Dissecting Multiple Sequence Alignment Methods

The Analysis, Design and Development of Generic Multiple Sequence Alignment Components in SeqAn.

Dem Fachbereich Mathematik und Informatik der
Freien Universität Berlin
zur Erlangung des akademischen Grades eines
Dr. rer. nat.

eingereichte Dissertation

von
Herr Tobias Rausch

Freie Universität Berlin
Mai 2010

Erstgutachter: Prof. Dr. Knut Reinert
Zweitgutachter: Prof. Dr. Alun Thomas

Tag der Disputation: 11. Mai 2010

Abstract

Multiple sequence alignments are an indispensable tool in bioinformatics. Many applications rely on accurate multiple alignments, including protein structure prediction, phylogeny and the modeling of binding sites. In this thesis we dissected and analyzed the crucial algorithms and data structures required to construct such a multiple alignment. Based upon that dissection, we present a novel graph-based multiple sequence alignment program and a new method for multi-read alignments occurring in assembly projects. The advantage of the graph-based alignment is that a single vertex can represent a single character, a large segment or even an abstract entity such as a gene. This gives rise to the opportunity to apply the consistency-based progressive alignment paradigm to alignments of genomic sequences. The proposed multi-read alignment method outperforms similar methods in terms of alignment quality and it is apparently one of the first methods that can readily be used for insert sequencing. An important aspect of this thesis was the design, the development and the integration of the essential multiple sequence alignment components in the SeqAn library. SeqAn is a software library for sequence analysis that provides the core algorithmic components required to analyze large-scale sequence data. SeqAn aims at bridging the current gap between algorithm theory and available practical implementations in bioinformatics. Hence, we always describe in conjunction to the theoretical development of the methods, the actual implementation of the data structures and algorithms in order to strengthen the use of SeqAn as an experimental platform for rapidly developing and testing applications. All presented methods are part of the open source SeqAn library that can be downloaded from our website, www.seqan.de.

Zusammenfassung

Multiple Sequenzvergleiche sind ein entscheidendes Hilfsmittel in der Bioinformatik. Zahlreiche Anwendungen, wie zum Beispiel Proteinstrukturvorhersagen, Phylogenie oder die Modellierung von Bindungsstellen beruhen auf der effizienten und biologisch korrekten Berechnung von multiplen Sequenzvergleichen. Aufgrund dieser enormen Bedeutung wurden in der vorliegenden Doktorarbeit Methoden zum multiplen Sequenzvergleich detailiert untersucht, analysiert und in elementare Algorithmen und Datenstrukturen zergliedert. Diese strukturelle Zerlegung bildet die Grundlage für unsere eigenen Weiterentwicklungen. Insbesondere diskutieren und beschreiben wir hier zwei erweiterte Ansätze zum graphbasierten, multiplen Sequenzvergleich und zum Konsensusalignment. Für beide Methoden zeigen wir die Vorteile unserer Algorithmen gegenüber bisherigen Ansätzen. Ein weiterer zentraler Bestandteil der Arbeit ist der Entwurf, die Implementierung und die Integration dieser grundlegenden Algorithmen und Datenstrukturen zum multiplen Sequenzvergleich in der SeqAn Bibliothek. SeqAn ist eine Softwarebibliothek zur Sequenzanalyse. SeqAn hat das Ziel die neuesten Erkenntnisse aus der Algorithmentheorie für praktische Anwendungen verfügbar zu machen und im Rahmen einer experimentellen Plattform anzubieten, in der Algorithmen einfach entworfen, entwickelt und verglichen werden können. Daher beschreiben wir in der gesamten Arbeit neben der theoretischen Entwicklung der Algorithmen ebenso deren softwaretechnische Umsetzung in SeqAn und zeigen zum Beispiel anhand von paarweisen Alignmentalgorithmen deren Überlegenheit in Zeit- und Platzbedarf verglichen mit bisherigen Implementierungen. Am Ende der Arbeit werden Einschränkungen und mögliche Erweiterungen der vorgestellten Methoden diskutiert. Alle Algorithmen und Datenstrukturen sind im Rahmen der SeqAn Bibliothek frei verfügbar: www.seqan.de

Acknowledgement

A big “Thank you!” to all my advisors, supporters, co-workers, friends and family members. Doing a PhD and writing a thesis was enriching, exciting, fulfilling and fun and from time to time frustrating, troublesome, stagnant and annoying. The positive part is no big deal but to be carried and pampered through the tough part makes the difference. The support of Knut Reinert, my primary advisor, was truly remarkable. Knut, I felt that your guidance, your ideas and your encouragements were extraordinarily valuable. I greatly enjoyed working in your group and I hope, we are able to stay in touch for future projects. Thanks a lot! I am also very happy to got to know Cedric Notredame. I have rarely seen anybody, who is as enthusiastic as you are, Cedric. Your famous aircraft survivability story probably unites all of your students and I am actually very proud to be one of the guys, who heard about it, thanks! Alun Thomas, you truly enjoy research. Without having seen someone like you before, I probably would have never started this PhD project. I am very grateful for the time I could spend with you, many thanks! Another big “Thank you!” to Andreas, Anne-Katrin, David and Marcel for joint projects or helping me out with SeqAn. Many thanks to the other group and “floor” members for fun and thoughts at work and in the free time: thanks Alexandra, Anja, Chris, Christopher, Clemens, Eva, Gesine, Laura, Markus, Martin, Ole, Sandro, Stephan and all the others. Many thanks to my best friends Christian (Hyvää päivää), Andi, Berni, Johannes and Holli (Servus, Tirol isch scho was) and Divya, Richard and Sebastian. I feel very privileged to have friends like you I can rely on. Last but not least I would like to thank my family and Shan. I know, the past three years were quite difficult but having people like you by my side is indeed impressive. Vielen Dank!

谢谢你，人海之中找到了你，一切变得都有意义，自有山比此山更高，但爱心找不到比你更好。

Contents

List of Figures

List of Tables

Part I

Introduction

Motivation

1.1 Biological Background

The genetic information of all living organisms is encoded in deoxyribonucleic acid (DNA). The DNA is a double-stranded polymer, wounded as a helix that has a backbone and only four simple building blocks called nucleotides shown in Table 1.1(a). The nucleotides are adenine (A), cytosine (C), guanine (G) and thymine (T). A and G are purines whereas T and C are pyrimidines. The nucleotides pair up and give rise to the characteristic double-stranded, helical shape of DNA molecules. Since adenine only pairs up with thymine and cytosine only pairs up with guanine, each strand is a reverse complemented copy of the other strand. Hence, reversing one strand and replacing each A with T, T with A, C with G and G with C gives rise to the other strand as shown below. By convention, both DNA strands are read in five prime ($5'$) to three prime ($3'$) direction.

$$
\begin{array}{cccccc}
\overrightarrow{5'} & & & & & 3' \\
\cdots & \mathrm{T} & \mathrm{C} & \mathrm{A} & \mathrm{G} & \cdots \\
 & \| & \| & \| & \| & \\
\cdots & \mathrm{A} & \mathrm{G} & \mathrm{T} & \mathrm{C} & \cdots \\
3' & & & & & \overleftarrow{5'}
\end{array}
$$

Each strand can serve as a template for building the other strand, which is an essential precondition for DNA replication. In human beings the DNA resides in organized structures called chromosomes. The human genome consists of 46 such chromosomes. The chromosomes appear in 23 pairs. There is one pair of sex chromosomes and 22 remaining pairs of chromosomes called autosomes. In every pair one chromosome was inherited from the mother and the other chromosome was in-

Nucleotide	1-Letter	Class
Adenine	A	Purine
Cytosine	C	Pyrimidine
Guanine	G	Purine
Thymine	T	Pyrimidine
Uracil (Rna)	U	Pyrimidine

(a) The 4 DNA nucleotides and their 1-letter abbreviation. In RNA thymine is replaced by uracil.

Amino acid	3-Letter	1-Letter	Amino acid	3-Letter	1-Letter
Alanine	Ala	A	Leucine	Leu	L
Arginine	Arg	R	Lysine	Lys	K
Asparagine	Asn	N	Methionine	Met	M
Aspartic acid	Asp	D	Phenylalanine	Phe	F
Cysteine	Cys	C	Proline	Pro	P
Glutamic acid	Glu	E	Serine	Ser	S
Glutamine	Gln	Q	Threonine	Thr	T
Glycine	Gly	G	Tryptophan	Trp	W
Histidine	His	H	Tyrosine	Tyr	Y
Isoleucine	Ile	I	Valine	Val	V

(b) The 20 amino acids with their 3-letter and 1-letter abbreviations.

Table 1.1: The 20 amino acids and 4 DNA nucleotides.

herited from the father. These inherited chromosomes are, however, not a mere copy of the chromosomes of our parents. Genetic recombination events such as crossing over or mutations can induce changes ranging from single nucleotide differences to shuffled pieces of DNA.

However, not DNA but proteins are required in nearly every cellular process. Proteins have a plethora of functions such as structural, catalytic, signaling or mechanical responsibilities. Proteins or polypeptides are made of twenty standard amino acids shown in Table 1.1(b). The crucial link between the genetic information carrier DNA and the proteins is the transcription of coding DNA, called genes, into ribonucleic acid (RNA) and the translation of RNA into the primary sequence

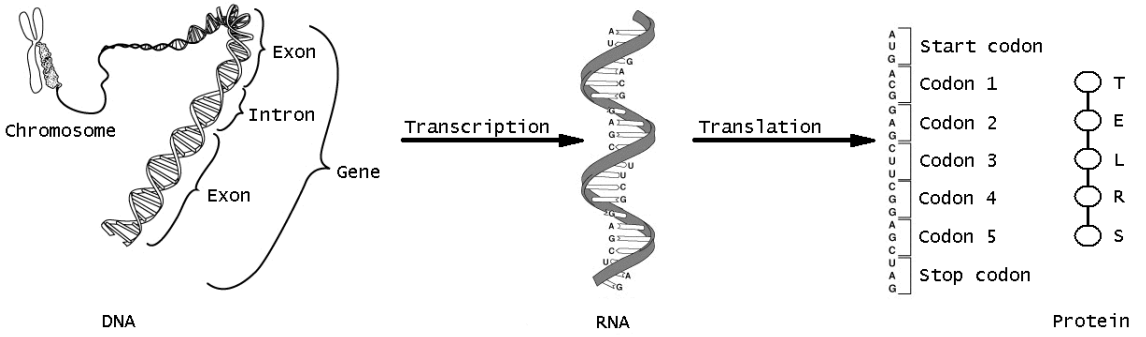


Figure 1.1: Genes are transcribed into RNA and by means of the genetic code translated into proteins. The figure was adapted from a public domain illustration of the National Human Genome Research Institute (www.genome.gov).

Amino acid	Codons	Amino acid	Codons
A	GCU, GCC, GCA, GCG	L	UUA, UUG, CUU, CUC, CUA, CUG
R	CGU, CGC, CGA, CGG, AGA, AGG	K	AAA, AAG
N	AAU, AAC	M	AUG
D	GAU, GAC	F	UUU, UUC
C	UGU, UGC	P	CCU, CCC, CCA, CCG
E	GAA, GAG	S	UCU, UCC, UCA, UCG, AGU, AGC
Q	CAA, CAG	T	ACU, ACC, ACA, ACG
G	GGU, GGC, GGA, GGG	W	UGG
H	CAU, CAC	Y	UAU, UAC
I	AUU, AUC, AUA	V	GUU, GUC, GUA, GUG
START	AUG	STOP	UAA, UGA, UAG

Table 1.2: The standard genetic code showing for each amino acid the involved codons. The start codon initiates translation and the stop codons terminate translation.

of amino acids in a protein. This two-step process of transcription and translation is well-known as the central dogma of molecular biology, summarized in Figure 1.1 and explained hereafter.

The transcribed RNA is a single-stranded molecule. It has the same four building blocks as DNA, except that thymine is replaced by uracil (U). The translation from the four letter RNA alphabet to the twenty letter amino acid alphabet occurs by means of the genetic code shown in Table 1.2. Each non-overlapping, contiguous

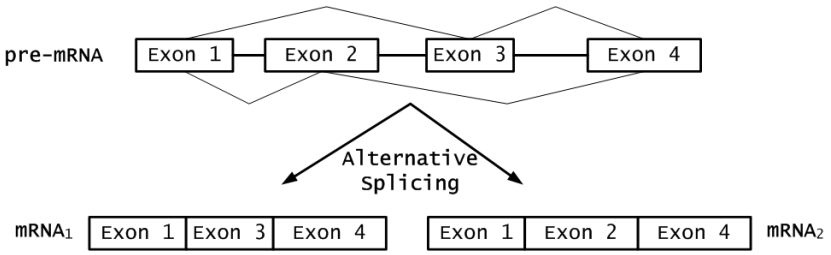


Figure 1.2: Alternative splicing: The exons of a single pre-mRNA can be spliced together in different combinations. This leads to distinct mRNAs that may be translated to different proteins. Hence, a single gene can code for multiple proteins.

tri-nucleotide sequence, called codon, is mapped to an amino acid. Since there are 4^3 different tri-nucleotides but only 20 amino acids, the genetic code is a degenerated or redundant code, multiple codons can encode the same amino acid. This implies a limited robustness of the genetic code against single nucleotide mutations. In many eukaryotes a gene is composed of coding and non-coding segments called exons and introns, respectively. The introns are spliced out after the initial transcription. This splicing process is quite sophisticated because different splicing patterns can occur in a single gene, a mechanism called alternative splicing shown in Figure 1.2. Hence, even a single gene can code for different proteins. Besides the problem of finding the genes and elucidating the regulatory mechanisms controlling transcription, alternative splicing or post-translational modifications, one of the most important research questions is what functions do the encoded proteins perform. An indispensable tool to answer this specific question are multiple sequence alignments (MSAs).

1.2 Multiple Sequence Alignment in Computational Biology

A DNA sequence and thus, each and every protein sequence has an evolutionary history. Due to that history, sequence similarity might imply functional or structural conservation. Biologically important residues or nucleotides are assumed to be less-likely to mutate than unimportant ones and thus, observed sequence conservation might be a hint to a functionally important region. Using sequence similarity, a sequence with a known function can be used to annotate, classify or find similar sequences by means of pairwise or multiple alignments. Hence, the main goal of a

Figure 1.3: A local (left) and a global (right) multiple sequence alignment of three sequences. The full sequences are depicted by gray lines and only the aligned parts are shown in black.

MSA is to group conserved residues or nucleotides that have the highest sequence similarity. This is achieved by rewriting the sequences in such a way that conserved residues or nucleotides appear in the same column. A gap character '−' is used to introduce spaces into the sequences. An example MSA is shown below for three sequences $GAAT$, AAC, and $GAACT$.

$$\begin{array}{ccccc} G & A & A & - & T \\ - & A & A & C & - \\ G & A & A & C & T \end{array}$$

The advantage of multiple alignments compared to pairwise alignments is that multiple alignments are more powerful to detect weakly conserved sequence similarities. In a pairwise alignment these weak and faint but biologically important sequence similarities might vanish. They still stand out in a multiple sequence alignment if a number of sequences highlight and delineate the extent of conservation.

MSA problems are characterized by (1) the number of sequences, (2) the length of the sequences, (3) the alphabet of the sequences (usually DNA, RNA, or amino acids), and (4) the relatedness of the sequences. Here, relatedness refers to both, the divergence of the sequences and whether the sequences are globally or locally related (see Figure 1.3). Many applications rely on accurate MSAs and the most prominent ones are explained hereafter, namely structure prediction, function prediction, domain identification, modeling binding sites, phylogeny and deriving a consensus sequence.

1.2.1 Structure prediction

Translating an RNA sequence into a linear chain of amino acids by means of the genetic code allows us to retrieve the primary structure of a protein, its sequential composition of amino acids. The secondary structure of a protein describes the

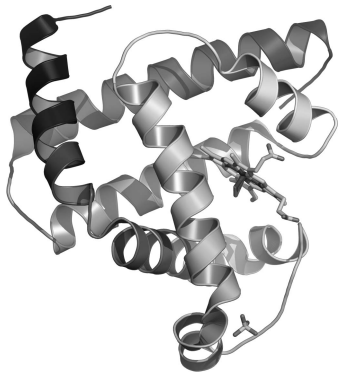


Figure 1.4: A 3D model showing the coiled, helical domains of myoglobin. Public domain illustration from Wikipedia: http://en.wikipedia.org/wiki/Myoglobin.

three-dimensional layout of common structural elements occurring in proteins such as alpha helices and beta sheets. An alpha helix has a characteristic coiled shape whereas a beta sheet is rather flat and bended as shown in Figure 1.4. The secondary structure does not describe the exact position of each and every atom, which is the so-called tertiary structure of a protein.

Inferring the three-dimensional protein fold from the primary structure is non-trivial. For the secondary structure, however, multiple alignments can give a very good hint as exemplarily shown in Table 1.3 and Figure 1.4 for a set of globin sequences.

1.2.2 Function prediction

In the recent past, a number of protein sequences have been characterized and annotated. This information is collected in public databases such as UniProt (The UniProt Consortium, 2007). Given an unannotated query sequence, tools such as BLAST (Altschul et al., 1990) can be used to search such databases. These tools return the database sequences that gave rise to high-scoring local alignments with the initial query sequence. Using a local or global alignment of the retrieved sequences with the query sequence, one can identify shared and distinct sequence patterns, facilitating an annotation of the uncharacterized sequence. Similarly, we can align the unannotated sequence to a protein family present in a protein family database such as Pfam (Finn et al., 2008) to verify that the new protein sequence belongs to that family.

```
HBA_HUMAN    .MVLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHF.DLSH
HBB_HUMAN    MVHLTPEEKSAVTALWGKV..NVDEVGGEALGRLLVVYPWTQRFFESFGDLST
HBA_HORSE    .MVLSAADKTNVKAAWSKVGGHAGEYGAEALERMFLGFPTTKTYFPHF.DLSH
HBB_HORSE    .VQLSGEEKAAVLALWDKV..NEEEVGGEALGRLLVVYPWTQRFFDSFGDLSN
MYG_PHYCA    .MVLSEGEWQLVLHVWAKVEADVAGHGQDILIRLFKSHPETLEKFDRFKHLKT
LGB2_LUPLU   MGALTESQAALVKSSWEEFNANIPKHTHRFFILVLEIAPAAKDLFSFLKGTSE

HBA_HUMAN    .....GSAQVKGHGKKVADALTNAVAHVDD...M..PNALSALSDLHAHKLRVD
HBB_HUMAN    PDAVMGNPKVKAHGKKVLGAFSDGLAHLDN...L..KGTFATLSELHCDKLHVD
HBA_HORSE    .....GSAQVKAHGKKVGDALTLAVGHLDD...L..PGALSNLSDLHAHKLRVD
HBB_HORSE    PGAVMGNPKVKAHGKKVLHSFGEGVHHLDN...L..KGTFAALSELHCDKLHVD
MYG_PHYCA    EAEMKASEDLKKHGVTVLTALGAILKKKGH...H..EAELKPLAQSHATKHKIP
LGB2_LUPLU   VPQ..NNPELQAHAGKVFKLVYEAAIQLQVTGVVVTDATLKNLGSVHVSK.GVA

HBA_HUMAN    PVNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKYR......
HBB_HUMAN    PENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH......
HBA_HORSE    PVNFKLLSHCLLSTLAVHLPNDFTPAVHASLDKFLSSVSTVLTSKYR......
HBB_HORSE    PENFRLLGNVLVVVLARHFGKDFTPELQASYQKVVAGVANALAHKYH......
MYG_PHYCA    IKYLEFISEAIIHVLHSRHPGDFGADAQGAMNKALELFRKDIAAKYKELGYQG
LGB2_LUPLU   DAHFPVVKEAILKTIKEVVGAKWSEELNSAWTIAYDELAIVIKKEMNDAA...
```

Table 1.3: MSA of 6 globin sequences: Human hemoglobin subunit alpha (UniProt accession: P69905), human hemoglobin subunit beta (P68871), horse hemoglobin subunit alpha (P01958), horse hemoglobin subunit beta (P02062), sperm whale myoglobin (P02185) and european yellow lupin leghemoglobin-2 (P02240). The common helix secondary structure elements are shown in bold font. UniProt (The UniProt Consortium, 2007) is a comprehensive public protein sequence database providing functional annotations.

1.2.3 Domain identification

Proteins consist of several functionally active regions called domains. The domains of a given protein can occur shuffled or repeated in a different protein. Hence, to identify these domains one needs a more sensitive method than a mere global alignment of protein sequences. One approach is to build a profile $\mathcal{P}$ or position specific probability matrix of a given domain. A very small example of such a profile is shown in Table 1.4. For each column, a profile stores the relative frequency

of each letter. That is, the number of occurrences of a given letter normalized by the total amount of letters in the given column. The strength of profiles is their ability to distinguish evolutionary conserved sites from variable sites. A preliminary step to construct such a profile is the multiple alignment of a set of sequences describing the domain. Once we have such a profile at hand, we can scan other sequences for high-scoring local alignments to the profile.

1.2.4 Modeling binding sites

Another application of MSAs is the modeling of binding sites. A specific class of binding sites are, for instance, promoter binding sites. Promoters regulate the expression of a gene and they are located upstream of the 5' end of a gene. A compact graphical representation of such a binding site are so-called sequence logos (Schneider and Stephens, 1990; Crooks et al., 2004) that can be generated from an input MSA. Characteristic features of an alignment are readily apparent in such a logo as exemplarily shown in Figure 1.5.

For each column in the alignment, the logo has a stack of characters usually DNA or RNA letters. In each of these stacks, the height of a symbol represents the relative frequency of that character in the given column. The total height of the stack represents the conservation or information content (in bits) of the given column. Hence, sequence logos highlight the conserved positions in the alignment. In addition, they provide more information than a mere consensus sequence, where a multiple alignment is simply condensed to the most frequent letter in each column.

A	G	C	T
A	G	C	C
A	–	C	A
C	G	C	A

$\mathcal{P}$	1	2	3	4
A	0.75	0	0	0.5
C	0.25	0	1.0	0.25
G	0	0.75	0	0
T	0	0	0	0.25
–	0	0.25	0	0

Table 1.4: A profile of an alignment of four DNA sequences. $\mathcal{P}_{s,u}$ is the frequency of character $s \in \tilde{\Sigma}$ in column u. $\tilde{\Sigma}$ is the DNA alphabet augmented by the gap character '–'.

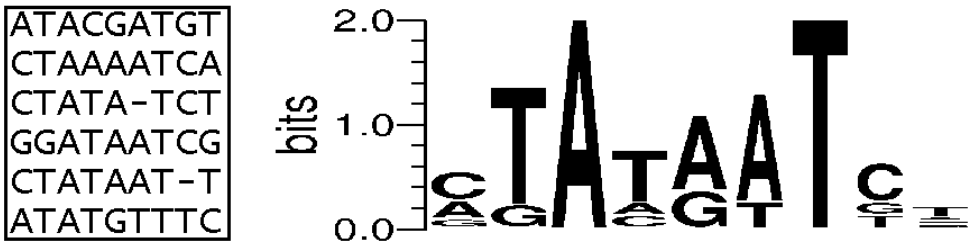


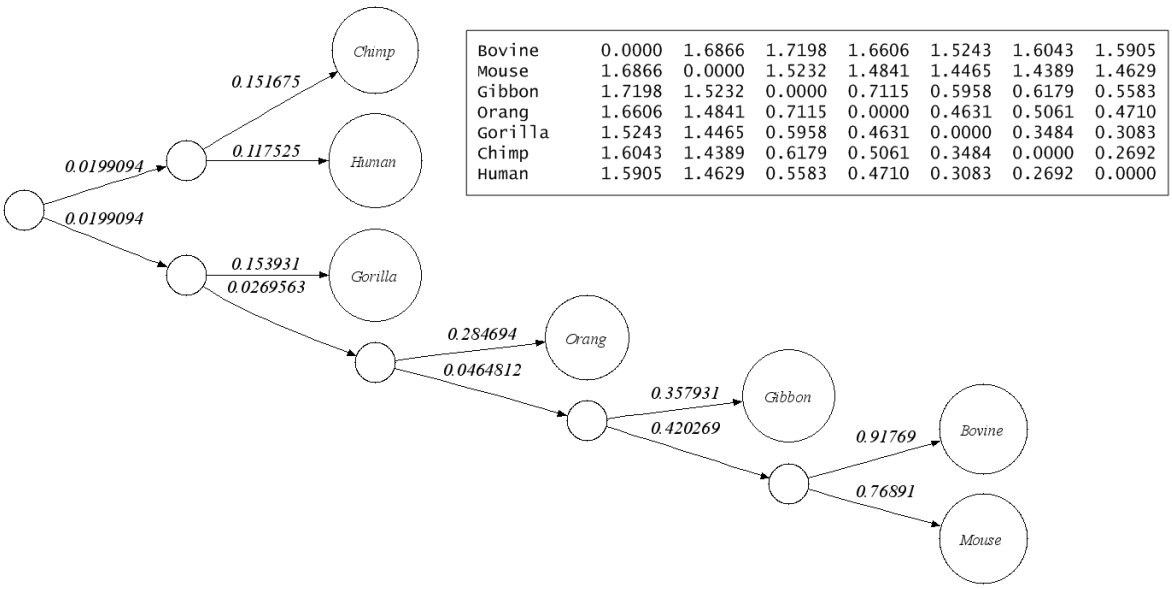
Figure 1.5: A putative binding site sequence logo (right) derived from a multiple sequence alignment (left). The sequence logo has been created with WebLogo 3 (Crooks et al., 2004).

