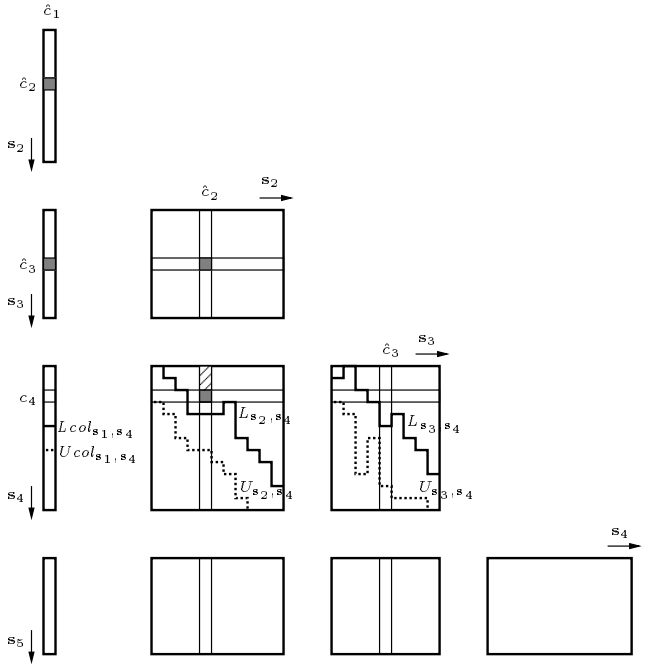

Figure 5.8: How to utilize monotony bounds: If the sum of the values in the shaded boxes is larger than the upper bound $\widehat{C}$, then all smaller indices for c_4 (the hatched region) can be skipped for this setting of $\langle \hat{c}_2, \hat{c}_3 \rangle$.

Of course, a similar statement holds for $Ucol$ and U, respectively.

With this technique, we were able to save up to a quarter of the computation time required for the search of slicing positions for $k = 14$ sequences. For that many sequences, the relative speed-up is – to our own surprise – even larger than for less sequences. For exact running time improvements, see Section 7.4.

5.4.2 Preprocessing κ-Subfamilies

A different approach to deal with, say, more than ten sequences, is a procedure which utilizes further preprocessing. When considering slicing positions c_p, $2 \leq p \leq k$, the multiple additional cost of some $\kappa < k$ sequences not including sequence $\mathbf{s}_p$ can be used as an additional estimate.

For example, consider the case of $k = 5$ sequences, and let $\kappa = 3$. Assume, for fixed $\hat{c}_1$, that we have chosen $\hat{c}_2$ as slicing position of $\mathbf{s}_2$. We still have to determine appropriate values for c_3, c_4, and c_5. In this case, we can make use of the minimal additional cost contributed, for instance, by the subfamily $\langle \mathbf{s}_1, \mathbf{s}_4, \mathbf{s}_5 \rangle$: We pre-compute

$$C_{1,4,5} := \min_{\substack{c_4 \in \{l_4, \ldots, u_4\} \\ c_5 \in \{l_5, \ldots, u_5\}}} \{C(\langle \hat{c}_1, c_4, c_5 \rangle)\}.$$

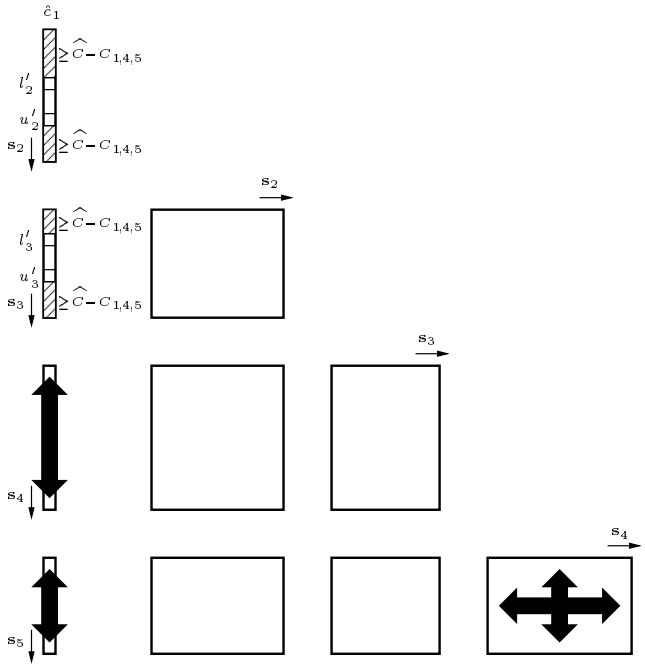

Figure 5.9: Computing an additional estimate $C_{1,4,5}$ and the corresponding improved lower and upper bounds (l'_2, l'_3 and u'_2, u'_3, respectively).

Then, an index $\hat{c}_3$ with $C_{1,4,5} + \alpha_{1,3}\ Col^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_3}[\hat{c}_3] \geq \widehat{C}$ or $C_{1,4,5} + \alpha_{2,3}\ C_{\mathbf{s}_2,\mathbf{s}_3}[\hat{c}_2, \hat{c}_3] \geq \widehat{C}$ cannot lead to a smaller sum $C(\langle \hat{c}_1, \hat{c}_2, \hat{c}_3, c_4, c_5 \rangle)$ for any $(c_4, c_5) \in \{0, \ldots, |\mathbf{s}_4|\} \times \{0, \ldots, |\mathbf{s}_5|\}$. Thus, no values c_4 and c_5 have to be evaluated for such a $\hat{c}_3$.

Of course, a pre-computed $C_{1,4,5}$ can also be used to adapt the relevant parts of the columns $Col^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_2}$ and $Col^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_3}$: Let $q \in \{2, 3\}$. We re-define l_q and u_q, denoted by l'_q and u'_q, respectively, such that $C_{1,4,5} + \alpha_{1,q}\ Col^{\hat{c}_1}_{\mathbf{s}_1,\mathbf{s}_q}[j] \geq \widehat{C}$ for all $j < l'_q$ and for all $j > u'_q$ (see Figure 5.9).

This process can be generalized in the obvious way to the case where $3 \leq \kappa < k$: A combination of κ of the k sequences is selected, using a map

$$\pi : \{1, 2, \ldots, \kappa\} \to \{1, 2, \ldots, k\} \quad \text{with} \quad \pi(i) < \pi(j) \text{ for } i < j,$$

and the *additional estimate*

$$C_{\pi(1),\ldots,\pi(\kappa)} \quad := \quad \min\left\{C(\langle c_{\pi(1)}, \ldots, c_{\pi(\kappa)} \rangle) | 0 \leq c_i \leq |\mathbf{s}_i| \text{ for all } i \in \{\pi(1), \ldots, \pi(\kappa)\}\right\}$$

is computed. For larger κ, also a recursive call with a smaller $\kappa' < \kappa$ is possible.

Note that, compared to all previous heuristics based on that upper bound $\widehat{C}$, an important step was taken here: Rather than a single value, a sum of $\binom{\kappa}{2}$ additional-cost values is compared with $\widehat{C}$. This way, the dependence of the relative speed-up on the number of sequences can be reduced. For more than twelve sequences

where the "exhaustive" methods alone perform worse and worse, we obtained much faster procedures. Of course, the performance also largely depends on the additional estimate, i.e. which κ-subfamily of the k sequences is selected. A detailed discussion of this dependence is given in Section 6.1.2. Section 7.4 shows the corresponding running times of the different variations.

Note also the relatedness of this approach to the function *siMinC* of computing an upper bound $\widehat{C}$ by simultaneous variation of multiple indices (see Section 5.3.3). The simultaneous minimization of s indices is nothing but the computation of a κ-subfamily with $\kappa = s$. But while *siMinC* uses the exact additional cost of the subfamily of sequences merely to improve the upper bound $\widehat{C}$, here we use it more directly to prune the search space utilizing its origin as a sum of $\binom{\kappa}{2}$ additional-cost values.

Since the technique of preprocessing κ-subfamilies and the monotony bounds are based on rather independent principles, a combination of both approaches is possible. Despite the large preprocessing overhead, for twelve and more sequences the combined version yields the shortest running times (see the measurements in Section 7.4). The resulting alignment algorithm allows to align simultaneously up to fourteen closely related proteins within less than a minute. However, for feebly related sequences, also with the improvements described here, fairly long running times can – and often will – occur when more than ten sequences are under consideration.

5.5 Approximate Slicing Positions

As noted above, for too many sequences the calculation of a C-optimal family of slicing positions may take rather a long time if the sequences are not very similar. In these cases, it is possible to use a slightly less accurate method for computing the slicing positions. The minimization over all positions $c_2, \ldots, c_k$ can be skipped and a family of *approximate slicing positions* $\langle \hat{c}_1, \hat{c}_2, \ldots, \hat{c}_k \rangle$ which gives rise to the upper bound $\widehat{C}$ can be used to cut up the sequences. This allows to obtain reasonably good alignments in time $\mathcal{O}(k^2n^2 + 2^k n L^{k-1})$, making our procedure for small L comparable to iterative alignment procedures, the advantage being that even these approximate algorithms align the sequences simultaneously.

Note that the different possibilities of computing an estimate $\widehat{C}$ might result in different approximate slicing positions and thus in different alignments. By our general heuristic, those methods leading to good, i.e. small, upper bounds $\widehat{C}$ are likely to produce alignments near to the optimal one. A discussion of the quality of alignments computed from approximate slicing positions is given in Section 7.2.4.

By giving up the heuristic of (proper) C-optimal families of slicing positions, of course, the alignment may even improve – especially when measured in biological terms. So, it might be possible to define other objectives which may take biological properties of the sequences into account and/or are faster to compute. However, such criteria still have to be developed.

Chapter 6

The Program DCA

6.1 Some Implementational Aspects

The program DCA is an implementation of the divide-and-conquer multiple sequence alignment algorithm in the programming language C [115]. The largest part of the program is a direct implementation of the functional specifications given in the previous chapters. In the following, we concentrate our discussion on the points where non-trivial solutions were necessary or inaccuracies were unavoidable.

6.1.1 Incorporation of MSA

One of the main difficulties arising with the implementation of DCA was the incorporation of a fast algorithm for the optimal SP alignment of the families of remaining short (sub)sequences. A first, straightforward self-implementation of the standard k-way dynamic programming algorithm was a tolerable solution for up to six sequences when the recursion stop size L was set to small values. However, compared to sophisticated speed-up techniques such as those implemented in the program MSA (see Section 3.4, [39, 128, 90]), this was a fairly slow solution. Another reason for a combination of DCA with MSA was its feature to accept partial alignments of the input sequences as fixed pieces of the alignment to be computed, much like anchor points described in the context of the fragment-based heuristic alignment methods (see Section 3.5). Since a family of slicing positions can be seen as an anchor of length zero, a single MSA-call with the additional information about all the slicing positions would be an elegant solution of the entire problem.

Unfortunately, the current version 2.1 of MSA accepts anchors only if they have length ≥ 1.[1] So it is necessary to call MSA once for each family of (sub)sequences. But the alignment of the subsequence families independently of each other raises difficult questions concerning the free shift and the handling of affine gaps at slicing positions.

[1] Upon our request, the authors of MSA are working on an extension of their program to allow also anchors of length zero [170].

While, in principle, MSA allows to turn the free shift option on and off, the left and the right termini are always treated identically. This, of course, makes sense when aligning uncut sequences. However, for our purpose we require a left-only free shift for the leftmost and a right-only free shift for the rightmost subsequence family. Fortunately, this problem can be circumvented by a simple trick (here explained for the leftmost family): At the right end of each sequence, we add an additional letter, say A. These letters are forced to build an anchor of length one. Then, MSA is called with the free shift option switched on, thus scoring left gaps like terminal gaps while gaps endig immediately before the fixed last column of As are scored like internal gaps. Finally, the As are removed from the obtained alignment.

For the other problem, the scoring of gaps running through slicing positions, we could not find an accurate solution. Consider two adjacent families of subsequences. Terminal gaps in the left-hand alignment and initial gaps in corresponding rows of the right-hand alignment unify to a single larger gap upon concatenation of the alignments. Hence, the alignments cannot be computed and scored independently. An option which allows – based on the endings in the left-hand alignment – to switch between the initiation of a new gap and the elongation of an existing gap individually for each sequence when computing the right-hand alignment might be an acceptable solution. But none of the efficient optimal alignment procedures known to the author allows this flexibility. So we had to accept an inaccuracy: We align the families of (sub)sequences independently of each other, charging all gaps (except initial gaps of the leftmost and terminal gaps of the rightmost family) like internal gaps.

Of course, when re-aligning parts of the sequences as described in Section 4.5.2, the same problems arise at the window bounds, and we evaded them in the same inexact way.

Note that this inaccuracy does not influence the quality of alignments too much when one is mainly interested in the detection of corresponding regions of the sequences. The overall structure of the alignment is dominated by the relative shift of the sequences against each other due to the location of slicing positions (which are computed correctly). In particular for biological sequences, minor effects of gaps in the proximity of slicing positions become quite insignificant since the structurally important regions which are of highest interest are mostly gap-free. However, in terms of SP score or when aligning for phylogeny (where each single match or mismatch is counted), the effect might become perceptible.

Of course, the problems discussed above do not arise when dealing with a strongly additive alignment score. So, for homogeneous gap costs, the alignments of the (sub)sequences approximate the correct objective function.

However, also with homogeneous gap costs it is not completely certain that one always obtains the expected results since – as noted at the end of Section 3.4 – MSA sometimes only finds a suboptimal alignment. Even more problematic is that the current version of MSA from time to time, mainly when called with more than five or six sequences, terminates abnormally due to errors in the program and does not return any alignment. In such cases, DCA inserts a heuristic multiple alignment which

is also computed by MSA and whose score is normally used as an upper bound for reducing the search space (as explained in Section 3.4). Of course, this results in a much worse than optimal (or near-optimal) multiple alignment.

Another disadvantage of the current version of MSA is the unability to handle arbitrary sequence weights. It is only possible to choose between the *unweighted* SP score and sequence weights which are computed internally by MSA based on the *neighbor joining* method for phylogenetic tree reconstruction [165]. So, in the current implementation of DCA, our sequence weights (see Section 2.5) are only used for the computation of the slicing positions while the optimal alignments are computed with respect to the unweighted SP score.

6.1.2 Preprocessing κ-Subfamilies

Our first attempts of implementing the approach of preprocessing the minimal additional cost of a subfamily of κ of the k sequences (see Section 5.4.2) has not led to the speed-up we expected: We computed the value $C_{\pi(1),\ldots,\pi(\kappa)}$ for *all* $\binom{k}{\kappa}$ combinations π. While this took comparatively a long time, in most cases the obtained values were rather small, often even zero such that they did not lead to any reduction of the search space.

A first reason was quickly detected: If sequence $\mathbf{s}_1$ is not among the sequences of the subfamily from which the additional estimate is computed, then the value $C_{\pi(1),\ldots,\pi(\kappa)}$ is always zero due to the trivial cuts[2] $\langle 0,\ldots,0\rangle$ and $\langle|\mathbf{s}_{\pi(1)}|,\ldots,|\mathbf{s}_{\pi(\kappa)}|\rangle$. Thus, to obtain useful additional estimates, $\pi(1)$ has to be set to 1.

More difficult was the selection of the remaining $\kappa-1$ sequences. Here, the following heuristic proved to be successful: The least compatible sequences are expected to result in the highest additional cost. The compatibility of sequences can be estimated by the pairwise additional costs they contribute to the upper bound $\widehat{C}$. So, we select those $\kappa-1$ sequences which are involved in the largest projections $C_{\mathbf{s}_p,\mathbf{s}_q}[\hat{c}_p,\hat{c}_q]$ of $\widehat{C}$. This way, we obtain comparatively large values $C_{\pi(1),\ldots,\pi(\kappa)}$, and the time of the pre-processing becomes moderate when we consider only one subfamily of κ sequences.

However, the running time of our first implementation of this technique was still larger than the time required for the exhaustive search. The indirect indexing when computing the value $C_{\pi(1),\ldots,\pi(\kappa)}$ (see Figure 6.1) and the more complex control structure during the later search for C_{opt} made up more than the time saved by the decrease of search space.

Then, we implemented the algorithm in a less elegant way which indeed runs faster than the exhaustive version: Before computing the additional estimate, the corresponding sequences are copied (in fact by switching some pointers) such that

[2]This is only true if the relevant parts of the matrices are maximally large. Otherwise, the slicing positions given here are not within the relevant parts of the additional-cost matrices, and the values $C_{\pi(1),\ldots,\pi(\kappa)}$ are not necessarily zero but nevertheless very low.

2.1 $C_{min} := +\infty$
2.2 for $c_{\pi(2)} \in \{0, \ldots, |\mathbf{s}_{\pi(2)}|\}$ do
for $c_{\pi(3)} \in \{0, \ldots, |\mathbf{s}_{\pi(3)}|\}$ do
$\ddots$
for $c_{\pi(\kappa)} \in \{0, \ldots, |\mathbf{s}_{\pi(\kappa)}|\}$ do
if $C(\langle \hat{c}_1, c_{\pi(2)}, c_{\pi(3)}, \ldots, c_{\pi(\kappa)} \rangle) < C_{min}$ then
$C_{min} := C(\langle \hat{c}_1, c_{\pi(2)}, c_{\pi(3)}, \ldots, c_{\pi(\kappa)} \rangle)$
2.3 return C_{min}

Figure 6.1: Implementation of the search for an additional estimate $C_{\pi(1),\ldots,\pi(\kappa)}$ with indirect addressing of sequence numbers. We assume that $\pi(1) = 1$. For simplicity we show the naive version corresponding to that shown in Figure 4.7.

they receive new (temporary) sequence numbers $1, \ldots, \kappa$. Thus, no indirect addressing is necessary, and the original exhaustive procedure can be used to compute the values $C_{\pi(1),\ldots,\pi(\kappa)}$.

6.2 Parameters of DCA

In general terms, DCA takes as input a family of sequences and a matrix of pairwise letter distances with an affine gap function and produces a multiple alignment of the sequences. The input formats of both the sequences and the cost function are the same as those of MSA (see [171]). Several other parameters can be chosen for DCA:

- The recursion stop size can be set to any number $L \geq 1$. Of course, too large an L (e.g. $L > 100$) can result in very long running times and very large memory usage due to the resulting MSA-runs. On the other hand, too small an L (e.g. $L < 5$) can result in empty subsequences which may lead to bad alignments. We obtained a value of $L = 40$ as a good compromise between computation time and alignment quality (see Section 7.2).

- The window size can be set to any value $W \geq 0$. Note that for $W > \frac{L}{2}$, the windows laying across adjacent slicing positions may overlap which is not in accordance with the original intention of this approach.

- The weight intensity λ can be set to any value between $\lambda = 1$ (maximum weighting) and $\lambda = 0$ (no weighting). However, by reasons mentioned above, the weights are only used when computing the slicing positions while the alignments of the (sub)sequences are optimized with regard to the unweighted SP score.

- The free shift of the sequences can be turned on and off.

- The search phase for a C-optimal cut can be deactivated such that the sequences are cut at the approximate slicing positions which give rise to the estimates $\widehat{C}$.

Finally, DCA can also be run in an interactive mode. Here, a series of alignments of the same sequence family can be computed with one or more parameters altered while the others remain unchanged. This provides an environment for a comfortable exploration of the parameter space of multiple sequence alignment.

We implemented an html-based user interface for the use of DCA via the World-Wide Web. An installation has been set up [191].

Chapter 7

Evaluation of the Algorithm

In this chapter we evaluate the performance of the divide-and-conquer alignment procedure. Because there is no "benchmark problem" for multiple sequence alignment containing comparable problems differing only in the number, length, and/or relatedness of the sequences and because the behavior of alignment methods depends very much on the sequence family under consideration, we decided to test our algorithm on families of random sequences.

In the first part, we show how we create sequence families by a tree-guided mutation process simulating evolution. Proceeding this way, we are aware of the true evolutionary history of the sequences which is an advantage when evaluating methods for the reconstruction of phylogenetic relationships. Then, we use these families to assess the quality of the alignments produced by `DCA`; we compare the different running time optimizations; and we validate the theoretical results about space and time requirements of our procedure.

Alignments of biological sequences are presented in Chapter 8.

7.1 Families of Random Sequences

It is our aim to create sequence families most similar to "real world" biological data. Of course, a family consisting of k unrelated random sequences does not resemble such a typical problem instance since species are part of a hierarchically structured phylogeny, and thus cannot be regarded as if drawn independently from the same distribution [65].

We simulate an evolutionary process by iterated mutation of a *common ancestor* sequence following the edges of a pre-given rooted *mutation guide tree*. Beneath the relatedness of the sequences, the major advantage of this approach is the knowledge about the history of the sequences. So it is – in contrast to biological applications – easily possible to verify predictions about the phylogeny of the sequences simply by comparing the predicted phylogenetic tree to the tree that was used in the creation process of the sequences.

In the literature, some methods of imitating the evolution of (mostly DNA-like)

A	R	N	D	C	Q	E	G	H	I
0.087	0.041	0.040	0.047	0.033	0.038	0.050	0.089	0.034	0.037

L	K	M	F	P	S	T	W	Y	V
0.085	0.081	0.015	0.040	0.051	0.070	0.058	0.010	0.030	0.065

Table 7.1: Normalized frequencies of the amino acids in the random sequence families.

random sequences have been suggested (e.g. [112, 94, 176]). But in none of the methods known to the author, the length of the sequences is altered by insertions and/or deletions of subsequences making these approaches unsuitable for the creation of alignment test families. So we had to develop a new procedure which serves our purposes.

We describe the creation of a family of k sequences of average length n. The letters of our sequences are independently chosen from a fixed alphabet $\mathcal{A}$ with fixed but different probabilities for different letters. We use $\mathcal{A} = \{$A, R, N, D, C, Q, E, G, H, I, L, K, M, F, P, S, T, W, Y, V$\}$ denoting the set of twenty amino acids.

Initially, we randomly choose a sequence of length n using the distribution of amino acids calculated by Dayhoff *et al.* [50] (see Table 7.1). This sequence is the starting point for a mutation process guided by the branching order of the mutation guide tree which we require to contain at least k nodes. The root of the tree is identified with the initial sequence. Then, at the nodes connected with the root, "child sequences" are created by randomly substituting, inserting, and deleting letters. The exact way mutations are performed in our model is described below. The process of assigning mutated sequences to nodes adjacent to those which are already provided with a sequence is carried on until the leaves of the guide tree are reached and thus each node of the tree is provided with a sequence which is more or less related to the common ancestor. If the mutation guide tree contains more than k nodes, from the sequences created this way, k sequences are selected randomly, forming the final sequence family.

The creation of a child sequence from its immediate ancestor is performed as follows. We assume a constant mutation rate over the whole length of the sequence. At each position of the ancestor sequence, amino acid substitutions are performed according to the mutation probability matrix of one *accepted amino acid substitution per hundred sites* (1 PAM) given in [50]. (In fact, most sequence positions remain unchanged as in 99 percent of the cases an amino acid is substituted by itself.) Additionally, with fixed probabilies ι and δ, we allow at most ten consecutive amino acids to be inserted or deleted at a randomly chosen position of the sequence. (Insertions maintain the overall distribution of amino acids.) We denote the new unit of measure for the distance of a child sequence created this way from its ancestor including insertions and deletions by 1 PAM*. Note that, if we set the insertion probability ι to the same value as the deletion probability δ, the expected average length of the sequences remains n, the length of the common ancestor sequence.

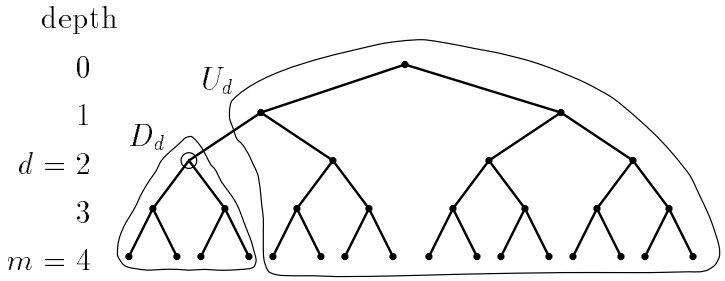

Figure 7.1: Uniform binary tree of depth $m = 4$ with $M = 2^5 - 1 = 31$ nodes. For a node in depth $d = 2$ (marked by the circle), those nodes contributing to D_d and U_d, respectively, are shown.

Of course, this model is only an approximation to biological reality. The mutation rate of genomic sequences found in nature is not constant for all positions in the genome of a single species nor seems it to be the same for different species: Mutations in genomic regions with strong functional and/or structural importance are less often observed than elsewhere [87, 88, 178, 24], and higher reproduction rates tend to result in higher mutation rates. Nevertheless, constant mutation rates are a commonly used approximation [112, 63, 116, 94, 24, 176] and suffice for the purpose of this model. Also, our model does not reflect "evolutionary pressure" since all mutations are reversible.[1] More sophisticated methods of creating random sequence families taking these and other features into consideration have been suggested in the literature [61, 43, 175, 14].