1.2.5 Phylogeny

A phylogenetic tree indicates the evolutionary history of a set of sequences. The sequences are represented by the leaves of the tree and internal nodes are putative ancestor sequences. Almost all phylogenetic inference algorithms start with a MSA of a set of sequences. One specific class of phylogenetic tree reconstruction algorithms are distance-based tree reconstruction algorithms. These methods first estimate a distance matrix from pairwise or multiple alignments. This distance matrix is then used to reconstruct the tree as explained later in Chapter 2. An example of such a phylogenetic tree derived from a distance matrix is shown in Figure 1.6. Alternative methods to reconstruct a phylogenetic tree from a given MSA are maximum parsimony (Fitch, 1971) and maximum likelihood (Felsenstein, 1981) methods.

1.2.6 Sequence consensus

Using current DNA sequencers, whole genomes cannot be sequenced front to back in a single run. Depending on the sequencing technology, machines are limited to sequence only 35 to at most 1000 nucleotides. These small, sequenced DNA chunks are called reads. Fortunately, these tiny reads can still be used to sequence eukaryotic genomes hundreds or even thousands of mega bases long. The idea is to copy the initial DNA several times and to shear these copies randomly into thousands of overlapping fragments. The final reads are then sequenced from the ends of these fragments. Due to the copying and random shearing we hope to sequence every single base of the genome multiple times by means of overlapping reads. Such overlaps can then be used to position every read and reconstruct the sequenced genome. A very small example is shown in Figure 1.7. In this clipped example,



Bovine	0.0000	1.6866	1.7198	1.6606	1.5243	1.6043	1.5905
Mouse	1.6866	0.0000	1.5232	1.4841	1.4465	1.4389	1.4629
Gibbon	1.7198	1.5232	0.0000	0.7115	0.5958	0.6179	0.5583
Orang	1.6606	1.4841	0.7115	0.0000	0.4631	0.5061	0.4710
Gorilla	1.5243	1.4465	0.5958	0.4631	0.0000	0.3484	0.3083
Chimp	1.6043	1.4389	0.6179	0.5061	0.3484	0.0000	0.2692
Human	1.5905	1.4629	0.5583	0.4710	0.3083	0.2692	0.0000

Figure 1.6: A phylogenetic tree of 7 sequences reconstructed from a distance matrix in Phylip format. The pairwise distances may, for instance, be derived from normalized alignment scores.

seven reads have been ordered and positioned according to their overlaps. Each read has an orientation because unfortunately, we do not know whether the sequencer has read the sequence in forward or reverse direction. In addition, sequencers might produce sequencing errors as in column 25 where a G is missing in $Read_2$ or in column 13 where we observe an A instead of a G in $Read_1$. Similarly, we might encounter true polymorphisms. For instance, as noted in the introduction humans have pairs of chromosomes, one is inherited from the mother and the other from the father. These so-called polyploid organisms with multiple haplotypes result in sequenced reads stemming from either haplotype. Since the haplotypes are not identical we might observe these deviations, called polymorphisms, in the multi-read alignment. In the above example, column 16 and column 20 might be true polymorphisms instead of sequencing errors. The distinction of sequencing errors and polymorphisms is non-trivial. If a certain letter is supported by many other reads it might be a true variation otherwise it is assumed to be a sequencing error. In sequencing projects, one is usually interested in the originally sequenced genome. Because of that, the multi-read alignment is usually augmented by a consensus sequence. This consensus sequence can be derived from the alignment by taking, for instance, the most frequent letter in each column as shown in the example.

	...	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	...
Consensus	G	A	T	T	G	A	G	A	C	T	G	T	A	–	C	T	G	A	T	C
← $Read_1$	G	A	T	T	A	A	G	A	C											
→ $Read_2$		A	T	T	G	A	G	A	C	T	G	T	A	–	C	T	–	A		
← $Read_3$				T	G	A	G	–	C	T	G	C	A	T	C	T	G	A	T	
← $Read_4$					G	A	G	A	C	T	G	T	A	–	C	T				
→ $Read_5$						A	G	–	C	T	G	C	A	–	C	T	G	A	A	C
→ $Read_6$							G	A	C	T	G	T	A	–	C	T	G	A		
→ $Read_7$							G	–	C	T	G	C	A	–	C	T	G	A	T	C

Figure 1.7: A clipped multi-read alignment of seven reads. The most frequent letter in each column is part of the consensus shown at the top with ties broken arbitrarily.

1.3 History of the Problem

Throughout the history of MSAs one can distinguish two types of algorithms, optimal ones and heuristics. The former algorithms compute an optimal alignment with respect to some scoring function such as the sum of pairs score. The latter algorithms compute an alignment based on some kind of biological sound procedure such as progressive alignment. Both classes of algorithms are reviewed in Chapter 2.

The first optimal methods could align three sequences simultaneously using standard dynamic programming (Gotoh, 1986; Murata et al., 1985). A few years later, the program MSA (Gupta et al., 1995; Lipman et al., 1989) could align up to eight sequences of average protein length by using a clever bounding technique for the dynamic programming lattice. Time and space was further reduced using the $\mathcal{A}^*$ algorithm (Lermen and Reinert, 2000; Reinert et al., 1997) and (partly heuristic) divide and conquer techniques (Reinert et al., 2000). Besides bounding techniques for the dynamic programming formulation, other algorithms used a so-called alignment graph or trace graph (Kececioglu, 1993; Sankoff and Kruskal, 1983). This alignment graph was used in an integer linear programming (ILP) formulation (Reinert, 1999) extended by various methods from combinatorial optimization (Althaus and Canzar, 2008; Althaus et al., 2002, 2006).

Computing an optimal alignment is, however, NP-hard using the sum of pairs score (Wang and Jiang, 1994). Because of that a vast number of heuristics has been

developed enabling the alignment of more sequences of greater length. Heuristic methods were difficult to compare in the beginning but gained enormous leverage with the advent of protein benchmark data sets of sometimes manually refined MSAs such as BAliBASE (Thompson et al., 1999a, 2005), PREFAB (Edgar, 2004b), OXBENCH (Raghava et al., 2003), SABmark (Walle et al., 2005) and IRMBASE (Subramanian et al., 2005). For protein alignments, these benchmarks are the de facto standard for judging the performance of individual methods. The first heuristic *progressive* aligner was published in 1987 (Feng and Doolittle, 1987) followed by a great variety of other heuristics, most prominently the Clustal series of programs (Higgins and Sharp, 1988; Larkin et al., 2007; Thompson et al., 1994). Throughout the past years, the progressive alignment paradigm has been extended using approaches outlined in the next chapter such as consistency (Gotoh, 1990; Vingron and Argos, 1991; Notredame et al., 2000; Do et al., 2005) and refinement (Edgar, 2004b; Katoh et al., 2002).

The advent of genome sequencing spurred the development of software packages to assemble entire genomes. An integral part of these so-called assemblers is a multiread alignment module to compute a consensus sequence. The most prominent assembler is the Celera Assembler (Myers et al., 2000) developed to sequence the human genome in 2001 (Venter et al., 2001). However, a number of other assemblers are frequently used today, such as Velvet (Zerbino and Birney, 2008), the Newbler assembler from Roche or Arachne (Batzoglou et al., 2002).

The increasing number of available genomic sequences also stimulated the development of so-called genome aligners or genome comparison tools in the past 10 years. The MUMmer series of programs (Delcher et al., 1999, 2002; Kurtz et al., 2004) remarkably pioneered this research area but lately, a number of other interesting anchor-based alignment tools appeared (Brudno et al., 2003; Darling et al., 2004).

1.4 Guide to the Thesis

Having read the motivation you are hopefully convinced that multiple sequence alignments are useful and one of the main workhorses ubiquitously used in computational biology. The rest of the introduction covers all facets of a MSA in-depth by giving a succinct MSA definition, a detailed account of available MSA representa-

tions and an overview of current alignment methods. At the end of the introduction a brief overview of the main contributions of this thesis is given.

Part II and Part III of this thesis cover then in detail our own contribution whereas the last, fourth part contains a discussion of the strength and weaknesses of our method and an outlook on upcoming trends and challenges in the field of multiple sequence alignments. To facilitate an easy reading of the next chapters we summarized here some notation and some graph theoretical concepts used in the rest of the thesis. We also introduce the SeqAn library that was used and augmented extensively throughout the entire thesis.

1.4.1 Notation

Sequences:

S	A single sequence.
$\mathcal{S}$	An ordered set of sequences.
$S^i \in \mathcal{S}$	S^i is the i-th sequence in the ordered sequence set $\mathcal{S}$.
$\mathcal{S}' \subseteq \mathcal{S}$	$\mathcal{S}'$ is a subset of $\mathcal{S}$.
$\lvert\mathcal{S}\rvert$	The cardinality of a set.
$\lvert S^i\rvert$	The length of sequence S^i.
$S^i = s^i_0, s^i_1, \cdots, s^i_{n-1}$	A sequence of length n. The first character is s^i_0 and the last character is s^i_{n-1}. The first character is always indexed with 0. The comma in-between characters is sometimes omitted.
$[u, x($	An integer interval ranging from u (including) to x (excluding), that is, $[u, u+1, \cdots, x-1]$.
$S^i_{ux} = s^i_u, s^i_{u+1}, \cdots, s^i_{x-1}$	A $[u, x($ segment of string S^i starting at u and ending at $x-1$. Hence, S^i_{0n} denotes the full string S^i of length n.
Σ	The set of alphabet symbols.
$S \in \Sigma^*$	S is a finite sequence over the alphabet Σ.

Further notations, used font styles and pronoun usage:

$\mathcal{O}$	The big $\mathcal{O}$ notation is used to describe the worst case running time and memory usage of an algorithm.
`./pair_align`	Typewriter font is used for pseudocode or shell commands.
`connected_components`	Functions or data types that are one-to-one available in SeqAn are boxed and written in typewriter font.
we/I	I enjoyed working on this thesis project quite a lot and in particular, I enjoyed working with other people from my own and other research groups. Unfortunately, I had personally some difficult past months but thanks to my co-workers, my advisors, friends and family I carried on. To honor their help and contribution I prefer to use "we" instead of "I" throughout this thesis.

1.4.2 Graph theory

Graphs have been central to this thesis. In particular, a large part of this thesis is concerned about a data structure called alignment graph. This alignment graph has been built on top of some basic graph types and standard graph algorithms can be applied to it. Hence, a brief overview of graph theory is given hereafter.

A graph $G = (V, E)$ consists of a set of vertices V and a set of edges E, which is defined as a binary relation on V. An edge e is an element of E and e can be either directed $e = (u, v) \in E$ or undirected $e = \{u, v\} \in E$ where $u, v \in V$. Directed graphs have directed edges, a set of ordered pairs of vertices, whereas undirected graphs have undirected edges, a set of unordered pairs of vertices. Directed edges are also called arcs. Each undirected and directed edge has two endpoints $u, v \in V$. We call an edge a self-loop if $u = v$. For a directed edge $e = (u, v) \in E$, we call u the source vertex and v the target vertex of edge e. Tail and head are alternative names for the source and target vertex, respectively. For a graph $G = (V, E)$, $|V|$ denotes the number of vertices and $|E|$ denotes the number of edges. Weighted graphs associate a weight to each edge, denoted as w_e.

In an undirected graph, two vertices u and v are adjacent if and only if there exists an edge $e = \{u, v\} \in E$. In a directed graph, an edge $e = (u, v) \in E$ means that v is adjacent to u but not vice versa. If a vertex $v \in V$ is an endpoint of an undirected edge $e \in E$, e is called incident on v. For a directed edge $e = (u, v) \in E$, e is incident from u and incident to v.

The degree of a vertex v in an undirected graph is the number of adjacent vertices. In a directed graph the in-degree of vertex v is the number of edges directed to v, that is, the number of edges where v is the target vertex. The out-degree of vertex v is the number of edges directed from v, that is, the number of edges where v is the source vertex. We also say the out-degree specifies the number of outgoing edges whereas the in-degree specifies the number of incoming edges.

A path of length k in G from vertex u to vertex w is a sequence of vertices $(v_0, v_1, \cdots, v_k)$ such that $(v_i, v_{i+1}) \in E$, $0 \leq i < k$, $v_0 = u$ and $v_k = w$. Hence, the length of a path is equal to the number of edges on the path. A vertex v_t is reachable from v_s if and only if there is a path from v_s to v_t. A path is a cycle if $v_0 = v_k$ and it is called simple if all vertices are distinct. A clique in an undirected graph $G = (V, E)$ is a vertex subset $V_C \subseteq V$ where all pairs of vertices are adjacent in G. $G^* = (V^*, E^*)$ is a subgraph of $G = (V, E)$ if and only if $V^* \subseteq V$ and $E^* \subseteq E$.

In a complete undirected graph of n vertices, denoted as K_n, every pair of vertices is connected by an edge.

A tree is a rooted directed graph. A tree of n vertices has $n-1$ edges and no cycles. Except for the root, every tree vertex v_c has one incoming edge going from the parent vertex v_p to the child vertex v_c. The out-degree of a tree vertex is equal to the number of children. The tree vertices with out-degree equal to zero are the leaves of the tree, all other vertices are called internal vertices. The root vertex is the only vertex with an in-degree equal to zero, all other vertices have an in-degree equal to one. A binary tree is a tree where all internal vertices have an out-degree equal to two.

A k-partite graph $G = (V = \{V^0 \cup V^1 \cup ... \cup V^{k-1}\}, E)$ is a graph where the node set V is divided into k distinct partitions. The defining property of a k-partite graph is that $i \neq j$, $\forall e = \{v^i, v^j\} \in E$ with $v^i \in V^i$ and $v^j \in V^j$. In other words, vertices from the same partition are never adjacent. The 2-partite graph is also called bipartite graph. The complete k-partite graph is denoted as $K_{|V^0|,|V^1|,\cdots,|V^{k-1}|}$.

The complete De Bruijn graph is a directed graph consisting of all possible contiguous sequences of length k over a given alphabet Σ. It has $|\Sigma|^k$ vertices, one for each possible k-mer. The directed edges represent $(k+1)$-mers, that is, the source vertex represents the prefix of length k of the $(k+1)$-mer and the target vertex represents the suffix of length k of the $(k+1)$-mer. Hence, in the complete De Bruijn graph each vertex has an in-degree equal to $|\Sigma|$ and an out-degree equal to $|\Sigma|$. In this thesis, we will consider De Bruijn graphs, where the set of edges is limited to $(k+1)$-mers occurring in one of the input sequences.

A breadth-first search on a graph starts from a source vertex v_s and enumerates all other reachable vertices of the graph in ascending order of their distance from v_s. The distance is defined as the smallest number of edges on a path from v_s to a reachable vertex v_i. That is, the algorithm first enumerates all vertices at distance 0, which is simply v_s, followed by all vertices at distance 1, followed by the vertices at distance 2 and so on. Note that this procedure creates a so-called breadth-first search tree where v_s is the root. One level underneath the root are the vertices at distance 1 and each subsequent tree level corresponds to the vertices at this specific distance.

A depth-first search enumerates the vertices from a source vertex v_s by always going as deep as possible into the graph until no more undiscovered vertices exist

and a backtracking is required. Hence, we start at v_s, go to one adjacent vertex v_1 and mark it as discovered. From v_1 we go to another unmarked adjacent vertex v_2 and mark it as discovered. We continue this process until we reach a vertex v_k with no more adjacent unmarked vertices available. Then we backtrack to the last vertex with unmarked adjacent vertices and start to go deeper in the graph from that vertex. We repeat the whole process until all reachable vertices have been discovered.

A topological sort enumeration of the vertices of a graph can only be applied to a directed acyclic graph whereas breadth-first search and depth-first search can be applied to directed and undirected graphs that may contain cycles. A topological sort of a directed acyclic graph $G = (V, E)$ is an ordered enumeration of the vertices $(v_0, v_1, \cdots, v_k)$ such that for any edge $e = (v_i, v_j) \in E$ the vertex v_i appears before the vertex v_j in the ordered enumeration, that is, $i < j$.

A connected component of an undirected graph $G = (V, E)$ is a maximal set of vertices $V_C \subseteq V$ such that for each pair of vertices v_u and v_w there is a path from v_u to v_w. In a directed graph we require that there is a path from v_u to v_w and from v_w to v_u and call such a component strongly connected component. An undirected or directed graph G can be decomposed into its connected or strongly connected components, respectively. An undirected graph itself is called connected if it has only one connected component. Similarly, a directed graph is strongly connected if it has only one strongly connected component.

1.4.3 The SeqAn library

The LEDA library (Mehlhorn et al., 1999) for algorithms on graphs and efficient data types or the CGAL library (Overmars, 1996) for computational geometry have shown to be very successful in bringing down the required time for prototyping algorithms and applications. Because of these successes, Knut Reinert started developing a library called SeqAn (Döring et al., 2008) that aims at providing generic and integrated implementations of core algorithms and key data structures required in the sequence analysis domain of bioinformatics (see www.seqan.de). Throughout the past years, a number of algorithmic components are reoccurring in many sequence analysis applications, including methods such as string matching, alignment or filter algorithms and data structures such as suffix arrays, q-gram indices, alignment holders or graphs. A library-based provision of these indispensable algorithms

and data structures is the main goal of SeqAn and a large part of this thesis was devoted to augment SeqAn with new functionality.

SeqAn is a C++ library that makes heavily use of templates because we favored performance over the ease of use of an object-oriented library. We believe that the use of templates has a number of advantages compared to object-oriented programming in terms of performance, generality, integration and extensibility. In addition, a mechanism called template argument subclassing is able to mirror the neat object-oriented class derivation concept. This mechanism avoids, however, the additional runtime necessary for dynamic binding required in class derivation. The main SeqAn design principles (Döring et al., 2008) are briefly reviewed below.

1. **Generic programming**
   Generic programming code using templates can be easily optimized by the compiler since it favors static binding over dynamic binding and therefore avoids the overhead of calling virtual functions. This is a prerequisite for high performance algorithms to handle large input data sets.

2. **Metafunctions or traits classes**
   Generic algorithms usually have to know certain types associated with their input arguments. A classical example is a function reverse complementing a string that needs to know the alphabet type of the input string (DNA or RNA). A metafunction is a construct that maps types or constants to other C++ entities like types, constants, functions, or objects. Metafunctions are evaluated at compile time.

3. **Template argument subclassing**
   Different specializations of a given class template are specified by template arguments. For instance, `Graph<Directed<> >` is the directed specialization of a graph whereas `Graph<Undirected<> >` is used for an undirected graph. Hence, a single function can be overloaded using both specializations to adapt algorithms to the specific input graph.

This basic design proved to be very successful and a number of applications already made use of the core data structures and algorithms provided by the library (Schulz et al., 2008; Weese and Schulz, 2008; Weese et al., 2009; Rausch et al., 2008a,b, 2009; Langmead et al., 2009). A brief summary of the main components of the library is given below.

- **Sequences**
  Various string types, including an external string using secondary memory, a bit-packed string, a stack-allocated string and a data structure to store gapped strings; Sequence modifiers that provide distinct views of a given sequence, including an infix view or a reverse complement view.

- **Alignments**
  Pairwise alignment algorithms with configurable gap penalties (Needleman and Wunsch, 1970; Smith and Waterman, 1981); Local and global alignments; Progressive multiple sequence alignment (Feng and Doolittle, 1987); Algorithms for chaining alignment fragments and computing the longest and heaviest common subsequence (Jacobson and Vo, 1992); Various alignment data structures.

- **Indices**
  Enhanced suffix array (Kurtz et al., 2004); Gapped and ungapped q-gram indices; Lazy suffix trees (Giegerich et al., 2003); An index for frequency based string mining (Weese and Schulz, 2008); Algorithms on these indices to iterate through the suffix tree, finding maximal repeats, super maximal repeats or maximal unique matches (Kurtz et al., 2004).

- **Searching**
  Various algorithms for exact or approximate string matching, including the Horspool, Shift-AND, Shift-OR, Pex, BOM, BNDM, Aho-Corasick, Myer's bit vector algorithm and DP based algorithms (Navarro and Raffinot, 2002); SWIFT (Rasmussen et al., 2005) filter algorithm; Indexed based algorithms for exact string matching.

- **Biologicals**
  DNA, RNA and amino acid alphabets; Scoring matrices, including BLOSUM (Henikoff and Henikoff, 1992) and PAM (Dayhoff et al., 1979); Various file formats; Modified alphabets to store gaps.

- **Graphs**
  Various graph types, including directed and undirected graphs as well as trees and automata; Basic graph algorithms, including minimum spanning tree, connected components, shortest path and network flow algorithms (Cormen et al., 2001).

- **Miscellaneous**
  Pipelining architecture for efficient external algorithms; Allocator classes; Fundamental data structures such as Heap, Map or a Priority queue; External memory strings and memory mapped strings.

Multiple Sequence Alignments

Over the past years, numerous research projects have contributed to a steady progress in the area of MSA. Albeit such a long history, methods are still far from being optimal in a biological sense. The main obstacles are (1) that we still lack a precise mathematical formulation of such a biologically optimal alignment and (2) that the problem is already NP-hard if we use a very simplified formulation such as the alignment score maximization. This very question of finding an alignment of maximum score has driven the field significantly in the past years and many sequence based methods, both heuristics and optimal ones, have been developed to solve this problem. The nuts and bolts of these methods are described in-depth in Section 2.4 and Section 2.5. Recently, the sequence based methods have been complemented by methods that go beyond the raw sequence data. These structure based methods use a great variety of structure prediction methods and databases with structural information. The goal is either to substantiate a possibly weak signal of sequence similarity or to identify novel domains where conservation only manifests itself on a structural level. The progress in alignment methods was accompanied by the development of different computational models to represent a MSA. Alignment representations range from the classical alignment matrix, over profiles to the increasingly popular De Bruijn graphs (see Section 2.3).

The genomic sequencing efforts demanded new methods to align and compare very large sequences and methods that are able to build overlap alignments out of thousands of reads. Some properties of these novel algorithms are introduced at the end of this chapter. We conclude this review of available MSA approaches with a listing of available programs categorized by the nature of the used alignment algorithm.

2.1 Alignment Definition

The predominant representation of an alignment is the well-known alignment matrix. An example is shown on the right in Figure 2.1. Based upon that matrix, we can formally define the properties of a multiple alignment of n sequences $\mathcal{S} = \{S^0, S^1, \ldots, S^{n-1}\}$.

- $S^i \in \mathcal{S}$ is a string over the finite ordered alphabet Σ, that is $S^i \in \Sigma^*$. Σ is, for instance, the DNA or amino acid alphabet. Each string S^i is a sequence of letters $s_0^i s_1^i \ldots s_{|S^i|-1}^i$ of length $|S^i|$ where $s_u^i \in \Sigma$.
- The alphabet $\tilde{\Sigma} = \Sigma \cup \{-\}$ is the extended alphabet including a gap '$-$' symbol.

A multiple alignment $\mathcal{A}$ of the strings in $\mathcal{S}$ is a $n \times l$ matrix consisting of n strings $\tilde{S}^0, \tilde{S}^1, \ldots, \tilde{S}^{n-1} \in \tilde{\Sigma}^*$ such that

- The strings $\tilde{S}^0, \tilde{S}^1, \ldots, \tilde{S}^{n-1}$ are of length l.
- The string $\tilde{S}^i$ with gaps removed is equal to S^i.
- The matrix entry a_u^i in row i and column u is either from the alphabet Σ or equal to the gap character '$-$': $\quad a_u^i \in \tilde{\Sigma} \quad \forall\, 0 \leq i < n,\ 0 \leq u < l$
- No column consists entirely of gap characters. This implies:

$$\max_{i=0,\ldots,n-1} |S^i| \leq l \leq \sum_{i=0,\ldots,n-1} |S^i|$$

Alternatively, one can think of an alignment as a path through an n-dimensional lattice as shown in Figure 2.1.

A given alignment $\mathcal{A}$ can be projected onto a sequence subset $\mathcal{I} \subseteq \{0, 1, \cdots, n-1\}$. In the matrix representation, the projected alignment $\mathcal{A}^{\mathcal{I}}$ can be obtained by

1. Selecting row i in $\mathcal{A}$ if and only if $i \in \mathcal{I}$.
2. Deleting column u in $\mathcal{A}^{\mathcal{I}}$ if and only if column u contains only gap characters.

Hence, a pairwise alignment projection is a mere selection of 2 distinct rows in a given alignment and a subsequent removal of all columns containing only gaps. For

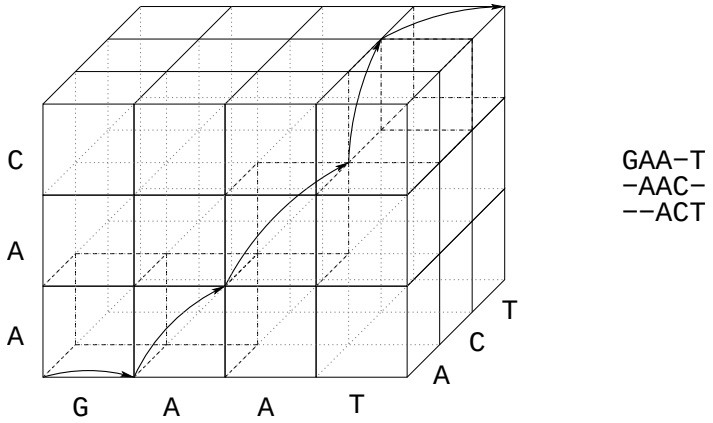


Figure 2.1: The MSA path in a 3-dimensional lattice corresponding to the alignment shown on the right.

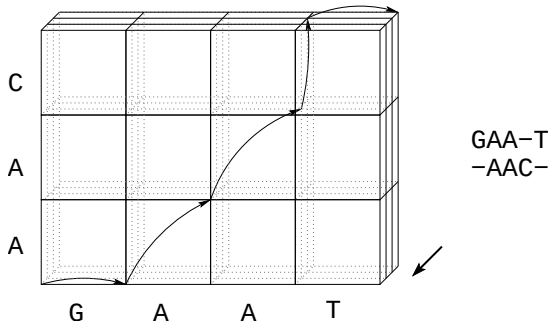


Figure 2.2: A projection of a 3-dimensional lattice to a 2-dimensional matrix corresponding to the projection of an alignment of 3 sequences onto a subset of 2 sequences.

example, the projection $\mathcal{A}^{\{0,1\}}$ of the alignment in Figure 2.1 results in the pairwise alignment:

$$\begin{array}{cccccc} \mathrm{G} & \mathrm{A} & \mathrm{A} & - & \mathrm{T} \\ - & \mathrm{A} & \mathrm{A} & \mathrm{C} & - \end{array}$$

This definition of an alignment projection respects the requirement that $\mathcal{A}^{\{i\}} = S^i$. Similarly, one can project the path through the n-dimensional lattice onto a subspace as shown in Figure 2.2.

2.2 Alignment Scoring

Given a number of different multiple alignments for a set of sequences, we need a quantitative measure to decide which one is the best. For exact methods the ubiquitously used measure is the sum of pairs multiple alignment score, which is an

extension of the pairwise alignment score to more than two sequences.

2.2.1 Sum of pairs score

The most common pairwise scoring function uses linear gap costs. Linear gap costs penalize a gap of length γ with a cost of $g + e \cdot (\gamma - 1)$ where g is the constant gap opening penalty for the first gap and e is the constant gap extension penalty for each subsequent gap, with $g \leq e$ and $g, e \leq 0$. If $g = e$ the number of gap openings is irrelevant and such gap costs are called constant hereafter. Using linear gap costs, the score of a pairwise alignment is

$$Score(\mathcal{A}^{\{i,j\}}) = \Bigg(\sum_{\substack{u=0,\ldots,\tilde{l}-1 \\ a_u^i \neq -;\ a_u^j \neq -}} \delta(a_u^i, a_u^j) \Bigg) + g \cdot \ \#\text{GapOpen} \ + e \cdot \ \#\text{GapExtension}$$

where $\tilde{l}$ is the length of the projected alignment and δ a scoring function or substitution matrix for all pairs of characters $a_u^i, a_u^j \in \Sigma$. The BLOSUM (Henikoff and Henikoff, 1992) and PAM matrices (Dayhoff et al., 1979) are commonly used substitution matrices for protein alignments. For DNA alignments most tools use a simple match / mismatch scoring function. The alignment

G A T A T A − − T
− A T G T A C C −

evaluated with linear gap costs (gap opening penalty $g = -4$, gap extension penalty $e = -1$) and a scoring function defined by a match score of $\delta(x, x) = 4$ and a mismatch score of $\delta(x, y) = -2$ results in a total score of $14 + (-4) \cdot 3 + (-1) \cdot 1 = 1$. The sum of pairs multiple alignment score SP_{Score} can be simply defined as

$$SP_{Score}(\mathcal{A}) = \sum_{0 \leq i < j < n} Score(\mathcal{A}^{\{i,j\}})$$

Besides the sum of pairs score other quantitative alignment quality measures are available, most notably the weighted sum of pairs score, the tree alignment score and the consensus score (Gotoh, 1999). These alignment scores are, however, less common in practice.

A	G	A	C
T	–	A	C
T	–	–	C
T	G	A	C

$\mathcal{P}$	1	2	3	4
A	0.25	0	0.75	0
C	0	0	0	1.0
G	0	0.5	0	0
T	0.75	0	0	0
–	0	0.5	0.25	0

Table 2.1: A Profile $\mathcal{P}$ of an alignment matrix $\mathcal{A}$. $\mathcal{P}_{a,u}$ is the frequency of character $a \in \tilde{\Sigma}$ in column u of $\mathcal{A}$.

2.3 Alignment Representation

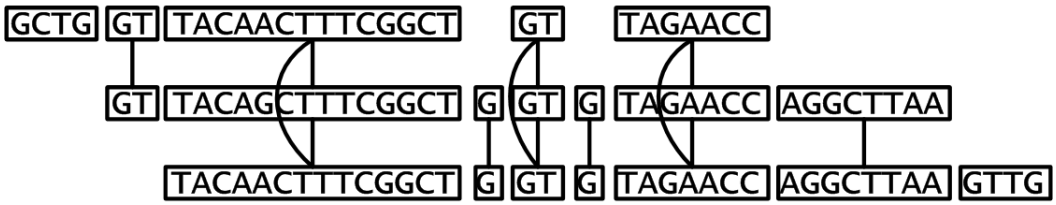
Besides the classical alignment matrix representation, a number of other computational models has been used to represent MSAs. The most important ones are briefly reviewed in this section, namely profiles, alignment graphs, De Bruijn graphs and partial order graphs.

2.3.1 Alignment matrices and profiles

The classical alignment matrix has one row for each sequence. A condensed view of such an alignment matrix is a profile. An alignment profile of a multiple alignment $\mathcal{A}$ of length l is a $|\tilde{\Sigma}| \times l$ matrix $\mathcal{P}$, where $\mathcal{P}_{a,u}$ is the frequency of character $a \in \tilde{\Sigma}$ in column u of $\mathcal{A}$. An example is shown in Table 2.1. Profiles are frequently used as a fast method to align two subalignments and we will review this application later in this chapter.

2.3.2 Alignment graphs

An alignment can be represented as an alignment graph of sequence segments as shown in Figure 2.3. The alignment edges of this graph represent possible matching sequence segments. Gaps are implicitly represented by the topology of the graph. For instance, a vertex without any outgoing edge is aligned to gaps in all other sequences. An alignment graph can be easily converted into a classical alignment matrix using standard graph algorithms, namely connected components and topological sort (Cormen et al., 2001). In contrast to the alignment matrix, the graph makes no assumption about the order of adjacent insertions and deletions. The



GCTG	GT	TACAACTTTCGGCT	-	GT	-	TAGAACC	--------	----
----	GT	TACAGCTTTCGGCT	G	GT	G	TAGAACC	AGGCTTAA	----
----	--	TACAACTTTCGGCT	G	GT	G	TAGAACC	AGGCTTAA	GTTG

Figure 2.3: An alignment graph of three sequences (top) and its corresponding alignment matrix (bottom). The aligned ungapped sequence segments do not have to be identical. They may contain mismatches.

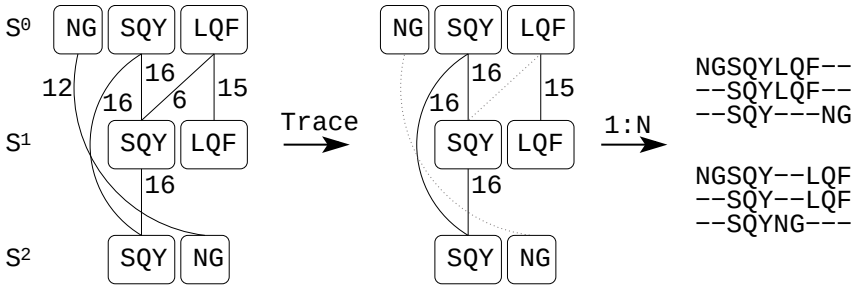


Figure 2.4: An alignment graph of three amino acid sequences with arbitrary alignment edges (left), its best subset of edges called trace (middle) and its conversion to an alignment matrix (right). The alignment graph does not define the order of adjacent indels.

strength of the alignment graph representation is, however, that the graph can contain arbitrary match information. Because of that, the graph can be used as a store for all putative aligned segments. In the remainder of this thesis, we will encounter different methods that use such an initial alignment graph with possibly contradicting edges as input. All of these methods then select a subset of edges, called a trace, that constitute an alignment as shown in Figure 2.4. The simplest solution is to select the subset of edges of maximum cardinality. A more practical approach is to augment the alignment graph by edge weights that capture some kind of quantitative measure of alignment quality. The objective is then to find the maximum weight trace (Kececioglu, 1993), that is, the subset of edges with maximum weight.

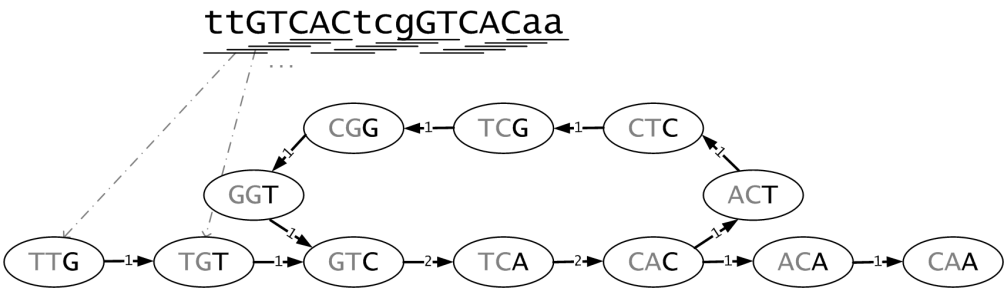


Figure 2.5: A De Bruijn graph of a single sequence using a k-mer of length 3. Each k-mer is mapped to a vertex and two vertices are connected if and only if the corresponding $(k+1)$-mer is present in the sequence. Note how the two edges with multiplicity 2 (GTCA and TCAC) cover the repeat shown in upper case letters.

2.3.3 De Bruijn graphs

Given long matching sequence segments, the alignment graph packs a large-scale alignment of long sequences into a few matching vertices. The De Bruijn graph is somehow orthogonal to that approach since it is able to pool an alignment of thousands of sequences (Zhang and Waterman, 2003). In other words, alignment graphs enable a compact representation of long alignments whereas De Bruijn graphs enable the compact representation of deep alignments. De Bruijn graphs are constructed by subdividing each sequence into overlapping k-mers, where a k-mer is a contiguous substring of length k. Two adjacent k-mers overlap by $k-1$ letters. For each distinct k-mer, we define a vertex v_k. Vertices are connected by an edge if there is at least one pair of adjacent k-mers present in one of the sequences. In other words, the $(k+1)$-mer defined by two adjacent vertices must be present in one of the sequences. The edge weight usually indicates how often this $(k+1)$-mer was observed. An example of such a De Bruijn graph for a single sequence is shown in Figure 2.5. Note how a De Bruijn graph of a single sequence is able to highlight repeats. In a De Bruijn graph of multiple sequences, each original input sequence is mapped to a path traversing the graph. Conserved subsequences are highlighted by heavy weight edges, that is, edges that are part of almost all sequence paths. Hence, the De Bruijn graph can be used to extract these conserved patterns (Zhang and Waterman, 2003) as shown in Figure 2.6. Similarly, the De Bruijn graph can be used to find the consensus sequence of thousands of reads and thus, it can be used as a computational model for an assembler (Zerbino and Birney, 2008). A practical k-mer size is usually larger than 20 to avoid spurious read overlaps.

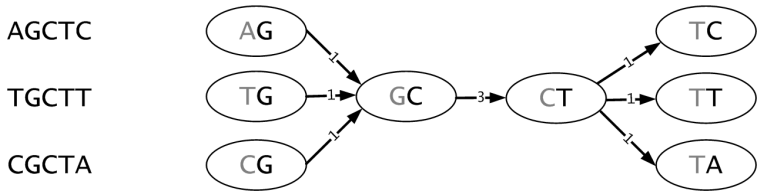


Figure 2.6: A De Bruijn graph of three sequences using a k-mer of length 2. The conserved pattern GCT is represented by the edge of weight 3.

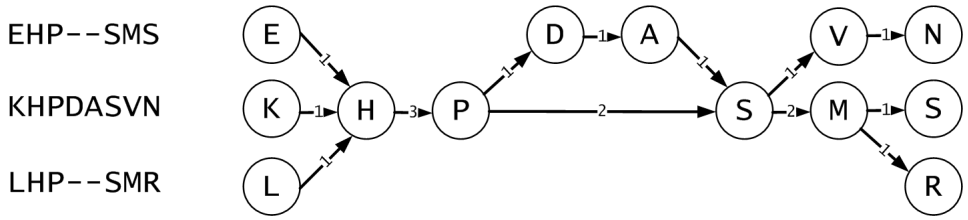


Figure 2.7: A partial order graph (right) for a multiple sequence alignment (left).

2.3.4 Partial order graphs

An individual sequence can be represented by a trivial partial order graph. Each character is converted to a vertex and all vertices have exactly one outgoing edge to the vertex for the subsequent character, except for the vertex of the last sequence character. In a MSA these single-sequence graphs are merged. Similar to the De Bruijn graph, each sequence is mapped to a path traversing the graph but since it is a partial order graph no cycles are allowed. That is, the partial order graph is a directed acyclic graph. Consequently, the partial order graph enforces the collinearity condition and does not allow two crossing aligned regions where the order of characters in the sequences is not preserved. The key characteristic of the partial order graph is that matching sequence characters are merged to a single node whereas mismatches cause the graph to bifurcate. This is shown in Figure 2.7.

2.4 Exact Alignment Algorithms

Alignment algorithms can be broadly classified into exact algorithms and heuristic algorithms. In this section, we first review the exact algorithms that solve the multiple alignment score maximization problem optimally using either natural extensions of the dynamic programming algorithm (Gupta et al., 1995; Lermen and Reinert, 2000; Lipman et al., 1989; Reinert et al., 1997, 2000) or a graph theoretic formula-

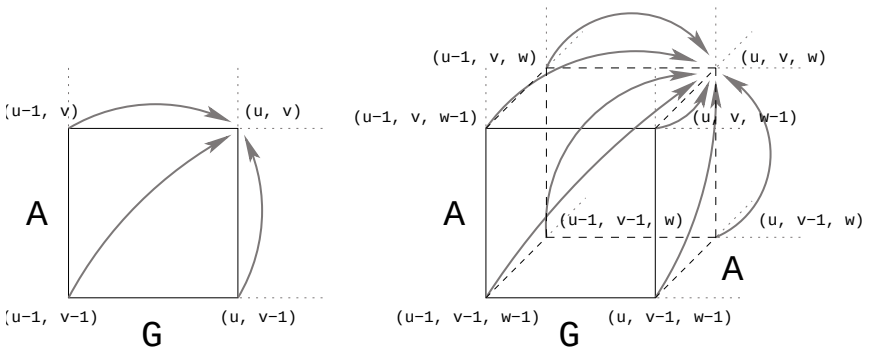


Figure 2.8: In each cell of the dynamic programming matrix / cube $(2^n - 1)$ predecessor have to be evaluated where n is the number of sequences.

tion that facilitates the use of combinatorial algorithms (Althaus and Canzar, 2008; Althaus et al., 2002, 2006; Reinert, 1999).

In practice, however, optimal methods are only feasible for a few, relatively short sequences. Hence, many fast and accurate heuristics to solve the multiple alignment problem have been proposed. We review the most important heuristic, the progressive alignment strategy (Feng and Doolittle, 1987), extensively in the next Section 2.5, including recent additions such as consistency and refinement.

2.4.1 Dynamic programming

The dynamic programming recursion to compute the optimal pairwise alignment between sequence $S^0 = s_0^0 s_1^0 \dots s_{|S^0|-1}^0$ and sequence $S^1 = s_0^1 s_1^1 \dots s_{|S^1|-1}^1$ is

$$M_{u,v} = \max \begin{cases} M_{u-1,v-1} + \delta(s_u^0, s_v^1) \\ M_{u-1,v} + \delta(s_u^0, -) \\ M_{u,v-1} + \delta(-, s_v^1) \end{cases}$$

where $M_{u,v}$ is the 2-dimensional dynamic programming matrix and δ is the scoring function. For a constant gap penalty e and the Blosum_{62} substitution matrix one could define δ as $\delta(s_u^0, s_v^1) = \text{Blosum}_{62}(s_u^0, s_v^1)$ and $\delta(s_u^0, -) = \delta(-, s_v^1) = e$. The extension to 3 sequences involves two changes. First, a 3-dimensional dynamic programming lattice has to be computed and second, for each entry we have to evaluate $(2^n - 1) = (2^3 - 1) = 7$ predecessors as shown in Figure 2.8. The recursion

is

$$M_{u,v,w} = \max \begin{cases} M_{u-1,v-1,w-1} + \tilde{\delta}(s_u^0, s_v^1, s_w^2) \\ M_{u,v-1,w-1} + \tilde{\delta}(-, s_v^1, s_w^2) \\ M_{u-1,v,w-1} + \tilde{\delta}(s_u^0, -, s_w^2) \\ M_{u-1,v-1,w} + \tilde{\delta}(s_u^0, s_v^1, -) \\ M_{u,v,w-1} + \tilde{\delta}(-, -, s_w^2) \\ M_{u-1,v,w} + \tilde{\delta}(s_u^0, -, -) \\ M_{u,v-1,w} + \tilde{\delta}(-, s_v^1, -) \end{cases}$$

For the sum of pairs score with constant gap costs, $\tilde{\delta}$ can be defined in terms of δ as $\tilde{\delta}(a,b,c) = \delta(a,b) + \delta(b,c) + \delta(a,c)$ with $a,b,c \in \tilde{\Sigma}$ and $\delta(-,-) = 0$. This can be extended to higher dimensions n. As in the pairwise case, the key idea is that larger alignments are constructed from already computed subsolutions. Any $M_{u,v,\ldots,z}$ is the best score of aligning the prefixes $s_0^0 s_1^0 \ldots s_u^0$, $s_0^1 s_1^1 \ldots s_v^1$, …, $s_0^{n-1} s_1^{n-1} \ldots s_z^{n-1}$. In addition, the optimal alignment can be retrieved through the standard traceback operations extended to the n-dimensional lattice. Note that it is also possible to apply Gotoh's algorithm (Gotoh, 1982) for linear gap costs to more than two sequences. Similar to the pairwise case, we then require additional lattices for the best gapped alignment in each dimension. The size of the lattice is exponential in the number of sequences $\mathcal{O}(\prod_{i=0}^{n-1} |S^i|)$. For each cell of this lattice, $(2^n - 1)$ predecessor cells have to be evaluated. Thus, the time complexity is $\mathcal{O}((2^n - 1) \cdot \prod_{i=0}^{n-1} |S^i|)$ if and only if the computation of the δ function is constant $\mathcal{O}(1)$. This is roughly $\mathcal{O}((2\tilde{n})^n)$ where $\tilde{n}$ is the average sequence length. Bounding techniques try to minimize the actually computed lattice alignment space by using lower and upper bounds (Gupta et al., 1995; Lermen and Reinert, 2000; Lipman et al., 1989; Reinert et al., 1997) or a combination of an exact algorithm with a heuristic divide and conquer approach (Reinert et al., 2000).

2.4.2 Combinatorial algorithms

In Section 2.3.2 we introduced the alignment graph. We showed a graph with arbitrary alignment edges in Figure 2.4, where only a subset of the edges can be realized in an actual alignment. This subset is called a trace (Sankoff and Kruskal, 1983). Augmenting the graph with edge weights that capture the alignment quality of a given match leads to the maximum weight trace problem (Kececioglu, 1992, 1993).

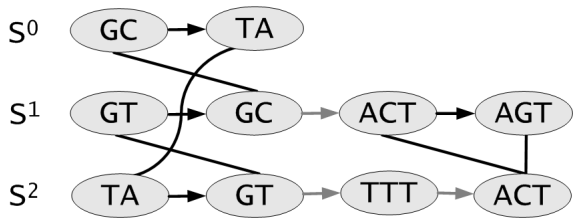


Figure 2.9: An alignment graph augmented by directed edges connecting adjacent vertices in each of the sequences. The graph contains two critical mixed cycles shown in black.

Given an alignment graph $G = (V, E)$ and edge weights $w_{e \in E}$ the maximum weight trace problem is

$$\max \sum_{e \in T} w_e \text{ where } T \subseteq E \text{ is a trace.}$$

Since the graph theoretic formulation has a favorable combinatorial structure - an edge is either part of the solution or not - it can be solved by methods from combinatorial optimization (Reinert, 1999; Kececioglu et al., 2000) such as integer linear programming (ILP). An ILP formulation of the maximum weight trace problem introduces a binary variable $x_{e \in E}$ for each edge. It indicates whether edge e is part of the trace or not and hence, the objective is $\max \sum_{e \in E} w_e x_e$. Additional linear inequalities have to ensure that the selected edges constitute a valid alignment. Let T be the set of selected edges then T is a valid trace if and only if there is no critical mixed cycle in $G^* = (V, T \cup H)$ (Reinert, 1999). G^* is the original graph G with all edges $e \notin T$ removed. G^* is further augmented by a set of directed edges H connecting two adjacent vertices in a single sequence as shown in Figure 2.9. A mixed cycle in G^* is an alternating sequence $C = (v_0, e_0, v_1, e_1, \cdots, v_k)$ of distinct vertices and edges where $e_i \in T$ or $e_i \in H$, $v_i \in V$, $v_0 = v_k$ and at least two edges from T and one edge from H. Such a cycle C is called critical if all vertices $v^i \in C$ that are on the same sequence S^i appear consecutively in C. Hence, a *valid* solution to the maximum weight trace problem can be found by the following ILP.

$$\begin{aligned} & \max \textstyle\sum_{e \in E} w_e x_e \\ \text{subject to:} \quad & \textstyle\sum_{e \in C \cap E} x_e \leq |C \cap E| - 1, \forall C \text{ (critical mixed cycles)} \\ & x_e \in \{0, 1\} \end{aligned}$$

The alignment graph can be further extended with so-called gap arcs to incorporate positional gap penalties (Reinert, 1999).

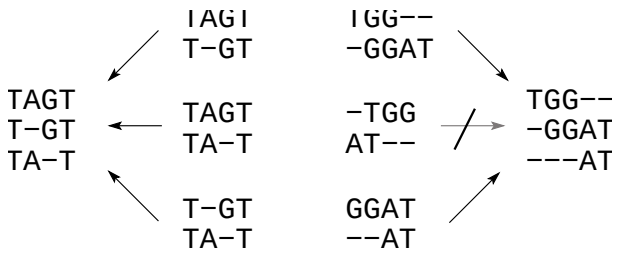


Figure 2.10: A set of pairwise alignments that are compatible (left) or incompatible (right).

There is a great variety of combinatorial optimization methods that can be applied to solve ILPs. One such approach that can be applied to the above graph theoretic model is branch-and-cut (Althaus et al., 2002, 2006). Recently, also a Lagrangian relaxation approach was proposed to solve the ILP formulation even more efficiently (Althaus and Canzar, 2008).

2.5 Heuristic Alignment Algorithms

The rapid accumulation of sequence data demanded the development of heuristic methods that are able to align more sequences of greater length than the optimal methods with exponential runtime. For an ordinary protein alignment, the predominant heuristic strategy is called progressive alignment.