To be able to create sequence families of a pre-given average pairwise distance, it is necessary to know, for a given mutation guide tree, the average sequence distance d_{av} (in units of PAM*), i.e. the expected length of a shortest path between two randomly chosen nodes in the tree where each edge has length 1 PAM*. As it allows a fairly simple theoretical analysis, we consider a binary uniform tree of depth m with $M = 2^{m+1} - 1$ nodes (see Figure 7.1).

We obtain d_{av} by computing the sum of all pairwise distances in the tree divided by M^2, the number of pairs of nodes: Consider a node in level d of the uniform binary tree, $0 \leq d \leq m$. In the example of Figure 7.1, we have chosen $d = 2$. The corresponding node is indicated by a circle. In each level i, $0 \leq i \leq m - d$, of the subtree "below" the observed node, there are 2^i nodes with distance i. The sum of distances to all these nodes is

$$D_d \quad := \quad \sum_{i=0}^{m-d} (2^i \cdot i) \quad = \quad 2^{m-d+1}(m - d - 1) + 2.$$

Additionally, there are d nodes "above" the observed node, each being starting point

[1] In fact, because of the reversibility, there is no need of locating a root of the mutation guide tree: Any node can be used as the starting point to which the common ancestor sequence is attached.

of a subtree. Summing the distances to all these nodes gives

$$\begin{aligned} U_d &:= \sum_{i=1}^{d} \left(i + \sum_{j=1}^{m-(d-i)} 2^{j-1}(i+j) \right) \\ &= 2^{m-d+1}(d-m+3) + 2^{m+1}(d+m-3) + d. \end{aligned}$$

Hence, the total sum of distances from a node in level d to all other nodes is

$$\begin{aligned} N_d &:= D_d + U_d \\ &= 2^{m-d+2} + 2^{m+1}(d+m-3) + d + 2. \end{aligned}$$

Averaging this value over all pairs of nodes, we obtain

$$\begin{aligned} d_{av} &= \frac{\sum_{d=0}^{m}(2^d \cdot N_d)}{M^2} \\ &= 2 \cdot 2^{m+1} \, \frac{4 + m(1+2^{m+1}) - 2^{m+2}}{(2^{m+1}-1)^2} \end{aligned}$$

which approximates

$$2 \, \frac{2^{m+1}m - 2^{m+2}}{2^{m+1}} = 2\,(m-2)$$

for m sufficiently large.

In conclusion, to obtain sequences of a pre-given relatedness, we simply have to alter the depth m of the tree:

$$m \approx \frac{d_{av}}{2} + 2.$$

Unfortunately, for large distances d_{av}, the number of sequences $M = 2^{m+1}-1$ involved in our creation process increases enormously (e.g. to $M = 2^{128} \approx 10^{38}$ for $d_{av} = 250$ PAM*). For computation time and memory size reasons, it is not possible to create such a large number of sequences. So we decided to use a higher mutability rate, say R PAM* each time creating a new sequence from its immediate ancestor. This leads to an average distance of $d_{av} \approx 2(m-2)R$ PAM* between two of the resulting sequences.

To be able to create sequences of some thousand letters in length, we use a mutation guide tree of depth $m = 9$, creating $M = 2^{10} - 1 = 1023$ sequences. Choosing $R = 18 \approx \frac{250}{2(9-2)}$, we obtain an average pairwise distance $d_{av} \approx 250$ PAM*. The probability for insertions and deletions is set to $\iota = \delta = 0.3$ percent. With this setting, the pairwise identity between two sequences (i.e. the number of identically paired letters in an optimal pairwise alignment due to unit edit distance divided by the total number of letters in the two sequences) lies between twenty and thirty percent. Except stated otherwise, all measurements described in this chapter are obtained on sequence families generated with these parameters.

(a)
```
FSAEAALVSPGKGDDEQVPNKDKCVYHGHKDGKRMNVKTPPTGPLVVGVHQ
YEGANEVGATCEESSYCYVKEQAIQVKESQECTDFARHEVKSFRGVPGKLTEVIPVPL
YGAAHPVGDPIKLGSLFLNHYESKGHTAAMCLLGMKTELIEPIEVQA
SGVTEPVPNPVPATGIKLDKYTREENCLGMCLMGMGPPMVTIGEVGI
```

(b)
```
FSAEAALVSP--------GKGDDEQVPNKDKCVYHGHKDGKRMNVKTPPTGPLVVGVHQ
YEGANEVGATCEESSYCYVKEQAIQVKESQECTDFARHEVKSFRGVPGKLTEV-IPVPL
YGAAHPVGDP--------IKLGSLFLNH---YESKGHTAAMCLLGMKTELIEP-IEVQA
SGVTEPVPNP--------VPATGIKLDK---YTREENCLGMCLMGMGPPMVTI-GEVGI
```

(c)
```
FSAEAALVSP--------GKGDDEQVPNKDKCVYHGHKDGKRMNVKTPPTGPLVVGVHQ
YEGANEVGATCEESSYCYVKEQAIQVKESQECTDFARHEVKSFRGVPGKLTE-VIPVPL
YGAAHPVGDP--------IKLGSLFL---NHYESKGHTAAMCLLGMKTELIE-PIEVQA
SGVTEPVPNP--------VPATGIKL---DKYTREENCLGMCLMGMGPPMVT-IGEVGI
```

Figure 7.2: (a) Sample family of random sequences obtained with the procedure described in the text for $n = 50$ and $k = 4$; (b) "true" alignment of these sequences; (c) optimal alignment according to PAM 250 substitution matrix and gap function $g(l) = 8 + 12\,l$.

Note that due to the assumptions we made, aligning biological sequences – even if they have the same pairwise identity – might require less or more time than aligning our random sequences. Yet, the aim of this chapter is the validation of the asymptotic behavior of the divide-and-conquer procedure as derived in the previous chapters. Our random sequence families seem to be suitable for this purpose.

In Figure 7.2 (a), a sample family with $k = 4$ sequences of average length $n = 50$ is shown. The alignment given in Figure 7.2 (b) is the "true" alignment corresponding to the creation process of the sequences. Figure 7.2 (c) shows the "optimal" alignment according to the PAM 250 substitution matrix [50] (in distance form with values between 0 and 24) and gap function $g(l) = 8 + 12\,l$. While the overall optimal alignment is correct, the exact location of the gaps does not coincide in all cases. The scores for both alignments show these differences as well: The "true" alignment has an alignment score of 5184, while the "optimal" alignment has a score 5166. This shows that – as is well-known – an optimal alignment is not necessarily the correct one, i.e. the parsimony principle does not always hold strictly.

Figure 7.3 shows the corresponding relatedness tree of these sequences.

7.2 Quality of the Alignments

In this section, we assess the quality of the alignments computed with `DCA` depending on several parameters. All alignments are optimized with respect to the substitution matrix of 250 PAM as described above. Yet, here we use a homogeneous gap function $g(l) = 15\,l$ since otherwise the results are distorted by the incorrect scoring of affine gaps at slicing positions. We also set all sequence weight factors $\alpha_{p,q}$ to 1. As the

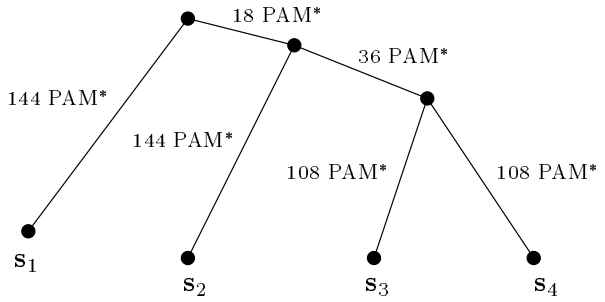

Figure 7.3: Relatedness tree for the sequences shown in Figure 7.2.

standard of truth, we use the optimal SP score computed with `MSA` (or with `DCA` when L is set to a value larger than the length of the longest sequence under consideration).

We have run the tests on a Sun SparcStation 10/41 running Solaris 2.4 with 32 megabytes of RAM. The programs were compiled with the `gcc` C compiler version 2.7.0 with the `-O` flag for optimization. To measure time, we have used the `times` command from the `<sys/times>` C library. The space usage was measured with the `set time` environment of `csh`.

As our basic test set we have computed for each $k \in \{3, 4, \ldots, 14\}$ one hundred families of random sequences with average expected length $n = 250$. We present the results of our measurements mostly in the following form: First, we give for alignments of some individual sequence families the exact scores and other values of interest and discuss distinctive features. These families are randomly selected and thus do not always show the the typical behavior. Then, we present the corresponding results averaged over the whole sets of one hundred sequence families for each $k \in \{3, 4, 5, 6\}$. For larger k, the optimal alignment score averaged over all one hundred sequence families was unobtainable due to several incomplete runs of `MSA`, making an assessmant of the alignment quality impossible.

7.2.1 Dependence on the Recursion Stop Size

We begin with an assessment of the alignment score and computation time depending on the recursion stop size L. Table 7.2 shows the absolute alignment scores (first row of each box), the relative score error as percentage of the optimal score (second row), and the corresponding running times (third row) for sequence families with $k = 3, \ldots, 14$ sequences. While for $k = 3, 4, 5, 6$ the sequence family is chosen randomly from our one hundred test families, the families with $7 - 9$ sequences were selected more carefully due to the problems of `MSA`. However, also for the family with $k = 5$ sequences, in two cases `MSA` could not compute optimal alignments of all subsequence families. This is marked by an asterisk at the score value. For $k > 9$, we could not compute the optimal alignment score with `MSA` (indicated by the question marks in Table 7.2). Here, only the score and running time of the `DCA`-runs with smaller L are shown.

	$L = 10$	$L = 20$	$L = 30$	$L = 40$	$L = 50$	$L = 100$	$L = 200$	$L = 300$
$k = 3$	12 018 .033 % 2.79 sec	12 018 .033 % 1.85 sec	12 018 .033 % 1.72 sec	12 018 .033 % 1.32 sec	12 018 .033 % 1.21 sec	12 014 .000 % 1.28 sec	12 014 .000 % 1.95 sec	12 014 .000 % 2.55 sec
$k = 4$	23 961 .058 % 3.47 sec	23 950 .013 % 2.65 sec	23 950 .013 % 2.64 sec	23 947 .000 % 2.23 sec	23 947 .000 % 2.27 sec	23 947 .000 % 2.44 sec	23 947 .000 % 3.14 sec	23 947 .000 % 4.65 sec
$k = 5$	39 015 .018 % 4.31 sec	39 015 .018 % 3.66 sec	39 015 .018 % 3.64 sec	39 008 .000 % 3.47 sec	39 008 .000 % 3.56 sec	*39 020 .031 % 3.97 sec	*39 020 .031 % 5.36 sec	39 008 .000 % 10.09 sec
$k = 6$	60 184 .088 % 7.01 sec	60 176 .075 % 6.16 sec	60 176 .075 % 5.92 sec	60 171 .067 % 6.10 sec	60 171 .067 % 5.97 sec	60 141 .017 % 6.55 sec	60 131 .000 % 9.11 sec	60 131 .000 % 15.98 sec
$k = 7$	83 653 .010 % 8.03 sec	83 653 .010 % 7.69 sec	83 653 .010 % 7.29 sec	83 652 .008 % 7.57 sec	83 652 .008 % 7.62 sec	83 652 .008 % 8.73 sec	83 652 .008 % 11.32 sec	83 645 .000 % 26.39 sec
$k = 8$	112 411 .060 % 10.73 sec	112 407 .056 % 10.20 sec	112 407 .056 % 10.19 sec	112 370 .023 % 10.68 sec	112 370 .023 % 10.74 sec	112 370 .023 % 12.79 sec	112 344 .000 % 23.14 sec	112 344 .000 % 70.30 sec
$k = 9$	147 150 .176 % 15.94 sec	147 086 .132 % 15.56 sec	147 086 .132 % 15.18 sec	146 992 .068 % 16.24 sec	146 992 .068 % 16.22 sec	*147 079 .127 % 19.56 sec	146 942 .034 % 30.31 sec	146 892 .000 % 319.35 sec
$k = 10$	182 007 — 20.99 sec	181 908 — 20.18 sec	181 908 — 19.93 sec	181 896 — 21.50 sec	181 896 — 21.82 sec	181 812 — 24.52 sec	*181 803 — 44.19 sec	?
$k = 11$	223 697 — 30.31 sec	223 563 — 29.13 sec	223 563 — 29.45 sec	223 550 — 30.24 sec	223 550 — 33.87 sec	*223 554 — 34.45 sec	*223 879 — 438.82 sec	?
$k = 12$	274 839 — 56.27 sec	274 564 — 55.63 sec	274 564 — 55.41 sec	274 498 — 56.30 sec	274 498 — 56.04 sec	*274 879 — 65.00 sec	?	?
$k = 13$	322 813 — 202.48 sec	322 688 — 199.27 sec	322 688 — 204.79 sec	*322 854 — 182.63 sec	*322 854 — 180.74 sec	*323 289 — 162.98 sec	?	?
$k = 14$	377 655 — 1704.48 sec	*377 795 — 1698.37 sec	*377 795 — 1702.96 sec	*377 742 — 1702.43 sec	*377 742 — 1783.18 sec	*377 913 — 69.15 sec	?	?

Table 7.2: The trade-off between alignment score and computation time. The first row of each box shows the absolute alignment score, the second row the relative score error, and the third row shows the running time in seconds. Where `MSA` could not compute optimal alignments of all (sub)sequence families, the score is marked with an asterisk.

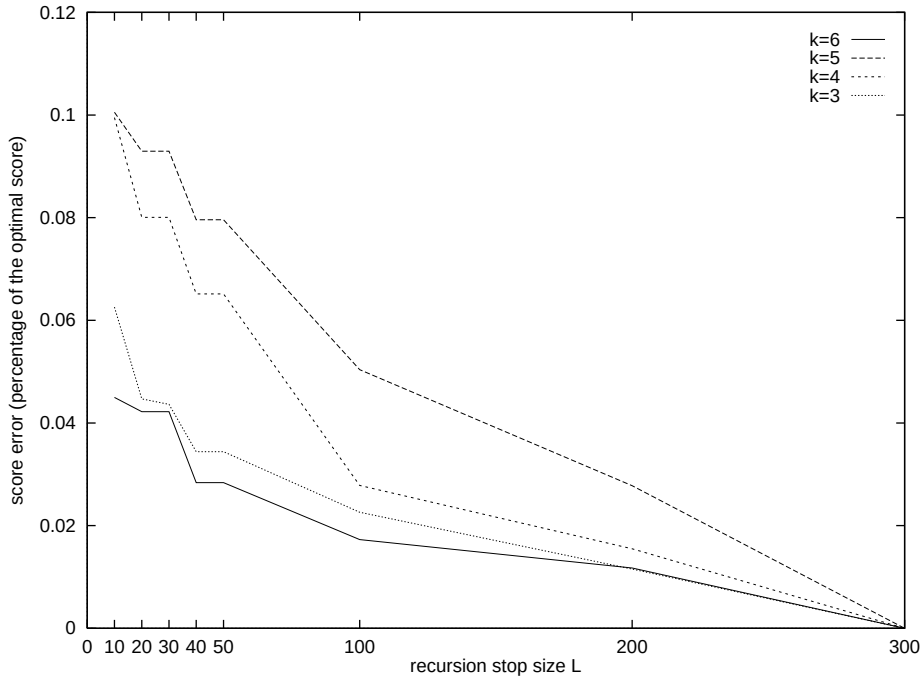

Figure 7.4: Relative deviation from the optimal alignment score for different values of L and k.

Doubtless, even for small L the quality of the alignments is very high. And, as expected, for larger L – at least where `MSA` is able to compute optimal alignments of all subsequence families –, the alignment quality increases further. Of course, this rise of quality is accompanied with an increase of computation time. However, for small L, this trade-off cannot be observed: In most cases, the shortest running time is achieved for L between 20 and 40 while $L = 10$ leads for none of the observed families to the shortest computation time. This indicates that for very short sequences, the interaction with the operating system due to the larger number of `MSA`-calls takes more time than is additionally required for aligning the longer (but still comparatively short) sequences of length ≤ 40.

Note that the incorrect runs of `MSA` result in some unexpected effects concerning the computation time: For example, the abnormally short running time for $k = 14$, $L = 100$ results from errorneous terminations of `MSA` for all subsequence families. The increase of alignment score also confirms this.

The corresponding average values for the complete test sets of one hundred sequence families are shown in Figures 7.4 (average score error) and 7.5 (average running time; note the logarithmic time scale). The general score vs. time trade-off is confirmed. For the small sequence families in this study, a value for L between 40 and 100 seems to be a good compromise with rather high alignment quality and still comparatively short running times. For larger sequence families, of course, a value for L between 20 and 40 should be preferred.

It is noteworthy that the values for $L = 20$ and $L = 30$, and the values for $L = 40$

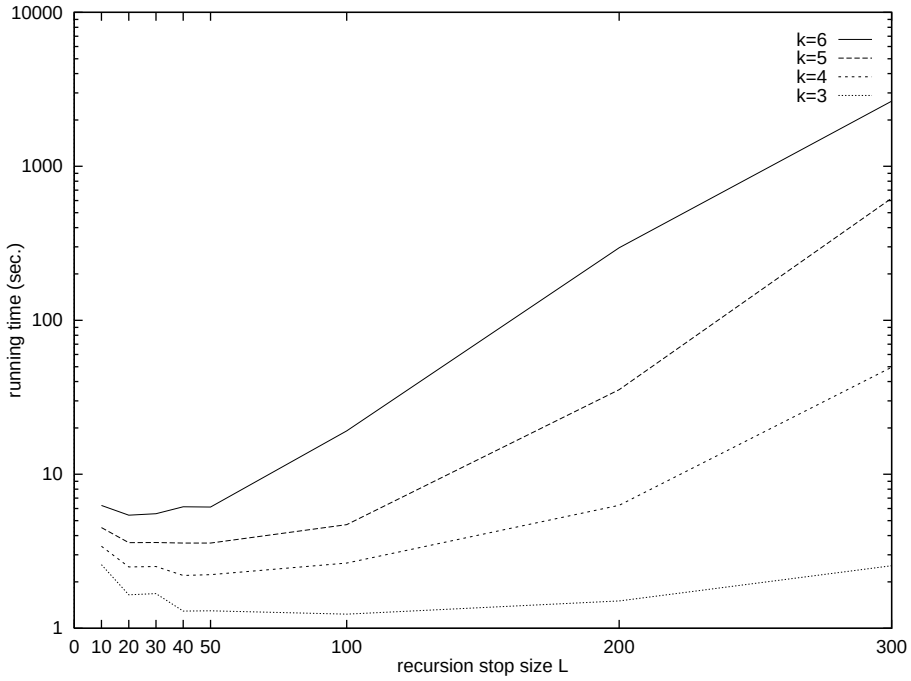

Figure 7.5: Running times of DCA for different values of L and k.

and $L = 50$ almost always coincide. This can be easily understood by observing the series of average sequence lengths of the subsequences when starting with an initial length of $n = 250$: $125, 63, 31, 16, \ldots$ Both $L = 20$ and $L = 30$ (as well as $L = 40$ and $L = 50$) fall in the same class, and thus they have (in most cases) the same number of recursions, resulting in the same subsequence families being aligned by MSA.

A clear dependence of the alignment quality on the number of sequences cannot be observed.

7.2.2 Improvement by Windowing

We also examined the influence of a re-alignment of parts of the sequences in the proximity of slicing positions (see Section 4.5.2) on the alignment score and running time. Table 7.3 shows for different values of L the absolute scores obtained with windows of size $W = L/4$ (for $L = 10$, we used $W = 3$), $W = L/2$, and $W = L$. In most cases the alignment score can be improved. Often, the improvement is rather impressive (e.g. $k = 6$, $L = 40$) while in other cases no improvement can be achieved (e.g. $k = 8$, $L = 10$). Of course, the computation time grows with increasing window size (except for fluctuations in the time measurements). Yet, in some cases (e.g. $k = 7$, $L = 20$), the window approach allows a much faster computation of an optimal alignment than can be achieved by increasing the stop size L.

Also the average values over the complete test set, where we fixed $L = 40$, show a slight improvement of the score for increasing window size (see Figure 7.6). As expected, the running time increases correspondingly (see Figure 7.7).

		$W = 0$	$W = \frac{1}{4}L$	$W = \frac{1}{2}L$	$W = L$
$k = 3$	$L = 10$	12 018 (2.79 sec)	12 018 (4.65 sec)	12 018 (4.82 sec)	12 014 (4.48 sec)
	$L = 20$	12 018 (1.85 sec)	12 018 (2.64 sec)	12 014 (2.75 sec)	12 014 (2.96 sec)
	$L = 40$	12 018 (1.32 sec)	12 014 (1.79 sec)	12 014 (1.86 sec)	12 014 (1.97 sec)
	$L = 100$	12 014 (1.28 sec)	12 014 (1.58 sec)	12 014 (1.78 sec)	12 014 (2.14 sec)
	$L = 200$	12 014 (1.95 sec)	12 014 (1.64 sec)	12 014 (1.87 sec)	12 014 (2.78 sec)
	$L = 300$	12 014 (2.55 sec)			
$k = 4$	$L = 10$	23 961 (3.47 sec)	23 961 (5.36 sec)	23 961 (5.59 sec)	23 953 (5.71 sec)
	$L = 20$	23 950 (2.65 sec)	23 950 (3.74 sec)	23 950 (3.57 sec)	23 947 (3.97 sec)
	$L = 40$	23 947 (2.23 sec)	23 947 (2.95 sec)	23 947 (3.03 sec)	23 947 (3.27 sec)
	$L = 100$	23 947 (2.44 sec)	23 947 (3.11 sec)	23 947 (3.33 sec)	23 947 (3.98 sec)
	$L = 200$	23 947 (3.14 sec)	23 947 (3.39 sec)	23 947 (3.89 sec)	23 947 (5.92 sec)
	$L = 300$	23 947 (4.65 sec)			
$k = 5$	$L = 10$	39 015 (4.31 sec)	39 015 (6.60 sec)	39 015 (6.77 sec)	39 008 (7.34 sec)
	$L = 20$	39 015 (3.66 sec)	39 015 (4.92 sec)	39 008 (5.01 sec)	*39 020 (5.26 sec)
	$L = 40$	39 008 (3.47 sec)	39 008 (3.98 sec)	*39 020 (4.33 sec)	*39 020 (5.30 sec)
	$L = 100$	*39 020 (3.97 sec)	*39 020 (4.14 sec)	*39 020 (4.99 sec)	*39 020 (6.31 sec)
	$L = 200$	*39 020 (5.36 sec)	*39 020 (5.84 sec)	*39 020 (5.84 sec)	*39 020 (9.02 sec)
	$L = 300$	39 008 (10.09 sec)			
$k = 6$	$L = 10$	60 184 (7.01 sec)	60 170 (9.04 sec)	60 170 (9.27 sec)	60 169 (9.95 sec)
	$L = 20$	60 176 (6.16 sec)	60 170 (7.43 sec)	60 170 (8.01 sec)	60 151 (8.30 sec)
	$L = 40$	60 171 (6.10 sec)	60 166 (6.65 sec)	60 138 (7.40 sec)	60 131 (8.50 sec)
	$L = 100$	60 141 (6.55 sec)	60 141 (7.30 sec)	60 134 (8.25 sec)	60 131 (11.10 sec)
	$L = 200$	60 131 (9.11 sec)	60 131 (9.69 sec)	*60 134 (10.88 sec)	*60 135 (20.97 sec)
	$L = 300$	60 131 (15.98 sec)			
$k = 7$	$L = 10$	83 653 (8.03 sec)	83 652 (10.40 sec)	83 645 (10.59 sec)	83 645 (11.29 sec)
	$L = 20$	83 653 (7.69 sec)	83 645 (8.61 sec)	83 645 (9.28 sec)	83 645 (10.01 sec)
	$L = 40$	83 652 (7.57 sec)	83 645 (8.27 sec)	83 645 (8.89 sec)	83 645 (10.59 sec)
	$L = 100$	83 652 (8.73 sec)	83 645 (9.62 sec)	83 645 (11.30 sec)	*83 656 (14.02 sec)
	$L = 200$	83 652 (11.32 sec)	83 645 (12.23 sec)	*83 652 (17.88 sec)	*83 839 (26.94 sec)
	$L = 300$	83 645 (26.39 sec)			
$k = 8$	$L = 10$	112 411 (10.73 sec)	112 411 (13.13 sec)	112 411 (13.42 sec)	112 411 (14.07 sec)
	$L = 20$	112 407 (10.20 sec)	112 407 (12.29 sec)	112 387 (11.96 sec)	*112 431 (13.49 sec)
	$L = 40$	112 370 (10.68 sec)	112 370 (11.49 sec)	112 370 (12.67 sec)	112 344 (15.01 sec)
	$L = 100$	112 370 (12.79 sec)	112 370 (13.95 sec)	*112 514 (16.77 sec)	*112 370 (21.88 sec)
	$L = 200$	112 344 (23.14 sec)	*112 514 (24.40 sec)	*112 465 (28.04 sec)	*112 427 (41.21 sec)
	$L = 300$	112 344 (70.30 sec)			
$k = 9$	$L = 10$	147 150 (15.94 sec)	147 150 (20.05 sec)	*147 166 (20.28 sec)	*147 160 (21.20 sec)
	$L = 20$	147 086 (15.56 sec)	147 085 (18.31 sec)	147 085 (19.04 sec)	147 040 (20.51 sec)
	$L = 40$	146 992 (16.24 sec)	146 991 (18.50 sec)	146 983 (19.70 sec)	146 942 (22.91 sec)
	$L = 100$	*147 079 (19.56 sec)	*146 997 (22.07 sec)	*146 942 (24.81 sec)	*147 071 (32.46 sec)
	$L = 200$	146 942 (30.31 sec)	146 942 (31.71 sec)	*147 062 (32.98 sec)	*147 001 (100.67 sec)
	$L = 300$	146 892 (319.35 sec)			

Table 7.3: Alignment score and computation time for varying recursion stop size L and window size W.