For genomic DNA sequences, most tools use a heuristic called anchor-based alignment. Besides the progressive and the anchor-based alignment strategy, we also briefly review in this section the novel structural aligners. Structural aligners go beyond the raw sequence data and take into account protein structure information.

2.5.1 Progressive alignment

A sound multiple alignment of n sequences should induce $(\frac{n\cdot(n-1)}{2})$ projected pairwise alignments that are as close as possible to the optimal pairwise alignments. Unfortunately, pairwise alignments may be incompatible as shown in Figure 2.10. Progressive alignment resolves these inconsistencies in a greedy manner. The multiple alignment is started from the most similar pair and then gradually, the other less similar sequences are added to the growing alignment. The intuitive assumption is that a pairwise alignment of closely related sequences is more trustable than an

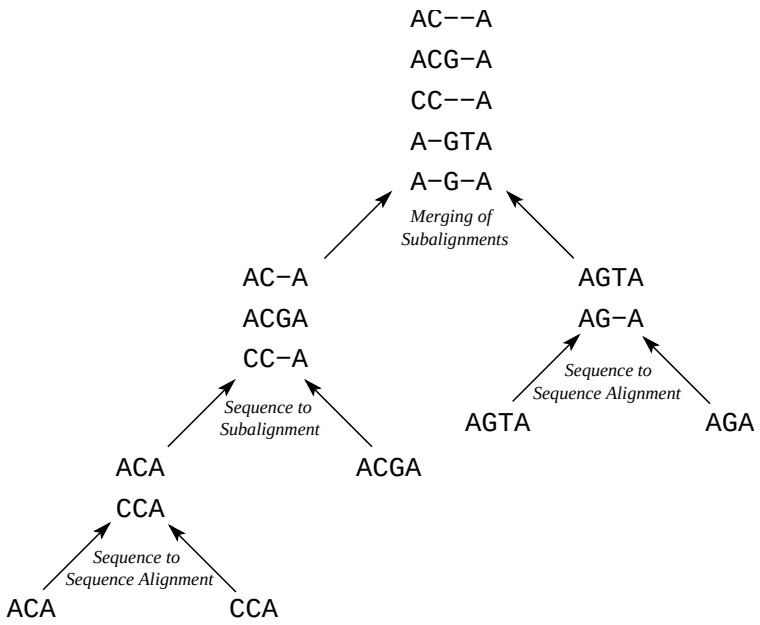


Figure 2.11: The progressive alignment greedily builds a final alignment along the guide tree using a given method to merge subalignments.

alignment of distantly related sequences (Feng and Doolittle, 1987). The method thus requires 2 things. First, a binary tree, called guide tree, that indicates when every sequence (a leaf of the tree) is merged into a growing multiple alignment and second, a means of aligning already finished subalignments with another sequence or another subalignment. The latter situation arises if the progressive alignment is started from multiple seeding alignments as shown in Figure 2.11.

The guide tree can be obtained in 2 steps. First, a distance score between all pairs of sequences is computed and second, the phylogenetic tree is reconstructed using clustering methods such as UPGMA (Sokal and Michener, 1958) or neighbor-joining (Saitou and Nei, 1987). Several pairwise distance measures are in common use. Examples are the percent identity between two sequences or the fractional number of common k-mers where a k-mer is a contiguous substring of length k. For large alphabets the percent identity and the number of common k-mers are less applicable, unless the sequences are closely related or both measures are applied over a compressed alphabet (Edgar, 2004c). More precise measures are based upon pairwise global or local alignment scores (Needleman and Wunsch, 1970; Smith and Waterman, 1981), which are usually normalized by alignment length.

UPGMA is, besides neighbor-joining, a widely used distance based tree reconstruction method. The algorithm requires a set of n elements (e.g. sequences) and

all pairwise distances $d_{i,j}$. Initially, each element is in its own group and thus, the sequences are the leaves of the tree. The algorithm proceeds in 4 steps:

1. Select the minimum distance $d_{i,j}$.
2. Create a new group u that joins i and j.
3. Compute the distances $d_{k,u}$ of any group k to the new group u.
4. Remove i, j from the set of elements. Go to step 1 if more than a single group is left, otherwise terminate.

The UPGMA algorithm reconstructs the correct tree only for ultrametric distances. Such distances imply that all sequences have evolved from a common ancestor at constant rate. This assumption is, in general, not true and thus, UPGMA is not used very often in phylogenetic studies. It is, however, widely used in progressive alignment tools because some authors argue (Edgar, 2004b) that a reliable evolutionary tree is not as important as a tree that guarantees that the subalignments with the fewest differences are merged first. In step (3) the new distance $d_{k,u}$, from any group k to the new group u that joined i and j, can be computed using different methods:

1. Single linkage clustering: $d_{k,u} = \min(d_{k,i}, d_{k,j})$
2. Complete linkage clustering: $d_{k,u} = \max(d_{k,i}, d_{k,j})$
3. Average linkage clustering: $d_{k,u} = \frac{d_{k,i}+d_{k,j}}{2}$
4. Weighted average linkage clustering: $d_{k,u} = \frac{n_i \cdot d_{k,i}+n_j \cdot d_{k,j}}{n_i+n_j}$

In the last method n_i and n_j are the number of elements in group i and j, respectively.

The neighbor joining method has a time complexity of $\mathcal{O}(n^3)$ compared to $\mathcal{O}(n^2)$ for the UPGMA algorithm, where n is the number of sequences. The guide tree obtained with that method is, however, regarded as a better evolutionary tree because the neighbor joining method does not assume a molecular clock. The idea of the method is to start with a star tree that has a single root with n children for n sequences. The algorithm then gradually groups pairs of sequences so that the overall tree length is minimized.

The final guide tree obtained with the UPGMA or neighbor joining algorithm is then used to progressively align all input sequences. Aligning the children of an

internal node in the guide tree either involves an ordinary sequence alignment or an alignment of subalignments. In the latter case, one possible objective is to optimize the already mentioned sum of pairs multiple alignment score.

$$SP_{Score}(\mathcal{A}) = \sum_{0 \leq i < j < n} Score(\mathcal{A}^{\{i,j\}})$$

Using linear gap costs, an optimal merging of subalignments is NP-complete (Kececioglu and Starrett, 2004; Ma et al., 2003). Sophisticated exact algorithms can, however, be reasonable fast in practice (Kececioglu and Starrett, 2004). Other methods favor speed over optimality and use approximations of gap opening counts (Kececioglu and Zhang, 1998). More often, however, practical tools use their own way of merging subalignments with quite different objective functions (Edgar and Sjolander, 2004). These methods are usually subsumed under the generic term profile-profile alignments.

Assuming constant gap costs, a string $S = s_0 s_1 \cdots s_{|S|-1}$ can be quickly aligned to a profile with a standard pairwise dynamic programming algorithm. Only the scoring function δ has to be adapted.

$$\delta_{\text{New}}(s_w, u) = \sum_{a \in \tilde{\Sigma}} \mathcal{P}_{a,u} \cdot \delta(s_w, a)$$

In this case, δ_{New} scores a column u against a character $s_w \in \Sigma$. The δ function has to be extended to handle the special case of scoring a gap character against another gap character.

$$\delta(a,b) = \begin{cases} \text{Blosum}_{62}(a,b) & \text{if and only if } a,b \in \Sigma \\ e & \text{if and only if } a = \text{"}-\text{"} \text{ or } b = \text{"}-\text{"} \\ 0 & \text{if and only if } a = b = \text{"}-\text{"} \end{cases}$$

Note that in a projected alignment gap columns are removed and hence, the score for two aligned gaps is set to 0. An example of a string to profile alignment is shown in Figure 2.12. Constant gap penalties simplify the sum of pairs score of a multiple alignment $\mathcal{A}$ of length l to

$$SP_{Score}(\mathcal{A}) = \sum_{0 \leq i < j < n} Score(\mathcal{A}^{\{i,j\}}) = \sum_{i,j} \sum_{u=0}^{l-1} \delta(\tilde{s}_u^{\,i}, \tilde{s}_u^{\,j}) = \sum_{u=0}^{l-1} \sum_{i,j} \delta(\tilde{s}_u^{\,i}, \tilde{s}_u^{\,j})$$

The last equality stems from the independence of the alignment columns using the δ scoring function with constant gap penalties. Using dynamic programming, the

A	G	–	C	T
A	G	–	C	C
A	–	–	C	A
C	G	–	C	A
A	–	C	C	A

$\mathcal{P}$	1	2	–	3	4
A	0.75	0		0	0.5
C	0.25	0		1.0	0.25
G	0	0.75		0	0
T	0	0		0	0.25
–	0	0.25	1.0	0	0

S	A	–	C	C	A
δ_{New}	2.25	−1.5	−2	4	0.5

Figure 2.12: A string to profile alignment of the string $S = ACCA$ and the profile shown above. Assuming $\delta(x, x) = 4$, $\delta(x, y) = -3$, $\delta(x, -) = \delta(-, x) = -2$ and $\delta(-, -) = 0$ the score of the full string to profile alignment is 3.25.

optimal string to profile alignment can be found in quadratic time $\mathcal{O}(|\tilde{\Sigma}| \cdot l \cdot |S|)$ where l is the length of the profile, $|S|$ the length of the sequence and $|\tilde{\Sigma}|$ a small constant, e.g., 5 for the DNA alphabet or 21 for the amino acid alphabet including a gap character. Similarly, a profile-profile alignment can be carried out. The only difference is an extra sum over the alphabet $\tilde{\Sigma}$.

$$\delta_{\text{New}}(u, w) = \sum_{a \in \tilde{\Sigma}} \sum_{b \in \tilde{\Sigma}} \mathcal{P}_{a,u} \cdot \mathcal{P}_{b,w} \cdot \delta(a, b)$$

Numerous other profile-profile column scoring functions have been published (Vingron and Argos, 1989; Edgar, 2004b; Edgar and Sjolander, 2004; Katoh et al., 2002).

In summary, an optimal merging of subalignments with linear gap costs $g + e \cdot (\gamma - 1)$ is NP-complete. A merging with $g = e$ remains polynomial because gap opening counts are irrelevant. In this case, each column can be treated as a meta-character in an extended alphabet. Given a scoring function for such meta-characters, the problem is to find an alignment of two strings of meta-characters, which is clearly solvable with a pairwise dynamic programming algorithm.

Consistency and refinement

The choice of the binary guide tree and the method to merge subalignments has great influence on the final alignment. Once a new sequence is added to the growing alignment all the aligned characters and inserted gaps are fixed ("Once a gap, always

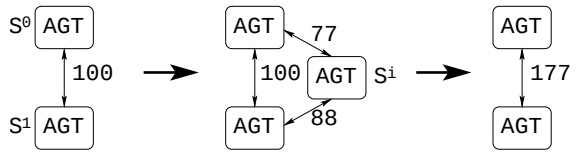


Figure 2.13: A possible means of consistency extension: Every supported alignment is increased by the minimum of the two connecting edges.

a gap." (Feng and Doolittle, 1987)). But this is also true for alignment errors: once made they are preserved and they may even cause new alignment errors in the subsequent progressive steps. There are two strategies, called consistency and refinement, to handle alignment errors, one aims at preventing errors and the other one aims at correcting errors (Wheeler and Kececioglu, 2007). The prevention approach tries to substantiate pairwise alignments by multiple sequence information. That is, it tries to make pairwise alignments consistent with all the other sequences and hence, the name consistency (Gotoh, 1990; Vingron and Argos, 1991; Notredame et al., 2000). The refinement approach takes a possibly erroneous alignment, iteratively splits this alignment into two subalignments and merges these alignments together again. These methods, thus, iteratively refine or realign a given alignment. In other publications authors sometimes use the term iterative alignment to describe such techniques (Pirovano and Heringa, 2008).

Although current algorithms use slightly different means of consistency the basic idea is always the same: the confidence of aligning substrings of a pair of sequences S^0 and S^1 is the greater, the more intermediate sequences S^i support this alignment. In other words, the alignments $S^0 \leftrightarrow S^i$ and $S^i \leftrightarrow S^1$ induce a putative transitive alignment $S^0 \leftrightarrow S^1$ that is either consistent or inconsistent with a precomputed alignment of S^0 and S^1. If it is consistent, greater confidence in the alignment of these substrings of S^0 and S^1 is established and the scores are somehow increased. In an alignment graph, this consistency extension or triplet extension (Notredame et al., 2000) corresponds to a search for three-way cliques (see Figure 2.13).

The refinement approach (Edgar, 2004b; Katoh et al., 2002) splits a full alignment randomly or following a deterministic order into subalignments and then merges these subalignments using, for example, profile-profile alignment methods. Random cutting is usually stopped if no improvement in alignment score was observed during a fixed number of past iterations.

2.5.2 Methods using structure and sequence homologs

The improvements in de-novo structure prediction methods and the growth of sequence and structural databases opened up new possibilities to extend the sequence based alignment methods. These extended methods tend to deliver more accurate alignments on standard benchmarks (see Section 2.7), especially in the so-called twilight zone of highly diverged sequences with less than 20% identity. Three combinable techniques are in common use: homology extension (Katoh et al., 2005; Pei and Grishin, 2007; Simossis et al., 2005; Zhou and Zhou, 2005), secondary structure prediction (Pei and Grishin, 2006, 2007; Simossis and Heringa, 2005; Zhou and Zhou, 2005) and the use of a known 3D structure (O'Sullivan et al., 2004; Pei et al., 2008).

Homology extension augments the raw sequence information using, for example, database searches with PSI-BLAST (Altschul et al., 1997). Given a set of retrieved database homologs, a profile can be built for each input sequence. The profiles can then be readily used in the progressive alignment as outlined in Section 2.5.1. The use of profiles turned out to be beneficial because profiles differentiate between conserved and variable sites.

Predicted or known secondary structures can further improve the alignment quality because, in most cases, structure is more conserved than sequence information. Structural elements can be predicted, for instance, with PSIPRED (McGuffin et al., 2000) or similar tools (Rost, 2001). The pairwise sequence alignment is then carried out under structural constraints. For instance, one could add a simple secondary structure weight function to the profile-profile alignment that indicates if the two corresponding structural elements at a given position match or mismatch.

Similarly, a known 3D structure eases the alignment of highly diverged sequences. Methods such as SAP (Taylor, 1999) employ a double dynamic programming algorithm to compute a structural alignment. The time complexity is, however, $\mathcal{O}(\tilde{n}^4)$ where $\tilde{n}$ is the average sequence length. Hence, structure based methods are usually significantly slower than sequence based heuristics. Results are, however, highly accurate because the structural constraints are of great value to build the final sequence alignment. The consistency-based methods usually employ these constraints during the consistency extension. That is, the weights of aligned substrings are adapted depending on intermediated sequences *and* structural information.

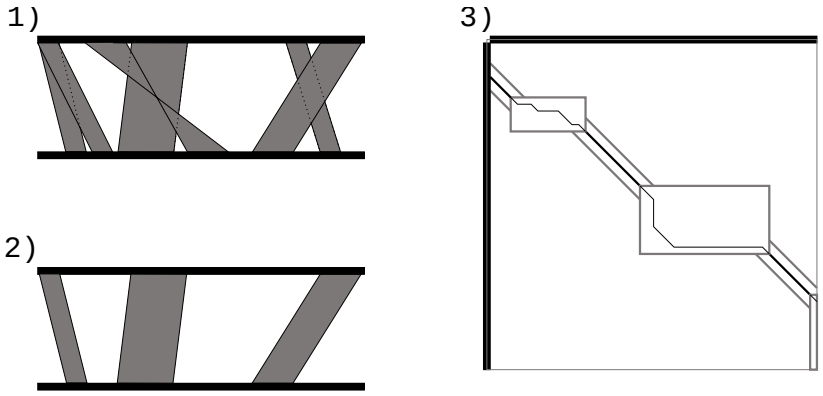


Figure 2.14: Anchor-based alignment: (1) computation of initial segment matches, (2) collinear chaining of non-overlapping segment matches and (3) dynamic programming to close the alignment gaps.

2.5.3 Anchor-based alignment

Even the heuristic progressive alignment becomes prohibitively expensive when aligning *genomic* DNA sequences. In these cases any approach involving a full pairwise dynamic programming is impossible. Nevertheless, so-called genome alignments or genome comparisons are more important than ever before because of several vertebrate genomes at hand and thousands of on-going sequencing projects. The applications are numerous, ranging from the comparison of different assemblies, annotation tasks, regular elements identification and phylogenetic studies to analyzing principal questions addressing mechanisms of genome evolution. Almost all genome aligners make use of the same strategy: anchor-based alignment or synonymously seeded alignment. Anchor-based alignment has three steps: (1) the computation of small segment matches of high similarity shared by multiple sequences, (2) the ordering of these segment matches into a collinear chain of non-overlapping segment matches (the fixed alignment anchors) and (3) closure of gaps in-between the anchors. The sole purpose of step 1 and step 2 is to abandon a large chunk of the possible alignment space as shown in Figure 2.14. Only small indels are allowed within the anchors. Hence, the time-consuming dynamic programming is only required in-between the fixed anchors. Some programs also try to extend anchors first to the left and right to further reduce the search space. Note that step 1 does not yet imply collinearity as shown in Figure 2.14.

The initial segment matches can be, for example, maximal unique or exact matches (Kurtz et al., 2004), maximal multiple exact matches (Hohl et al., 2002) and

exact or hashed k-mers (Buhler, 2001; Subramanian et al., 2005). Segment matches are optionally extended and finally, the quality of a segment match is assessed using some weight function. Chaining algorithms (Abouelhoda and Ohlebusch, 2003; Myers and Miller, 1995) can be applied to compute the heaviest (best) collinear chain of these segment matches. The resulting list of anchors is refined by applying the above procedure iteratively (e.g. by using a smaller k-mer) or by filling the gaps in-between the anchors using more sensitive approaches such as pairwise dynamic programming. Since genomic rearrangements such as transposition, duplication or inversion are rather likely, novel methods try to cover at least some of these operations, for example, by computing only local chains (Darling et al., 2004; Ovcharenko et al., 2005).

2.5.4 Others

The POA (Lee et al., 2002) tool uses partial order graphs to represent multiple sequence alignments. As noted previously, each individual sequence is a trivial partial order graph such that each character is a node connected to the subsequent node for the following character. POA progressively aligns these trivial graphs by (1) merging the nodes of aligned characters and (2) by introducing bifurcations if the subsequent nodes cannot be aligned.

The ABA (Raphael et al., 2004) MSA program uses a De Bruijn graph. In contrast to partial order graphs, a De Bruijn graph allows cycles and hence, it can be used to detect repeated or shuffled domains. In other words, the De Bruijn graph does not enforce a collinear alignment where one alignment column precedes the next one. Exact k-mer matches are, however, inappropriate for diverged protein sequences and thus, a classical De Bruijn graph cannot readily be applied. Because of that, ABA replaces the exact k-mer matches with local alignment information. Such local alignment information is quite often inconsistent, that is, matches might contradict each other. In order to delineate larger blocks that potentially represent domains, ABA applies a set of heuristics (Pevzner et al., 2004) on the initial graph to find a consistent set of similarities. These heuristics include the removal of so-called bulges and whirls in the De Bruijn graph. These operations appear to be similar to the ones employed by the De Bruijn graph-based, short-read assembler Velvet (Zerbino and Birney, 2008). The desired outcome of these heuristics for an alignment of n sequences is a graph with n source and n sink nodes. In this

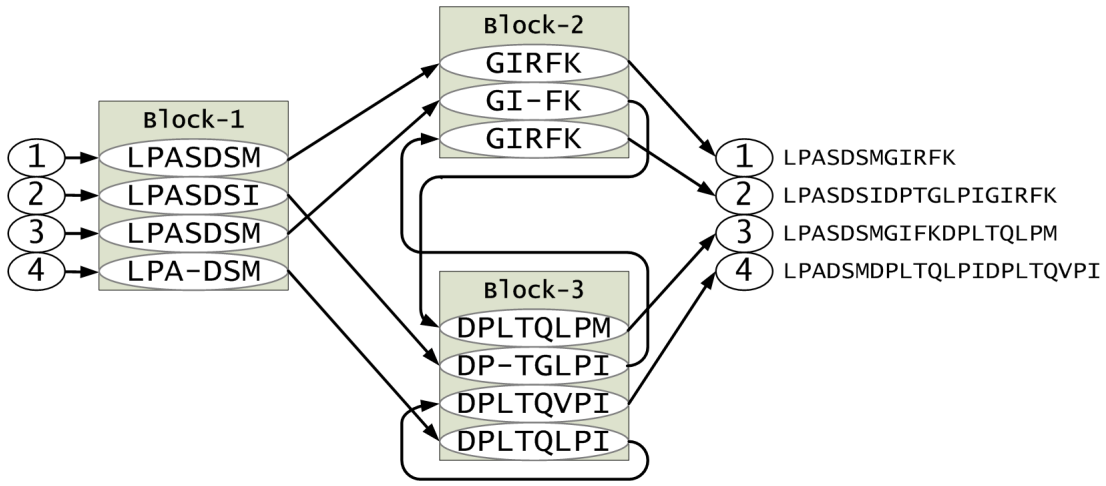


Figure 2.15: An alignment of shuffled and repeated domains of four sequences shown on the right using a De Bruijn graph created from local alignments. This figure was adapted from Figure 2 of the ABA paper (Raphael et al., 2004)

graph each original sequence is mapped to a path from the sequence's source to the sequence's sink node, which traverses a number of well-defined blocks as shown in Figure 2.15.

Current multi-read aligners usually use some-kind of greedy progressive alignment scheme without consistency and refinement. So far, there is no clear methodology used for multi-read alignments and most of the aligners are an integral part of some fragment assembler such as the Celera (Myers et al., 2000), Arachne (Batzoglou et al., 2002) or Atlas (Havlak et al., 2004) assembler. Unfortunately, the most interesting assembly regions harboring genetic variations such as SNPs or small indels are the most difficult regions to align and thus, the most error-prone regions for greedy progressive aligners. Hence, current multi-read alignments are not yet optimal and this implies that downstream analyses such as SNP calling or haplotype separation are hampered.

2.6 RNA Alignment Algorithms

RNA is a single stranded molecule that in contrast to DNA can fold onto itself by forming base pairs between C and G and A and U. Sometimes one can also observe the weaker bond between G and U, a so-called wobble base pair. The characteristic folding of a single RNA sequence is called the RNA's secondary structure. A number of algorithms has been developed to predict the secondary structure of an RNA sequence. The most prominent ones are the Nussinov (Nussinov et al., 1978) and Zuker algorithm (Zuker and Stiegler, 1981). Comparing RNA sequences is fundamentally different from a classical DNA sequence alignment since the RNA structure conservation outweighs the sequence conservation. Hence, an RNA alignment primarily aims at aligning common structural elements whereas the preservation of sequence similarity is only a subordinate goal. The algorithm of Sankoff is, for instance, a dynamic programming based RNA sequence-structure alignment algorithm that solves the alignment problem and the problem of finding an RNA secondary structure mapping simultaneously (Sankoff, 1985).

RNA sequence-structure alignment algorithms have not been considered in this thesis. Nevertheless, the approach described in Part II of this thesis can be applied to extrapolate pairwise sequence-structure alignments to a multiple RNA alignment.

2.7 Alignment Benchmarks

For the heuristic methods described in Section 2.5, protein benchmark data sets such as BAliBASE (Thompson et al., 1999a, 2005), PREFAB (Edgar, 2004b) or SABmark (Walle et al., 2005) are used to measure the performance of individual tools.

BAliBASE (Thompson et al., 1999a, 2005) is the most widely used reference benchmark for heuristic MSA tools. It was specifically designed for the evaluation and comparison of protein aligners and contains a comprehensive set of manually refined alignment instances. The reference alignments are annotated and the evaluation is limited to core blocks where the sequences can be reliably aligned without any ambiguities. The initial BAliBASE benchmark contained 142 reference alignments of more than a 1000 sequences. The benchmark also provides a scoring program that evaluates a third-party alignment against the reference benchmark.

The benchmark is further subdivided into 6 standard reference sets RV11, RV12, RV20, RV30, RV40 and RV50. The reference sets RV11 and RV12 contain alignment instances of equidistant sequences of similar length. RV20 contains alignment instances of protein families plus up to three orphan sequences per instance. Orphan sequences are distant members of the family with less than 20% identity. RV30 contains up to four protein sub-families per alignment instance that shall be aligned in a single alignment. RV40 contains long extensions to the left and right of individual sequences and RV50 contains alignment instances with sequences having large internal insertions.

In contrast to BAliBASE, the PREFAB (Edgar, 2004b) benchmark has been created using an automated protocol, resulting in more than 1500 alignment instances. The protocol first structurally aligns two proteins with disregard of sequence similarity. Then each of the two sequences is used to query a database. The high-scoring database sequences are included in the alignment instance that is subsequently cut to a maximum number of 50 sequences. The evaluation program assesses a MSA by means of projecting the alignment to the original pair and comparing this projected alignment to the precomputed structural alignment.

The SABmark benchmark contains alignment instances for pairwise and multiple aligners. In contrast to the other two benchmarks, SABmark also provides alignment instances with false positives, that is, sequences not related to the other sequences in the given instance. Each alignment instance has at most 25 sequences. All instances contain sequences with less than 50% identity since most MSA tools perform well above that threshold.

2.8 Available Implementations

In Table 2.2, 2.3 and 2.4 we compiled a list of current multiple sequence alignment tools. Given the plethora of available tools, this list is necessarily incomplete but should include most of the frequently used programs. Online web servers hosting the different alignment algorithms are frequently available, except for the genome aligners. Nevertheless, we restrained ourselves from providing web addresses of these servers because they tend to change frequently and can be easily found online by searching the name of the tool.

Category	Method	Protein / DNA
Sequence-based	LASA (Althaus and Canzar, 2008)	Both
exact	• Lagrangian ILP approach	
	MSA (Lipman et al., 1989)	Both
	• Bounded dynamic programming	
Sequence-based	ABA (Raphael et al., 2004)	Both
heuristic	• A-Bruijn alignment	
	AMAP (Schwartz and Pachter, 2007)	Both
	• Sequence annealing	
	CLUSTAL W (Thompson et al., 1994)	Both
	• Progressive alignment	
	DIALIGN-TX (Subramanian et al., 2008)	Both
	• Progressive, segment-based alignment	
	Kalign (Lassmann and Sonnhammer, 2005)	Both
	• Progressive alignment	
	POA (Lee et al., 2002)	Protein
	• Partial order graph alignment	
	MAFFT (Katoh et al., 2002)	Both
	• Progressive with refinement	
	MUSCLE (Edgar, 2004b)	Both
	• Progressive with refinement	
	Opal (Wheeler and Kececioglu, 2007)	Both
	• Prog. with consistency and refinement	
	ProbCons (Do et al., 2005)	Protein
	• Progressive with consistency	
	SeqAn::T-Coffee (Rausch et al., 2008b)	Both
	• Progressive, segment-based alignment	
	T-Coffee (Notredame et al., 2000)	Both
	• Progressive with consistency	

Table 2.2: Available MSA programs, categorized according to the used information sources (sequence / structure), the nature of the algorithm (exact / heuristic) and the ability to align genomic sequences. The method column highlights only the *predominant* technique. Thus, a progressive aligner using refinement might also use some kind of consistency extension. Continued on the next two pages.

Category	Method	Protein / DNA
Sequence-based meta-alignment	M-Coffee (Wallace et al., 2006)	Both
	• Progressive with consistency	
	SeqAn::T-Coffee (Rausch et al., 2008b)	Both
	• Progressive, segment-based	
Using secondary structure and database homologs	MUMMALS (Pei and Grishin, 2006)	Protein
	• Progressive with consistency	
	PRALINE (Simossis and Heringa, 2005)	Protein
	• Progressive alignment	
	PROMALS (Pei and Grishin, 2007)	Protein
	• Progressive with consistency	
	SPEM (Zhou and Zhou, 2005)	Protein
	• Progressive with consistency	
Using 3D structure	3D-Coffee (Wallace et al., 2006)	Protein
	• Progressive with consistency	
	Expresso (Armougom et al., 2006)	Protein
	• Progressive with consistency	
	PROMALS3D (Pei et al., 2008)	Protein
	• Progressive with consistency	
Genome aligners	M-GCAT (Treangen and Messeguer, 2006)	DNA
	• Anchor-based alignment	
	Mauve (Darling et al., 2004)	DNA
	• Anchors, local collinear blocks	
	MGA (Hohl et al., 2002)	DNA
	• Anchor-based, chaining	
	Mulan (Ovcharenko et al., 2005)	DNA
	• Anchor-based alignment	
	Multi-LAGAN (Brudno et al., 2003)	DNA
	• Anchor-based alignment	
	MUMmer (Kurtz et al., 2004)	DNA
	• Anchor-based, suffix-tree	
	TBA (Blanchette et al., 2004)	DNA
	• Anchor-based alignment	

Table 2.3: Available MSA programs. Table continued from last page.

Category	Method	Protein / DNA
Multi-read alignment	AMOS Consensus (Sommer et al., 2007)	DNA
	• Part of the Minimus assembler	
	Celera Consensus (Myers et al., 2000)	DNA
	• Part of the Celera assembler	
	ReAligner (Anson and Myers, 1997)	DNA
	• Iterative read to consensus alignment	
	SeqCons (Rausch et al., 2009)	DNA
	• Progressive MSA as well as realignment	

Table 2.4: Available MSA programs. Table continued from last page.

Contribution

The analysis, design and development of generic multiple sequence alignment components for the SeqAn software library (Döring et al., 2008) guided the progress of the entire thesis. The deliberate dissecting of state-of-the-art sequence analysis methods resulted in a fair amount of highly efficient and reusable algorithmic components such as pairwise alignment algorithms or guide-tree reconstruction algorithms. Given the wide range of applications for multiple sequence alignments such a novel library-based provision of the required key algorithmic components is an important addition to the field that enables a rapid prototyping of new algorithms and applications. It also strengthens the use of SeqAn as an experimental platform where different algorithms can easily be evaluated or where the collection of algorithms and data structures is used to develop and build novel, functionally enhanced applications. The careful analysis of current methods also provided the opportunity to implement improved algorithms for multiple sequence alignments. In particular, we contributed a new method to compute heuristic protein and DNA alignments and a new method to compute accurate consensus sequences in a reference-guided or de novo genome assembly. The performance, accuracy and strength and weaknesses of these tools are discussed in detail in Part II and Part III of this thesis. In this chapter, we briefly introduce and review the main contributions to guide the reader through the next chapters.

3.1 Dissecting Multiple Sequence Alignment Tools

As part of our effort to create a comprehensive, generic sequence analysis library called SeqAn (Döring et al., 2008) we dissected state-of-the-art methods that com-

pare and align multiple DNA and protein sequences. Using such a top-down approach, we identified the key components that are reoccurring in a number of these tools and decided to reengineer these components in the context of SeqAn to facilitate the rapid development and testing of new applications and algorithms (Rausch and Reinert, 2010).

First of all, we designed, developed and implemented a fairly comprehensive set of graph types and graph algorithms, including directed and undirected graphs, trees and automatons, methods to iterate such graphs and attach auxiliary information to edges or vertices and graph algorithms such as breadth-first and depth-first search, minimum spanning tree or shortest path algorithms. We used these core graph data structures, for instance, to implement tree reconstruction algorithms and the alignment graph model.

Second, we put special emphasis on an efficient implementation of configurable pairwise alignment algorithms that are heavily used as a preliminary step in MSA programs. During the design and development of global and local alignment algorithms we once again emphasized genericity. A generic `AlignConfig` template class allows, for example, an initialization of the dynamic programming matrix suitable for overlap and semi-global alignments.

Third, tree reconstruction algorithms such as neighbor joining or UPGMA and methods for profile alignment were added to the SeqAn library. Using algorithm tags, different specializations of a given algorithm can be seamlessly plugged in. For instance, the tags `UpgmaMin`, `UpgmaMax` and `UpgmaAvg` can be used to specify the desired UPGMA method to merge clusters, i.e., either single linkage, complete linkage or average linkage clustering.

Finally, we assembled the core components to a couple of new command-line tools, including a comprehensive pairwise alignment tool called `pair_align`, a tree reconstruction tool called `tree_recon`, a multiple sequence alignment tool called `seqan_tcoffee` and a multi-read alignment tool called `seqcons`. The pairwise alignment program is faster and equally good in memory usage than state-of-the-art methods from the EMBOSS library and the NCBI C++ toolkit (Rice et al., 2000). The tree reconstruction program encapsulates all the guide tree algorithms described in this thesis. The remaining two projects are described hereafter because they include in addition to the reengineered components novel techniques to efficiently build and construct multiple sequence alignments in various settings.

3.2 Segment-Based Multiple Sequence Alignment

In this project we focused on the problem of aligning up to two hundred, globally related, DNA or amino acid sequences with high accuracy measured on standard benchmarks. Based-upon previous work on small-scale multiple sequence alignments using an alignment graph (Althaus et al., 2002, 2006), we designed and developed a new model to compute and represent large-scale alignments. The basic idea was to represent arbitrary match information of k sequences within a k-partite graph and to compute a generic graph based progressive alignment independent of the sequence characters using solely the alignment graph edges, a so-called trace computation (Sankoff and Kruskal, 1983). For the graph based progressive alignment we adopted the consistency means invented by T-Coffee (Notredame et al., 2000) and hence, the name of our tool SeqAn::T-Coffee (Rausch et al., 2008b). The most advantageous property of the alignment graph compared to previous computation models is that a single vertex can represent a single character, a large segment or even an abstract entity such as a gene. Thus, given long matching segments our own method is able to extend the consistency-based progressive alignment paradigm to genomic sequences.

Our own pairwise global and local alignment algorithms turned out to be among the fastest and most memory-efficient algorithms currently available. We also redesigned consistency (Notredame et al., 2000) and progressive alignment methods (Feng and Doolittle, 1987) to facilitate the graph-based MSA construction. On standard MSA benchmark sets such as BAliBASE 3.0 (Thompson et al., 2005) and PREFAB 4.0 (Edgar, 2004b), we perform similar to the best MSA methods. The tool also supports a meta-alignment of subalignments, which delivered the best results on standard benchmarks among all sequence-based methods.

3.3 Multi-Read Alignment

In this project we designed, developed and experimentally verified a flexible multi-read alignment tool (Rausch et al., 2008a, 2009) that is robust in case of high sequencing error rates. The two main application scenarios of multi-read alignments, synonymously consensus methods, are reference-guided and de novo sequence assembly projects. Both scenarios create use cases that are quite distinct and require a flexible multi-read alignment method. We adapted the original ReAligner (Anson and Myers, 1997) algorithm to handle accurate layout positions of reads. The method scales well to large resequencing projects. For this algorithm and the read mapper RazerS developed by David Weese (Weese et al., 2009), we also designed a data structure that efficiently stores deep-coverage, large scale multi-read alignments.

For less accurate layout positions occurring in insert sequencing scenarios or for high-error reads we developed a graph-based multi-read alignment strategy. This strategy is due to all-against-all pairwise overlap computations slower than the ReAligner method. It does, however, consistently outperform competing methods in terms of quality, especially for insert sequencing. For this method, we extended our alignment algorithms to handle semi-global and overlap alignments. As a result, the Needleman-Wunsch and Gotoh algorithms have been augmented by complementary banded alignment algorithms. We experimentally verified the ReAligner and the graph based multi-read alignment method on a comprehensive set of simulated multi-read alignment instances using varying read lengths, error rates and coverage assumptions. We had to use simulated data since so far, there are no generally accepted benchmarks for multi-read alignment methods available. Both methods compared in almost all simulated cases favorably to existing consensus methods.

Part II

Algorithms and Data Structures

Alignment Data Structures

For diverse alignment tasks, ranging from ordinary protein sequence alignments to multi-read alignments of thousands of reads, specialized data structures are required. The two main reasons are that (1) no single alignment representation fits the needs of all alignment tasks and that (2) different representations allow either a more efficient access to different parts of the alignment or a more efficient storage of large-scale alignments.

4.1 Alignment Containers

The three main alignment containers in SeqAn are alignment matrices (see Section 4.1.1), alignment graphs (see Section 4.1.2) and a fragment store for multi-read alignments (see Section 4.1.3).

4.1.1 Alignment matrix

The Align data structure in SeqAn, developed by Andreas Gogol-Döring, is a direct representation of a classical alignment matrix (Gogol-Döring and Reinert, 2009). For an $n \times l$ alignment matrix $\mathcal{A}$ of n sequences, the Align data structure uses a set of n gapped sequences $\{\tilde{S}^0, \tilde{S}^1, \ldots, \tilde{S}^{n-1}\}$ to store the alignment. Hence, the Align data structure stores an alignment line by line. There are three distinct gapped sequence implementations available.

1. The SequenceGaps specialization is the straight forward implementation of a gapped sequence. It simply inserts the '−' characters into the actual sequence.

2. The ArrayGaps specialization stores the sizes of gaps and gap free parts. The

strength of this specialization is the efficient storage of long gaps.

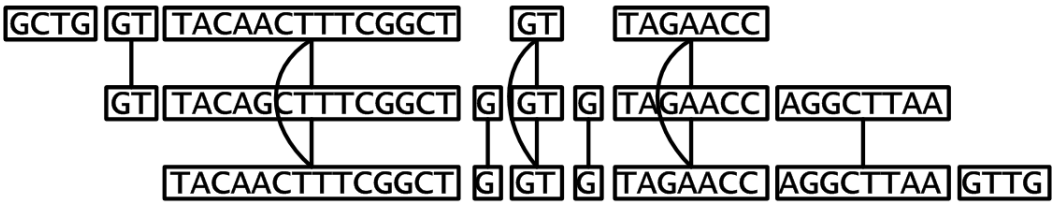
3. The SumlistGaps specialization stores pairs of integers representing the length of a contiguous sequence of non-gaps and the size of that non-gap plus the preceding gap characters.

The details of the Align data structure can be found in the SeqAn book (Gogol-Döring and Reinert, 2009).

4.1.2 Alignment graphs

The alignment graph introduces an additional abstraction layer for the representation of an alignment. Instead of storing actual aligned sequence characters such as the Align data structure or the FragmentStore, it represents an alignment as an n-partite graph for n sequences as shown in Figure 4.1. Vertices represent non-overlapping sequence segments, edges represent ungapped aligned sequence segments and gaps are implicitly represented by the topology of the graph. For example, the *GCTG* vertex in Figure 4.1 has no outgoing edges (degree zero) and thus, it is aligned to gaps in all other sequences. The alignment graph is a very compact and versatile description of an alignment. Large-scale alignments can be efficiently stored since long segments are represented by only a single vertex. Furthermore, the extension and direction of an alignment is completely defined by the alignment edges. That is, the graph formulation is equally suitable to align globally related sequences or thousands of reads where only subsets are related by mutual overlaps (see Figure 4.2). The properties of an alignment graph G are:

- For a set $\mathcal{S} = \{S^0, S^1, ..., S^{n-1}\}$ of n sequences the alignment graph $G = (V = \{V^0 \cup V^1 \cup ... \cup V^{n-1}\}, E)$ is an n-partite graph.
- Each vertex $v_p^i \in V^i$ represents a sequence segment in S^i of arbitrary length. We also say that v_p^i covers all positions of the segment. For instance, v_p^i might cover the sequence segment $S_{u1,u2}^i = s_{u1}^i s_{u1+1}^i ... s_{u2-1}^i$.
- Every position in $S^i = s_0^i s_1^i .. s_{|S^i|-1}^i$ is covered by one and only one vertex $v_p^i \in V^i$.
- Three integers are associated with each vertex: (1) the sequence identifier it belongs to, (2) the beginning of the segment and (3) the length of the segment.



GCTG	GT	TACAACTTTCGGCT	-	GT	-	TAGAACC	--------	----
----	GT	TACAGCTTTCGGCT	G	GT	G	TAGAACC	AGGCTTAA	----
----	--	TACAACTTTCGGCT	G	GT	G	TAGAACC	AGGCTTAA	GTTG

Figure 4.1: An alignment graph and the corresponding alignment matrix for three reads. Vertices represent non-overlapping sequence segments, edges represent ungapped aligned sequence segments, and gaps are implicitly represented by the topology of the graph.


```
GCTG-GTC
GATG-GTC
GCTGAGTC
```

```
GCTGGTC--------
----GTCTACA----
-------TACAGTTG
```

Figure 4.2: The alignment on the left shows globally related sequences whereas the one on the right shows a simplified multi-read alignment. The direction of the alignment solely depends on the alignment edges.

- An edge $e = \{v_p^i, v_q^j\} \in E$ with $i \neq j$ indicates that vertex v_p^i can be aligned with vertex v_q^j. In other words, the sequence substring in S^i covered by v_p^i can be aligned without gaps to the substring in S^j covered by v_q^j.

- The benefit of aligning v_p^i with v_q^j is given by an edge-weight w_e.

Besides representing actual alignments, the graph can also be used to store arbitrary match information as illustrated in Figure 4.3. This is, for instance, convenient to store multiple overlapping local alignments as computed by the Waterman and Eggert algorithm (Waterman and Eggert, 1987).

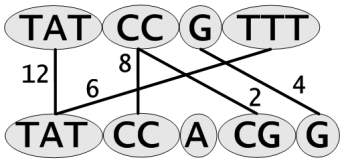


Figure 4.3: A general alignment graph of two sequences with weighted match information. Only a subset of the edges can be realized in an alignment.

4.1.3 Fragment store

The FragmentStore alignment data structure targets the large-scale storage of multi-read alignments occurring in de novo sequence assembly and resequencing projects. It was developed together with David Weese. Its main strength is the efficient storage of the alignment of a short read to a large contig or reference sequence. The FragmentStore uses a number of subclasses to store all the additional alignment information required in such projects, such as mate pair information and fragment library characteristics. It also supports the storage of alignments using clipped sequences. Before explaining the data structure in-depth, we want to illustrate the main features by means of a very small example.

Contig		–	C	T	–	A	C	–	–	A	C	G	G	–	–	–		
→ *Read*$_1$			C	T	C	A	C	G	–	A	C	G						
← *Read*$_2$		A	–	T	–	A	C	–	–	A	C	a	a					
→ *Read*$_3$						A	C	T	G	A								
→ *Read*$_4$	g	g	C	T	–	A	C	–	–	A	C	G	G	C	C	T	g	g

The first row in the multi-read alignment is the putative consensus sequence. Underneath the consensus are four aligned reads. Each aligned read has an orientation, shown as an arrow preceding the read name. Clipped sequence characters are shown in lower-case letters. Not shown in this example are mate-pairs, mapping quality information or multiple contigs. The design of the FragmentStore is database oriented. Basically, there is one table, called store, for each of the required elements, namely a read store, a mate-pair store, a library store, a contig store, an aligned read store and an annotation store. To link the information in the different tables, each read, mate-pair, library and contig has an associated id. This id is used to index the corresponding table, except for the aligned read store that has no such index id. Hence, the aligned read store is the only store that can be arbitrarily sorted. This is very convenient, for instance, to efficiently find all reads belonging

Category	Characteristics	Storage
Directed graph	Edges are directed, $e_1 = (u, v) \neq (v, u) = e_2$	Adjacency list
Undirected graph	Edges are undirected, $e = \{u, v\}$	Adjacency list
Automaton	Directed edges labeled with characters	Edge table
WordGraph	Directed edges labeled with sequences	Edge table
Tree	Directed edges with parent links, rooted graph	Adjacency list
HMM	Hidden Markov model using a directed graph	Adjacency list

Table 4.1: Listing of available graph types.

to a contig or to enumerate all reads in increasing order of their alignment position. Nevertheless, each element of the aligned read store has a unique id. Although this unique id cannot be used as an index into the aligned read store, it can be used to associate additional information with the aligned read such as a mapping quality or annotation data.

4.2 Implementation

4.2.1 Alignment matrix

The Align data structure uses a set of n gapped sequences stored in a String. The type of the gapped sequence can be one of the previously introduced specializations, namely SequenceGaps, ArrayGaps or SumlistGaps. The data structure supports several gap modifying functions such as procedures to insert gaps, to remove gaps, to count gaps or to clear all gaps. The details of how to use these functions and iterate over an alignment are described in the SeqAn book (Gogol-Döring and Reinert, 2009).

4.2.2 Alignment graphs

The alignment graph data structure is built on top of some basic graph types available in SeqAn. The main graph types available are shown in Table 4.1. Except for the automatons and word graphs, all graphs are stored as adjacency lists. An exemplary adjacency list for a directed graph is shown in Figure 4.4. As can be seen in the Figure, the graphs allow self-edges, multiple edges between two vertices (multigraphs) and multiple, disjoint components. The rounded rectangles in the

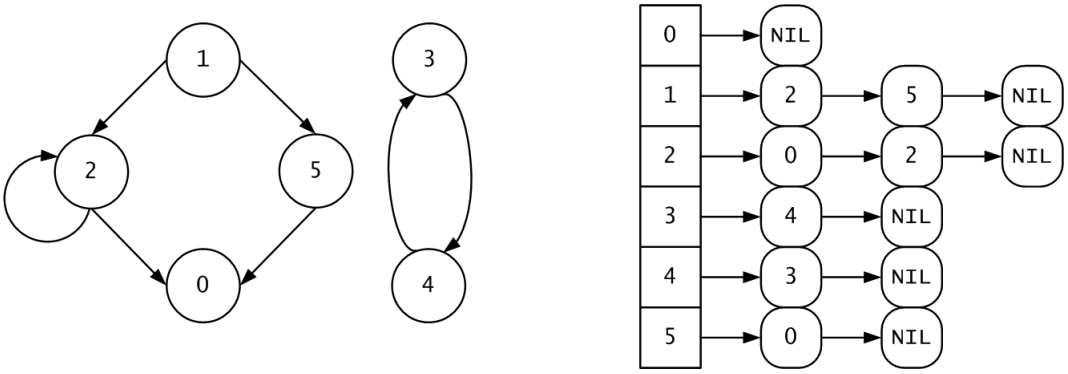


Figure 4.4: A directed graph using an adjacency list. The rounded rectangles on the right are so-called edge-stumps storing only the target vertex, where NIL denotes the end of the list.

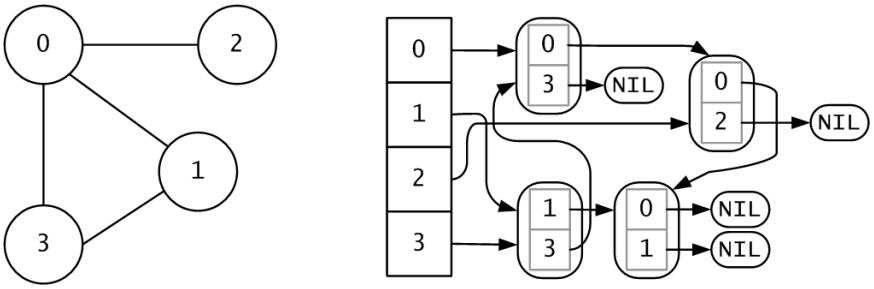


Figure 4.5: An undirected graph using an adjacency list. The rounded rectangles on the right are so-called edge-stumps storing the source and target vertex. Each undirected edge is stored only once.

adjacency list are the so-called edge-stumps. A minimal edge-stump stores only the target vertex and a pointer to the next edge-stump. They can, however, also be configured to store a unique id, the source vertex or an arbitrary cargo. The cargo can be used to store any edge information, such as integers to store distances or structs to store arbitrary complex information. The unique ids can be used to attach edge information by means of an external property map, explained in-depth later in this chapter. The undirected graphs store each edge $e = \{u, v\}$ only once by using the optional source vertex in each edge-stump and two pointers to the next source and target edge-stump. An example is shown in Figure 4.5. The adjacent vertices of a given vertex v can simply be traversed by following the link that originates from v in each edge-stump until NIL is reached. Since each edge is stored only once, edge cargos can be used as in directed graphs.

Trees are directed graphs with edges going from parent to child vertices. To

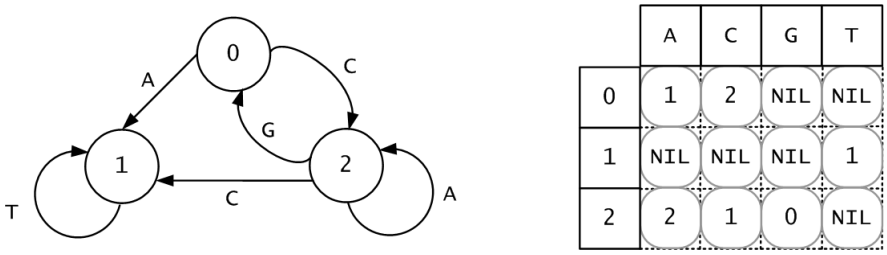


Figure 4.6: An automaton using an edge-table. The rounded rectangles on the right are the so-called edge-stumps storing the target vertex.

efficiently retrieve the parent of a given child vertex, trees store an additional parent link. Furthermore, trees have a distinct root, which is the only vertex without a parent. Besides the standard graph functions (see Table 4.2), trees support a number of specialized tree functions such as functions to add or remove child vertices (addChild and removeChild), functions to set, get and test for the root vertex (assignRoot, getRoot and isRoot) or a function to test if a given vertex is a leaf (isLeaf).