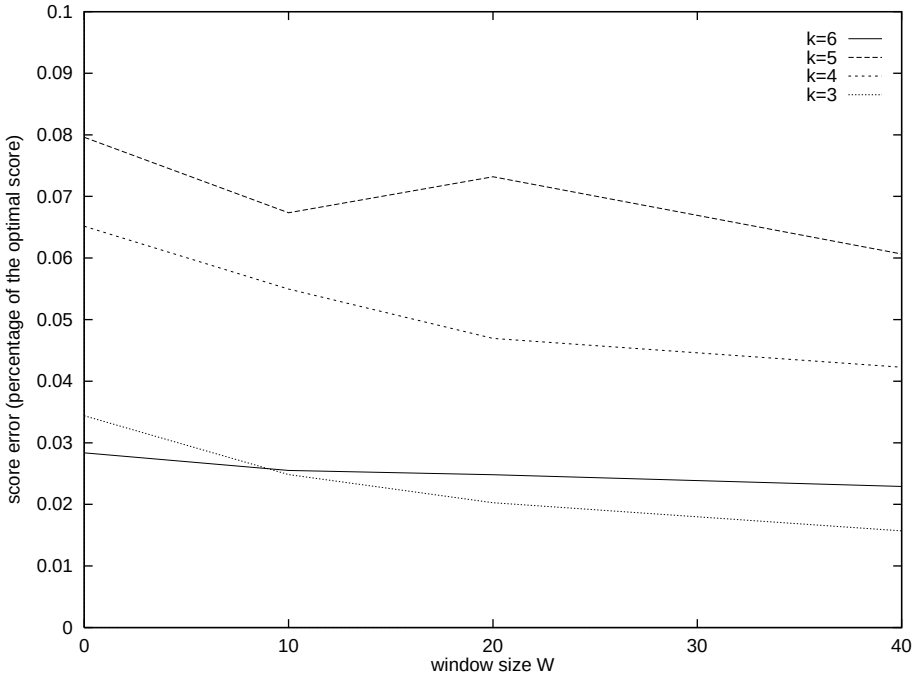

Figure 7.6: Relative deviation from the optimal alignment score for different values of W and k ($L = 40$ fixed). The non-monotonicity of the curve for $k = 5$ results from several non-optimal `MSA`-alignments for $W = 20$.

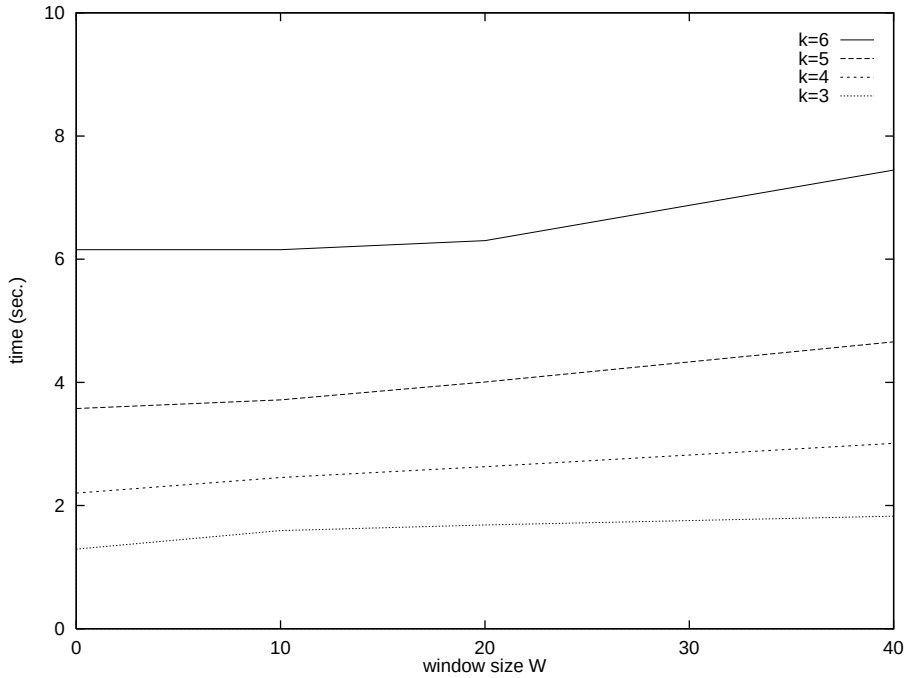

Figure 7.7: Running time of `DCA` for different values of W and k ($L = 40$ fixed).

	$\delta = -10$	$\delta = -5$	$\delta = -2$	$\delta = 0$	$\delta = +2$	$\delta = +5$	$\delta = +10$
$k = 3$	12 025 1	12 014 1	12 025 1	12 018 1	12 014 1	12 014 1	12 016 0
$k = 4$	23 957 8	23 947 4	23 947 1	23 947 3	23 948 2	23 958 0	23 948 6
$k = 5$	39 028 0	39 020 2	39 026 2	39 008 2	39 020 2	39 021 5	39 027 0
$k = 6$	60 138 12	60 140 0	60 179 0	60 171 0	60 161 0	60 152 0	60 202 7
$k = 7$	83 687 26	83 645 34	83 656 23	83 652 20	83 686 16	83 645 13	83 678 17
$k = 8$	112 372 32	112 421 3	112 370 3	112 370 3	112 417 1	112 401 1	112 370 1
$k = 9$	147 389 0	147 100 0	147 151 0	146 992 1	147 023 0	147 032 1	147 043 18

Table 7.4: Alignment score for different slicing positions $\hat{c}_1 := \left\lceil \frac{|\mathbf{s}_1|}{2} \right\rceil + \delta$ of sequence $\mathbf{s}_1$. The second row shows the corresponding minimal multiple additional cost $C_{opt}(\hat{c}_1)$ in the first recursion of *DCA*.

7.2.3 Relaxing $\hat{c}_1$

We also examined the influence of the position $\hat{c}_1$ on the alignments. Table 7.4 shows the alignment scores for different offsets $\delta = -10, -5, -2, 0, +2, +5, +10$ from the midpoint of sequence $\mathbf{s}_1$. (The recursion stop size is fixed to $L = 40$ for all these measurements.) The second row of each box shows the minimal additional cost $C_{opt}(\hat{c}_1)$ computed in the first recursion of *DCA*. (Note that the alignment score depends on *all* cuts. So it is no contradiction when different alignment scores occur although the values of C_{opt} coincide.) A tendency of best alignments to be obtained for $\hat{c}_1$ fixed at or near the midpoint of $\mathbf{s}_1$ can only be estimated. However, the mean values for the complete sets of test families show that in the average case, the best alignments are obtained when δ is not too large (see Figure 7.8).

Even when setting $\hat{c}_1$ inside a region $-\Delta, \ldots, +\Delta$ around the midpoint of $\mathbf{s}_1$ to that position where the value $C_{opt}(\hat{c}_1)$ reaches its absolute minimum, a clear improvement of the alignment score cannot be observed for increasing Δ (see Table 7.5 and Figure 7.9).

7.2.4 Approximate Slicing Positions

Finally, we have examined how much the alignment score gets worse upon the usage of approximate rather than C-optimal families of slicing positions. For these measurements, we have also used the larger sequence families with up to 14 sequences

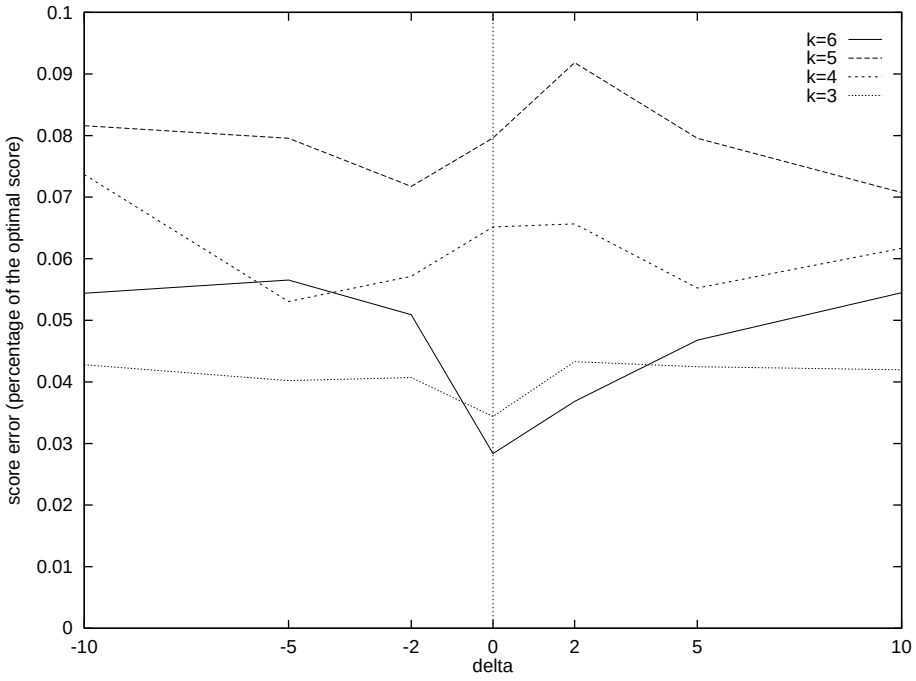

Figure 7.8: Relative deviation from the optimal score for different slicing positions $\hat{c}_1 := \left\lceil \frac{|s_1|}{2} \right\rceil + \delta$.

---

because the speed-up by use of approximate slicing positions becomes clearly apparent only for twelve and more sequences (see Table 7.6). For less sequences, the computation of the optimal alignments of the subsequences with `MSA` dominates the procedure such that no clear differences in the running times can be observed.

In the second row of each box in Table 7.6, the upper bounds $\widehat{C}$ are shown which give rise to the approximate slicing positions. Note again that these values are those only of the first recursion of *DCA* and thus a coinciding $\widehat{C}$ does not necessarily imply the same alignment score. The third row of each box shows the running time of `DCA`. In the last column of Table 7.6, the corresponding values for C-optimal slicing positions are listed.

An unexpected result is the running time when the approximate slicing positions computed with the variant *calc-Chat-firstRowZero* are used. For $k \leq 12$, these runs take often much longer than those where C-optimal families of slicing positions are computed. This is due to the less compatible sequences resulting from the comparatively bad slicing positions of *calc-Chat-firstRowZero* causing an increase of computation time of `MSA` which is larger than the time saved by skipping the search for C-optimal cuts. In contrast, the more sophisticated methods, especially the iterative version *calc-Chat-itSCRmin*, produce alignments whose scores are often very close to the scores of the alignments computed with C-optimal families of slicing positions. In three cases ($k = 6$, *itSCRoi*; $k = 9$, *itSCRmin*; $k = 14$, *itSCRmin* and

	$\Delta = 0$	$\Delta = 2$	$\Delta = 4$	$\Delta = 6$	$\Delta = 8$	$\Delta = 10$
$k = 3$	12 018 1 $\delta = 0$	12 018 1 $\delta = 0$	12 018 1 $\delta = 0$	12 018 1 $\delta = 0$	12 018 1 $\delta = 0$	12 018 0 $\delta = 10$
$k = 4$	23 947 3 $\delta = 0$	23 948 1 $\delta = -2$	23 948 0 $\delta = 3$	23 948 0 $\delta = 3$	23 948 0 $\delta = 3$	23 948 0 $\delta = 3$
$k = 5$	39 008 2 $\delta = 0$	39 008 2 $\delta = 0$	39 008 2 $\delta = 0$	39 008 2 $\delta = 0$	39 070 1 $\delta = -7$	39 020 0 $\delta = 9$
$k = 6$	60 171 0 $\delta = 0$	60 170 0 $\delta = 0$	60 170 0 $\delta = 0$	60 163 0 $\delta = 0$	60 163 0 $\delta = 0$	60 143 0 $\delta = 0$
$k = 7$	83 652 20 $\delta = 0$	83 671 16 $\delta = 2$	83 645 12 $\delta = 4$	83 645 12 $\delta = 4$	83 645 12 $\delta = 4$	83 645 12 $\delta = 4$
$k = 8$	112 370 3 $\delta = 0$	112 370 1 $\delta = 2$	112 370 1 $\delta = 2$	112 370 1 $\delta = 2$	112 370 1 $\delta = 2$	112 398 1 $\delta = 2$
$k = 9$	146 992 1 $\delta = 0$	146 992 0 $\delta = 2$	147 072 0 $\delta = 2$	147 072 0 $\delta = 2$	147 197 0 $\delta = 2$	147 036 0 $\delta = 2$

Table 7.5: Alignment scores when $\hat{c}_1$ is set inside a region around the midpoint of $\mathbf{s}_1$ to a position $\left\lceil \frac{|\mathbf{s}_1|}{2} \right\rceil + \delta$, $\delta \in \{-\Delta, \ldots, +\Delta\}$, where $C_{opt}(\hat{c}_1)$ is minimal. The second row of each box shows the optimal multiple additional cost $C_{opt}(\hat{c}_1)$ in the first recursion of *DCA*. The third row shows the offset δ to that position $\hat{c}_1$ where the minimum $C_{opt}(\hat{c}_1)$ is found.

itSCRoi), the approximate slicing positions even result in better alignments than the C-optimal cuts.

The average values for the complete test sets of one hundred sequence families for each $k \in \{3, 4, 5, 6\}$ are shown in Figure 7.10. The trend of *calc-Chat-itSCRmin* resulting in the best alignments with scores close to those computed with C-optimal families of slicing positions is confirmed.

7.3 Comparison of Different Realizations of *calc-Chat*

In the previous section, we already measured the differences between the variants of *calc-Chat* by the quality of the alignments when using the slicing positions directly

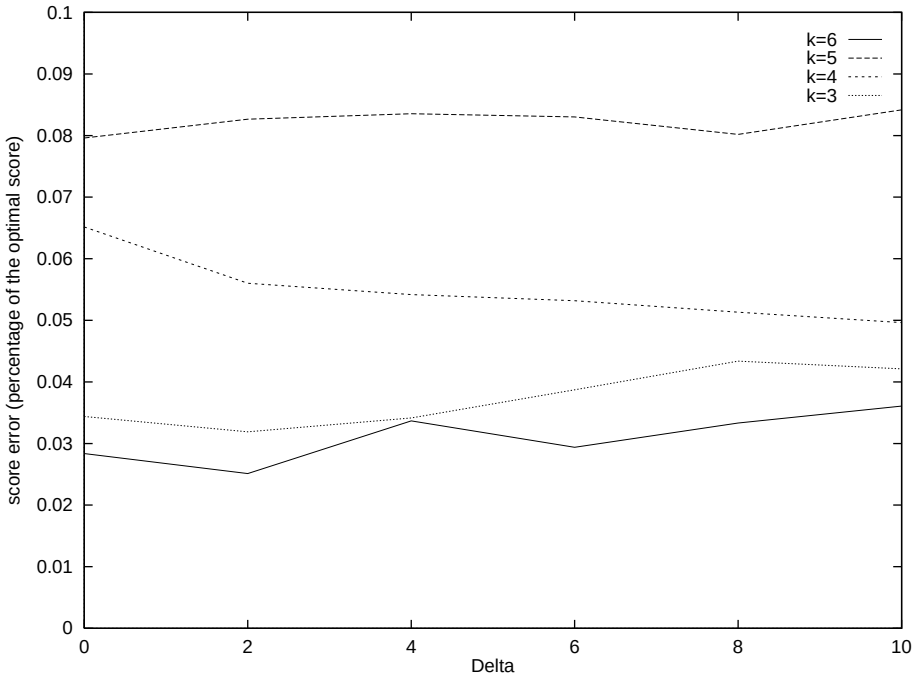

Figure 7.9: Relative deviation from the optimal score when $\hat{c}_1$ is set inside the region $\{\lceil \frac{|s_1|}{2} \rceil - \Delta, \ldots, \lceil \frac{|s_1|}{2} \rceil + \Delta\}$ to that position $\hat{c}_1 = \lceil \frac{|s_1|}{2} \rceil + \delta$, $|\delta| \leq \Delta$, where $C_{opt}(\hat{c}_1)$ is minimal.

---

to cut the sequences. In this section, we continue this comparison by showing the influence of the different methods on the search for C-optimal families of slicing positions. Since the shape of the additional-cost matrices and thus the size of the relevant part of the search space depends considerably also on the gap function, we now use affine gap penalties $g(l) := 8 + 12\,l$ with the PAM 250 substitution matrix. Table 7.7 shows the mean values of $\widehat{C}$ computed with the different algorithms *calc-Chat*. The corresponding ratios $\frac{r}{n}$ are shown in Table 7.8, and the running times are displayed in Table 7.9. Table 7.10 compares the average and maximal number of iterations of the iterative methods *calc-Chat-itSCRmin* and *calc-Chat-itSCRoi*.

For the time evaluation, we have skipped the calculation of the optimal alignments of the (sub)sequences since for some sequence families, the `MSA`-runs largely dominate the overall running time. Proceeding this way, we ensure that the time-differences observed indeed result from differences in the size of the search space due to the different upper bounds $\widehat{C}$. In all cases, the minimal additional cost C_{opt} is computed exhaustively within the relevant part of the search space.

The following observations are remarkable:

- For all variants, the upper bound $\widehat{C}$, the ratio $\frac{r}{n}$, and the running time behave similar, i.e. a lower $\widehat{C}$ results in a smaller ratio $\frac{r}{n}$ which leads to a faster search.

	firstRowZero	*SCmin*	*SCRmin*	*itSCRmin*	*SCRoi*	*itSCRoi*	C_{opt}
	12 018	12 018	12 018	12 018	12 018	12 018	12 018
$k = 3$	1	1	1	1	1	1	1
	1.21 sec	1.22 sec	1.22 sec	1.22 sec	1.25 sec	1.23 sec	1.32 sec
	24 246	23 979	23 947	23 947	24 112	24 126	23 947
$k = 4$	9	3	3	3	3	3	3
	2.36 sec	2.24 sec	2.17 sec	2.40 sec	2.31 sec	2.13 sec	2.23 sec
	39 008	39 026	39 008	39 008	39 008	39 008	39 008
$k = 5$	2	2	2	2	2	2	2
	3.49 sec	3.25 sec	3.37 sec	3.34 sec	3.31 sec	3.38 sec	3.47 sec
	61 355	60 174	60 171	60 171	60 762	60 143	60 171
$k = 6$	0	0	0	0	0	0	0
	5.50 sec	4.74 sec	5.26 sec	5.33 sec	5.62 sec	5.23 sec	6.10 sec
	84 095	83 725	83 652	83 652	83 695	83 652	83 652
$k = 7$	58	20	20	20	20	20	20
	7.84 sec	7.24 sec	7.37 sec	7.55 sec	7.48 sec	7.38 sec	7.57 sec
	113 206	112 543	112 370	112 370	112 370	112 370	112 370
$k = 8$	62	38	3	3	3	3	3
	11.43 sec	10.77 sec	11.22 sec	11.03 sec	10.64 sec	11.03 sec	10.68 sec
	149 490	147 085	147 350	146 892	147 814	147 283	146 992
$k = 9$	1	1	1	1	1	1	1
	25.74 sec	13.86 sec	16.13 sec	16.21 sec	16.68 sec	16.30 sec	16.24 sec
	184 959	182 261	182 245	181 913	182 575	182 016	181 896
$k = 10$	184	43	91	43	98	43	43
	65.90 sec	17.85 sec	20.50 sec	20.39 sec	20.82 sec	20.31 sec	21.50 sec
	228 527	224 331	223 638	223 638	224 440	223 638	223 550
$k = 11$	434	335	120	120	120	120	120
	218.19 sec	23.91 sec	27.05 sec	32.32 sec	27.28 sec	27.45 sec	30.24 sec
	280 694	276 594	275 067	274 498	275 723	274 554	274 498
$k = 12$	663	305	195	195	260	195	195
	493.61 sec	32.46 sec	43.80 sec	44.49 sec	45.42 sec	43.93 sec	56.30 sec
	329 369	327 791	324 342	323 722	326 116	323 919	322 854
$k = 13$	1341	1034	243	243	746	267	243
	75.03 sec	59.12 sec	70.62 sec	69.59 sec	72.65 sec	69.13 sec	182.63 sec
	382 466	378 853	377 919	377 698	378 326	377 698	377 742
$k = 14$	1031	105	105	105	258	105	105
	107.24 sec	41.43 sec	61.79 sec	61.95 sec	63.00 sec	61.86 sec	1702.43 sec

Table 7.6: Alignment score, upper bound $\widehat{C}$, and computation time when using approximate slicing positions. The last column shows the corresponding values when C-optimal families of slicing positions are used ($L = 40$).

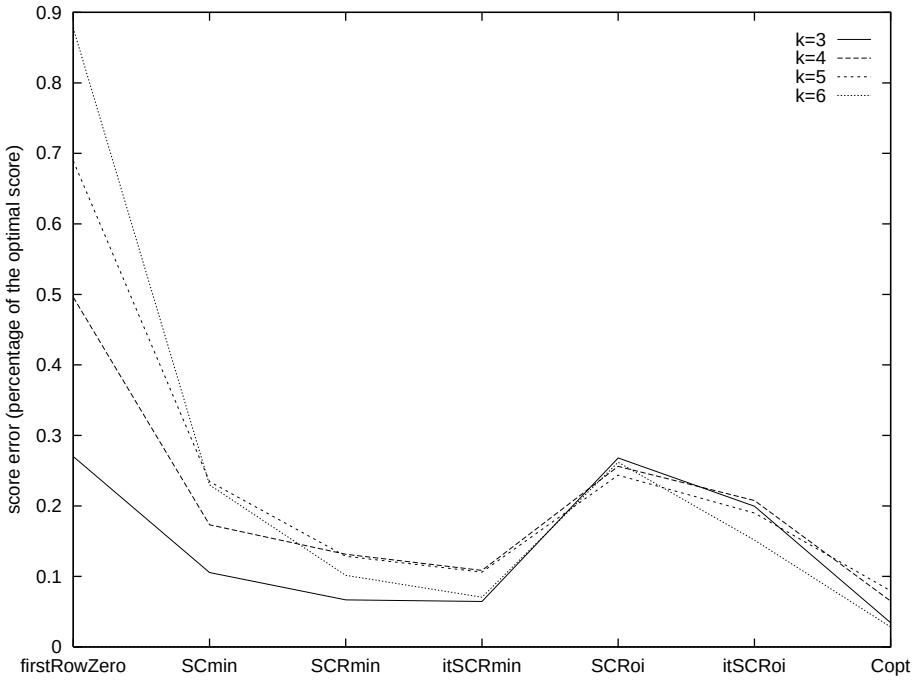

Figure 7.10: Relative deviation from the optimal score for the different algorithms *calc-Chat* used to compute approximate slicing positions ($L = 40$ fixed).

- Compared to the naive version without pruning the search space, even the method *calc-Chat-firstRowZero* diminishes the search space considerably. This speeds up the procedure by a factor of four and more for larger k.

- Further significant improvement is obtained with the greedy methods. Among the non-iterative variants, *calc-Chat-SCRmin* produces by far the lowest upper bounds $\widehat{C}$, again reducing the search time to less than a third. For the test families with 14 sequences, the speed-up factor is even larger than ten. For twelve and less sequences, the simpler method *calc-Chat-SCmin* has shorter running times due to the comparatively faster computation of the upper bounds.

- The iteration of *calc-Chat-SCRmin* allows a further acceleration by approximately ten percent.