Automatons are usually defined as a 5-tuple $\langle \mathcal{Q}, \Sigma, \delta, q_0, \mathcal{T} \rangle$ where $\mathcal{Q}$ is the set of states, Σ a finite alphabet, δ the transition function, q_0 the start state and $\mathcal{T}$ a set of terminal states. In SeqAn, the automatons have been implemented as graphs where the vertices are the states, the root vertex is the start state and the edge labels are drawn from Σ. The transition function δ is encoded by means of the directed edges. The source vertex is the source state, the edge label is the input symbol and the target vertex is the target state of the δ function. If a set of terminal states $\mathcal{T}$ is required, this has to be done by means of an external property map. Automatons are usually used to parse strings in pattern matching applications. Hence, given a symbol and a source state, one readily wants to determine the target state. Because of that, automatons use an edge table instead of an adjacency list to facilitate this frequent operation in $\mathcal{O}(1)$ as shown in Figure 4.6. Similar to the trees, automatons have specialized functions to set, for instance, the initial start state or to parse input strings.

Hidden Markov models (HMMs) are also characterized by a 5-tuple $\langle \mathcal{Q}, \Sigma, \mathcal{A}, \mathcal{E}, \pi \rangle$ where $\mathcal{Q}$ is the set of states, Σ a finite alphabet, $\mathcal{A}$ the transition probability matrix, $\mathcal{E}$ the emission probability matrix and π a vector of length $|\mathcal{Q}|$ giving the initial starting distribution. For sparse HMMs, SeqAn offers a graph-based HMM model, where the vertices as states are labeled with emission probabilities and the edges

Function	Characteristics
addVertex	Creates a new vertex
addEdge	Creates a new directed or undirected edge
removeVertex	Removes a vertex and all adjacent edges
removeEdge	Removes an edge
numEdges	Number of edges in the graph
numVertices	Number of vertices in the graph
empty	Checks whether a graph is empty or not
clearEdges	Removes all edges
clearVertices	Removes all vertices
clear	Removes all edges and vertices
outDegree	Number of outgoing edges of a given vertex
inDegree	Number of incoming edges of a given vertex
degree	Number of outgoing and incoming edges
transpose	Transposes the graph

Table 4.2: Listing of available graph functions supported by all graph types.

as possible transitions are labeled with transition probabilities. Non-emitting silent states are also supported. The initial vector π has to be modeled by a separate, silent begin state and outgoing edges labeled with the probabilities given by π.

Alignment graphs are implemented by means of an undirected graph as shown previously in Figure 4.1. Since the graph is built over a set of n sequences, the graph additionally stores a StringSet that holds all the sequences. Each vertex stores by means of a property map the sequence id it belongs to and the beginning and length of the sequence segment it covers. One of the most frequent operations on an alignment graph is the retrieval of a vertex given a sequence identifier and a position on that sequence. To facilitate this operation the graph uses internally a map. This data structure maps a given key consisting of a sequence identifier and a position to the corresponding vertex. Naturally, each graph modifying operation such as the addition or removal of a vertex needs to be mirrored in the map so that both data structures are consistent with each other. Operations such as label, sequenceId, fragmentBegin and fragmentLength can be used to retrieve the covered sequence segment of a given vertex, the sequence id, the begin position and the length of the segment, respectively. The alignment graph also offers some input and output

Function	Characteristics
resizeEdgeMap	Initializes an edge property map
resizeVertexMap	Initializes a vertex property map
assignProperty	Assigns a property value to a given edge or vertex
property	Accesses a property value from a given edge or vertex
getProperty	Retrieves the property value from a given edge or vertex

Table 4.3: Listing of available property map functions.

routines to write, for instance, a FASTA or MSF file or read another alignment from a file.

Property maps

As previously mentioned, all graph types support an attachment of arbitrary information to vertices and edges by means of so-called property maps. A classical example are graphs representing flight networks with city names and flight distances. In SeqAn, one could store the city names in a property map for the vertices and the flight distances in a property map for the edges. Similarly, an alignment graph stores in a vertex property map the sequence id, the beginning and the length of the segment the vertex covers. The interface of a property map is rather simple and easy to use (see Table 4.3). All of the graph functions use so-called vertex and edge descriptors. These descriptors are handles to the vertices and edges present in the graph and are also used to access the property values in the property map. In particular, these descriptors carry a unique vertex and edge id that can be used to index the property map. As a result, the property map can be any indexable container. As mentioned before, edges can also carry an arbitrary cargo object. These cargo objects are useful for persistent edge information whereas the aforementioned external property maps are useful if we have only temporary edge labels. Due to the generic SeqAn design, the internal cargos have been subsumed under the property map interface as so-called internal property maps. As a result, algorithms should always rely on the property map interface to access additional vertex or edge information because this efficiently shields them from the actual underlying storage that is either a cargo object or an entry in an external property map.

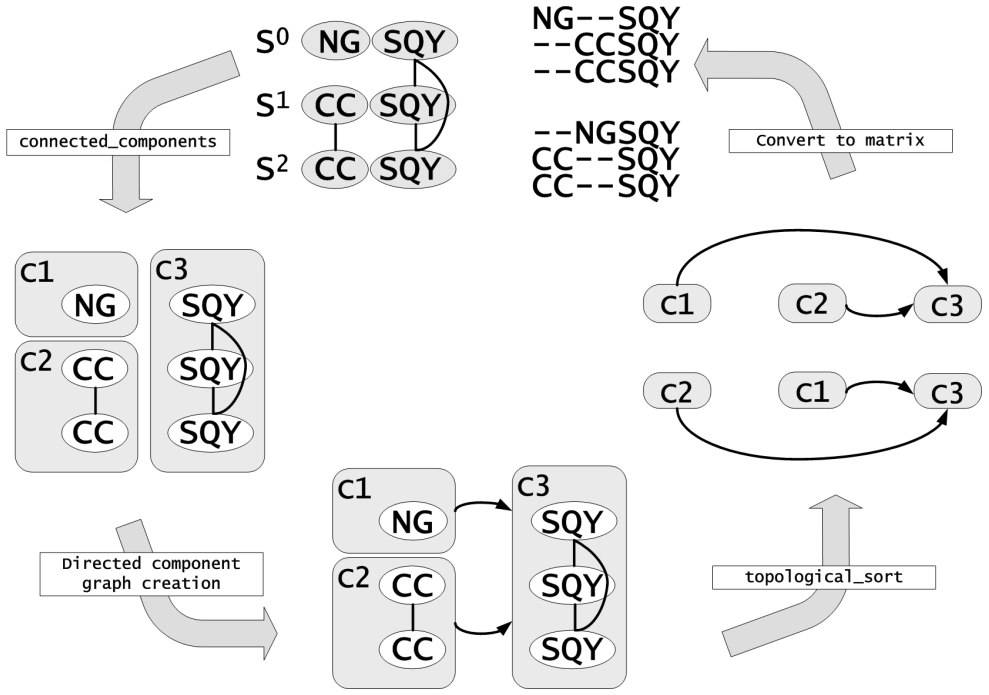


Figure 4.7: Conversion of an alignment graph to an alignment matrix. Due to the topological sort operation the order of adjacent indels is not fixed in the alignment graph representation.

Graph iterators

Graph iterators can be used to traverse the vertices and edges of a graph. The default vertex iterators are an adjacency iterator, an depth-first iterator, an breath-first iterator and a simple vertex iterator. The simple vertex iterator traverses all vertices in increasing order of their ids. The adjacency iterator traverses all adjacent vertices of a given vertex. The depth-first iterator traverses all vertices in depth-first search order and similarly, the breath-first iterator traverses all vertices in breath-first search order. The provided edge iterators are an out-edge iterator traversing all outgoing edges of a given vertex and a simple edge iterator traversing all edges of a graph.

Graph algorithms

SeqAn provides some standard graph algorithms shown in Table 4.4. Such basic graph algorithms are required in many bioinformatics applications and as an example, we show in Algorithm 1 a method to convert an alignment graph G into

Category	Algorithms
Vertex enumeration	Breadth-first search
	Depth-first search
	Topological sort
Minimum spanning tree	Prim's algorithm
	Kruskal's algorithm
Single-source shortest path	Directed acyclic graph (DAG) shortest path
	Bellman Ford algorithm
	Dijkstra
All-pairs shortest path	Floyd-Warshall
Connected components	Strongly connected components for directed graphs
	Connected components for undirected graphs
Network flow	Ford-Fulkerson algorithm

Table 4.4: Listing of available graph algorithms.

an ordinary alignment matrix $\mathcal{A}$ using two of the implemented graph algorithms, namely `connected_components` and `topological_sort`. A graphical illustration of this algorithm is shown in Figure 4.7. Both algorithms require a depth-first search as a preliminary step.

Algorithm 1 Alignment graph to alignment matrix conversion

Input: Alignment graph G

Output: Alignment matrix $\mathcal{A}$

1: $\mathcal{C} = \{C_0, C_1, ..., C_{k-1}\} \Leftarrow \text{connected\_components}(G)$
2: Build a component graph $G_{\mathcal{C}} = (V_{\mathcal{C}} = \{v_{C_0}, v_{C_1}, ..., v_{C_{k-1}}\}, E_{\mathcal{C}})$.
3: Insert directed edge $e = (v_{C_u}, v_{C_v}) \in E_{\mathcal{C}}$ if and only if a vertex in component C_u precedes a vertex in component C_v in one of the sequences.
4: $(C_{i_0}, C_{i_1}, ..., C_{i_{k-1}}) \Leftarrow \text{topological\_sort}(G_{\mathcal{C}})$
5: Write the vertices belonging to C_{i_j} underneath each other.
6: Replace vertices with sequence information.

4.2.3 Fragment store

The five most important stores belonging to the FragmentStore are the read store, the mate-pair store, the library store, the contig store and the aligned read store. The main elements of each store are shown below.

Read Store		Mate-pair Store	
•Index:	read id	•Index:	mate-pair id
•Members:	mate-pair id	•Members:	library id
			$read_1$ id
			$read_2$ id
Library Store		Contig Store	
•Index:	library id	•Index:	contig id
•Members:	mean	•Members:	seq
	std		gaps
Aligned Read Store			
•Index:	none		
•Members:	read id		
	contig id		
	pair match id		
	begin pos		
	end pos		
	gaps		
	unique id		

For each read, the mate-pair id indicates the fragment the read stems from. Similarly, a mate-pair element stores the two read ids that were sequenced from the given fragment. The fragments themselves are derived from mate-pair libraries. This library information is linked via the library id in the mate-pair element. The library element itself simply stores the mean and the standard deviation of the library. Each contig element stores the contig sequence and a gap anchor data structure that stores the occurrences of gaps in the contig sequence. The aligned read store is the only non-indexable store. That is, it is never accessed via an id but rather sorted and traversed according to some desired property. One can traverse, for instance, all aligned reads having the same contig id. For each aligned read, the read id links the original read data. The contig id, the begin pos, the end pos and the gap anchor data structure characterize the alignment of the given aligned read element. For reverse aligned reads the end pos is smaller than the begin pos. The pair match id is a unique id for multiple, distinct mate-pair matches and the unique id for each aligned read allows the storage of additional aligned read information such as mapping qualities in a separate store. The sequences of all reads are stored separately in a concatenated StringSet. This reduces the memory overhead

Position			0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		
Contig			–	C	T	–	A	C	–	–	A	C	G	G	–	–	–			
$\rightarrow$ *Read*$_1$				C	T	C	A	C	G	–	A	C	G							
$\leftarrow$ *Read*$_2$			A	–	T	–	A	C	–	–	A	C	a	a						
$\rightarrow$ *Read*$_3$							A	C	T	G	A									
$\rightarrow$ *Read*$_4$	g	g	C	T	–	A	C	–	–	A	C	G	G	C	C	T	g	g		

$\rightarrow$ *Read*$_1$: begin pos $= 1$, end pos $= 11$
gap anchors $= \langle \binom{6}{7} \rangle$ where $\binom{x}{y} = \binom{\text{Sequence position}}{\text{Alignment position}}$

$\leftarrow$ *Read*$_2$: begin pos $= 10$, end pos $= 0$,
gap anchors $= \langle \binom{1}{2}, \binom{2}{4}, \binom{4}{8}, \binom{8}{10} \rangle$

$\rightarrow$ *Read*$_3$: begin pos $= 4$, end pos $= 9$,
gap anchors $= \langle \rangle$

$\rightarrow$ *Read*$_4$: begin pos $= 1$, end pos $= 15$,
gap anchors $= \langle \binom{2}{0}, \binom{4}{3}, \binom{6}{7}, \binom{15}{14} \rangle$

Contig : gap anchors $= \langle \binom{0}{1}, \binom{2}{4}, \binom{4}{8}, \binom{8}{15} \rangle$

Figure 4.8: A multi-read alignment with gap anchors for each read and the contig. Read orientations are indicated by the arrow preceding the read's name. Clipped sequence characters are in lower case.

and additionally leaves the possibility to store the reads in an external string using secondary memory. To explain the gap anchor data structure we resume our introductory multi-read alignment example that is reproduced in Figure 4.8. The gap anchor data structure is a String of so-called gap anchors where each gap anchor describes a mapping from a sequence character position in the ungapped sequence to a sequence character position in the gapped sequence. In short, gap anchors are a means of translating from alignment space to sequence space and vice versa. In principle, one could store these gap anchors for each and every alignment position. A relative shift of alignment and sequence positions does, however, only occur if a gap or a clipped sequence character is reached. Read sequences get clipped or trimmed due to possible cloning vector content at the beginning or end of a read. Alignment gaps are usually caused by sequencing errors that delete true or introduce spurious

bases in the read's sequence.

In summary, it is sufficient to store a gap anchor only for those sequence characters that follow a gap or a clipped position (see Figure 4.8). By definition, this implies that a gapless alignment of a sequence that has not been clipped results in no gap anchors at all, as can be seen for $Read_3$. Gaps in an alignment cause an increment of the alignment position of a gap anchor, clipped sequence characters cause an increment in the sequence position and aligned characters cause an increment in both positions.

Pairwise alignment

5.1 Algorithms

In this chapter we briefly explain the algorithms of Needleman-Wunsch (Needleman and Wunsch, 1970), Gotoh (Gotoh, 1982), Smith-Waterman (Smith and Waterman, 1981) and Waterman-Eggert (Waterman and Eggert, 1987). In Section 5.2 we highlight some key features of our implementation in SeqAn and compare its space and time consumption to other state-of-the-art implementations.

5.1.1 Needleman-Wunsch

The Needleman-Wunsch algorithm computes a maximum score global alignment of two sequences $S^0 = s^0_0 s^0_1 \dots s^0_{|S^0|-1}$ and $S^1 = s^1_0 s^1_1 \dots s^1_{|S^1|-1}$ using a constant gap penalty e. The algorithm fills a matrix $M_{i,j}$ storing the maximum score of an optimal alignment $\mathcal{A}^{\{0,1\}}(S^0_i, S^1_j)$. The recursion of the algorithm is based upon the observation that the last aligned pair of an optimal alignment $\mathcal{A}^{\{0,1\}}(S^0_i, S^1_j)$ is either a pair of characters (s^0_i, s^1_j) or a character aligned to a gap, $(s^0_i, -)$ or $(-, s^1_j)$. Given that the last aligned pair is a pair of characters, $M_{i-1,j-1}$ is the maximum score of an alignment $\mathcal{A}^{\{0,1\}}(S^0_{i-1}, S^1_{j-1})$ and thus, $M_{i,j} = M_{i-1,j-1} + \delta(s^0_i, s^1_j)$ where δ is a user-defined scoring function. Similarly, we can conclude that $M_{i,j} = M_{i-1,j} + \delta(s^0_i, -)$ if and only if the last aligned pair is $(s^0_i, -)$ and $M_{i,j} = M_{i,j-1} + \delta(-, s^1_j)$ if and only if the last aligned pair is $(-, s^1_j)$. Hence, we can compute the final alignment score $M_{|S^0|-1,|S^1|-1}$ using the recursion

$$M_{u,v} = \max \begin{cases} M_{u-1,v-1} + \delta(s^0_u, s^1_v) \\ M_{u-1,v} + e \\ M_{u,v-1} + e \end{cases}$$

To facilitate a traceback that retrieves the actual alignment, it is sufficient to store for each cell $M_{i,j}$ a link to the predecessor cell that maximized the equation above. Hence, the algorithm uses $\mathcal{O}(\tilde{n}^2)$ time and space where $\tilde{n}$ is the average sequence length.

5.1.2 Gotoh

Gotoh extended the Needleman-Wunsch algorithm by linear gap costs that penalize a gap of length γ with a cost of $g + e \cdot (\gamma - 1)$ where g is the constant gap opening penalty, e is the constant gap extension penalty and $g \leq e$ with $g, e \leq 0$. Linear gap costs take into account the fact that in biological alignments gaps are often longer than a single character. Hence, opening a new gap should be more expensive than extending an existing one. Linear gap costs can be implemented using the Needleman-Wunsch recursion. Unfortunately, for each horizontal $(s_i^0, -)$ or vertical $(-, s_j^1)$ gap each predecessor cell to the left $M_{k,j}$ $(k < i)$ or top $M_{i,k}$ $(k < j)$ respectively has to be checked. This, however, requires $\mathcal{O}(\tilde{n}^3)$ time. To avoid this expensive horizontal or vertical look-up using k, Gotoh introduced two new matrices $V_{i,j}$ and $H_{i,j}$ that store the maximum score of an alignment $\mathcal{A}^{\{0,1\}}(S_i^0, S_j^1)$ ending in a vertical or horizontal gap, respectively. $M_{i,j}$ itself still stores the maximum score of an optimal alignment $\mathcal{A}^{\{0,1\}}(S_i^0, S_j^1)$. The new recursion is

$$
\begin{aligned}
H_{u,v} &= \max \begin{cases} M_{u-1,v} & + & g \\ H_{u-1,v} & + & e \end{cases} \\
V_{u,v} &= \max \begin{cases} M_{u,v-1} & + & g \\ V_{u,v-1} & + & e \end{cases} \\
M_{u,v} &= \max \begin{cases} M_{u-1,v-1} & + & \delta(s_u^0, s_v^1) \\ H_{u,v} \\ V_{u,v} \end{cases}
\end{aligned}
$$

The traceback requires for all three matrices the information from what cell the maximum was derived from, otherwise the backtrace will not be able to jump through each of the matrices. The algorithm still uses only $\mathcal{O}(\tilde{n}^2)$ time and space.

5.1.3 Smith-Waterman

The Smith-Waterman algorithm computes a maximum score *local* alignment of two sequences $S^0 = s_0^0 s_1^0 \dots s_{|S^0|-1}^0$ and $S^1 = s_0^1 s_1^1 \dots s_{|S^1|-1}^1$ using constant or linear gap costs. The algorithm augments the recursion with an additional 0 case, so that a new local alignment can be started whenever the score drops below 0. For linear gap costs the recursion is

$$
\begin{aligned}
H_{u,v} &= \max \begin{cases} M_{u-1,v} + g \\ H_{u-1,v} + e \end{cases} \\
V_{u,v} &= \max \begin{cases} M_{u,v-1} + g \\ V_{u,v-1} + e \end{cases} \\
M_{u,v} &= \max \begin{cases} M_{u-1,v-1} + \delta(s_u^0, s_v^1) \\ H_{u,v} \\ V_{u,v} \\ 0 \end{cases}
\end{aligned}
$$

The Smith-Waterman traceback starts at the highest scoring cell and ends at the first encountered 0. The algorithm uses $\mathcal{O}(\tilde{n}^2)$ time and space.

5.1.4 Waterman-Eggert

For multiple sequence alignment tools it is important to include the best local alignment *and* suboptimal local alignments, especially for highly divergent sequences. The suboptimal alignments should be mutually distinct. Hence, one cannot simply traceback from the second best score value since this is most likely an alignment that only differs marginally from the best one. To avoid this problem, the Waterman-Eggert algorithm resets all cells on the last traceback path to 0. This implies that all cells dependent on the reseted cells now contain incorrect values. Hence, the dynamic programming matrix needs to be recomputed from the first cell contained in the previous local alignment. Given such a recomputation, we can start another traceback from the new highest score and continue iteratively. The space consumption is still $\mathcal{O}(\tilde{n}^2)$. The required time depends on the number of suboptimal alignments and the first alignment position of every previous local alignment.

5.2 Implementation

All pairwise alignment algorithms have been subsumed under a common interface.

```
globalAlignment(TAlignDataStructure&, TScore&, TAlgorithmTag)
localAlignment(TAlignDataStructure&, TScore&, TAlgorithmTag)
```

Listing 5.1: Alignment interface

TAlignDataStructure is a placeholder for one of the alignment data structures such as the alignment graph or the alignment matrix. It can also be a simple StringSet if there is solely an alignment score required. TScore is a scoring object that is used by the alignment algorithm to score gaps and pairs of characters. All scoring objects implement the interface shown below.

```
score(TScore&, TPos1, TPos2, TSeq1&, TSeq2&)
scoreGapOpenHorizontal(TScore&, TPos1, TPos2, TSeq1&, TSeq2&)
scoreGapExtendHorizontal(TScore&, TPos1, TPos2, TSeq1&, TSeq2&)
scoreGapOpenVertical(TScore&, TPos1, TPos2, TSeq1&, TSeq2&)
scoreGapExtendVertical(TScore&, TPos1, TPos2, TSeq1&, TSeq2&)
```

Listing 5.2: Scoring interface

This interface allows a position-dependent scoring which is, for instance, required in profile alignments. The algorithm tag can be one of the implemented alignment algorithms, namely NeedlemanWunsch, Gotoh, BandedNeedlemanWunsch, BandedGotoh, SmithWaterman and SmithWatermanClump. The banded versions require the specification of a lower and upper diagonal. Only the part of the dynamic programming matrix enclosed by these diagonals is computed as shown in Figure 5.1. For the BandedGotoh algorithm the band is applied to all three dynamic programming matrices.

All of the above global alignment algorithms can be configured for overlap or semi-global alignments by using an AlignConfig<TTop, TLeft, TRight, TBottom>& object. This object has four template parameters one for each side of the dynamic programming matrix. TTop and TLeft indicate whether the first row and column of the matrix is initialized with appropriate gap costs or with 0's. TRight and TBottom indicate whether the alignment traceback can start anywhere in the last column or row or only in the cell in the bottom right corner. Hence, there are in total 2^4 different means of initializing the dynamic programming matrix available.

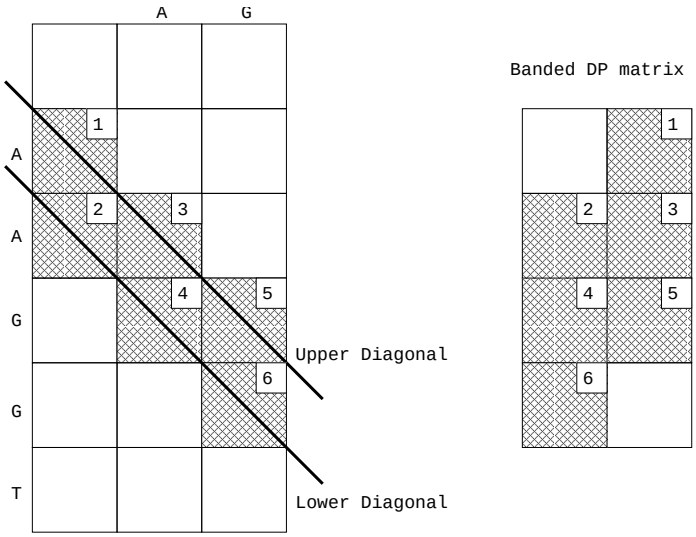


Figure 5.1: Shown on the left is how an upper and lower diagonal are used to delimit the desired band within the dynamic programming matrix. The space consumption of banded alignment algorithms solely depends on the length and the width of the band due to a transformation of the coordinate space shown on the right.

All options can also be applied to the banded alignment algorithms even if the band does not include the main diagonal.

All of the above alignment algorithms compute the dynamic programming matrix using only a single column (NeedlemanWunsch, Gotoh, SmithWaterman) or row (BandedNeedlemanWunsch, BandedGotoh) in memory. In Figure 5.2 we compared the banded alignment algorithms with the algorithms computing the full dynamic programming matrix. For the sake of illustration, we picked a huge bandwidth of 20.000 to get significant runtimes. This bandwidth implies that for sequences $< 10kb$ the banded versions actually computed the whole dynamic programming matrix. As can be seen in the plot for sequence lengths $< 10kb$, the banded algorithms induce only a very small overhead compared to the non-banded counterparts. In addition, it is clearly visible how the running time increases quadratically for the non-banded algorithms but linear for the banded versions. The Gotoh algorithm is more time-consuming than Needleman-Wunsch due to the three required matrices instead of only one. We then compared our own algorithms to state-of-the-art implementations. To our knowledge the two most efficient and widely used implementations are part of the EMBOSS library (Rice et al., 2000) and the NCBI C++ toolkit. The EMBOSS library provides both, an implementation of Gotoh's algorithm for global

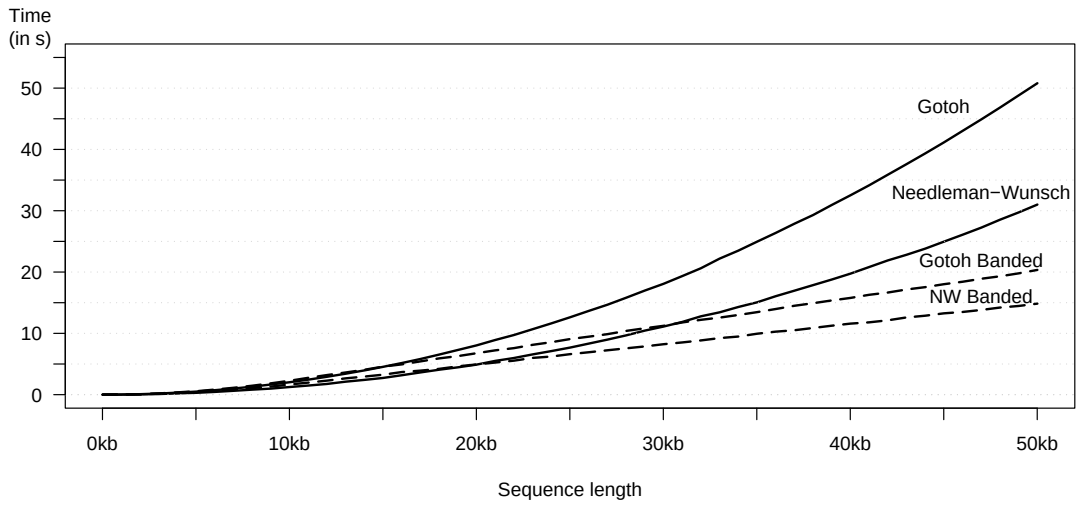
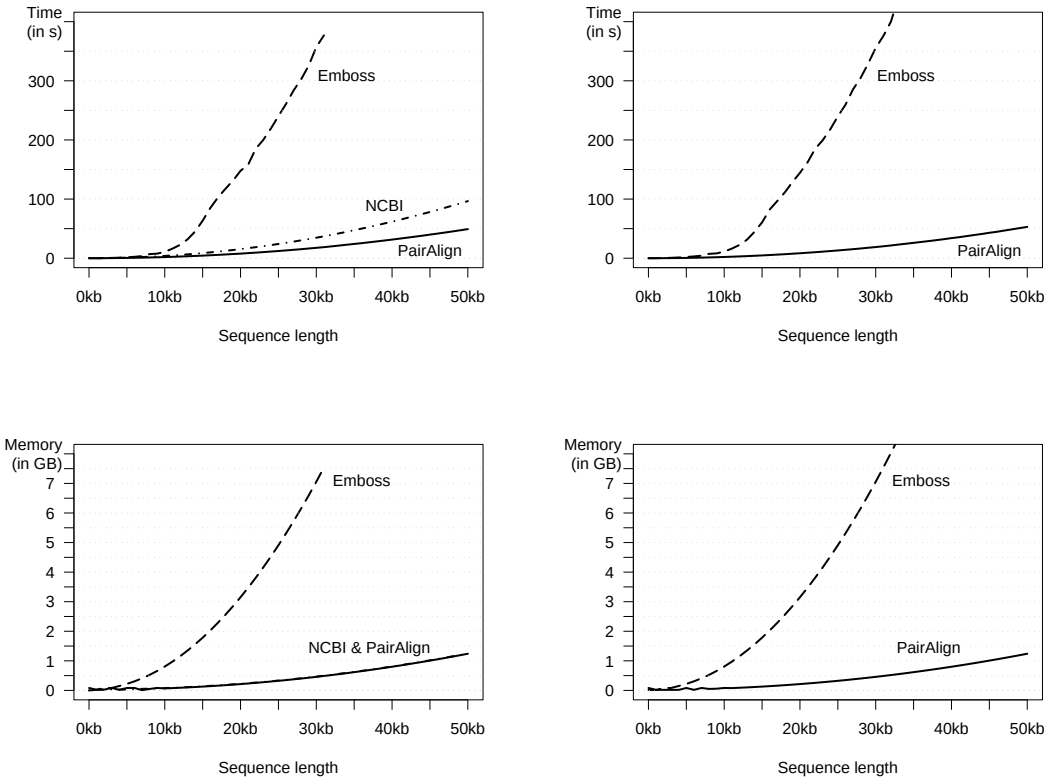


Figure 5.2: We simulated 50 pairs of random sequences of average length $1kb$, $2kb$, ..., $50kb$. Shown are the runtimes for each alignment algorithm on each alignment instance.

alignments and an implementation for local alignments. The NCBI C++ toolkit only offers the global version. We encapsulated all our own alignment algorithms in a tool called PairAlign. All tested alignment algorithms computed the same alignment score for each of the computed instances. Our global alignment algorithm is about twice as fast as the second best tool from the NCBI toolkit using a similar amount of memory (see Figure 5.3). EMBOSS is several magnitudes slower and failed on instances larger than $35kb$ due to insufficient memory. Note that our own tool and NCBI use less than the expected $50.000^2 = 2.5GB$ of memory for the traceback matrix in the $50kb$ case. The reason for PairAlign is that the traceback of two adjacent cells is stored in a single Byte. This is possible because in the horizontal and vertical matrix of Gotoh we can only traceback in the diagonal or horizontal / vertical direction. We encode this in two bits. We use two more bits for the diagonal matrix, where 0, 1 and 2 correspond to a diagonal, horizontal and vertical move. Hence, we need only four bits per cell and can store two traceback values in one Byte. For the Smith-Waterman local alignment this is still possible since we only need one additional stop value to indicate the end of a local alignment.



(a) Global alignment

(b) Local alignment

Figure 5.3: Our own alignment algorithm PairAlign in comparison to other alignment algorithms on a set of 50 simulated pairs of random sequences of average length $1kb$, $2kb$, ..., $50kb$. Global alignments are shown in (a), local alignments in (b).

Multiple Sequence Alignment

6.1 Overview

In Chapter 2, the main components of current MSA tools have been reviewed. All tools usually start with all-against-all pairwise alignments. Afterwards, core regions that reoccur in most of the pairwise alignments are amplified by using some kind of consistency scheme. Based upon these amplified regions and the background information from the pairwise alignments a progressive alignment along a guide tree is carried out. Some tools now further refine the MSA derived from the progressive alignment by splitting the alignment in two parts and realigning both parts together. Although most of the above steps have well-defined interfaces a majority of the present-day tools has been written from scratch. Therefore, we decided to dissect the main data structures and algorithms required in multiple sequence alignments and implement them in the SeqAn library. In the last Chapter 5, we already saw how such a careful refactoring of pairwise alignment algorithms can lead to highly efficient algorithms. Similarly, we reengineered algorithms to compute guide trees or to progressively align sequences along a guide tree. In addition to the dissecting, we also extended previous work on alignment graphs (Reinert, 1999; Althaus et al., 2006; Althaus and Canzar, 2008). Specifically, we present in this chapter a new graph-based method to progressively align sequences. The method can broadly be divided into seven distinct steps that are explained in the upcoming sections.

1. **Segment-match generation**
   One strength of our method is that the input can be *any* set of segment matches. That is, we can use fairly standard pairwise alignments, index-based comparisons such as maximal unique matches or even external matches derived

from tools such as BLAST (Altschul et al., 1990).

2. **Segment-match refinement**
   Initial segment matches might be contradictory, in the sense that matches overlap and intersect each other. Our own method therefore refines the initial set of segment matches so that all parts of all segment matches can be used.

3. **Alignment graph construction**
   Based upon the refined segment matches we define an alignment graph where vertices are gapless sequence segments and edges connect the matching sequence segments.

4. **Distance matrix computation**
   Several pairwise distance measures can be used to derive a distance matrix and construct a guide tree. Examples are pairwise alignment scores or the counting of common k-mers among the input sequences, where a k-mer is a contiguous subsequence of length k.

5. **Guide tree construction**
   Guide tree reconstruction algorithms such as UPGMA (Sokal and Michener, 1958) and neighbor-joining (Saitou and Nei, 1987) are used to build the required tree for the subsequent progressive alignment.

6. **Triplet extension**
   As a means of consistency, we adapted the triplet extension proposed in the T-Coffee (Notredame et al., 2000) package to our alignment graph.

7. **Progressive alignment**
   Finally, a graph-based progressive alignment is computed along the guide tree. At each internal node of the guide tree we compute the best pairwise trace by means of the heaviest common subsequence algorithm (Jacobson and Vo, 1992).

Figure 6.1: Subdivision of local or global alignments into gapless segment matches shown in black.

6.2 Algorithmic Components

6.2.1 Segment-match generation

Global or local alignments computed by our own alignment algorithms (see Chapter 5) or by external tools such as BLAST (Altschul et al., 1990) or MUMmer (Kurtz et al., 2004) are subdivided into gapless segment matches, as shown in Figure 6.1. Each segment match is labeled by its length, its beginning position in both sequences and the two identifiers of the two sequences belonging to this match. We perform this operation on all input alignments and store all the occurring segment matches in a segment match store $\mathcal{M}$.

6.2.2 Segment-match refinement

Segment matches might overlap and intersect each other as shown in Figure 6.2. In this example, the two segment matches containing the 'CC' segment are contradictory and only one of them can be realized in an alignment. In order to keep all of the available match information, we refine the set of segment matches so that all parts of all segment matches can be used. In contrast to this refinement approach, the DIALIGN series of programs (Morgenstern et al., 1998; Subramanian et al., 2008, 2005) leaves the set of segment matches unchanged. This implies that overlapping segment matches involving the same pair of sequences must be greedily resolved. The objective function of DIALIGN is to find a consistent, maximum score subset of segment matches whereas our method computes a heuristic maximum trace of all refined matches as explained later in this chapter.

The implemented multiple refinement algorithm is an extension of a pairwise refinement algorithm proposed by Halpern et al. (Halpern et al., 2002). The objective of the refinement method is to find a *minimal* subdivision of the segments so that all parts of all segment matches can be used. An example refinement for the three matches shown in Figure 6.2 is shown in Figure 6.3. Let $\mathcal{S} = \{S^0, S^1, ..., S^{n-1}\}$

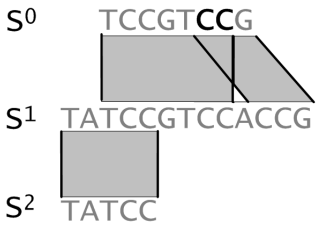


Figure 6.2: Three sequence S^0, S^1, and S^2 with three overlapping segment matches. Two matches are contradictory for the black 'CC' sequence segment.

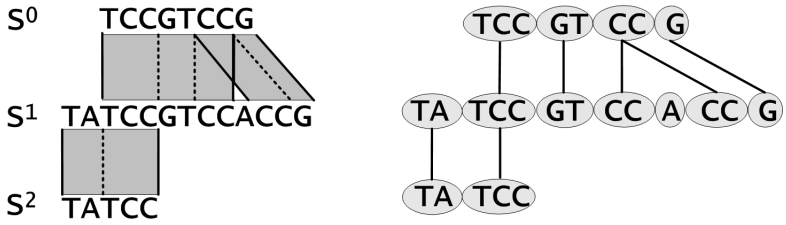


Figure 6.3: The refinement of overlapping segment matches is shown on the left. The dotted lines are the necessary cuts. On the right is the corresponding alignment graph for the refined segment matches.

be a set of n sequences with $S^i = s^i_0 s^i_1 ... s^i_{|S^i|-1}$ and $i \in \{0, 1, ..., n-1\}$. Let $\mathcal{M} = \{M^0, M^1, ..., M^{m-1}\}$ be a set of m segment matches with $M^k = (S^i_{uv}, S^j_{xy})$ and $k \in \{0, 1, ..., m-1\}$. M^k is an alignment between two segments $S^i_{uv} = s^i_u s^i_{u+1} ... s^i_{v-1}$ and $S^j_{xy} = s^j_x s^j_{x+1} ... s^j_{y-1}$ with $i, j \in \{0, 1, ..., n-1\}$, $i \neq j$, $0 \leq u < v \leq |S^i|$, $0 \leq x < y \leq |S^j|$ and $v - u = y - x$. For the segment S^i_{uv} the positions u and v are the boundary positions. We define the S^i-*support* of $\mathcal{M}$, in short $support_{S^i}(\mathcal{M})$, to be the set of all boundary positions of segments on sequence S^i.

The goal of the algorithm is to refine a set of input segment matches $\mathcal{M}$ into a set of segment submatches $\mathcal{M}_* = \{M^0_*, M^1_*, ..., M^{m'-1}_*\}$ where all submatches cover the original matches. A submatch of $M^k = (S^i_{uv}, S^j_{xy}) \in \mathcal{M}$ is a match $M^{k'} = (S^i_{u'v'}, S^j_{x'y'}) \in \mathcal{M}_*$ with $u \leq u' < v' \leq v$, $x \leq x' < y' \leq y$, $v' - u' = y' - x'$ and $u' - u = x' - x$. A set $\mathcal{M}_*$ is called a refinement of $\mathcal{M}$ if each $M^{k'} \in \mathcal{M}_*$ is a submatch of a $M^k \in \mathcal{M}$ and the set $\mathcal{M}_*$ tiles $\mathcal{M}$. That is, for each segment match $M^k = (S^i_{uv}, S^j_{xy}) \in \mathcal{M}$ we have a subset $\mathcal{M}'_* \subset \mathcal{M}_*$ where each $M^{k'} \in \mathcal{M}'_*$ is a submatch of M^k and the following two conditions are true:

$$[u, v-1] = \dot{\bigcup_{M^{k'} \in \mathcal{M}'_*}} [u', v'-1]$$

$$[x, y-1] = \dot{\bigcup_{M^{k'} \in \mathcal{M}'_*}} [x', y'-1]$$

In short, each original match must be tiled by submatches in $\mathcal{M}_*$.

We are, however, interested in a refinement $\mathcal{R}$ out of the set of possible refinements where all segments are either disjoint or identical, i.e., a refinement without partially overlapping segments. We call such a set of segment matches $\mathcal{R}$ *resolved*. In a resolved set any $(S^i_{uv}, S^j_{xy}) \in \mathcal{R}$ satisfies the requirement that

$$[u, v] \cap \mathit{support}_{S^i}(\mathcal{R}) = \{u, v\}$$

$$[x, y] \cap \mathit{support}_{S^j}(\mathcal{R}) = \{x, y\}$$

If we refine every segment match $M^k \in \mathcal{M}$ into single position matches we obtain a trivial resolved refinement. Hence, the objective is to find a refinement $\mathcal{R}$ of minimum cardinality. Such a refinement can be constructed by Algorithm 2 that successively applies only the necessary cuts to resolve partial overlaps and terminates when all segments are disjoint or identical. The algorithm refines a set of input segment matches $\mathcal{M}$ and returns an alignment graph constructed from the refined segment match set $\mathcal{R}$. At the end of the algorithm the node sets V^i contain the original boundary positions plus necessary cuts made from the projections of the initial boundaries. Hence, no superfluous cuts are made and the refinement is of minimum cardinality.

Algorithm 2 Segment-match refinement

Input: Set of segment matches $\mathcal{M}$

Output: Alignment graph G of refined segment matches

1: Build node sets $V^i = support_{S^i}(\mathcal{M})$
2: **for all** $M^k = (S^i_{uv}, S^j_{xy}) \in \mathcal{M}$ **do**
3: Build the set of boundary positions $\mathcal{B} = \{u, v, x, y\}$
4: **while** $\mathcal{B} \neq \emptyset$ **do**
5: Pick $w \in \mathcal{B}$
6: Remove w from $\mathcal{B}$
7: Retrieve all segment matches $\mathcal{L} = \{L^0, L^1, ..., L^{p-1}\}$ that contain w
8: **for all** $L^q = (S^k_{ab}, S^l_{cd}) \in \mathcal{L}$ **do**
9: Let $a < w < b$ and $h = c + (w - a)$ be the projected position of w.
10: **if** $h \notin V^l$ **then**
11: h is a cut, insert h into V^l
12: Insert h into $\mathcal{B}$
13: **end if**
14: **end for**
15: **end while**
16: **end for**
17: Create alignment graph $\mathcal{G}$
18: Derive from the node sets V^i the vertices of $\mathcal{G}$
19: Derive from the original segment match store $\mathcal{M}$ the edges of $\mathcal{G}$

6.2.3 Alignment graph construction

Given the refined set of segment matches $\mathcal{R}$ the construction of the alignment graph is rather trivial. As shown in Figure 6.3, we define for each gapless sequence segment a vertex according to the boundary positions present in V^i at the end of Algorithm 2. We define edges between two vertices v_1 and v_2 if and only if both covered sequence segments took part in an initial segment match. There are three different schemes to weight the matches during the alignment graph construction. The FractionalScore specialization uses the length of the refined segment match with respect to the original segment match to rescale the score of the refined match. The FrequencyCounting specialization simply counts how often a given refined match occurred in the initial

set of segment matches and the ReScore specialization scores all refined segment matches anew according to a user-defined scoring matrix such as BLOSUM.

6.2.4 Distance matrix computation

Progressive alignment requires a guide tree that indicates when each sequence is added to the growing MSA. There are different methods to construct such a guide tree or phylogenetic tree for a set of sequences. The most prominent tree reconstruction methods either use a distance matrix, the maximum likelihood principle or the maximum parsimony principle. In practice, the distance-based methods tend to be the fastest and therefore most MSA methods use such distance-based tree reconstruction methods. The input of these algorithms is a distance matrix $\mathcal{D}$ where $d_{i,j}$ is the distance of sequence S^i to sequence S^j. One possibility to derive such distances $d_{i,j}$ is to convert the pairwise alignment scores into distances, e.g., by normalizing with the highest observed pairwise alignment score.

$$d_{0,1} = 1 - \frac{Score(\mathcal{A}^{\{0,1\}})}{\max_{i,j}(Score(\mathcal{A}^{\{i,j\}}))}$$

An alternative is the so-called common k-mer counting method. A k-mer is a contiguous subsequence of length k. Two sequences S^0 and S^1 can share at most $\min(|S^0|, |S^1|) - k + 1$ common k-mers and thus,

$$d_{0,1} = 1 - \frac{\#\text{Common k-mers between } S^0, S^1}{\min(|S^0|, |S^1|) - k + 1}$$

can be used as a distance between sequence S^0 and S^1. Another option is to use the alignment graph from the last step to derive the distances $d_{i,j}$. In Section 6.2.7 we will see how a pairwise alignment can be carried out using the alignment graph. Hence, we can compute all pairwise alignments using the alignment graph edges as constraints. This method, however, requires $\mathcal{O}(n^2)$ additional pairwise alignments.

6.2.5 Guide tree construction

The neighbor-joining method (Saitou and Nei, 1987) as well as the UPGMA algorithm (Sokal and Michener, 1958) have been implemented in SeqAn. As mentioned in Chapter 2, the UPGMA algorithm has different options to compute the distances $d_{k,u}$ from any group k to a newly formed cluster u that joined group i and j.

We implemented each of the possible options using specialization tags of the form Upgma∗.

- Single linkage clustering: $d_{k,u} = \min(d_{k,i}, d_{k,j})$, Tag: UpgmaMin
- Complete linkage clustering: $d_{k,u} = \max(d_{k,i}, d_{k,j})$, Tag: UpgmaMax
- Average linkage clustering: $d_{k,u} = \frac{d_{k,i}+d_{k,j}}{2}$, Tag: UpgmaAvg
- Weighted avg. linkage clustering: $d_{k,u} = \frac{n_i \cdot d_{k,i}+n_j \cdot d_{k,j}}{n_i+n_j}$, Tag: UpgmaWeightAvg

The neighbor-joining method and the UPGMA algorithm transform a distance matrix into a tree where the leaves correspond to the sequences. The vertex id of a leaf equals the position of the sequence in the original StringSet of all sequences.

6.2.6 Triplet extension

Given the initial pairwise alignments, the triplet extension (Notredame et al., 2000) aims at substantiating true matches and degrading false matches by means of looking at all matches simultaneously. The principle was introduced in Chapter 2: Two pairwise matches $M^{0i} = (S^0_{uv}, S^i_{xy})$ and $M^{i1} = (S^i_{xy}, S^1_{qr})$ induce a putative transitive match $M^{01} = (S^0_{uv}, S^1_{qr})$ that is either consistent or inconsistent with a match occurring in the precomputed alignment $\mathcal{A}(S^0, S^1)$. If it is consistent, that is the match $M^{01} = (S^0_{uv}, S^1_{qr})$ is part of the precomputed alignment $\mathcal{A}(S^0, S^1)$, then we increase the weight w of M^{01} by $\min(w(M^{0i}), w(M^{i1}))$. If it is not consistent, the missing match will be created with $w(M^{01}) = \min(w(M^{0i}), w(M^{i1}))$. In terms of the alignment graph, we traverse all pairs of adjacent edges $(e_1 = \{v_k, v_i\}, e_2 = \{v_i, v_j\})$ and either insert a new edge $e_3 = \{v_k, v_j\}$ or adapt the weight of $e_3 = \{v_k, v_j\}$. Note that v_i, v_j and v_k have to cover segments on three different sequences. Since we have to enumerate all possible pairs of adjacent vertices (v_k, v_j) for a given vertex v_i, the runtime of the triplet extension greatly depends on the average out-degree of all vertices. This in turn depends on the number of input segment matches but in practice, the average out-degree is usually a small constant times $(n-1)$, where n is the number of input sequences.

Nevertheless, for a large number of sequences the triplet extension becomes quite expensive and because of that, we implemented a novel group-based triplet extension that takes into account the guide tree. As mentioned in the introduction, the triplet

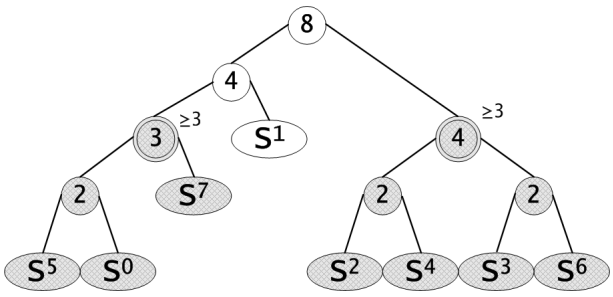


Figure 6.4: Binary guide tree for eight sequences $S^0, S^1, \cdots, S^7$. Internal nodes are labeled with the number of leaves underneath them. Subtrees with more than three members are shaded and their root is double-circled. Non-grouped sequences such as S^1 are added to the closest subtree. In the above case, S^1 is added to the left subtree.

extension aims at preventing greedy progressive alignment mistakes. These mistakes are especially likely as long as we are aligning the first two sequences or quite small profiles. If we go up in the guide tree and align profiles consisting of more than 10 sequences we have quite a lot of pairwise alignment information available and thus, we are less likely to make a mistake. Hence, the triplet extension is indispensable in the different subtrees of the guide tree but could be omitted among the subtrees to save time and space. To facilitate such a group-based triplet extension we cluster the guide tree into subtrees of a user-defined minimum size. The subtree clustering algorithm enumerates all guide tree vertices in reversed breadth-first search order and then labels each internal vertex with the number of leaves underneath it. If the number of leaves reaches the user-defined minimum size, we define that internal vertex as the root of a subtree. In the end, we have a number of clustered sequences and a few sequences belonging to subtrees with less than the required number of sequences as shown in Figure 6.4. These unclustered sequences are now added to the closest subtree. Therefore the worst-case behavior occurs in a completely unbalanced guide tree because all sequences would end up in the same group. In practice, however, we usually encounter quite balanced guide trees where a number of subtrees can be identified and hence, the triplet extension can be limited to these subtrees.