- The iterated order-independent method *calc-Chat-itSCRoi* computes upper bounds $\widehat{C}$ of a similar quality. In some cases, the running times are even lower than for *calc-Chat-itSCRmin*. For larger k, the independence from the sequence order seems to be an advantage preventing the procedure from running into bad local minima. This is also confirmed by the larger numbers of iterations compared to those of the greedy algorithm *calc-Chat-itSCRmin* (see Table 7.10).

average $\widehat{C}$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$	$k=9$	$k=10$	$k=11$	$k=12$	$k=13$	$k=14$
calc-Chat-firstRowZero	4.1	10.9	26.0	44.8	70.2	97.0	111.6	216.5	254.1	291.7	402.9	452.4
calc-Chat-SCmin	2.1	4.2	9.8	11.3	16.2	28.7	26.6	40.5	37.1	78.1	91.7	74.1
calc-Chat-SCRmin	1.8	2.8	7.8	11.1	15.9	17.7	18.0	32.2	36.4	67.7	69.3	55.6
calc-Chat-itSCRmin	1.8	2.8	7.5	9.5	11.4	16.3	18.0	25.2	31.1	57.7	61.8	52.6
calc-Chat-SCRoi	4.9	6.8	12.3	20.3	22.0	22.9	22.4	56.3	61.1	88.3	83.7	81.0
calc-Chat-itSCRoi	3.1	5.0	9.3	15.7	12.4	16.5	18.1	26.2	31.7	56.3	60.8	55.5
C_{opt}	1.3	2.3	6.7	8.0	11.1	15.3	17.7	24.8	29.7	51.3	53.6	50.8

Table 7.7: Average values of the upper bound $\widehat{C}$ for the different algorithms *calc-Chat*.

average ratio $\frac{r}{n}$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$	$k=9$	$k=10$	$k=11$	$k=12$	$k=13$	$k=14$
calc-Chat-firstRowZero	1.01 %	1.97 %	4.02 %	6.43 %	9.74 %	12.89 %	14.51 %	24.53 %	28.06 %	33.30 %	36.90 %	41.45 %
calc-Chat-SCmin	0.72 %	0.98 %	1.99 %	2.13 %	2.84 %	4.58 %	4.23 %	6.39 %	6.06 %	11.56 %	12.93 %	11.21 %
calc-Chat-SCRmin	0.64 %	0.81 %	1.74 %	2.15 %	2.74 %	3.10 %	3.14 %	5.27 %	6.06 %	10.14 %	10.39 %	8.98 %
calc-Chat-itSCRmin	0.64 %	0.81 %	1.70 %	1.94 %	2.22 %	2.90 %	3.14 %	4.43 %	5.37 %	8.97 %	9.50 %	8.55 %
calc-Chat-SCRoi	0.78 %	1.16 %	2.33 %	3.26 %	3.55 %	3.73 %	3.69 %	8.04 %	8.51 %	12.23 %	12.02 %	11.86 %
calc-Chat-itSCRoi	0.78 %	1.06 %	1.96 %	2.76 %	2.35 %	2.92 %	3.14 %	4.54 %	5.43 %	8.82 %	9.40 %	8.95 %
C_{opt}	0.55 %	0.75 %	1.63 %	1.74 %	2.16 %	2.80 %	3.10 %	4.39 %	5.21 %	8.22 %	8.60 %	8.39 %

Table 7.8: Average ratios $\frac{r}{n}$ for the different algorithms *calc-Chat*.

search time (sec.)	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$	$k=9$	$k=10$	$k=11$	$k=12$	$k=13$	$k=14$
full matrices ($r=n$)	1.19	3.01	5.59	9.33	14.41	21.25	30.80	57.67	129.14	483.12	?	?
calc-Chat-firstRowZero	0.49	1.06	1.90	3.12	4.79	7.00	10.66	15.80	47.02	120.10	196.30	804.92
calc-Chat-SCmin	0.49	1.06	1.84	2.88	4.18	6.10	7.80	10.51	14.27	32.94	83.44	75.92
calc-Chat-SCRmin	0.49	1.10	1.89	3.07	4.79	6.83	9.58	14.02	21.11	58.04	61.85	66.39
calc-Chat-itSCRmin	0.50	1.09	1.94	3.14	4.73	6.76	9.58	13.89	19.42	39.58	51.44	57.92
calc-Chat-SCRoi	0.49	1.10	1.89	3.13	4.75	6.87	9.81	14.50	24.30	72.57	82.07	90.95
calc-Chat-itSCRoi	0.49	1.06	1.89	3.09	4.70	6.79	9.50	13.89	19.26	40.23	50.74	57.49

Table 7.9: Average time (in seconds) used for the search for C-optimal families of slicing positions for the different algorithms *calc-Chat*.

# iterations	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$	$k=9$	$k=10$	$k=11$	$k=12$	$k=13$	$k=14$
calc-Chat-itSCRmin	0.42 (2)	0.50 (2)	0.84 (3)	0.98 (3)	1.21 (3)	1.26 (3)	1.33 (2)	1.42 (3)	1.46 (3)	1.68 (4)	1.59 (4)	1.66 (3)
calc-Chat-itSCRoi	0.33 (2)	0.47 (3)	0.86 (4)	1.02 (5)	1.26 (5)	1.34 (4)	1.37 (3)	1.52 (5)	1.53 (4)	1.81 (6)	1.71 (6)	1.74 (5)

Table 7.10: Average and maximal number of iterations of the iterative functions *calc-Chat-itSCRmin* and *calc-Chat-itSCRoi*.

- Further significant speed-up by improved upper bounds $\widehat{C}$ cannot be expected since the values $\widehat{C}$ computed by *calc-Chat-itSCRmin* and *calc-Chat-itSCRoi* are already very close to the minima C_{opt} shown in the last row of Table 7.7.

- As expected, the ratio $\frac{r}{n}$ increases with the number of sequences. However, this does not necessarily mean a lower speed-up factor: Let q_k denote the quotient $\frac{r}{n}$ obtained for k sequences. By definition, $0 < q_k \leq 1$. Then, $(q_{k+1})^{k+1}$ can be smaller than $(q_k)^k$ even if $q_{k+1} > q_k$. See e.g. $k = 9$, *calc-Chat-firstRowZero*: compared to the exhaustive search, the speed-up increases from a factor of about three ($k = 9$) to more than four ($k = 10$) although the ratio $\frac{r}{n}$ worsens considerably from $q_9 = 14.51$ to $q_{10} = 24.53$.

In fact, the least expected observation is that for $k = 14$ sequences the average values of $\widehat{C}$ and hence the fractions $\frac{r}{n}$ are smaller than those for $k = 12$ and $k = 13$. By observing the data set, one finds two families with running times of more than five minutes among the families of twelve as well as among those of thirteen sequences while among the families with fourteen sequences, there is only one such "killer family" with comparatively dissimilar sequences. (Of course, this indicates that our data sets of one hundred sequence families are still too small to obtain statistically well-founded results.) However, – due to the extra dimension – the running times do not show this effect in the same clarity.

We have also measured the relative size of the relevant parts $\frac{r}{n}$ depending on the length of the sequences. For this purpose, we have created a second set of test families with eight sequences of expected average length n for each $n \in \{100, 200, \ldots, 1000\}$. The results are shown in Table 7.13. As we anticipated in Section 5.1, for all variants the ratio $\frac{r}{n}$ decreases with increasing sequence length.

7.4 Comparison of Different Realizations of *calc-Copt*

In this section, we compare the basic ("exhaustive") method of computing the minimal additional cost C_{opt} to the more sophisticated solutions with monotony bounds and/or the preprocessing of the minimal additional cost of a subfamily of the sequences. For all variants, the upper bound $\widehat{C}$ is computed with the method *calc-Chat-itSCRmin* which performs very well in most cases (see the previous section).

We have already mentioned in Section 6.1.2 that the preprocessing of subfamilies of $\kappa < k$ sequences requires some care. While it is rather evident that those sequences which contribute with the highest pairwise additional costs to the upper bound $\widehat{C}$ give the highest values $C_{\pi(1),\ldots,\pi(\kappa)}$, it is not so obvious how many sequences should be involved. Consequently, we have run tests for all $\kappa \in \{3, \ldots, 13\}$. The computation times required by the search for C-optimal cuts are shown in Table 7.11.

search time (sec.)	$k = 8$	$k = 9$	$k = 10$	$k = 11$	$k = 12$	$k = 13$	$k = 14$
exhaustive search	6.76	9.58	13.89	19.24	39.58	51.44	57.92
$\kappa = 3$	6.90	9.77	14.44	20.87	43.04	74.34	63.00
$\kappa = 4$	7.00	9.80	14.68	20.74	38.41	55.41	58.31
$\kappa = 5$	6.89	9.86	14.40	20.18	31.90	42.84	52.57
$\kappa = 6$	6.96	9.78	14.29	20.66	32.13	41.56	47.98
$\kappa = 7$	6.92	9.73	14.38	19.86	31.25	39.67	44.37
$\kappa = 8$	—	9.74	14.38	20.50	32.77	37.94	42.16
$\kappa = 9$	—	—	14.50	21.31	28.76	42.58	45.99
$\kappa = 10$	—	—	—	20.88	32.45	46.12	45.70
$\kappa = 11$	—	—	—	—	48.30	67.55	46.70
$\kappa = 12$	—	—	—	—	—	57.05	54.76
$\kappa = 13$	—	—	—	—	—	—	59.08

Table 7.11: Average running times required by the search for C-optimal families of slicing positions where the minimal additional cost of a subfamily with $\kappa < k$ sequences is pre-computed.

Of course, for only a few sequences, preprocessing can even slow down the procedure: For less than twelve sequences, the exhaustive search is faster than the sophisticated method with any κ. However, for twelve and more sequences, the benefit of preprocessing becomes visible. The minimal running time seems to be obtained for $\kappa = \left\lceil \frac{k}{2} \right\rceil + 1$ leading to a speed-up of more than twenty-five percent for $k = 14$.

We also applied the method recursively with decrement ρ. Because – as we have seen above – the pre-computing has no benefit when the pre-processed sequence families are too small, we stop the recursion if the subfamily contains less than eight sequences (or, of course, if $\kappa - \rho < 3$). In Table 7.12, the running times for recursion decrements $\rho \in \{1, \ldots, 7\}$ are presented. For large k, a further small speed-up of about two percent is perceptible.

Finally, we present the effect of the monotony bounds on the time required for the search for C-optimal families of slicing positions. Table 7.14 compares the exhaustive search with the "pure" monotony bounds, a combination of monotony bounds and preprocessing of subfamilies of size $\kappa = \left\lceil \frac{k}{2} \right\rceil + 1$, and the recursive preprocessing with decrement $\rho = 4$ and its combination with the monotony bounds. For larger k, a further speed-up of ten to twenty-five percent is achieved by the use of monotony-bounds.

In conclusion, we have observed that for less than ten sequences, the exhaustive version performs best – mostly due to its simplicity. For more sequences, a reduction of the search space by additional upper bounds obtained from pre-computed minimal additional costs of subfamilies and the use of monotony bounds allows significant speed-ups.

search time (sec.)	$k = 8$	$k = 9$	$k = 10$	$k = 11$	$k = 12$	$k = 13$	$k = 14$
exhaustive search	6.76	9.58	13.89	19.42	39.58	51.44	57.92
$\rho = 1$	(6.92)	9.92	14.45	20.27	43.36	54.95	59.97
$\rho = 2$	(6.96)	(9.73)	14.85	19.85	30.92	46.74	46.66
$\rho = 3$	(6.98)	(9.78)	(14.38)	19.36	28.83	46.00	43.19
$\rho = 4$	(7.00)	(9.86)	(14.29)	(19.86)	29.57	36.71	41.81
$\rho = 5$	(6.90)	(9.80)	(14.40)	(20.66)	(31.25)	39.72	41.73
$\rho = 6$	—	(9.77)	(14.68)	(20.18)	(32.13)	(39.67)	(42.16)
$\rho = 7$	—	—	(14.44)	(20.74)	(31.90)	(41.56)	(44.37)

Table 7.12: Average running times required by the search for C-optimal families of slicing positions where the minimal additional cost of a subfamily is pre-computed recursively with recursion decrement ρ. The recursion is stopped when less than eight sequences remain. The values in parentheses require no recursion. They are taken from Table 7.11.

7.5 Dependence on Sequence Length and Number

The aim of this section is the empirical validation of the asymptotic time and space complexity of DCA as derived in Section 4.4. Unfortunately, for small problem size, often boundary effects make a validation of the asymptotic behavior impossible. Consequently, we use as many and as long sequences as possible. Moreover, the asymptotic behavior of the algorithm should be best observable for the basic version without running-time improvements. However, with that version only small families of short sequences can be aligned. For the measurements presented in the following, we have used the fastest variant of `DCA` where an upper bound for the multiple additional cost is computed with *calc-Chat-itSCRmin* and in the search phase, subfamilies are pre-processed recursively with a decrement $\rho = 4$ for the size of the subfamilies, combined with a pruning of the search space with monotony bounds.

Figures 7.11 and 7.12 show the running time of `DCA` for different sequence lengths. The curves show the expected quadratic behavior.

The corresponding memory usage is shown in Figures 7.13 and 7.14. Due to boundary effects, the quadratic dependence on the sequence length is not visible.

The time and memory requirements of `DCA` depending on the number of sequences are shown in Figures 7.15 and 7.16. An exponential increase of running time for increasing sequence number can be observed. By comparison of both figures, one can also hypothesize that the memory usage only grows quadratically with the number of sequences. This behavior would be in accordance with our theoretical analysis.

average ratio $\frac{r}{n}$	$n = 100$	$n = 200$	$n = 300$	$n = 400$	$n = 500$	$n = 600$	$n = 700$	$n = 800$	$n = 900$	$n = 1000$
calc-Chat-firstRowZero	47.23 %	19.44 %	8.99 %	5.15 %	3.37 %	4.04 %	2.79 %	2.36 %	1.94 %	1.89 %
calc-Chat-SCmin	22.70 %	6.82 %	3.45 %	2.07 %	1.64 %	1.19 %	1.03 %	0.93 %	0.73 %	0.69 %
calc-Chat-SCRmin	19.48 %	5.30 %	2.86 %	1.22 %	1.01 %	1.17 %	0.60 %	0.52 %	0.50 %	0.40 %
calc-Chat-itSCRmin	17.83 %	4.67 %	2.50 %	1.22 %	0.95 %	0.94 %	0.56 %	0.50 %	0.40 %	0.40 %
calc-Chat-SCRoi	25.19 %	6.13 %	3.82 %	1.22 %	1.12 %	1.64 %	0.80 %	0.54 %	0.55 %	0.78 %
calc-Chat-itSCRoi	21.09 %	4.66 %	2.59 %	1.22 %	0.98 %	0.99 %	0.67 %	0.50 %	0.40 %	0.41 %
C_{opt}	17.17 %	4.50 %	2.39 %	1.17 %	0.94 %	0.88 %	0.56 %	0.47 %	0.40 %	0.39 %

Table 7.13: The ratio $\frac{r}{n}$ depending on the length of the sequences for the different variants of computing an estimate. (All families contain $k = 8$ sequences.)

search time (sec.)	$k = 3$	$k = 4$	$k = 5$	$k = 6$	$k = 7$	$k = 8$	$k = 9$	$k = 10$	$k = 11$	$k = 12$	$k = 13$	$k = 14$
exhaustive search	0.50	1.09	1.94	3.14	4.73	6.76	9.58	13.89	19.42	39.58	51.44	57.92
monotony bounds	0.50	1.10	1.90	3.21	4.81	6.83	9.52	13.78	19.20	34.11	40.51	46.15
subfamilies ($\kappa = \lceil \frac{k}{2} \rceil + 1$)	—	—	—	—	4.83	6.89	9.78	14.29	19.86	31.25	37.94	42.16
with mon. bounds	—	—	—	—	4.77	6.93	9.76	14.25	19.57	29.81	34.86	41.14
rec. subfamilies ($\rho = 4$)	—	—	—	—	—	—	—	—	—	29.57	36.71	41.81
with mon. bounds	—	—	—	—	—	—	—	—	—	27.42	34.14	41.09

Table 7.14: Average running times required by the search for C-optimal families of slicing positions with the approach of monotony bounds and combinations of this approach with that of pre-processed subfamilies.

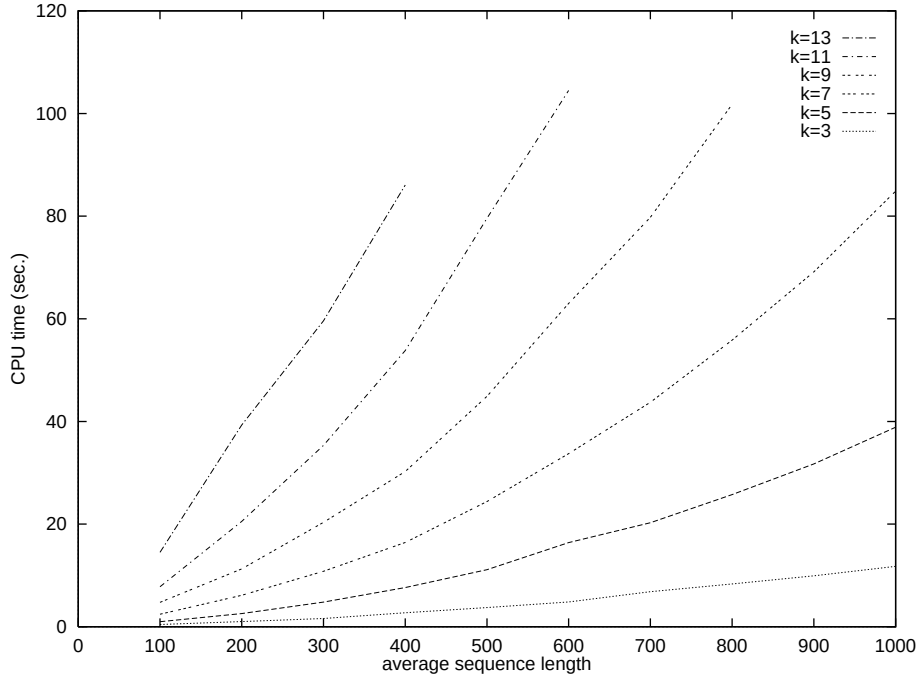

Figure 7.11: The time usage of `DCA` depending on the sequence length.

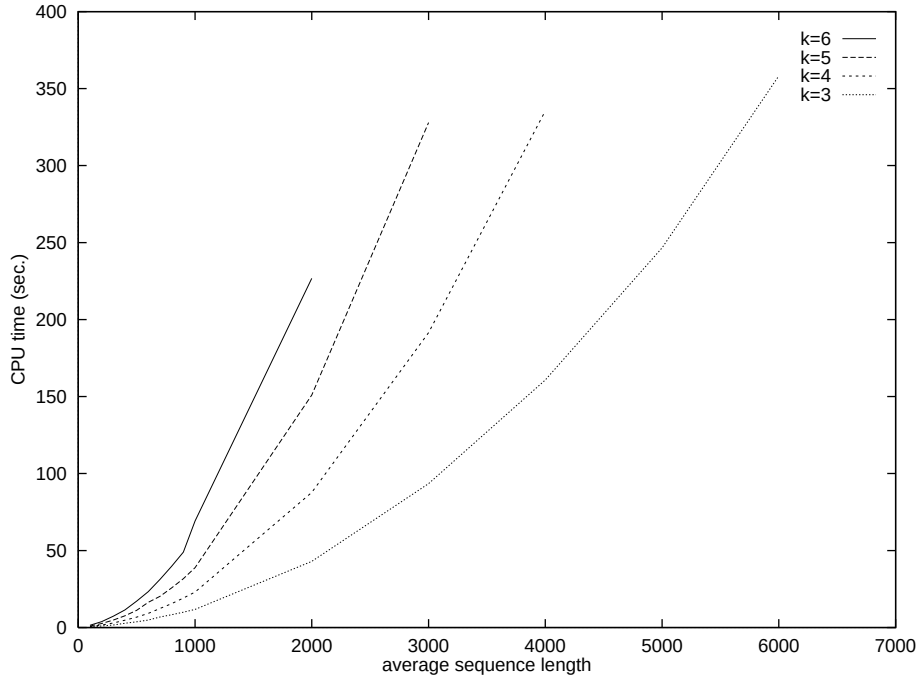

Figure 7.12: The time usage of `DCA` for longer sequences.

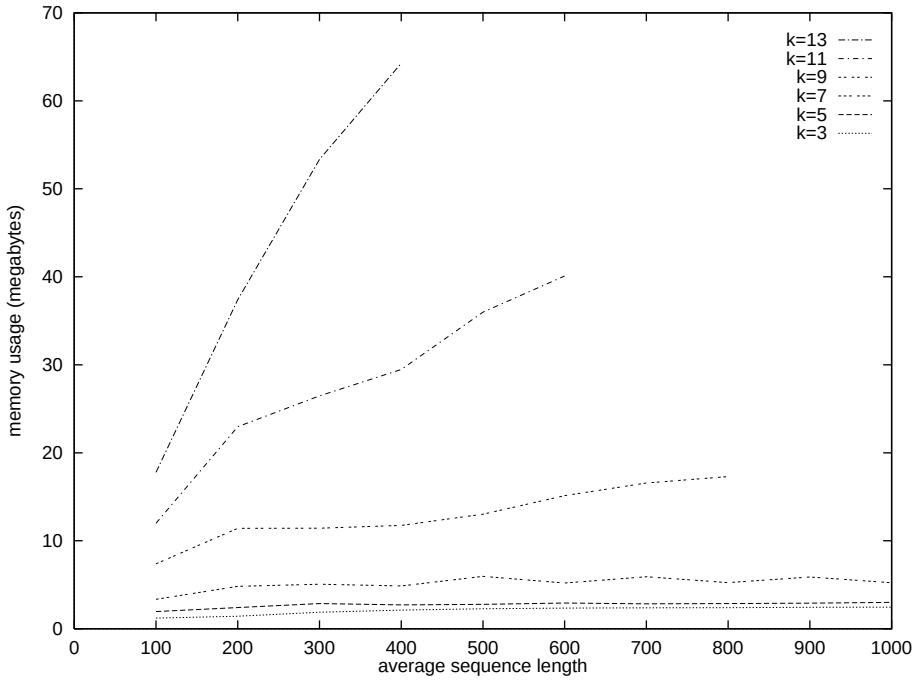

Figure 7.13: The memory usage of `DCA` depending on the sequence length.

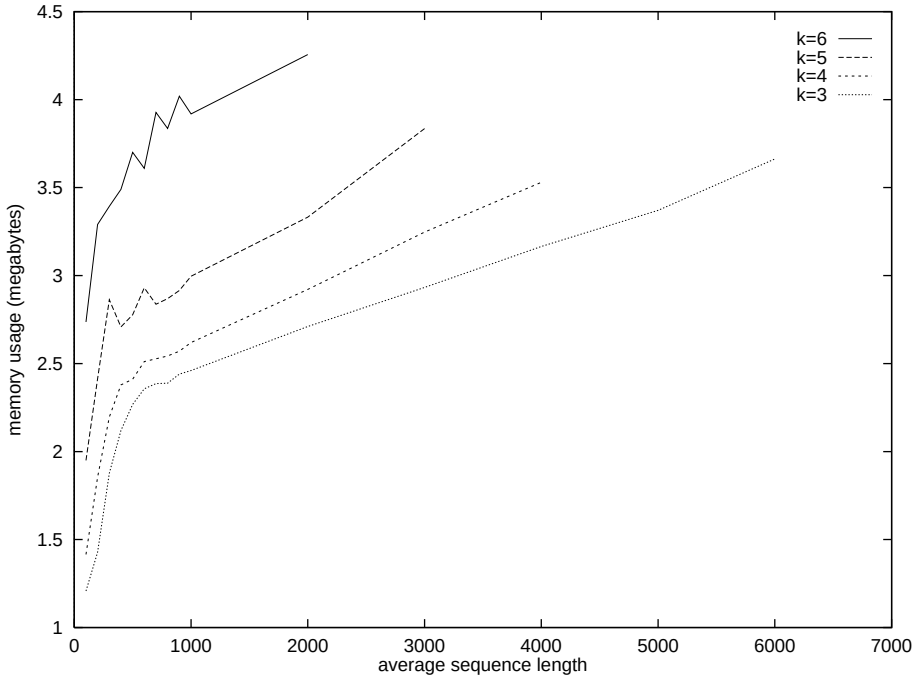

Figure 7.14: The memory usage of `DCA` for longer sequences.

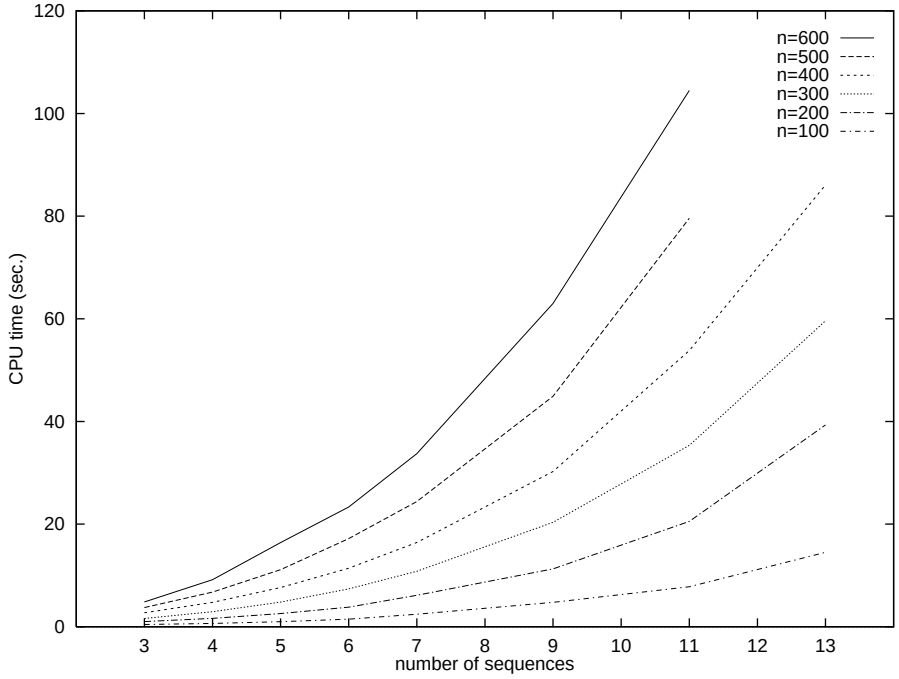

Figure 7.15: The time usage of `DCA` depending on the number of sequences.

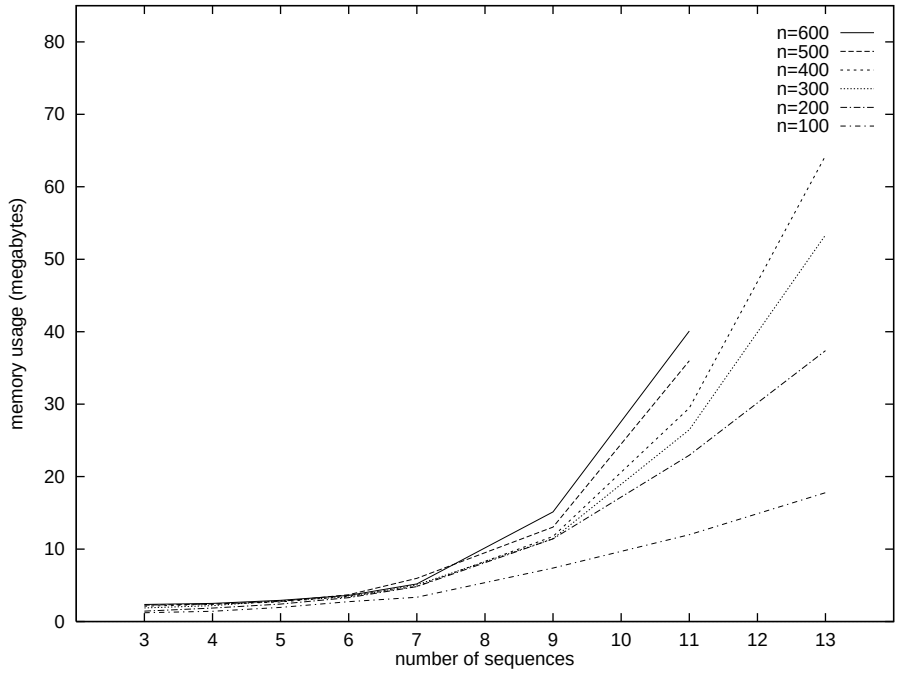

Figure 7.16: The memory usage of `DCA` depending on the number of sequences.

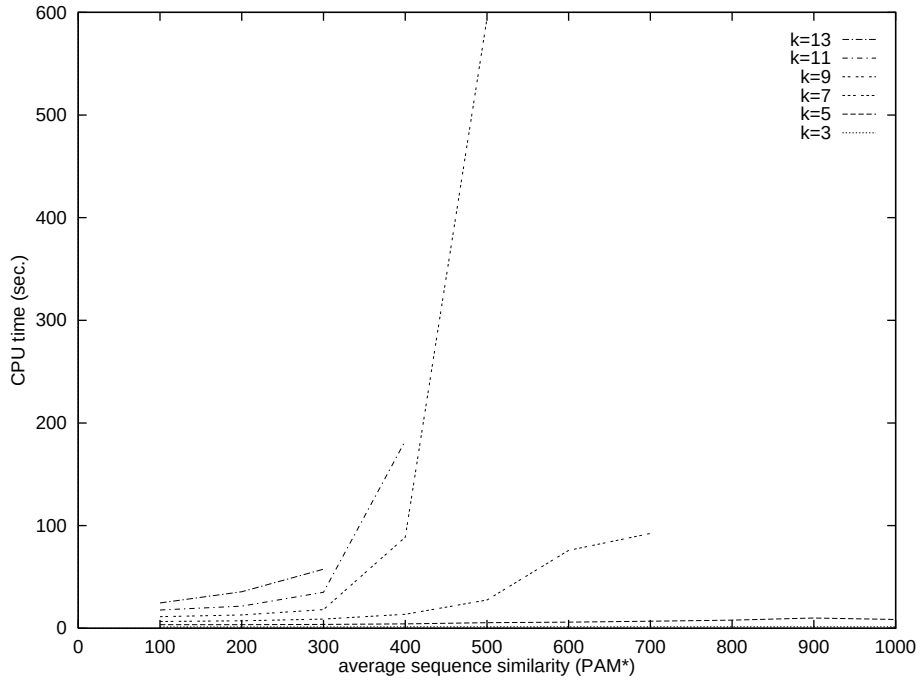

Figure 7.17: The time usage of DCA for different sequence similarities.

7.6 Dependence on Sequence Similarity

Finally, we have evaluated the dependence of DCA on the similarity of the sequences. We have created random sequence families with different average similarity from 100 up to 1000 PAM*. The sequences are of average expected length $n = 250$. Time and memory usage of DCA are presented in Figures 7.17 and 7.18. As presumed, the closer the sequences are related, the faster proceeds the algorithm and – due to the smaller relevant parts of the additional-cost matrices – the less memory is consumed.

We emphasize again that by reason of the large dependence on the sequence similarity, the results presented for the random sequences do not necessarily – although averaged over a large number of different sequence families – generalize to arbitrary data sets. As we will also see in the discussion of results on biological sequences in the following chapter, the (un)relatedness of sequences has a much larger influence on the practicability of DCA than the number or length of the involved sequences.

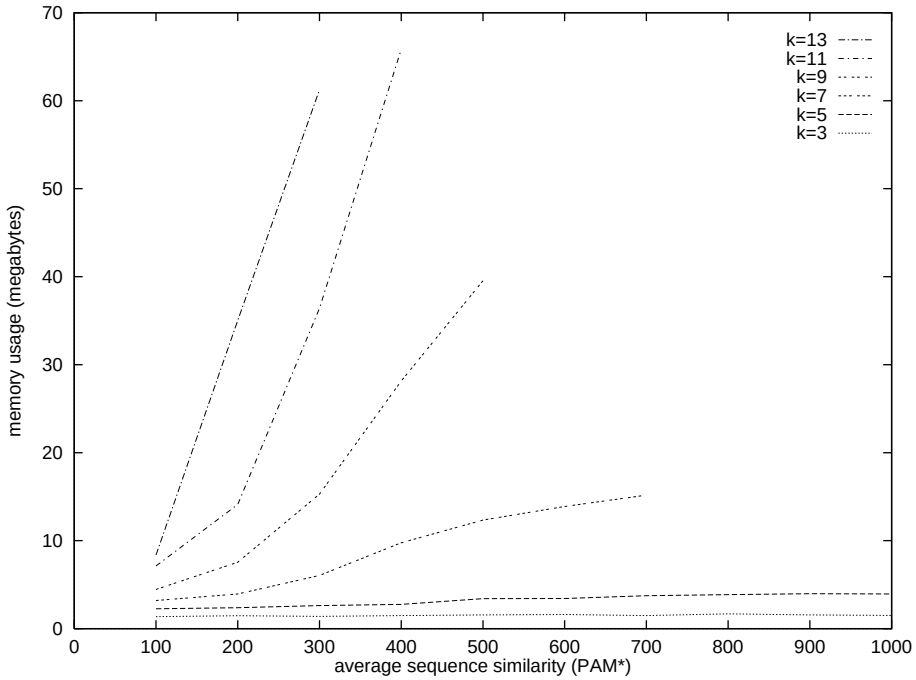

Figure 7.18: The memory usage of `DCA` for different sequence similarities.

Chapter 8

Results on Biological Sequences

8.1 Six Tyrosine Kinases

As a first example on biological data, we present an alignment of six tyrosine kinase protein sequences of length between 273 and 280 amino acids which were also aligned by Kececioglu using the *maximum weight trace* approach [114]. The sequences are rather similar such that even a (sum-of-pairs-) optimal alignment can be computed with `MSA` in 12.3 seconds using the PAM 250 score matrix in distance form with values between 0 and 24 and an affine gap function $g(l) = 8 + 12\,l$. However, with the recursion stop size set to $L = 40$, `DCA` computes an alignment which differs from the score-optimal one by only a single gap within 5.7 seconds.

This alignment is shown in Figure 8.1. The numbers above the alignment denote the slicing positions. The asterisks denote the region where the alignment differs from that computed by `MSA`. In Table 8.1, the scores of several alignments computed with `DCA` with different stop lengths L and window sizes W are listed. In parentheses, the number of alignment positions is shown where the alignments differ from the score-optimal one.

Note that, although from $L = 40$ to $L = 100$ the score improves, the number

	$W = 0$		$W = \frac{1}{4}L$		$W = \frac{1}{2}L$		$W = L$	
$L = 10$	61965	(22)	61965	(22)	62019	(30)	61963	(31)
$L = 20$	61917	(16)	61949	(19)	61917	(17)	61898	(9)
$L = 40$	61898	(9)	61898	(9)	61898	(9)	61898	(9)
$L = 100$	61883	(22)	61883	(22)	61883	(22)	61898	(9)
$L = 200$	61883	(22)	61883	(22)	61883	(22)	61883	(22)
$L = 300$	61880	(0)						

Table 8.1: Results of aligning a family of six tyrosine kinases with `DCA`. The alignment score and the number of positions where an alignment differs from the score-optimal one (in parentheses) are shown for different values of L and W.

```
                                    2  *********
-----GLAKDAWEIPRESLRLEAKLGQGCFGEVWMGTWND-TTRVAI-KTLKPG--TMSPEAFLQEAQ
-----GLAKDAWEIPRESLRLEVKLGQGCFGEVWMGTWNG-TTRVAI-KTLKLG--TMMPEAFLQEAQ
TIYGVSPNYDKWEMERTDITMKHKLGGGQYGEVYEGVWKKYSLTVAV-KTLKED--TMEVEEFLKEAA
-VLNRAVPKDKWVLNHEDLVLGEQIGRGNFGEVFSGRLRADNTLVAV-KSCRETLPPDIKAKFLQEAK
-VLTRAVLKDKWVLNHEDVLLGERIGRGNFGEVFSGRLRADNTPVAV-KSCRETLPPELKAKFLQEAR
-------SSYYWKMEASEVMLSTRIGSGSFGTVYKGKWHG-DVAVKILKVVDPT--PEQLQAFRNEVA

     1                                      2
VMKKLRHEKLVQLYAV-VSEEPIYIVIEYMSKGSLLDFLKGEMGKYLRLPQLVDMAAQIASGMAYVER
IMKKLRHDKLVPLYAV-VSEEPIYIVTEFMTKGSLLDFLKEGEGKFLKLPQLVDMAAQIADGMAYIER
VMKEIKHPNLVQLLGVCTREPPFYIITEFMTYGNLLDYLRECNRQEVSAVVLLYMATQISSAMEYLEK
ILKQYSHPNIVRLIGVCTQKQPIYIVMELVQGGDFLTFLRTEGAR-LRMKTLLQMVGDAAAGMEYLES
ILKQCNHPNIVRLIGVCTQKQPIYIVMELVQGGDFLSFLRSKGPR-LKMKKLIKMMENAAAGMEYLES
VLRKTRHVNILLFMGY-MTKDNLAIVTQWCEGSSLYKHLHVQETK-FQMFQLIDIARQTAQGMDYLHA

       0                                    2
MNYVHRDLRAANILVGENLVCKVADFGLARLIEDNEYTARQGAK-FPIKWTAPEAALY---GRFTIKS
MNYIHRDLRAANILVGDNLVCKIADFGLARLIEDNEYTARQGAK-FPIKWTAPEAALY---GRFTIKS
KNFIHRDLAARNCLVGENHLVKVADFGLSRLMTGDTYTAHAGAK-FPIKWTAPESLAY---NKFSIKS
KCCIHRDLAARNCLVTEKNVLKISDFGMSREAADGIYAASGGLRQVPVKWTAPEALNY---GRYSSES
KHCIHRDLAARNCLVTEKNTLKISDFGMSRQEEDGVYASTGGMKQIPVKWTAPEALNY---GWYSSES
KNIIHRDMKSNNIFLHEGLTVKIGDFGLATVKSRWSGSQQVEQPTGSVLWMAPEVIRMQDDNPFSFQS

            1                                      2
DVWSFGILLTELTTKGRVPYPGMVNR-EVLDQVERGYRMPCPP----ECPESLHDLMCQCWRKDPEER
DVWSFGILLTELVTKGRVPYPGMVNR-EVLEQVERGYRMPCPQ----GCPESLHELMKLCWKKDPDER
DVWAFGVLLWEIATYGMSPYPGIDLS-QVYELLEKDYRMERPE----GCPEKVYELMRACWQWNPSDR
DVWSFGILLWETFSLGASPYPNLSNQ-QTREFVEKGGRLPCPE----LCPDAVFRLMEQCWAYEPGQR
DVWSFGILLWEAFSLGAVPYANLSNQ-QTREAIEQGVRLEPPE----QCPEDVYRLMQRCWEYDPHRR
DVYSYGIVLYELMA-GELPYAHINNRDQIIFMVGRGYASPDLSRLYKNCPKAIKRLVADCVKKVKEER


PTFKYLQAQLLPACVLEVAE-
PTFEYIQSFLEDYFTAAEPSG
PSFAEIHQAFETMFQESSIS-
PSFSAIYQELQSIRKRHR---
PSFGAVHQDLIAIRKRHR---
PLFPQILSSIELLQHSLPKIN
```

Figure 8.1: Alignment of six tyrosine kinases computed with `DCA` ($L = 40$) using the PAM 250 substitution matrix. The region where this alignment differs from the score-optimal one computed with `MSA` is marked with asterisks above the alignment. The numbers denote slicing positions.

of incorrectly aligned positions increases. In contrast to the measurements of the previous chapter, the windowing approach more often worsens the alignment rather than improving it. Of course, this is caused by the incorrect handling of affine gap costs at window bounds.

8.2 Four Famous Benchmark Problems

In 1994, McClure *et al.* [135] applied a wide variety of multiple alignment programs to four different sequence families containing twelve globin, kinase, aspartic acid protease, and ribonuclease H (RH) proteins, respectively. They also defined subfamilies containing the six and ten sequences of each protein family with the widest distance distribution of sequence relationship. While the globins and the kinases are rather similar and hence the computation of reasonable alignments of these sequences is not difficult, the protease and RH sequences are much more diverse. Here most procedures in the study of McClure *et al.* did not perform very well, and some of the programs even could not at all align these sequences. Especially the fragment-based method ASSEMBLE [208] – which produces excellent alignments of the globins and the kinases – had enormous problems to detect reliable anchor subsequences in the protease sequences and the RH proteins.

The output of the alignment programs was scored by the following procedure: From structurally verified alignments of the test families, highly conserved regions – so-called *motifs* – of three to nine amino acids and some single completely conserved residues were extracted: five motifs in the globin family, eight in the kinase, three in the protease, and four motifs in the RH family. Then – individually for each motif – the percentage of the number of sequences in each data set for which the motif is correctly identified (i.e. *all* positions of the motif coincide) is measured. If a motif is aligned correctly in more than one subfamily of the sequences without aligning these motifs to one another, the total percent correct match is a combined score of the aligned subfamilies.

Also a condensed way of presenting the results has been used [90]: The scores of all individual motifs are added, and the sum is divided by 100. When motifs are spread over more than one subfamily of the aligned sequences, we mark this by an asterisk before the score. Thus, a single number gives an impression of the quality of an alignment. However, this method does not allow to determine the individual scores of the distinct motifs.

Table 8.2 shows both the detailed and the condensed scores of alignments computed with `DCA` using default paramters (PAM 250 substitution matrix with affine gap costs, stop size $L = 40$) as well as the corresponding running times of our program. While the globins and the kinases are fairly well aligned in rather a short time, the alignments of the proteases and of the RH sequences are of lower accuracy and their computation takes comparatively long time.

sequences	length	percent correctly aligned motifs	total	max.	CPU time
Globins 6	141-153	100 83 100 100 100	4.83	5.00	2.2 sec
Globins 10	141-153	100 90 100 100 100	4.90	5.00	7.9 sec
Globins 12	141-153	100 92 100 100 100	4.92	5.00	14.3 sec
Kinases 6	255-339	100 100 100 100 100 100 100 83	7.83	8.00	14.1 sec
Kinases 10	255-339	100 80 100 100 100 100 100 90	7.70	8.00	36.3 min
Kinases 12	255-339	100 75 92 100 100 100 100 100	7.67	8.00	24.1 min
Proteases 6	108-150	100(67,33) 50 33	*1.83	3.00	1.6 min
Proteases 10	98-160	90 40(20,20) 80	*2.10	3.00	71.8 min
Proteases 12	98-160	83 58(25,17,17) 75(50,25)	*2.17	3.00	4.7 h
RH 6	126-157	100 50 67 83(50,33)	*3.00	4.00	4.2 sec
RH 10	126-157	90 80(60,20) 90(70,20) 100(30,30,40)	*3.60	4.00	20.9 min
RH 12	126-158	92 67(50,17) 75(50,25) 92(42,33,16)	*3.25	4.00	83.5 min

Table 8.2: The percentages of correctly aligned motifs and the computation time of DCA for the protein families from McClure *et al.* [135] using the PAM 250 substitution matrix and recursion stop size $L = 40$.

In the measurements carried out by McClure *et al.*, the parameters of the individual programs were chosen optimally for each single test family. We have also run our algorithm with different parameters. While other values of L or a windowing only have minor effects, significant improvements can be achieved by a change of the substitution matrix. Tables 8.4 ($L = 20$) and 8.5 ($L = 40$) compare the results obtained using the PAM 250 matrix with those obtained with the substitution matrix of 160 PAM and for three different matrices from the BLOSUM series [99]. Some of the runs took rather a long time (more than 500 hours of CPU time) and have therefore been stopped. This is indicated by a question mark. Table 8.3 lists the ranges of the substitution matrices and the corresponding gap functions used in this study.

In general, we have observed that substitution matrices from the BLOSUM series produce on these data slightly better results than the corresponding PAM matrices. (Due to Henikoff and Henikoff [99], the PAM 250 matrix is comparable to BLOSUM 45, and PAM 160 is comparable to BLOSUM 62.) This result is in accordance with Henikoff and Henikoff [100] who also observed that the BLOSUM matrices perform better for distantly related proteins. It can be explained by recalling the different derivations of the matrices: The BLOSUM matrices are derived from the BLOCKS database [99] of highly conserved alignment regions, so they might be better suited for the detection of conserved motifs than the PAM matrices which are derived from an alignment database containing also less conserved protein regions.

The globins are very quickly aligned by DCA. Also for the kinases (which are comparatively long protein sequences) and for the smaller families of the kinase and

substitution matrix	min. (lowest distance)	max. (highest distance)	gap function
PAM 250	0	25	$g(l) = 8 + 12\,l$
PAM 160	0	29	$g(l) = 8 + 12\,l$
BLOSUM 62	0	15	$g(l) = 6 + 10\,l$
BLOSUM 45	0	20	$g(l) = 10 + 9\,l$
BLOSUM 30	0	27	$g(l) = 10 + 11\,l$

Table 8.3: The substitution matrices and corresponding gap functions used in this study.

RH sequences, DCA is relatively fast. However, the families of twelve RH sequences as well as those with ten and twelve protease sequences require extensively more time. As emphasized above, not the length or the number of sequences is the most relevant factor concerning the computation time of DCA but the relatedness. By closer observation, we have found that – despite all the efforts we have made to speeding up this stage of our algorithm – it is still the search for C-optimal families of slicing positions taking so much time.

One also observes an influence of the score function on the computation time: Some alignments with the PAM 160 matrix take more than 200 times as long as the corresponding runs with the BLOSUM 62 matrix. As noted in Section 5.1, this is by reason of the high dependence of the size of the relevant search space also on substitution matrix and gap function.

We have also computed alignments of these sequence families using approximate slicing positions. Table 8.6 shows the results for $L = 20$. Compared to Table 8.4, the running times are reduced enormously. Each of the sequence families can be aligned within some seconds up to maximally eight minutes. Where the computation of C-optimal slicing positions takes extremely long (e.g. for the family of ten protease sequences), a speed-up factor of more than 5 000 is achieved. Accompanied with this speed-up is only a low decrease of alignment accuracy. Often, the same number of motifs are aligned correctly. Occasionally, the score even increases (e.g. for the RH sequences with the PAM 250 substitution matrix).

Due to the enormous speed-up and the still high quality of the alignments, working on a refinement of the definition of approximate slicing positions seems to be rewarding.

In Table 8.7 we compare the best alignments obtained with DCA to the results of the alignment programs DFALIGN [66] and AMULT [20, 19] which were the best and second best scoring programs in the study of McClure *et al.* Both DFALIGN and AMULT are implementations of the progressive sequence alignment approach. McClure *et al.* do not give the running times of the methods they tested. So, we can only compare the quality of the alignments. DCA outperforms AMULT in all

sequences	max.	PAM 250		PAM 160		Blosum 62		Blosum 45		Blosum 30	
Globins 6	5.00	4.83	2.8 sec	4.83	3.1 sec	5.00	2.4 sec	4.83	2.8 sec	4.67	4.8 sec
Globins 10	5.00	4.90	9.4 sec	4.90	10.5 sec	5.00	7.5 sec	4.90	9.3 sec	*4.90	12.2 sec
Globins 12	5.00	4.92	17.5 sec	4.92	18.0 sec	5.00	12.1 sec	4.92	14.6 sec	4.83	20.8 sec
Kinases 6	8.00	7.92	17.0 sec	*8.00	13.4 sec	7.83	13.9 sec	8.00	11.6 sec	8.00	13.3 sec
Kinases 10	8.00	7.60	45.8 min	7.80	2.5 h	7.80	10.7 min	7.90	2.1 h	8.00	3.5 h
Kinases 12	8.00	7.83	25.6 min	8.00	1.6 h	7.92	8.2 min	7.92	52.8 min	8.00	46.8 min
Proteases 6	3.00	*1.83	6.6 sec	1.33	20.9 sec	1.00	4.3 sec	1.66	15.3 sec	2.17	37.7 sec
Proteases 10	3.00	*2.00	87.5 min	*2.40	2.5 h	*2.20	38.3 min	*2.40	4.3 h	*2.20	2.3 h
Proteases 12	3.00	*2.25	5.7 h		?	*2.00	78.1 h		?		?
RH 6	4.00	2.83	4.4 sec	*3.83	5.3 sec	3.00	3.0 sec	3.33	4.9 sec	3.67	6.8 sec
RH 10	4.00	*3.50	20.1 min	*3.50	66.8 min	*3.30	4.5 min	*3.70	80.1 min	3.60	16.2 h
RH 12	4.00	*2.75	1.7 h		?	*3.42	6.5 h		?		?

Table 8.4: Score and running time of DCA with $L = 20$ using different amino acid substitution matrices.

sequences	max.	PAM 250		PAM 160		Blosum 62		Blosum 45		Blosum 30	
Globins 6	5.00	4.83	2.2 sec	4.67	2.6 sec	4.83	2.9 sec	4.83	3.5 sec	4.67	3.0 sec
Globins 10	5.00	4.90	7.9 sec	5.00	38.9 sec	5.00	12.3 sec	4.90	11.4 sec	*4.90	10.0 sec
Globins 12	5.00	4.92	14.3 sec	4.83	14.5 sec	5.00	15.0 sec	4.92	17.8 sec	4.83	17.4 sec
Kinases 6	8.00	7.83	14.1 sec	8.00	14.2 sec	7.83	14.8 sec	7.83	14.6 sec	7.67	11.6 sec
Kinases 10	8.00	7.70	36.3 min	7.90	3.3 h	*7.70	13.0 min	7.90	2.6 h	8.00	2.8 h
Kinases 12	8.00	7.67	24.1 min	7.92	110.7 min	7.92	10.4 min	7.92	89.8 min	8.00	51.8 min
Proteases 6	3.00	*1.83	1.6 min	1.33	45.4 min	1.33	11.9 sec	1.83	1.8 min	*2.50	23.2 sec
Proteases 10	3.00	*2.10	71.8 min	*2.30	4.8 h	1.90	46.2 min	*2.40	7.8 h	*2.50	15.7 h
Proteases 12	3.00	*2.17	4.7 h		?	*2.08	91.7 h		?		?
RH 6	4.00	*3.00	4.2 sec	3.33	11.1 sec	3.17	3.7 sec	*3.00	10.6 sec	3.67	12.4 sec
RH 10	4.00	*3.60	20.9 min	*3.50	4.1 h	*3.30	66.5 min	3.50	1.8 h	3.50	15.3 h
RH 12	4.00	*3.25	83.5 min		?	*3.42	55.2 h		?		?

Table 8.5: Score and running time of DCA with $L = 40$ using different amino acid substitution matrices.

sequences	max.	PAM 250		PAM 160		Blosum 62		Blosum 45		Blosum 30	
Globins 6	5.00	4.83	2.4 sec	4.83	2.5 sec	*4.83	2.3 sec	*4.67	3.1 sec	*4.67	4.0 sec
Globins 10	5.00	5.00	7.8 sec	4.80	7.9 sec	5.00	6.9 sec	4.90	10.0 sec	5.00	11.0 sec
Globins 12	5.00	5.00	12.0 sec	4.83	12.1 sec	5.00	12.8 sec	4.92	15.9 sec	4.83	18.6 sec
Kinases 6	8.00	7.50	12.5 sec	7.50	9.7 sec	7.67	12.3 sec	7.50	11.8 sec	7.33	11.4 sec
Kinases 10	8.00	7.40	44.1 sec	7.50	45.6 sec	7.80	43.4 sec	7.90	48.7 sec	8.00	59.2 sec
Kinases 12	8.00	8.00	65.4 sec	*7.83	68.4 sec	7.83	63.9 sec	8.00	75.2 sec	8.00	78.2 sec
Proteases 6	3.00	1.50	3.1 sec	1.33	4.4 sec	0.67	2.6 sec	1.83	4.0 sec	*2.00	4.7 sec
Proteases 10	3.00	*1.90	22.8 sec	*2.20	10.4 sec	*1.90	9.2 sec	*2.40	10.4 sec	*2.50	45.4 sec
Proteases 12	3.00	*2.17	13.7 sec	*2.25	187.4 sec	*1.92	13.8 sec	*2.42	19.2 sec	*2.50	479.3 sec
RH 6	4.00	3.33	5.1 sec	3.50	3.6 sec	2.83	2.5 sec	2.67	3.0 sec	2.50	3.0 sec
RH 10	4.00	3.60	13.4 sec	3.40	13.1 sec	*3.50	11.0 sec	*3.60	11.7 sec	*3.10	11.0 sec
RH 12	4.00	*3.25	16.6 sec	*3.33	19.5 sec	*3.00	16.8 sec	3.25	20.3 sec	3.42	20.0 sec

Table 8.6: Score and running time of DCA with $L = 20$ when approximate slicing positions computed with *calc-Chat-itSCRmin* are used.

sequences	max.	DFALIGN	AMULT	DCA	
Globins 6	5.00	5.00	5.00	5.00	(BLOSUM 62, $L = 20$)
Globins 10	5.00	5.00	5.00	5.00	(e.g. BLOSUM 62, $L = 20$)
Globins 12	5.00	5.00	5.00	5.00	(e.g. BLOSUM 62, $L = 20$)
Kinases 6	8.00	7.67	7.33	8.00	(e.g. BLOSUM 30, $L = 20$)
Kinases 10	8.00	8.00	7.70	8.00	(e.g. BLOSUM 30, $L = 20$)
Kinases 12	8.00	8.00	7.75	8.00	(e.g. BLOSUM 30, $L = 20$)
Proteases 6	3.00	2.33	1.17	*2.50	(BLOSUM 30, $L = 40$)
Proteases 10	3.00	*3.00	*2.40	*2.50	(e.g. BLOSUM 30, $L = 40$)
Proteases 12	3.00	*3.00	2.33	*2.50	(BLOSUM 30, $L = 20$, approx.)
RH 6	4.00	3.67	*3.30	*3.83	(PAM 160, $L = 20$)
RH 10	4.00	3.30	*3.20	*3.70	(BLOSUM 45, $L = 20$)
RH 12	4.00	3.83	*2.92	3.42	(BLOSUM 30, $L = 20$, approx.)

Table 8.7: Numbers of correctly aligned motifs in alignments computed with the programs DFALIGN and AMULT compared to the highest scoring alignments computed with DCA.

cases and produces results comparable to those of DFALIGN. For four families, DCA computes alignments scoring higher than any of the programs evaluated in the study of McClure *et al.* [135]. This proves that – when the score function is selected carefully – the divide-and-conquer alignment method can compete with the best alignment programs currently available.

Of course, we wondered why for some sequence families the results obtained with DCA are still a bit different from the biologically correct alignments despite the proximity of our alignments to the sum-of-pairs optimal ones. And of course, the answer is that our alignment can hardly be better than the score function we approximate. Consequently, we have compared the score of alignments computed with DCA to that of the biologically correct, "true" alignments as published in [135]. The result of this comparison is presented in Tables 8.8 and 8.9. For the example of the PAM 250 score, Table 8.8 explains how we compute the *relative difference* of the score of the correct alignment from the score of the DCA-alignment. Table 8.9 shows the relative differences for all the examined sequence families and substitution matrices.

In all cases, the score of the alignment computed with DCA is lower than that of the corresponding true alignment. However, for the globins and the kinases – where we detected almost all motifs correctly – both scores differ much less than for the proteases and the RH proteins. It also can be observed that the subfamilies of six sequences are much harder to be aligned than the larger families which is in accordance with our results shown in Tables 8.4 – 8.6. Assuming that an alignment computed with DCA differs by less than one percent from the optimal score, this proves that the studied alignment score functions – even if we could compute a sum-of-pairs

sequences	true	DCA	difference	rel. difference
Globins 6	37 054	36 834	220	0.60 %
Globins 10	108 460	108 093	367	0.34 %
Globins 12	156 074	155 657	417	0.27 %
Kinases 6	73 685	71 249	2436	3.42 %
Kinases 10	217 760	214 661	3099	1.44 %
Kinases 12	314 288	308 662	5626	1.82 %
Proteases 6	36 089	34 138	1951	5.71 %
Proteases 10	107 085	103 972	3113	2.99 %
Proteases 12	156 051	151 663	4388	2.89 %
RH 6	40 334	37 596	2738	7.28 %
RH 10	118 720	112 129	6591	5.88 %
RH 12	178 069	168 600	9469	5.62 %

Table 8.8: Comparison of the absolute PAM 250-scores of the true alignments and of those computed with DCA ($L = 20$). The relative difference is the absolute difference divided by the score of the DCA-alignment.

sequences	PAM 250	PAM 160	Blosum 62	Blosum 45	Blosum 30
Globins 6	0.60 %	0.75 %	0.93 %	0.61 %	0.45 %
Globins 10	0.34 %	0.45 %	0.68 %	0.38 %	0.08 %
Globins 12	0.27 %	0.24 %	0.60 %	0.11 %	0.33 %
Kinases 6	3.42 %	3.30 %	4.63 %	2.90 %	1.99 %
Kinases 10	1.44 %	0.72 %	3.44 %	1.15 %	0.16 %
Kinases 12	1.82 %	0.97 %	3.70 %	1.81 %	0.74 %
Proteases 6	5.71 %	4.76 %	8.78 %	3.44 %	2.41 %
Proteases 10	2.99 %	1.19 %	5.04 %	0.88 %	0.24 %
Proteases 12	2.89 %	?	3.82 %	?	?
RH 6	7.28 %	5.58 %	10.69 %	6.66 %	4.54 %
RH 10	5.88 %	3.45 %	8.11 %	3.91 %	3.18 %
RH 12	5.62 %	?	8.78 %	?	?

Table 8.9: Relative difference of scores of the true alignments and of those computed with DCA ($L = 20$) for the different amino acid substitution matrices.

sequences	# (length)	max.	MSA (PAM250)		DCA (PAM250)		DCA (BLOSUM)	
Globins A	7 (141-153)	5.00	4.86	157 sec	4.86	4.3 sec	5.00	5.3 sec
Globins B	10 (141-153)	5.00	5.00	130 sec	4.90	10.4 sec	5.00	10.9 sec
Kinases A	5 (255-293)	8.00	8.00	10 min	8.00	10.6 sec	8.00	16.8 sec
Kinases B	6 (255-293)	8.00	8.00	118 min	8.00	9.7 sec	8.00	57.5 sec
Kinases C	4 (255-339)	8.00	6.75	210 sec	*7.50	4.6 sec	7.25	4.8 sec
Proteases A	5 (98-150)	3.00	2.80	37 sec	2.40	2.8 sec	2.80	18.3 sec
Proteases B	4 (113-150)	3.00	0.50	9 min	0.00	1.4 sec	1.00	3.3 sec
RH A	5 (126-157)	4.00	2.60	68 min	*2.60	3.4 sec	3.40	29.0 sec

Table 8.10: Running time and percent correctly aligned motifs in alignments computed with MSA (using the PAM 250 substitution matrix) and the corresponding values of DCA (PAM 250 and the best-scoring matrix from the BLOSUM series).

optimal alignment – will not allow to compute a biologically correct alignment of the RH sequences, for example. Further work on the development of better alignment score functions will be necessary.

Similar to the results shown above, this comparison of alignment scores shows that the alignments computed with the BLOSUM matrices mostly are nearer to the true alignments than those computed with the matrices from the PAM series. With this study, we have shown that due to its speed and the high accuracy of the results, DCA makes it possible to analyze directly the properties of *multiple* alignment score functions.

8.3 Comparison with MSA

Gupta *et al.* [90] applied the improved version 2.0 of MSA to the same sequences as were used by the comparison of McClure *et al.* described above. Because they still could not align the full data sets, they selected some subfamilies (denoted by the letters A, B, C) which MSA was able to align SP-optimally (with regard to PAM 250 and gap function $g(l) = 8 + 12\,l$). In Table 8.10, we report their results compared to the results of DCA on the same subfamilies. The speed-up factor of DCA over MSA ranges from 15 to 1 500. The memory usage of DCA lies by a factor of two to twenty below that of MSA.

One observes that our alignments with the same substitution matrix often find the same number of motifs as those computed with MSA. In four cases, there are less, in one case even more motifs are aligned correctly. Again, with matrices from the BLOSUM series, the results can be improved: For all sequence families, DCA can compute alignments which score higher than or equal to the SP-optimal one regarding PAM 250 score. This, again, supports our assertion that the alignment score function influences the alignment quality (in biological terms) much more than the remaining

sequences	max.	PAM250		BLOSUM	
		cyclic score	sum-of-pairs	cyclic score	sum-of-pairs
Globins A	5.00	*4.86	4.86	*5.00	5.00
Globins B	5.00	4.80	4.90	*4.80	5.00
Kinases A	8.00	8.00	8.00	8.00	8.00
Kinases B	8.00	8.00	8.00	8.00	8.00
Kinases C	8.00	7.50	*7.50	7.00	7.25
Proteases A	3.00	*3.00	2.40	2.40	2.80
Proteases B	3.00	0.50	0.00	1.50	1.00
RH A	4.00	1.80	*2.60	3.40	3.40

Table 8.11: Numbers of correctly aligned motifs in alignments computed with regard to the cyclic alignment score and the sum-of-pairs score.

difference of less than one percent between an alignment computed with our method and an SP-optimal one.

For these sequence families, we also have computed alignments with regard to the cyclic alignment score mentioned in Section 2.3.3: Based on the pairwise distances of the sequences, a shortest cycle is computed and the weight factors of `DCA` are set accordingly. Unfortunately, due to the impossibility of feeding `MSA` with user-defined sequence weights, only the slicing positions are computed using the cyclic score while the alignments of the remaining subsequences are optimized with regard to the (unweighted) sum-of-pairs score. In Table 8.11, we compare the percentage of correctly aligned motifs of both approaches for both the PAM 250 substitution matrix and the best-scoring matrices from the BLOSUM series. While the sum-of-pairs alignment in general seems to score slightly better, in three difficult cases the slicing positions computed with regard to the cyclic score yield better alignments. Hence, the cyclic score seems to be an interesting alternative which allows a much faster computation of multiple alignments (when implemented efficiently).

8.4 Alignment and Phylogeny of RNase MRP RNA Sequences

We have also compared alignments computed with `DCA` to those computed with iterative methods in order to find out if the simultaneity of our approach can yield advantages. We aligned a family of highly diverged RNase MRP RNA sequences. The eight currently known sequences[1] are from human, animals (bovine, mouse,

[1]There are actually two more tobacco sequences. Because they are almost identical to the Arabidopsis sequence we have excluded them here.

	organism	GenBank acc. no.	length
Human	Homo sapiens	X51867	264
Bovine	Bos taurus	Z25280	278
Mouse	Mus musculus	J03151	274
Rat	Rattus norvegicus	(see caption)	273
Frog	Xenopus laevis	Z11844	276
Yeast1	Saccharomyces cerevisiae	Z14231	339
Yeast2	Schizosaccharomyces pombe	X52530	399
Plant	Arabidopsis thaliana	(see caption)	261

Table 8.12: The eight currently known RNase MRP RNA sequences used for demonstrating the quality of alignments computed with DCA. The Arabidopsis sequence and the Rat sequence are not contained in GenBank. They were obtained from [172] and [117], respectively.

---

rat, frog), yeasts (Saccharomyces cerevisiae, Schizosaccharomyces pombe) and plant (Arabidopsis) (see Table 8.12). In the following, we will refer to Saccharomyces cerevisiae and Schizosaccharomyces pombe as yeast1 and yeast2, respectively.

There are three single-stranded regions common to all molecules which are responsible for the tertiary structure and which are crucial for the functionality of the molecules. They are structurally conserved and of slightly different lengths. In the DCA alignment shown in Figure 8.2, we have highlighted them by capital letters.

Note that all regions (except the second and third of yeast2) are reasonably well aligned. The reason why yeast2 appears to be difficult to align is that this sequence is significantly longer than the other sequences (in fact it contains an additional subsequence of length 60 between the first and the second single-stranded region) and the second and third motifs of yeast2 differ significantly from the according motifs of the other sequences. The gaps in the second motif of all sequences other than yeast correspond to an (unusual) loop in the secondary structure of the yeast sequence, unique for the RNase MRP RNA sequences found so far.

None of the commonly used programs we have tested on this data set (e.g. ClustalW [201], TreeAlign [98], Malign [227]) is able to align this set with respect to the motifs as well as DCA does. Neither they predict the loop in the second motif.

In order to analyze why progressive alignment procedures fail to produce a reasonable alignment of this data set, we have investigated the phylogenetic relationships between these species. In particular, we have looked at the potential trees for guiding a successive alignment. In Figure 8.3, we show the graph generated by the SplitsTree program [54] for the set of Hamming Distances derived from the alignment of each pair of sequences. This program does not always return (phylogenetic) *trees*, but more general *graphs* indicating conflicting information in the data which do not fit into just a tree.

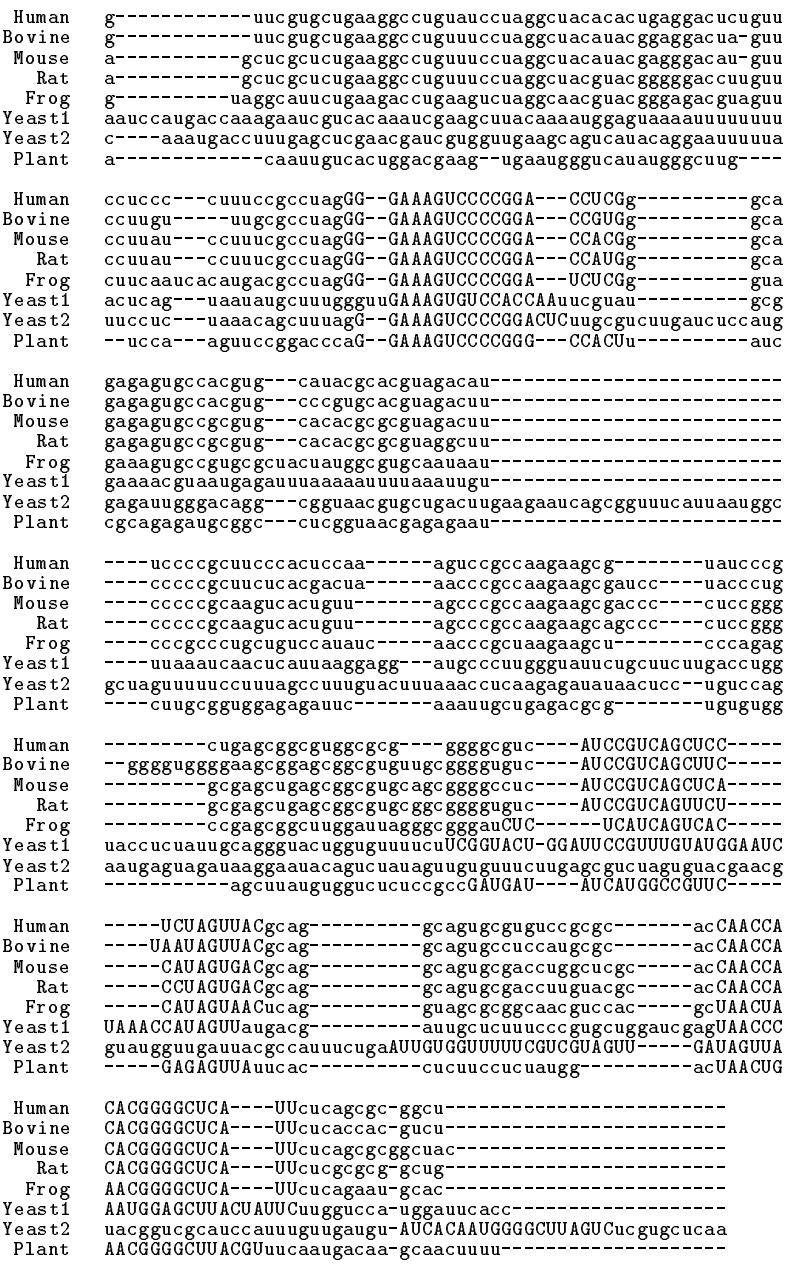

Figure 8.2: The alignment of eight RNase MRP RNA sequences, computed with `DCA` using a weighted transition/transversion matrix for substitutions and a linear gap penalty function of $g(l) = 6 + 2\,l$.

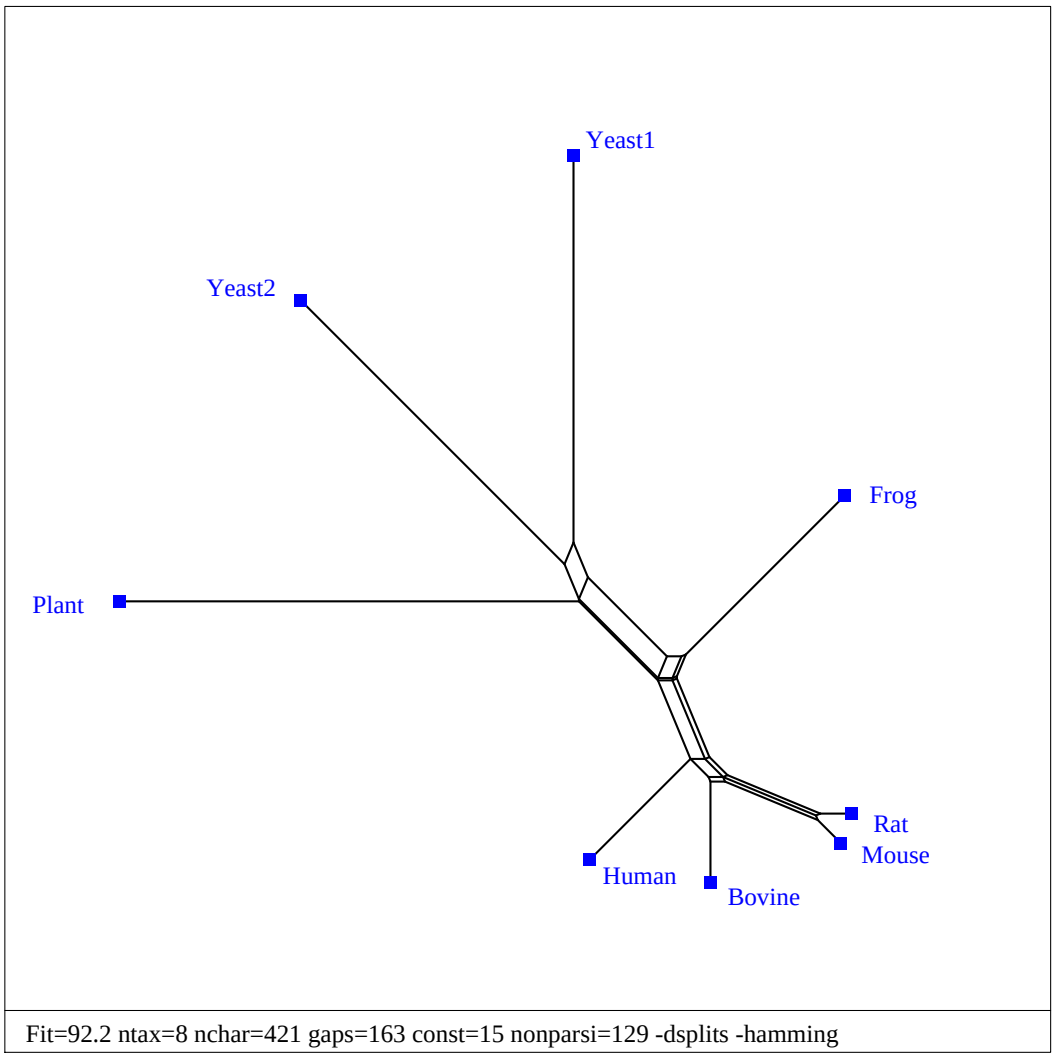

Figure 8.3: The SplitsTree graph for the pairwise Hamming Distances of eight RNase MRP RNA sequences.

Obviously, any reasonable guide tree puts the two yeasts and the Arabidopsis sequence close together. But any successive alignment of these three sequences is incorrect for the second and third motif due to the fact that pre-aligning any pair of these sequences goes wrong in this respect. Moreover, a simultaneous alignment of only these three sequences does neither identify motif 2 nor motif 3, even between any pair. But including the animal (and human) sequences, a simultaneous alignment identifies these motifs, at least for yeast1 and the plant sequence (see Figure 8.2).

This example shows the weak point of the progressive alignment method: *Once a gap, always a gap* [66]. Long indels in similar sequences, introduced in an early stage of the procedure, can disturb the complete multiple alignment. Common structures which are only detected as significant when all sequences are observed simultaneously can be suppressed.

8.5 How Many Sequences?

Finally, we have intended to find out how many (similar) sequences we could maximally align simultaneously with the DCA method. Therefore we have selected a family of several hundred sequences which is often used to obtain the evolutionary relationship of different species, the cytochrome C. Cytochrome C is a small, very old protein which plays an important role in the metabolic reactions in mitochondria. The amino acid sequence of cytochrome C is comparatively easy to obtain, so it was an early object of study for molecular evolutionists.

Due to the high similarity of the cytochrome C molecules, the correct alignment is no great challenge. It is easily obtained by hand and also most automatic methods are quite sucessful producing almost always the same alignment. However, for theoretical reasons, we wanted to find out where `DCA` reaches its limits. Unfortunately, the limits of `DCA` are highly correlated with the limits of `MSA` which – although we have even examined the source code – could not be persuaded into aligning more than forty-two sequences even if they were identical.

An alignment of forty cytochrome C sequences which was computed with `DCA` in less than a minute is shown in Figure 8.4. The sequences have length between 99 and 113 amino acids. In fact, the computation of slicing positions was possible even for more than sixty-five sequences in about two minutes. Table 8.13 shows the time and memory usage of `DCA` for different numbers of cytochrome C sequences. At first glance, the speed of `DCA` for these sequences seems to be in sharp conflict with the measurements of the previous chapter. However, due to the very high similarity, nearly all relevant parts of the matrices consist of one single cell only such that the search for a C-optimal family of slicing positions is trivial. This is supported by the mainly quadratical increase of running time for growing k.

This example shows that the memory usage and not the running time can be the limiting factor also for `DCA`: We stopped the run with $k = 70$ because of a drastic

	CPU time	memory usage
$k = 5$	1.11 sec	0.52 MB
$k = 10$	3.29 sec	2.06 MB
$k = 15$	6.79 sec	3.26 MB
$k = 20$	11.95 sec	5.03 MB
$k = 25$	18.92 sec	7.40 MB
$k = 30$	27.97 sec	9.99 MB
$k = 35$	41.23 sec	13.10 MB
$k = 40$	55.57 sec	16.22 MB
$k = 45$	(45.91 sec)	(11.40 MB)
$k = 50$	(58.57 sec)	(14.29 MB)
$k = 55$	(84.00 sec)	(26.43 MB)
$k = 60$	(103.01 sec)	(31.38 MB)
$k = 65$	(122.87 sec)	(36.17 MB)

Table 8.13: Running time and memory usage of `DCA` for different numbers k of cytochrome C sequences. The values in parentheses for $k > 40$ denote that `MSA` could not compute alignments of the subsequences in these cases, and the values are for the computation of the slicing positions, only.

---

swapping-out of computer memory and not because of too large computation time. In fact, the CPU times shown in Table 8.13 are somewhat misleading. The total (wallclock) time for the run with $k = 65$ was more than ten minutes on the computer with 32 megabytes of main memory we used.

Conclusively, it can be said that simultaneous alignment methods are not necessarily restricted to less than, say, twenty sequences. For the combination with `DCA`, alignment methods which – other than `MSA` – are specialized to the optimal alignment of several short sequences would allow a significant step towards the simultaneous alignment of many (sufficiently similar) sequences.

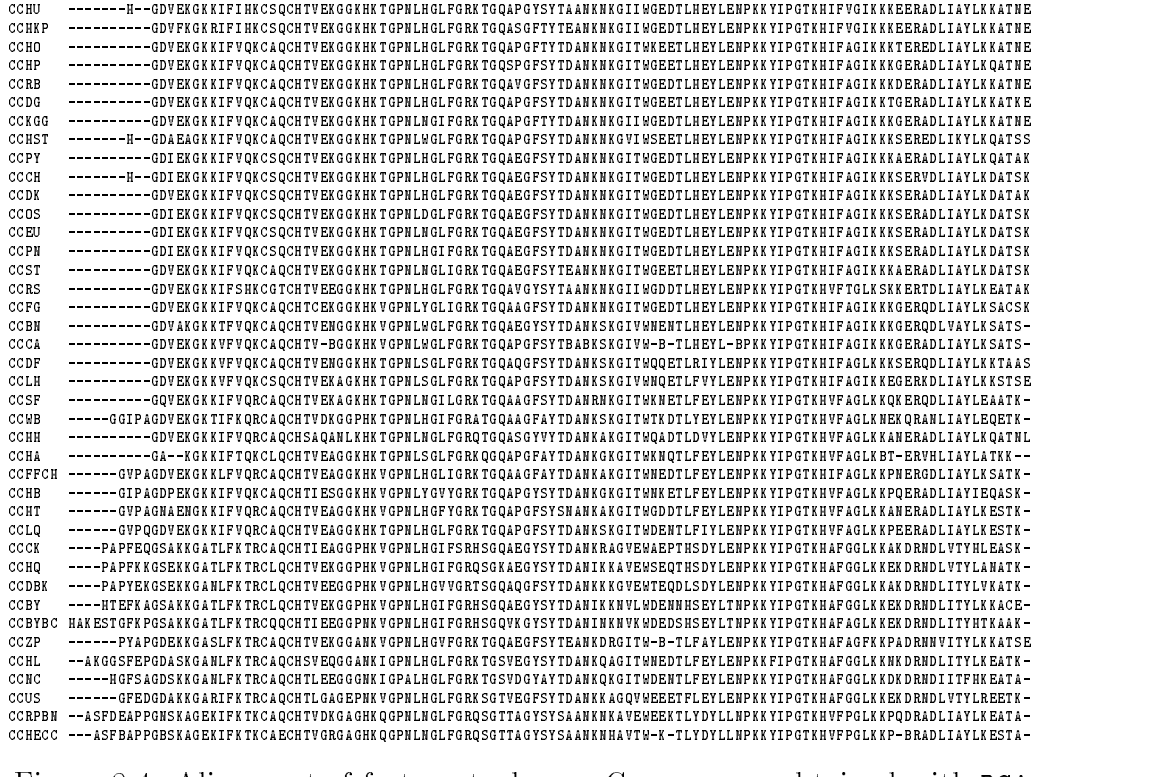

Figure 8.4: Alignment of forty cytochrome C sequences obtained with `DCA`.

Chapter 9

Conclusion

In this thesis we have developed a new algorithm for the approximate solution of the multiple sequence alignment problem, a well studied but still not satisfactorily solved problem in string processing, whose most important application is in computational molecular biology. We have given a formal description of the problem and discussed the standard solution and its high computational complexity. We have reviewed prior work on approximate solutions of the problem. We have studied, implemented, and improved a new principle for deriving approximative solutions of the multiple sequence alignment problem. Finally, we have extensively tested the behavior of the new algorithm with random sequences as well as with real biological data.

The main benefit of our divide-and-conquer algorithm is its simplicity. We have shown that even the most simple version allows a very fast computation of highly accurate alignments for three to, say, six sequences of relevant length. Due to its simplicity, the algorithm is highly suitable for being incorporated into larger systems which require a large number of reliable, but not necessarily optimal multiple sequence alignments. An example has already been implemented successfully.

Using branch-and-bound strategies for speeding up the basic algorithm, we also were able to develop a program for independent sequence alignment: `DCA`. The simultaneous alignment of more than a dozen of related sequences is possible providing the following advantages:

- The memory usage of `DCA` is in the magnitude required for pairwise alignments (about 30 megabytes for twelve sequences of average length 250).

- Compared to previous simultaneous alignment methods, our program is very fast (about 40 seconds for twelve sequences of average length 250).

- The alignments are of a very high quality, in mathematical as well as in biological terms. For none of the analyzed random sequence families did the sum-of-pairs score of the alignment computed with `DCA` differ by more than 0.3 percent from the optimal score. Applied to biological sequences, `DCA` can compete with the best alignment methods currently available.

- Due to the simultaneity, the computed alignments are also very well suited as an unbiased starting point for the reconstruction of evolutionary relationships.

- Because `DCA` approximates the optimal score very closely, new ways of testing and validating alternative choices for multiple alignment score functions are possible.

- Due to the stable interdependence of the parameters of `DCA` and its performance, the behavior of the program is transparent to the user.

- A state-of-the-art `html`-based user interface facilitates the use of the program via the World-Wide Web.

Our research has shown that – although the multiple sequence alignment problem has been a much studied subject over the last decades – the systematic application of the classical divide-and-conquer principle opens the way to a new, efficient, and effective simultaneous multiple sequence alignment algorithm.

In this thesis, not even the full potential of divide-and-conquer alignment could be worked out. For instance, further work on the following questions seems to be promising:

- Slicing positions which incorporate biological heuristics and are quickly computable might be an interesting alternative to C-optimal families of slicing positions.

- By a combination of DCA with a fragment-based alignment method, the development of an algorithm seems to be possible which is faster and produces higher accurate alignments than any of the separate approaches.

- Mainly for theoretical reasons, through the `DCA` program, certain principles such as the use of the same way of computing slicing positions regardless of the (sub)sequence length, for example, have been maintained. In special contexts, the practicability of the divide-and-conquer method might be improved by giving up some of these puristics.

- The development of a program for the optimal alignment of several short sequences, which is better than `MSA` suited for combinating with `DCA` would be valuable. This program should also allow a flexible handling of gaps at the sequence termini.

Beyond the evaluation of the divide-and-conquer algorithm, our studies on biological sequence data have raised some important aspects of the multiple sequence alignment problem in biology:

- The relatedness of the sequences has much more impact on computation time than the length and/or the number of sequences. While more than forty highly similar cytochrome C sequences can be aligned in few minutes, the alignment of ten less related protease sequences of about the same length takes more than an hour of computation time.

- Progressive alignment methods can be misled by long indels in otherwise highly similar sequences. In such cases, simultaneous alignments – like those computed by DCA – are to be preferred.

- With DCA we have reached a limit of what can be done with the sum-of-pairs model and the commonly used alignment score functions. For obtaining results which are still nearer to biologically correct alignments, it seems that more sophisticated score functions incorporating further biological criteria have to be considered.

We hope that our work has brought us a small step forward in applying computational methods to handle the data of life and that it may some day bring us closer to understanding life itself.

Bibliography

[1] P. Agarwal and D. J. States. A bayesian evolutionary distance for parametrically aligned sequences. *J. Comp. Biol.*, 3(1):1–17, 1996.

[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA, USA, 1974.

[3] B. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts, and J. D. Watson. *Molecular Biology of the Cell*. Garland Publishing, Inc., New York and London, 3rd edition, 1994.

[4] L. Allison. A fast algorithm for the optimal alignment of three strings. *J. theor. Biol.*, 164:261–269, 1993.

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

[6] S. F. Altschul. Leaf pairs and tree dissections. *SIAM J. Disc. Math.*, 2(3):293–299, 1989.

[7] S. F. Altschul. Amino acid substitution matrices from an information theoretic perspective. *J. Mol. Biol.*, 219:555–565, 1991.

[8] S. F. Altschul, R. J. Carroll, and D. J. Lipman. Weights for data related by a tree. *J. Mol. Biol.*, 207:647–653, 1989.

[9] S. F. Altschul and B. W. Erickson. Optimal sequence alignment using affine gap costs. *Bull. Math. Biol.*, 48(5/6):603–616, 1986.

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

[11] S. F. Altschul and D. J. Lipman. Trees, stars, and multiple biological sequence alignment. *SIAM J. Appl. Math.*, 49(1):197–209, 1989.

[12] P. Argos and M. Vingron. Sensitivity comparison of protein amino acid sequences. In R. F. Doolittle, editor, *Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences*, volume 183 of *Methods in Enzymology*, chapter 21, pages 352–365. Academic Press, Inc., San Diego, CA, USA, 1990.

[13] P. Argos, M. Vingron, and G. Vogt. Protein sequence comparison: Methods and significance. *Protein Engng.*, 4(4):375–383, 1991.

[14] R. Backofen and P. Clote. Evolution as a computational engine. Technical report, Institut für Informatik, Ludwig-Maximilians-Universität München, 1996.

[15] D. J. Bacon and W. F. Anderson. Multiple sequence alignment. *J. Mol. Biol.*, 191:153–161, 1986.

[16] V. Bafna, E. L. Lawler, and P. A. Pevzner. Approximation algorithms for multiple sequence alignment. In M. Crochemore and D. Gusfield, editors, *Combinatorial Pattern Matching, 5th Annual Symposium, CPM 94, Asilomar, CA, USA, June 5-8, 1994. Proceedings*, number 807 in Lecture Notes in Computer Science, pages 43–53, Berlin, 1994. Springer Verlag.

[17] W. Bains. MULTAN: A program to align multiple DNA sequences. *Nucl. Acids Res.*, 14(1):159–177, 1986.

[18] H.-J. Bandelt and A. W. M. Dress. Split decomposition: A new and useful approach to phylogenetic analysis of distance data. *Mol. Phyl. Evol.*, 1(3):242–252, 1992.

[19] G. J. Barton and M. J. E. Sternberg. Evaluation and improvements in the automatic alignment of protein sequences. *Protein Engng.*, 1(2):89–94, 1987.

[20] G. J. Barton and M. J. E. Sternberg. A strategy for the rapid multiple alignment of protein sequences – confidence levels from tertiary structure comparisons. *J. Mol. Biol.*, 198:327–337, 1987.

[21] D. Bashford, C. Chotia, and A. M. Lesk. Determinants of a protein fold – unique features of the globin amino acid sequences. *J. Mol. Biol.*, 196:199–216, 1987.

[22] L. H. Bell, J. R. Coggins, and E. J. Milner-White. Mix'n'match: An improved multiple sequence alignment procedure for distantly related proteins using secondary structure predictions, designed to be independent of the choice of gap penalty and scoring matrix. *Protein Engng.*, 6(7):683–690, 1993.

[23] S. A. Benner. Patterns of divergence in homologous proteins as indicators of tertiary and quaternary structure. *Adv. Enzyme Reg.*, 28:219–236, 1989.

[24] S. A. Benner, M. A. Cohen, and G. H. Gonnet. Amino acid substitution during functionally constrained divergent evolution of protein sequences. *Protein Engng.*, 7(11):1323–1332, 1994.

[25] D. Benson, D. J. Lipman, and J. Ostell. GenBank. *Nucl. Acids Res.*, 21(13):2963–2965, 1993.

[26] D. A. Benson, M. Boguski, D. J. Lipman, and J. Ostell. GenBank. *Nucl. Acids Res.*, 22(17):3441–3444, 1994.

[27] D. A. Benson, M. Boguski, D. J. Lipman, and J. Ostell. GenBank. *Nucl. Acids Res.*, 24(1):1–5, 1996.

[28] D. A. Benson, M. S. Boguski, D. J. Lipman, and J. Ostell. GenBank. *Nucl. Acids Res.*, 25(1):1–6, 1997.

[29] D. Benton. Recent changes in the GenBank on-line service. *Nucl. Acids Res.*, 18(6):1517–1520, 1990.

[30] M. P. Berger and P. J. Munson. A novel randomized iterative strategy for aligning multiple protein sequences. *CABIOS*, 7(4):479–484, 1991.

[31] K. Bergmann, A. Dress, J. Stoye, and M. Vingron. A practical approach to integrated alignment and phylogeny construction. In preparation.

[32] T. L. Blundell, B. L. Sibanda, M. J. E. Sternberg, and J. M. Thornton. Knowledge-based prediction of protein structures and the design of novel molecules. *Nature*, 326:347–352, 1987.

[33] D. R. Boswell and A. D. McLachlan. Sequence comparison by exponentially-damped alignment. *Nucl. Acids Res.*, 12(1):457–464, 1984.

[34] C. Branden and J. Tooze. *Introduction to Protein Structure*. Garland Publishing, Inc., New York, NY, USA, 1991.

[35] G. Brinkmann, A. W. M. Dress, S. W. Perrey, and J. Stoye. Two applications of the divide & conquer principle in the molecular sciences. In *Proc. of the International Symposium on Mathematical Programming, ISMP 97*, 1997. To appear.

[36] C. Burks, M. Cassidy, M. J. Cinkosky, K. E. Cumella, P. Gilna, J. E.-D. Hayden, G. M. Keen, T. A. Kelley, M. Kelly, D. Kristofferson, and J. Ryals. GenBank. *Nucl. Acids Res.*, 19(Suppl.):2221–2225, 1991.

[37] C. Burks, M. J. Cinkosky, W. M. Fischer, P. Gilna, J. E.-D. Hayden, G. M. Keen, M. Kelly, D. Kristofferson, and J. Lawrence. GenBank. *Nucl. Acids Res.*, 20(Suppl.):2065–2069, 1992.

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

[39] H. Carrillo and D. Lipman. The multiple sequence alignment problem in biology. *SIAM J. Appl. Math.*, 48(5):1073–1082, 1988.

[40] S. C. Chan, A. K. C. Wong, and D. K. Y. Chiu. A survey of multiple sequence comparison methods. *Bull. Math. Biol.*, 54(4):563–598, 1992.

[41] K.-M. Chao, R. C. Hardison, and W. Miller. Constrained sequence alignment. *Bull. Math. Biol.*, 55(3):503–524, 1993.

[42] C. Chappey, A. Danckaert, P. Dessen, and S. Hazout. MASH: An interactive program for multiple alignment and consensus sequence construction for biological sequences. *CABIOS*, 7(2):195–202, 1991.

[43] M. A. Charleston. Toward a characterization of landscapes of combinatorial optimization problems, with special attention to the phylogeny problem. *J. Comp. Biol.*, 2(3):439–450, 1995.

[44] V. Chvátal and D. Sankoff. Longest common subsequences of two random sequences. *J. Appl. Prob.*, 12:306–315, 1975.

[45] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT Press, Cambridge, MA, USA, 1989.

[46] F. Corpet. Multiple sequence alignment with hierarchical clustering. *Nucl. Acids Res.*, 16(22):10881–10890, 1988.

[47] J. Czelusniak, M. Goodman, N. D. Moncrief, and S. M. Kehoe. Maximum parsimony approach to construction of evolutionary trees from aligned homologous sequences. In R. F. Doolittle, editor, *Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences*, volume 183 of *Methods in Enzymology*, chapter 37, pages 601–615. Academic Press, Inc., San Diego, CA, USA, 1990.

[48] J. E. Darnell, H. Lodish, and D. Baltimore. *Molecular Cell Biology.* Scientific American Books, Inc., New York, NY, USA, 2nd edition, 1990.

[49] M. O. Dayhoff, W. C. Barker, and L. T. Hunt. Establishing homologies in protein sequences. In C. H. W. Hirs and S. N. Timasheff, editors, *Enzyme Structure*, volume 91 of *Methods in Enzymology*, chapter 47, pages 524–545. Academic Press, New York, NY, USA, 1983.

[50] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. A model of evolutionary change in proteins. In M. O. Dayhoff, editor, *Atlas of Protein Sequence and Structure*, volume 5, suppl. 3, pages 345–352. National Biomedical Research Foundation, Washington, D.C., 1979.

[51] A. Delcoigne and P. Hansen. Sequence comparison by dynamic programming. *Biometrika*, 62(3):661–664, 1975.

[52] E. Depiereux and E. Feytmans. Simultaneous and multivariate alignment of protein sequences: Correspondence between physicochemical profiles and structurally conserved regions (SCR). *Protein Engng.*, 4(6):603–613, 1991.

[53] R. F. Doolittle. Similar amino acid sequences: Chance or common ancestry? *Science*, 214:149–159, 1981.

[54] A. W. M. Dress, D. Huson, and V. Moulton. Analyzing and visualizing sequence and distance data using SplitsTree. *Discrete Applied Mathematics*, 71:95–110, 1996.

[55] A. W. M. Dress and A. von Haeseler, editors. *Trees and Hierarchical Structures. Proceedings of a Conference Held at Bielefeld, FRG, Oct. 5-9th, 1987.* Number 84 in Lecture Notes in Biomathematics. Springer Verlag, Berlin, 1990.

[56] S. R. Eddy and R. Durbin. RNA sequence analysis using covariance models. *Nucl. Acids Res.*, 22(11):2079–2088, 1994.

[57] M. Eigen, R. Winkler-Oswatitsch, and A. W. M. Dress. Statistical geometry in sequence space: A method of quantitative comparative sequence analysis. *Proc. Natl. Acad. Sci. USA*, 85:5913–5917, 1988.

[58] D. Eppstein, Z. Galil, R. Giancarlo, and G. F. Italiano. Sparse dynamic programming I: Linear cost functions. *Journal of the ACM*, 39(3):519–545, 1992.

[59] D. Eppstein, Z. Galil, R. Giancarlo, and G. F. Italiano. Sparse dynamic programming II: Convex and concave cost functions. *Journal of the ACM*, 39(3):546–567, 1992.

[60] B. W. Erickson and P. H. Sellers. Recognition of patterns in genetic sequences. In D. Sankoff and J. B. Kruskal, editors, *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, pages 55–91. Addison-Wesley, Reading, MA, USA, 1983.

[61] J. S. Farris. On the cophenetic correlation coefficient. *Syst. Zool.*, 18(3):279–285, 1969.

[62] J. S. Farris. Methods for computing Wagner trees. *Syst. Zool.*, 19(1):83–92, 1970.

[63] J. Felsenstein. Evolutionary trees from DNA sequences: A maximum likelihood approach. *J. Mol. Evol.*, 17:368–376, 1981.

[64] J. Felsenstein. A likelihood approach to character weighting and what it tells us about parsimony and compatibility. *Biol. J. Linn. Soc.*, 16:183–196, 1981.

[65] J. Felsenstein. Phylogenies and the comparative method. *Am. Nat.*, 125(1):1–15, 1985.

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

[67] D.-F. Feng, M. S. Johnson, and R. F. Doolittle. Aligning amino acid sequences: Comparison of commonly used methods. *J. Mol. Evol.*, 21:112–125, 1985.

[68] W. M. Fitch. An improved method of testing for evolutionary homology. *J. Mol. Biol.*, 16:9–16, 1966.

[69] W. M. Fitch. Further improvements in the method of testing for evolutionary homology among proteins. *J. Mol. Biol.*, 49:1–14, 1970.

[70] W. M. Fitch. Toward defining the course of evolution: Minimum change for a specific tree topology. *Syst. Zool.*, 20(4):406–416, 1971.

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

[72] W. M. Fitch and T. F. Smith. Optimal sequence alignments. *Proc. Natl. Acad. Sci. USA*, 80(5):1382–1386, 1983.

[73] M. L. Fredman. Algorithms for computing evolutionary similarity measures with length independent gap penalties. *Bull. Math. Biol.*, 46(4):553–566, 1984.

[74] Z. Galil and R. Giancarlo. Speeding up dynamic programming with applications to molecular biology. *Theor. Comput. Sci.*, 64:107–118, 1989.

[75] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco, CA, USA, 1979.

[76] L. Goldstein and M. S. Waterman. Mapping DNA by stochastic relaxation. *Adv. Appl. Math.*, 8:194–207, 1987.

[77] G. H. Gonnet. New heuristics for the protein multiple alignment problem. In *Function and Structure – Fundamental Concepts in Theoretical Biology*, 150. WE-Heraeus-Seminar at Physikzentrum Bad Honnef (Germany), 11 - 14 December 1995. WE-Heraeus-Stiftung.

[78] G. H. Gonnet, M. A. Cohen, and S. A. Benner. Exhaustive matching of the entire protein sequence database. *Science*, 256:1443–1445, 1992.

[79] A. D. Gordon. A sequence-comparison statistic and algorithm. *Biometrika*, 60(1):197–200, 1973.

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

[81] O. Gotoh. Alignment of three biological sequences with an efficient traceback procedure. *J. theor. Biol.*, 121:327–337, 1986.

[82] O. Gotoh. Optimal sequence alignment allowing for long gaps. *Bull. Math. Biol.*, 52(3):359–373, 1990.

[83] O. Gotoh. Optimal alignment between groups of sequences and its application to multiple sequence alignment. *CABIOS*, 9(3):361–370, 1993.

[84] O. Gotoh. Further improvement in methods of group-to-group sequence alignment with generalized profile operations. *CABIOS*, 10(4):379–387, 1994.

[85] O. Gotoh. A weighting system and algorithm for aligning many phylogenetically related sequences. *CABIOS*, 11(5):543–551, 1995.

[86] R. Grantham. Amino acid difference formula to help explain protein evolution. *Science*, 185:862–864, 1974.

[87] J. Greer. Comparative model-building of the mammalian serine proteases. *J. Mol. Biol.*, 153:1027–1042, 1981.

[88] J. Greer. Comparative modeling methods: Applications to the family of the mammalian serine proteases. *PROTEINS: Structure, Function, and Genetics*, 7:317–334, 1990.

[89] M. Gribskov, A. D. McLachlan, and D. Eisenberg. Profile analysis: Detection of distantly related proteins. *Proc. Natl. Acad. Sci. USA*, 84(13):4355–4358, 1987.

[90] S. K. Gupta, J. D. Kececioglu, and A. 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(3):459–472, 1995.

[91] D. Gusfield. Efficient methods for multiple sequence alignment with guaranteed error bounds. *Bull. Math. Biol.*, 55(1):141–154, 1993.

[92] J. E. Haber and D. E. Koshland Jr. An evaluation of the relatedness of proteins based on comparison of amino acid sequences. *J. Mol. Biol.*, 50:617–639, 1970.

[93] R. W. Hamming. Error detecting and error correcting codes. *Bell System Tech.*, 24(2):147–160, 1950.

[94] M. Hasegawa, H. Kishino, and T. Yano. Dating of the human-ape splitting by a molecular clock of mitochondrial DNA. *J. Mol. Evol.*, 22:160–174, 1985.

[95] J. Hein. A new method that simultaneously aligns and reconstructs ancestral sequences for any number of homologous sequences, when the phylogeny is given. *Mol. Biol. Evol.*, 6:649–668, 1989.

[96] J. Hein. A tree reconstruction method that is economical in the number of pairwise comparisons used. *Mol. Biol. Evol.*, 6:669–684, 1989.

[97] J. Hein. Unified approach to alignment and phylogenies. In R. F. Doolittle, editor, *Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences*, volume 183 of *Methods in Enzymology*, chapter 39, pages 626–645. Academic Press, Inc., San Diego, CA, USA, 1990.

[98] J. Hein. TreeAlign. In A. M. Griffin and H. G. Griffin, editors, *Computer Analysis of Sequence Data, Part II*, volume 25 of *Methods in Molecular Biology*, chapter 28, pages 349–364. Humana Press, Inc., Totowa, NJ, USA, 1994.

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

[100] S. Henikoff and J. G. Henikoff. Performance evaluation of amino acid substitution matrices. *PROTEINS: Structure, Function, and Genetics*, 17:49–61, 1993.

[101] C. M. Henneke. A multiple sequence alignment algorithm for homologous proteins using secondary structure information and optionally keying alignments to functionally important sites. *CABIOS*, 5(2):141–150, 1989.

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

[103] M. Hirosawa, M. Hoshida, M. Ishikawa, and T. Toya. MASCOT: Multiple alignment system for protein sequences based on three-way dynamic programming. *CABIOS*, 9(2):161–167, 1993.

[104] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18(6):341–343, 1975.

[105] P. Hogeweg and B. Hesper. The alignment of sets of sequences and the construction of phyletic trees: An integrated method. *J. Mol. Evol.*, 20:175–186, 1984.

[106] X. Huang. A contig assembly program based on sensitive detection of fragment overlaps. *Genomics*, 14(1):18–25, 1992.

[107] X. Huang. Alignment of three sequences in quadratic space. *Applied Computing Review*, 1(2):7–11, 1993.

[108] M. J. Hunt, M. Lennig, and P. Mermelstein. Use of dynamic programming in a syllable-based continuous speech recognition system. In D. Sankoff and J. B. Kruskal, editors, *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, chapter 5, pages 163–187. Addison-Wesley, Reading, MA, USA, 1983.

[109] M. S. Johnson and R. F. Doolittle. A method for the simultaneous alignment of three or more amino acid sequences. *J. Mol. Evol.*, 23:267–278, 1986.

[110] M. S. Johnson, M. J. Sutcliffe, and T. L. Blundell. Molecular anatomy: Phyletic relationships derived from three-dimensional structures of proteins. *J. Mol. Evol.*, 30:43–59, 1990.

[111] D. T. Jones, W. R. Taylor, and J. M. Thornton. The rapid generation of mutation data matrices from protein sequences. *CABIOS*, 8(3):275–282, 1992.

[112] T. H. Jukes and C. R. Cantor. Evolution of protein molecules. In H. N. Munro, editor, *Mammalian Protein Metabolism*, volume 3, pages 21–132. Academic Press, New York, NY, USA, 1969.

[113] R. M. Karp. Mapping the genome: Some combinatorial problems arising in molecular biology. In *Proc. of the 25th Annual ACM Symposium on the Theory of Computing, San Diego, CA, USA, May 16-18, 1993*, pages 278–285, New York, NY, USA, 1993. ACM Press.

[114] J. Kececioglu. The maximum weight trace problem in multiple sequence alignment. In A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, editors, *4th Annual Symposium, CPM 93, Padova, Italy, June 2-4, 1993. Proceedings*, number 684 in Lecture Notes in Computer Science, pages 106–119, Berlin, 1993. Springer Verlag.

[115] B. W. Kernighan and D. M. Ritchie. *The C Programming Language*. Prentice-Hall, Englewood Cliffs, NJ, 1978.

[116] M. Kimura. A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *J. Mol. Evol.*, 16:111–120, 1980.

[117] T. Kiss, C. Marshallsay, and W. Filipowicz. 7-2/MRP RNAs in plant and mammalian cells: Association with higher order structures in the nucleolus. *The EMBO Journal*, 11(10):3737–3746, 1992.

[118] A. Krogh, M. Brown, I. S. Mian, K. Sjölander, and D. Haussler. Hidden Markov models in computational biology: Applications to protein modeling. *J. Mol. Biol.*, 235(3):1501–1531, 1994.

[119] J. B. Kruskal. An overview of sequence comparison. In D. Sankoff and J. B. Kruskal, editors, *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, pages 1–44. Addison-Wesley, Reading, MA, USA, 1983.

[120] J. B. Kruskal and M. Liberman. The symmetric time-warp problem: From continuous to discrete. In D. Sankoff and J. B. Kruskal, editors, *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, pages 125–161. Addison-Wesley, Reading, MA, USA, 1983.

[121] J. B. Kruskal and D. Sankoff. An anthology of algorithms and concepts for sequence comparison. In D. Sankoff and J. B. Kruskal, editors, *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, pages 265–310. Addison-Wesley, Reading, MA, USA, 1983.

[122] J. A. Lake. Origin of the eukaryotic nucleus determined by rate-invariant analysis of rRNA sequences. *Nature*, 331:184–186, 1988.

[123] J. A. Lake. The order of sequence alignment can bias the selection of tree topology. *Mol. Biol. Evol.*, 8(3):378–385, 1991.

[124] H. T. Laquer. Asymptotic limits for a two-dimensional recursion. *Stud. Appl. Math*, 64:271–277, 1981.

[125] A. M. Lesk, M. Levitt, and C. Chotia. Alignment of the amino acid sequences of distantly related proteins using variable gap penalties. *Protein Engng.*, 1(1):77–78, 1986.

[126] U. Lessel and D. Schomburg. Similarities between protein 3D structures. *Protein Engng.*, 7(10):1175–1187, 1994.

[127] J. M. Levin, B. Robson, and J. Garnier. An algorithm for secondary structure determination in proteins based on sequence similarity. *FEBS Lett.*, 205(2):303–308, 1986.

[128] D. J. Lipman, S. F. Altschul, and J. D. Kececioglu. A tool for multiple sequence alignment. *Proc. Natl. Acad. Sci. USA*, 86:4412–4415, 1989.

[129] D. J. Lipman and W. R. Pearson. Rapid and sensitive protein similarity searches. *Science*, 227:1435–1441, 1985.

[130] R. Lowrance and R. A. Wagner. An extension of the string-to-string correction problem. *Journal of the ACM*, 22(2):177–183, 1975.

[131] R. Lück, G. Steger, and D. Riesner. Thermodynamic prediction of conserved secondary structure: Application to the RRE element of HIV, the tRNA-like element of CMV and the mRNA of prion protein. *J. Mol. Biol.*, 258:813–826, 1996.

[132] Martinez. H. M. An efficient method for finding repeats in molecular sequences. *Nucl. Acids Res.*, 11(13):4629–4634, 1983.

[133] H. M. Martinez. A flexible multiple sequence alignment program. *Nucl. Acids Res.*, 16(5):1683–1691, 1988.

[134] B. W. Matthews and M. G. Rossmann. Comparison of protein structures. In H. W. Wyckoff, C. H. W. Hirs, and S. N. Timasheff, editors, *Diffraction Methods for Biological Macromolecules*, volume 115 of *Methods in Enzymology*, chapter 27, pages 397–420. Academic Press, Inc., Orlando, FL, USA, 1985.

[135] M. A. McClure, T. K. Vasi, and W. M. Fitch. Comparative analysis of multiple protein-sequence alignment methods. *Mol. Biol. Evol.*, 11(4):571–592, 1994.

[136] W. Miller and E. W. Myers. Sequence comparison with concave weighting functions. *Bull. Math. Biol.*, 50(2):97–120, 1988.

[137] T. Miyata, S. Miyazawa, and T. Yasunaga. Two types of amino acid substitutions in protein evolution. *J. Mol. Evol.*, 12:219–236, 1979.

[138] S. Miyazawa and R. L. Jernigan. A new substitution matrix for protein sequence searches based on contact frequencies in protein structures. *Protein Engng.*, 6(3):267–278, 1993.

[139] J. K. Mohana Rao. New scoring matrix for amino acid residue exchanges based on residue characteristic physical parameters. *Int. J. Peptide Protein Res.*, 29(2):276–281, 1987.

[140] B. Morgenstern, A. W. M. Dress, and T. Werner. Multiple DNA and protein sequence alignment based on segment-to-segment comparison. *Proc. Natl. Acad. Sci. USA*, 93(22):12098–12103, 1996.

[141] M. Murata. Three-way Needleman-Wunsch algorithm. In R. F. Doolittle, editor, *Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences*, volume 183 of *Methods in Enzymology*, chapter 22, pages 365–375. Academic Press, Inc., San Diego, CA, USA, 1990.

[142] M. Murata, J. S. Richardson, and J. L. Sussman. Simultaneous comparison of three protein sequences. *Proc. Natl. Acad. Sci. USA*, 82:3073–3077, 1985.

[143] E. W. Myers. An $O(ND)$ difference algorithm and its variations. *Algorithmica*, 1:251–266, 1986.

[144] E. W. Myers. An overview of sequence comparison algorithms in molecular biology. Technical Report TR 91-29, University of Arizona, Tucson, Department of Computer Science, 1991.

[145] E. W. Myers and W. Miller. Optimal alignments in linear space. *CABIOS*, 4(1):11–17, 1988.

[146] E. W. Myers and W. Miller. Row replacement algorithms for screen editors. *Trans. on Prog. Lang. and Systems*, 11(1):33–56, 1989.

[147] E. W. Myers and W. Miller. Chaining multiple-alignment fragments in sub-quadratic time. In *Proc. of the 6-th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 38–47, 1995.

[148] D. Naor and D. L. Brutlag. On near-optimal alignments of biological sequences. *J. Comp. Biol.*, 1(4):349–366, 1994.

[149] 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.*, 48:443–453, 1970.

[150] K. Niefind and D. Schomburg. Amino acid similarity coefficients for protein modeling and sequence alignment derived from main-chain folding angles. *J. Mol. Biol.*, 219:481–497, 1991.

[151] K. Nishikawa and T. Ooi. Amino acid sequence homology applied to the prediction of protein secondary structures, and joint prediction with existing methods. *Biochim. Biophys. Acta*, 871:45–54, 1986.

[152] J. Overington, D. Donnelly, M. S. Johnson, A. Šali, and T. L. Blundell. Environment-specific amino acid substitution tables: Tertiary templates and prediction of protein folds. *Protein Science*, 1:216–226, 1992.

[153] S. Pascarella and P. Argos. Analysis of insertions/deletions in protein structures. *J. Mol. Biol.*, 224:461–471, 1992.

[154] W. R. Pearson. Using the FASTA program to search protein and DNA sequence databases. In A. M. Griffin and H. G. Griffin, editors, *Computer Analysis of Sequence Data, Part I*, volume 24 of *Methods in Molecular Biology*, chapter 26, pages 307–331. Humana Press, Totowa, NJ, USA, 1994.

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

[156] S. W. Perrey, A. W. M. Dress, and J. Stoye. Statistics for fragment comparison – a biologically motivated approach to sequence alignment. In D. Schomburg and U. Lessel, editors, *Bioinformatik, Informatik in den Biowissenschaften, Braunschweig, Germany, October 9-11*, 1995.

[157] S. W. Perrey and J. Stoye. Fast approximation to the NP-hard problem of multiple sequence alignment. Information and mathematical sciences reports, Series B:96/06, May 1996. (ISSN 1171-7637).

[158] S. W. Perrey, J. Stoye, V. Moulton, and A. W. M. Dress. On simultaneous versus iterative multiple sequence alignment. In preparation.

[159] P. A. Pevzner. Multiple alignment, communication cost, and graph matching. *SIAM J. Appl. Math.*, 52(6):1763–1779, 1992.

[160] P. A. Pevzner. Multiple alignment with guaranteed error bounds and communication cost. In A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, editors, *Combinatorial Pattern Matching, Third Annual Symposium, Tucson, AZ, USA, April 29-May 1, 1992. Proceedings*, number 644 in Lecture Notes in Computer Science, pages 205–213, Berlin, 1992. Springer Verlag.

[161] T. A. Reichert, D. N. Cohen, and A. K. C. Wong. An application of information theory to genetic mutations and the matching of polypeptide sequences. *J. theor. Biol.*, 42:245–261, 1973.

[162] J. L. Risler, M. O. Delorme, H. Delacroix, and A. Henaut. Amino acid substitutions in structurally related proteins: A pattern recognition approach. *J. Mol. Biol.*, 204:1019–1029, 1988.

[163] M.-F. Sagot, A. Viari, and H. Soldano. Multiple sequence comparison – a peptide matching approach. In Z. Galil and E. Ukkonen, editors, *Combinatorial Pattern Matching, 6th Annual Symposium, CPM 95, Espoo, Finland, July 5-7, 1995. Proceedings*, number 937 in Lecture Notes in Computer Science, pages 366–385, Berlin, 1995. Springer Verlag.

[164] N. Saitou. Maximum likelihood methods. In R. F. Doolittle, editor, *Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences*, volume 183 of *Methods in Enzymology*, chapter 36, pages 584–598. Academic Press, Inc., San Diego, CA, USA, 1990.

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

[166] D. Sankoff. Minimal mutation trees of sequences. *SIAM J. Appl. Math.*, 28(1):35–42, 1975.

[167] D. Sankoff and R. J. Cedergren. Simultaneous comparison of three or more sequences related by a tree. In D. Sankoff and J. B. Kruskal, editors, *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, pages 253–263. Addison-Wesley, Reading, MA, USA, 1983.

[168] D. Sankoff, R. J. Cedergren, and G. LaPalme. Frequency of insertion-deletion, transversion, and transition in the evolution of 5S ribosomal RNA. *J. Mol. Evol.*, 7:133–149, 1976.

[169] D. Sankoff, C. Morel, and R. J. Cedergren. Evolution of 5S RNA and the non-randomness of base replacement. *Nature New Biology*, 245:232–234, 1973.

[170] A. A. Schäffer. Personal communication.

[171] A. A. Schäffer. MSA version 2.1. http://alfredo.wustl.edu/msa/msa.tar.gz, 1995.

[172] M. E. Schmitt, J. L. Bennett, D. J. Dairaghi, and D. A. Clayton. Secondary structure of RNase MRP RNA as predicted by phylogenetic comparison. *The FASEB Journal*, 7(1):208–213, 1993.

[173] S. Schneckener and D. Schomburg. Optimal gap penalties and similarity matrices for sequence alignment. In R. Hofestädt, T. Lengauer, M. Löffler, and D. Schomburg, editors, *Computer Science and Biology – Proceedings of the German Conference on Bioinformatics (GCB'96)*, pages 292–293, University of Leipzig, Germany, 1996.

[174] M. Schöniger and A. von Haeseler. A stochastic model for the evolution of autocorrelated DNA sequences. *Mol. Phyl. Evol.*, 3:240–247, 1994.

[175] M. Schöniger and A. von Haeseler. Performance of the maximum likelihood, neighbor joining, and maximum parsimony methods when sequence sites are not independent. *Syst. Biol.*, 44(4):533–547, 1995.

[176] M. Schöniger and A. von Haeseler. Simulating efficiently the evolution of DNA sequences. *CABIOS*, 11(1):111–115, 1995.

[177] G. D. Schuler, S. F. Altschul, and D. J. Lipman. A workbench for multiple alignment construction and analysis. *PROTEINS: Structure, Function, and Genetics*, 9:180–190, 1991.

[178] G. E. Schulz, E. Schiltz, A. G. Tomasselli, R. Frank, M. Brune, A. Wittinghofer, and R. H. Schirmer. Structural relationships in the adenylate kinase family. *Eur. J. Biochem.*, 161:127–132, 1986.

[179] P. H. Sellers. On the theory and computation of evolutionary distances. *SIAM J. Appl. Math.*, 26(4):787–793, 1974.

[180] P. H. Sellers. Pattern recognition in genetic sequences. *Proc. Natl. Acad. Sci. USA*, 76(7):3041, 1979.

[181] P. H. Sellers. The theory and computation of evolutionary distances: Pattern recognition. *J. Algor.*, 1:359–373, 1980.

[182] P. R. Sibbald and P. Argos. Weighting aligned protein or nucleic acid sequences to correct for unequal representation. *J. Mol. Biol.*, 216:813–818, 1990.

[183] R. F. Smith and T. F. Smith. Pattern-induced multi-sequence alignment (PIMA) algorithm employing secondary structure-dependent gap penalties for use in comparative protein modelling. *Protein Engng.*, 5(1):35–41, 1992.

[184] T. F. Smith and M. S. Waterman. Comparison of biosequences. *Adv. Appl. Math.*, 2:482–489, 1981.

[185] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *J. Mol. Biol.*, 147:195–197, 1981.

[186] T. F. Smith, M. S. Waterman, and W. M. Fitch. Comparative biosequence metrics. *J. Mol. Evol.*, 18:38–46, 1981.

[187] E. Sobel and H. M. Martinez. A multiple sequence alignment program. *Nucl. Acids Res.*, 14(1):363–374, 1986.

[188] J. L. Spouge. Speeding up dynamic programming algorithms for finding optimal lattice paths. *SIAM J. Appl. Math.*, 49(5):1552–1566, 1989.

[189] R. Staden. A strategy of DNA sequencing employing computer programs. *Nucl. Acids Res.*, 6:2601–2610, 1979.

[190] M. A. Steel, M. D. Hendy, L. A. Szkely, and P. L. Erdös. Spectral analysis and a closest tree method for genetic sequences. *Appl. Math. Lett.*, 5:63–67, 1992.

[191] J. Stoye. DCA: Divide and conquer multiple sequence alignment. http://bibiserv.TechFak.Uni-Bielefeld.DE/dca/, 1996.

[192] 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(2):67–73, 1997.

[193] L. Stryer, editor. *Biochemistry.* Freeman, New York, NY, USA, 3rd edition, 1988.

[194] S. Subbiah and S. C. Harrison. A method for multiple sequence alignment with gaps. *J. Mol. Biol.*, 209:539–548, 1989.

[195] D. L. Swofford and G. J. Olsen. Phylogeny reconstruction. In D. M. Hillis and C. Moritz, editors, *Molecular Systematics*, chapter 11, pages 411–501. Sinauer Associates Inc., Sunderland, MA, USA, 1990.

[196] K. Tajima. Multiple DNA and protein sequence alignment on a workstation and a supercomputer. *CABIOS*, 4(4):467–471, 1988.

[197] P. Taylor. A fast homology program for aligning biological sequences. *Nucl. Acids Res.*, 12(1):447–455, 1984.

[198] W. R. Taylor. Pattern matching methods in protein sequence comparison and structure prediction. *Protein Engng.*, 2(2):77–86, 1988.

[199] W. R. Taylor. Hierarchical method to align large numbers of biological sequences. In R. F. Doolittle, editor, *Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences*, volume 183 of *Methods in Enzymology*, chapter 29, pages 456–474. Academic Press, Inc., San Diego, CA, USA, 1990.

[200] W. R. Taylor. An investigation of conservation-biased gap-penalties for multiple protein sequence alignment. *Gene*, 165:GC27–GC35, 1995.

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

[202] I. Tinoco Jr., O. C. Uhlenbeck, and M. D. Levine. Estimation of secondary structure in ribonucleic acids. *Nature*, 230:362–367, 1971.

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

[204] E. Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64:100–118, 1985.

[205] M. Vingron. *Multiple Sequence Alignment and Applications in Molecular Biology.* Dissertation, Interdisziplinäres Zentrum für Wissenschaftliches Rechnen der Universität Heidelberg, 1991.

[206] M. Vingron and P. Argos. A fast and sensitive multiple sequence alignment algorithm. *CABIOS*, 5(2):115–121, 1989.

[207] M. Vingron and P. Argos. Determination of reliable regions in protein sequence alignments. *Protein Engng.*, 3(7):565–569, 1990.

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

[209] M. Vingron and P. R. Sibbald. Weighting in sequence space: A comparison of methods in terms of generalized sequences. *Proc. Natl. Acad. Sci. USA*, 90:8777–8781, 1993.

[210] M. Vingron and A. von Haeseler. Towards integration of multiple alignment and phylogenetic tree construction. Arbeitspapiere 852, GMD, 1994.

[211] M. Vingron and A. von Haeseler. Towards integration of multiple alignment and phylogenetic tree construction. *J. Comp. Biol.*, 4(1):23–34, 1997.

[212] G. Vogt, T. Etzold, and P. Argos. An assessment of amino acid exchange matrices in aligning protein sequences: The twilight-zone revisited. *J. Mol. Biol.*, 249:816–831, 1995.

[213] R. A. Wagner. On the complexity of the extended string-to-string correction problem. In D. Sankoff and J. B. Kruskal, editors, *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, pages 215–235. Addison-Wesley, Reading, MA, USA, 1983.

[214] R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.

[215] L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *J. Comp. Biol.*, 1(4):337–348, 1994.

[216] M. S. Waterman. Introduction to computational biology. maps, sequences and genomes – corrections and clarifications. http://hto-e.usc.edu/books/msw-/msg/corrections/.

[217] M. S. Waterman. Sequence alignments in the neighborhood of the optimum with general application to dynamic programming. *Proc. Natl. Acad. Sci. USA*, 80:3123–3124, 1983.

[218] M. S. Waterman. Efficient sequence alignment algorithms. *J. theor. Biol.*, 108:333–337, 1984.

[219] M. S. Waterman. General methods of sequence comparison. *Bull. Math. Biol.*, 46(4):473–500, 1984.

[220] M. S. Waterman. Multiple sequence alignment by consensus. *Nucl. Acids Res.*, 14(22):9095–9102, 1986.

[221] M. S. Waterman. *Introduction to Computational Biology. Maps, Sequences and Genomes*. Chapman & Hall, London, UK, 1995.

[222] M. S. Waterman and T. H. Byers. A dynamic programming algorithm to find all solutions in a neighborhood of the optimum. *Math. Biosci.*, 77:179–188, 1985.

[223] M. S. Waterman and R. Jones. Consensus methods for DNA and protein sequence alignment. In R. F. Doolittle, editor, *Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences*, volume 183 of *Methods in Enzymology*, chapter 14, pages 221–237. Academic Press, Inc., San Diego, CA, USA, 1990.

[224] M. S. Waterman and M. D. Perlwitz. Line geometries for sequence comparisons. *Bull. Math. Biol.*, 48(4):567–577, 1984.

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

[226] M. S. Waterman and M. Vingron. Sequence comparison significance and poisson approximation. *Statistical Science*, 9(3):367–381, 1994.

[227] W. C. Wheeler and D. S. Gladstein. MALIGN: A multiple sequence alignment program. *J. Hered.*, In press.

[228] W. J. Wilbur and D. J. Lipman. The context dependent comparison of biological sequences. *SIAM J. Appl. Math.*, 44(3):557–567, 1984.

[229] Z. Zhang, B. Raghavachari, R. Hardison, and W. Miller. Chaining multiple-alignment blocks. *J. Comp. Biol.*, 1(3):217–226, 1994.

[230] M. Zuker. On finding all suboptimal foldings of an RNA molecule. *Science*, 244:48–52, 1989.

[231] M. J. Zvelebil, G. J. Barton, W. R. Taylor, and M. J. E. Sternberg. Prediction of protein secondary structure and active sites using the alignment of homologous sequences. *J. Mol. Biol.*, 195:957–961, 1987.

Bisher erschienene Reports an der Technischen Fakultät
Stand: 18. April 1997

94–01 Modular Properties of Composable Term Rewriting Systems
(Enno Ohlebusch)

94–02 Analysis and Applications of the Direct Cascade Architecture
(Enno Littmann und Helge Ritter)

94–03 From Ukkonen to McCreight and Weiner: A Unifying View
of Linear-Time Suffix Tree Construction
(Robert Giegerich und Stefan Kurtz)

94–04 Die Verwendung unscharfer Maße zur Korrespondenzanalyse
in Stereo Farbbildern
(Andrè Wolfram und Alois Knoll)

94–05 Searching Correspondences in Colour Stereo Images
— Recent Results Using the Fuzzy Integral
(Andrè Wolfram und Alois Knoll)

94–06 A Basic Semantics for Computer Arithmetic
(Markus Freericks, A. Fauth und Alois Knoll)

94–07 Reverse Restructuring: Another Method of Solving
Algebraic Equations
(Bernd Bütow und Stephan Thesing)

95–01 PaNaMa User Manual V1.3
(Bernd Bütow und Stephan Thesing)

95–02 Computer Based Training-Software: ein interaktiver Sequenzierkurs
(Frank Meier, Garrit Skrock und Robert Giegerich)

95–03 Fundamental Algorithms for a Declarative Pattern Matching System
(Stefan Kurtz)

95–04 On the Equivalence of E-Pattern Languages
(Enno Ohlebusch und Esko Ukkonen)

96–01 Static and Dynamic Filtering Methods for Approximate String Matching
(Robert Giegerich, Frank Hischke, Stefan Kurtz und Enno Ohlebusch)

96–02 Instructing Cooperating Assembly Robots through Situated Dialogues
in Natural Language
(Alois Knoll, Bernd Hildebrandt und Jianwei Zhang)

96–03 Correctness in System Engineering
(Peter Ladkin)

96–04 An Algebraic Approach to General Boolean Constraint Problems
(Hans-Werner Güsgen und Peter Ladkin)

96–05 Future University Computing Resources
(Peter Ladkin)

96–06 Lazy Cache Implements Complete Cache
(Peter Ladkin)

96–07 Formal but Lively Buffers in TLA+
(Peter Ladkin)

96–08 The X-31 and A320 Warsaw Crashes: Whodunnit?
(Peter Ladkin)

96–09 Reasons and Causes
(Peter Ladkin)

96–10 Comments on Confusing Conversation at Cali
(Dafydd Gibbon und Peter Ladkin)

96–11 On Needing Models
(Peter Ladkin)

96–12 Formalism Helps in Describing Accidents
(Peter Ladkin)

96–13 Explaining Failure with Tense Logic
(Peter Ladkin)

96–14 Some Dubious Theses in the Tense Logic of Accidents
(Peter Ladkin)

96–15 A Note on a Note on a Lemma of Ladkin
(Peter Ladkin)

96–16 News and Comment on the AeroPeru B757 Accident
(Peter Ladkin)

97–01 Analysing the Cali Accident With a WB-Graph
(Peter Ladkin)