6.2.7 Progressive alignment

The progressive alignment builds a MSA along a guide tree using the previously created alignment graph that contains all the weighted, refined matches. In contrast to

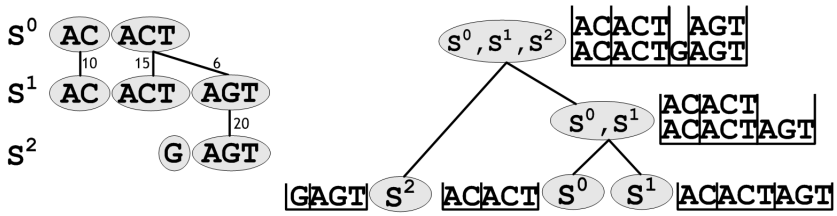


Figure 6.5: The progressive alignment uses a guide tree shown on the right and an alignment graph with edge-weights shown on the left. Next to each internal guide tree node a vertex profile of the already aligned subtree is shown. At each profile position (separated by vertical bars) we store only vertex descriptors but show the sequence information here for a better understanding.

other tools, our algorithm progressively aligns strings of vertices instead of the usual strings of sequence characters. This additional level of indirection has the benefit that vertices can represent diverse information such as sequence characters, sequence segments or even abstract entities such as genes. The benefit of aligning a vertex v_1 with v_2 is given by the edge-weight $w_{e_{v_1,v_2}}$. In the pairwise case, we are interested in finding the heaviest set of edges that constitutes a valid alignment. This problem can be solved by means of the heaviest common subsequence algorithm (Jacobson and Vo, 1992). Although the original algorithm assumed common subsequence characters, it can be applied to a bipartite alignment graph. Each weighted edge present in the graph simply connects two 'common' entities in the sequence of vertices. The outcome of the heaviest common subsequence algorithm is a set of trace edges. This trace is then condensed to a vertex profile. That is, we create a new string where at each position we encounter either a single vertex (aligned to gaps) or two vertices connected by one of the trace edges. This new string or vertex profile can now be used to align another string of vertices, again by means of the heaviest common subsequence algorithm. If multiple vertices are present at one profile position we set the edge weight to the average of the original edge weights. This bottom-up progressive alignment procedure is summarized in Figure 6.5.

6.3 Implementation

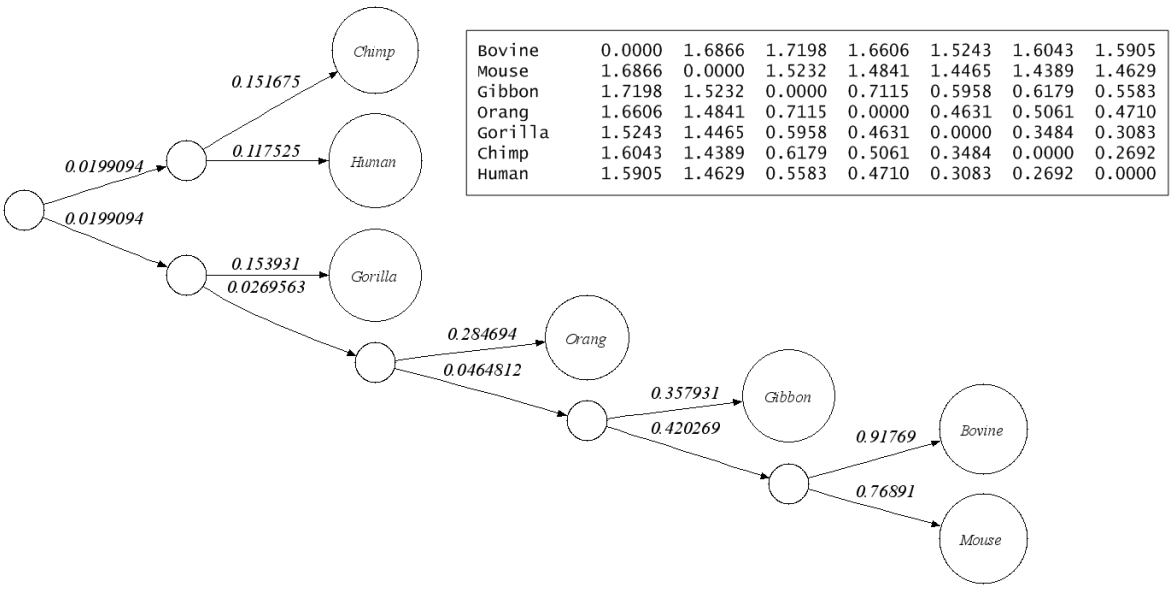
The initial segment matches are collected in a String. Each segment match is characterized by the two sequence ids taking part in the match, the two begin

positions and the length of the segment match. We also support reversible segment matches by means of the template specialization ExactReversibleFragment instead of the default ExactFragment specialization. Reversible segment matches have an additional boolean value indicating the orientation of the match. All built-in methods to generate segment matches have been subsumed under the function appendSegmentMatches, which can be specialized for different methods to generate segment matches. Examples are all-against-all pairwise global and local alignments (Algorithm tags: GlobalPairwise_Library and LocalPairwise_Library) or all-against-all comparisons using the longest common subsequence algorithm (Tag: Lcs_Library). The String of segment matches can also be augmented by external segment matches such as matches derived from BLAST, MUMmer or a T-Coffee library.

Subsequently, all collected segment matches are refined using a recursive implementation of the pseudo-code shown in Algorithm 2 developed by Anne-Katrin Emde (Emde, 2007). The key data structure used in the refinement algorithm is an interval tree (Edelsbrunner, 1980). Given a boundary position w, an interval tree T^i for each sequence S^i is used to efficiently retrieve all segments that contain w. The algorithm performs recursively all necessary cuts until all segment matches are disjoint or identical. At the end of the refinement algorithm all refined segment matches are inserted into the alignment graph and possibly rescored.

The binary guide tree $\mathcal{T}_G$ required for the progressive alignment is constructed from a distance matrix $\mathcal{D}$ using the functions njTree or upgmaTree. Both tree reconstruction methods are also available as a stand-alone application requiring a distance matrix in Phylip (Felsenstein, 1989) format as input. The output is a tree in Newick or DOT graph format that can be easily rendered as shown in Figure 6.6. The neighbor-joining method does, however, create unrooted trees. Since our subsequent progressive alignment requires a binary guide tree, we root the unrooted tree artificially at the edge created last. Hence, the original unrooted tree can be retrieved by collapsing the root, i.e., by means of merging its two outgoing edges into a single edge with a weight equal to the sum of the former outgoing edges.

The final progressive alignment algorithm originally worked recursively but has now been turned into an iterative algorithm. It enumerates all vertices of the guide tree $\mathcal{T}_G$ in reversed breadth-first search order. Thus, we traverse the guide tree bottom up and finish at the root node. For each leaf, we create a trivial String of



Bovine	0.0000	1.6866	1.7198	1.6606	1.5243	1.6043	1.5905
Mouse	1.6866	0.0000	1.5232	1.4841	1.4465	1.4389	1.4629
Gibbon	1.7198	1.5232	0.0000	0.7115	0.5958	0.6179	0.5583
Orang	1.6606	1.4841	0.7115	0.0000	0.4631	0.5061	0.4710
Gorilla	1.5243	1.4465	0.5958	0.4631	0.0000	0.3484	0.3083
Chimp	1.6043	1.4389	0.6179	0.5061	0.3484	0.0000	0.2692
Human	1.5905	1.4629	0.5583	0.4710	0.3083	0.2692	0.0000

Figure 6.6: A binary guide tree reconstructed from a Phylip distance matrix (upper right corner). The edge labels are the distances.

vertex descriptors. For each internal node, we retrieve the two strings or profiles of vertices from the child nodes, apply the heaviestCommonSubsequence algorithm and associate the new vertex profile with the internal node. In the end, the vertex profile at the root node is the desired alignment. We return the vertex profile at the root node as a new alignment graph that constitutes a valid trace. This alignment graph is then converted to an alignment matrix using the algorithm previously shown in Figure 4.7 and Algorithm 1 on page 64.

Meta-alignments as a means of combining several subalignments are also supported. We simply subdivide the input alignments into pairwise segment matches and then apply the algorithm pipeline described before. Since our algorithm is segment-based, we benefit from long conserved regions present in all subalignments. Such a situation allows a rapid computation of a new meta-alignment.

Multi-Read Alignment

7.1 Overview

With hundreds of ongoing de novo assembly and resequencing projects a new kind of alignment problem, called multi-read alignment, is of increasing importance. Frequently used synonyms are consensus alignment or consensus computation. In order to sequence the genome of an organism, the so-called target genome, scientists use sequencing platforms. The outcome of such a sequencer run is a set of reads and possibly paired-end or mate-pair information as shown in Figure 7.1 and explained hereafter. The initial target genome is copied and randomly broken into small fragments. In paired-end sequencing, two reads are sequenced per fragment, one from each end. The two reads stemming from the same fragment are called a mate-pair. The reads have varying length ranging from as short as 30 nucleotides to more than a 1000 nucleotides depending on the used sequencing technology. The new sequencing platforms such as 454 Life Sciences (www.454.com), Illumina's Solexa sequencing technology (www.illumina.com) and Applied Biosystems SOLiD Sequencing (www.appliedbiosystems.com) produce shorter reads (< 500 nucleotides) than the old Sanger technology (≈ 1000 nucleotides), which was used in the first human genome sequencing project. The massive throughput of the new sequencing platforms, however, outweighs the disadvantage of producing shorter reads. In addition, the read lengths are expected to further increase. For instance, the very short Solexa reads already have increased in length from about 35 nucleotides to more than 75 nucleotides.

Before the reads are sequenced, the fragments are usually grouped by mean length. Each of these groups, called fragment or mate-pair library, has a mean length and a standard deviation describing the variation of fragment lengths within

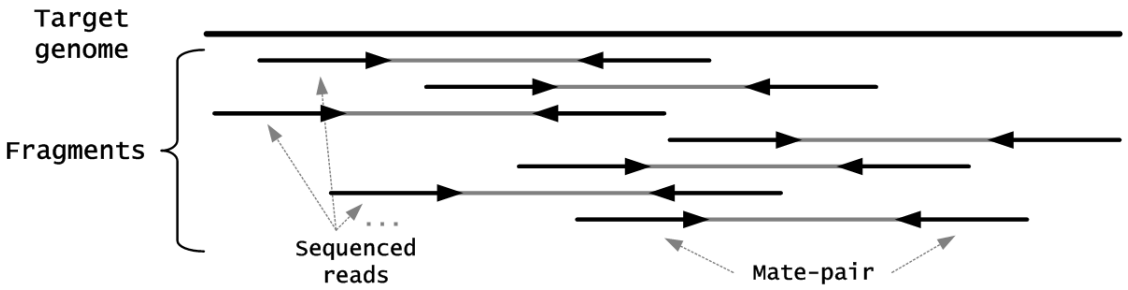


Figure 7.1: A target genome is copied and randomly broken into small fragments. In paired-end sequencing, each fragment is sequenced from both ends. The sequenced regions are called reads and the two reads belonging to the same fragment are called a mate-pair.

the library.

Once a genome has been sequenced, an assembler program is used to reconstruct the target genome from the set of reads. This set can contain thousands or even several millions of sequenced reads. There are two main assembly strategies: (1) de novo assembly and (2) reference-guided assembly.

The term de novo sequence assembly refers to the assembly of a genome from the raw read data without the help of an already sequenced reference genome. Classical de novo genome assemblers follow a three phase protocol: overlap phase, layout phase and consensus phase. In the overlap phase every read is compared to every other read and based upon the putative overlaps an overlap graph of the reads is computed. The overlap graph contains true overlaps as well as spurious overlaps introduced by sequencing errors, repeats or random alignments. The layout phase identifies a subgraph in the overlap graph that defines a consistent layout of the reads. That is, conflicting overlaps are heuristically resolved. The resulting resolved overlap graph determines an approximate placement of each read. Given such an approximate layout, a multi-read aligner computes the consensus sequence as well as a multi-read alignment. Consequently, the multi-read alignment problem is quite distinct from the alignments discussed so far, since it has to deal with a huge number of short reads that overlap only by a few bases.

In reference-guided assembly projects, we encounter a very similar multi-read alignment problem. In this scenario, the raw read data is mapped to a close relative whose genomic sequence is already available. For this reason, this kind of an assembly is also called template assembly, comparative assembly or resequencing in case of the same organism. The strength of this approach is that the overlap and layout

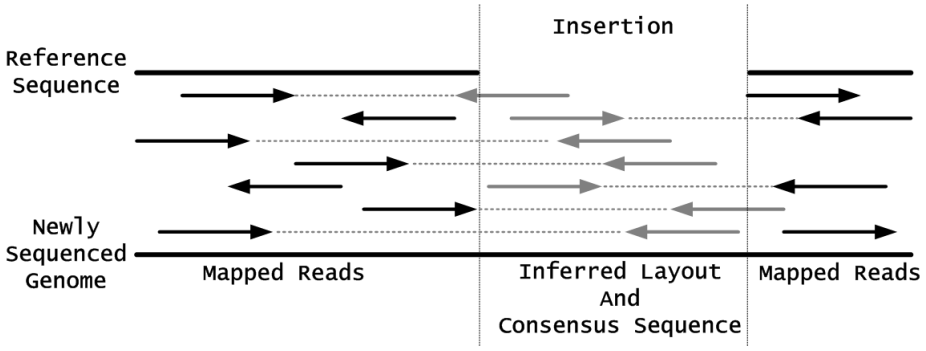


Figure 7.2: A newly sequenced genome with an unknown insertion with respect to a reference genome. The mapped reads (black lines) can be used to infer the layout of the mate pairs (gray lines). Mate pairs are indicated by arrows pointing to each other and the connecting, dotted line in-between them. From this inferred layout a multi-read alignment can be computed.

phase are unnecessary. The weakness, however, is that we have no placement information for unmapped reads, except possible mate-pair information. Consequently, one needs to keep the number of unmapped reads small by choosing an appropriate reference genome.

A great variety of tools has been designed and developed specifically for the purpose of mapping short reads. Examples are MAQ (Li et al., 2008a), SOAP (Li et al., 2008b), Bowtie (Langmead et al., 2009) or RazerS (Weese et al., 2009) from the SeqAn library. Almost all programs use a two step protocol: (1) A filtration algorithm is applied in order to identify candidate regions that possibly contain a match and (2) these candidate regions are verified for true matches. Filtration methods are based on single (Kent, 2002; Ma et al., 2002) or multiple seeds (Li et al., 2003), the pigeonhole principle (Navarro and Raffinot, 2002; Li et al., 2008a,b), or counting lemmas using (gapped) q-grams (Burkhardt et al., 1999; Rasmussen et al., 2005). Verification methods encompass semi-global alignment algorithms (Myers, 1999) or local-alignment algorithms (Smith and Waterman, 1981).

Given the final set of mapped reads, we can, however, only infer the mutual alignment of reads to themselves from this reference-based mapping. This implies that we cannot infer a correct multi-read alignment in novel insertions that are not present in the reference sequence. For small insertions, we might encounter reads bridging the insertion but for large-scale insertions we can only use anchored mate-pairs where exactly one read of the pair mapped to the reference (see Figure 7.2).

	...	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	...
Consensus	G	A	T	T	G	A	G	A	C	T	G	T	A	–	C	T	G	A	T	C
← *Read*$_1$	G	A	T	T	A	A	G	A	C											
→ *Read*$_2$		A	T	T	G	A	G	A	C	T	G	T	A	–	C	T	–	A		
← *Read*$_3$				T	G	A	G	–	C	T	G	C	A	T	C	T	G	A	T	
← *Read*$_4$					G	A	G	A	C	T	G	T	A	–	C	T				
→ *Read*$_5$						A	G	–	C	T	G	C	A	–	C	T	G	A	A	C
→ *Read*$_6$							G	A	C	T	G	T	A	–	C	T	G	A		
→ *Read*$_7$							G	–	C	T	G	C	A	–	C	T	G	A	T	C

Figure 7.3: A multi-read alignment showing seven reads. The read orientation is indicated by the arrow in front of the read name. The top row shows the consensus sequence. The consensus letter in each column is the most frequent letter with ties broken arbitrarily. By iterating through the alignment column by column, one can identify sequencing errors in column 13, 22 and 25, and putative polymorphisms in column 16 and 20.

Current mate-pair libraries produce, however, mate-pairs of quite varying insert sizes. Libraries with about 10% size deviation are rather the rule than the exception.

In summary, we encounter two multi-read alignment scenarios in de novo and reference guided sequence assembly projects. In the first scenario the reads have quite accurate layout positions. This case corresponds to a situation where, for instance, all the reads could be mapped and we are only unsure about small insertions. In the second scenario, however, we have a number of unmapped reads and only with the help of mate-pair information we can infer the positioning of the reads. In de novo assembly projects the accuracy of the read layout largely depends on the assembler's layout module, so we might encounter both situations here. To address both scenarios, we designed, developed and experimentally verified two algorithms for multi-read alignments, a ReAligner algorithm (Anson and Myers, 1997) for accurate layout positions and a robust graph-based multi-read alignment algorithm for inaccurate layout positions. Both algorithms are described in detail in the next two sections.

To conclude this overview we also want to explain the desired properties of a final multi-read alignment. In contrast to protein alignments, multi-read alignments are

only seldomly inspected manually. Quite often researchers automatically process such large-scale alignments by tools that call SNPs, detect genomic variations or separate haplotypes. Because of that, it is highly beneficial if the alignments are so accurate that a simple column-based consensus calling is possible. A very small example highlighting this important property is shown in Figure 7.3. The illustrated multi-read alignment allows the distinction of sequencing errors from true variations. For instance, the 'A' in column 13, the 'T' in column 22 or the '−' in column 25 are most likely sequencing errors because they are not supported by any other read overlapping this position. Column 16 and column 20, however, are more likely to be true polymorphisms since both variants 'A/−' and 'C/T' are supported by a number of reads. Note that sequencing errors as well as true polymorphisms can introduce spurious overlaps, which is one of the main obstacles that needs to be overcome by a consensus method.

7.2 ReAligner

In this section we describe the initial ReAligner algorithm (Anson and Myers, 1997) and the extensions made by us. As before, we first describe the algorithmic components and then explain our implementation.

7.2.1 Algorithmic components

The ReAligner algorithm is a so-called round-robin algorithm. It starts with an initial, possibly erroneous multi-read alignment, removes every read one by one and realigns each removed read to the multi-read alignment defined by the remaining reads. The assumption of the algorithm is that the initial errors are only local whereas the multi-read alignment itself is globally correct. Hence, we want to optimize the alignment with respect to the approximate layout. As for protein alignments, it is not immediately obvious what mathematical function describes an optimal alignment. A possible objective is to minimize the number of errors with respect to a consensus sequence. The simplest consensus sequence can be derived by choosing the most frequent letter in each column. Hence, for a single alignment column u, we pick the most frequent letter c that minimizes the number of errors of all other aligned reads

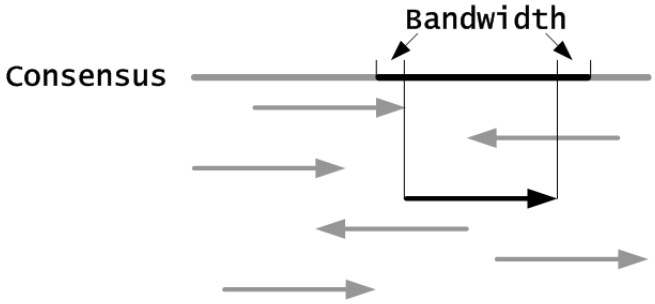


Figure 7.4: During each realignment iteration a read is removed from the multi-read alignment and realigned to a banded portion of the consensus derived from the remaining reads.

spanning the given column.

$$\min_{c \in \tilde{\Sigma}} |\{a_u^i : a_u^i \neq c\}|$$

Remember that a_u^i is the alignment character in row i and column u. i iterates over all reads spanning column u. The constraint that reads have to span the given column ensures that gaps preceding and following a read are not considered. In a multi-read alignment these leading and trailing gaps are of no interest. In summary, the objective function is to find a multi-read alignment $\mathcal{A}$ of length $\tilde{l}$ that minimizes

$$Score(\mathcal{A}) = \sum_{0 \leq u < \tilde{l}} \min_{c \in \tilde{\Sigma}} |\{a_u^i : a_u^i \neq c\}|$$

and respects the initial global layout. The last condition is ensured by realigning each read only within a band of size b surrounding the read's original position as shown in Figure 7.4. The actual removal of a read, its realignment and its reintegration into the alignment is described next.

Profile generation and read removal

In a preliminary step, we condense the initial multi-read alignment to a profile $\mathcal{P}$. $\mathcal{P}$ is a string of profile characters p_u and $\mathcal{P}$ is of length $\tilde{l}$ where $\tilde{l}$ is the length of the initial multi-read alignment. p_u stores the number of gaps and the number of occurrences of each letter $\sigma \in \tilde{\Sigma}$ appearing in column u of the alignment. Such a profile can be immediately used to score the full multi-read alignment according to the above objective function. We retrieve the most frequent letter in every column and subtract its occurrence count from the total number of letters and gaps in the given column. The subsequent removal of $read_i$ involves a decrement of the character

counts in all spanned columns and the possible removal of those alignment columns that contain only gaps.

ReAlignment and Scoring

The realignment of $read_i$ has to be bounded to an ϵ-environment in order to keep the global structure. Hence, $read_i$ is only realigned to a subsequence of the profile. Let $begin_i$ and end_i be the original begin and end position of $read_i$ in profile $\mathcal{P}$. The desired subsequence is $\mathcal{P}_{xy}$ such that $x = begin_i - b$ and $y = end_i + b$ where b is a user-defined bandwidth. Given the profile subsequence and the read, we can realign both sequences using dynamic programming and a user-defined scoring function. Two default scoring functions have been implemented and are described hereafter. The first scoring function uses the set of consensus letters $\mathcal{C}_u$ for each $p_u \in \mathcal{P}_{xy}$. $\mathcal{C}_u$ contains the most frequent letters appearing in column u. We then score a read character $s \in \Sigma$ using

$$\delta_c(s, \mathcal{C}_u) = \begin{cases} 0 & : s \in \mathcal{C}_u \\ -1 & : s \notin \mathcal{C}_u \end{cases}$$

Such a scoring is in general very useful but has the drawback of losing all the information about the letters that are not in $\mathcal{C}_u$. In other words, an alignment column with 9 A's and 10 '−'s would score a gap with 0 but all other characters with -1 even the quite plausible A. To circumvent such cases one could use a fractional score δ_f. Let a_u^i be again the alignment character in row i and column u and n be the number of reads spanning column u. Then δ_f is defined as

$$\delta_f(s, \mathcal{A}_u) = \begin{cases} -|\{a_u^i : a_u^i \neq s\}|/n & : n > 0 \\ -1 & : n = 0 \end{cases}$$

δ_f is 0 if all characters in column u are equal to s and the more negative the more characters disagree with s. Hence, the score mirrors the fractional content of the column equal to s. In our implementation both scoring functions δ_f and δ_c can be used independently or as a weighted average. The default implementation uses the weighted combination proposed in the original ReAligner paper (Anson and Myers, 1997).

$$\delta_{cf}(s, \mathcal{A}_u, \mathcal{C}_u) = \frac{1}{2} \cdot \delta_c(s, \mathcal{C}_u) + \frac{1}{2} \cdot \delta_f(s, \mathcal{A}_u)$$

Unfortunately, both scoring functions are only heuristics to approximate the objective function. An optimal layout cannot be guaranteed but in practice the algorithm

works very well. Using the above scoring function and a banded alignment algorithm, the removed read is realigned to the profile subsequence $\mathcal{P}_{xy}$.

Inserting the read back into the global multi-read alignment

Based upon the profile to read alignment we can reinsert the read. Any gap in the profile results in a new gap column in the multi-read alignment containing only the read's character and gaps. Gaps in the read as well as matches and mismatches can be simply inserted into the existing alignment and profile sequence without adding any new alignment column.

This process is then iterated for all other reads. At the end, the new profile is rescored. Let $Score(\mathcal{A}')$ denote the multi-read alignment after one full realignment iteration of all reads. Then the process is terminated if $Score(\mathcal{A}') \geq Score(\mathcal{A})$ since we are minimizing the objective function

$$Score(\mathcal{A}) = \sum_{0 \leq u < \tilde{l}} \min_{c \in \tilde{\Sigma}} |\{a_u^i : a_u^i \neq c\}|$$

Extensions

We extended the above basic ReAligner algorithm to handle the increasingly popular reference guided sequence assemblies as well as RNA-Seq or ChIP-Seq approaches. RNA-Seq refers to the use of the new sequencing platforms to study the RNA content of a sample whereas ChIP-Seq highlights binding sites of proteins. For all of these methods, we will observe reference genome parts that are covered with reads and other parts that are not covered with reads. For instance, most non-coding regions in RNA-Seq will be uncovered unless there are random matches. In the above cases, we cannot readily apply the realignment algorithm because it would shrink the whole alignment to the covered parts and eliminate all uncovered regions. In particular, for RNA-Seq and ChIP-Seq the relationship to the original reference genome is crucial. Because of that, we added an option to the algorithm that allows the inclusion of a reference sequence. That sequence is used during the consensus generation and the realignment of reads but it is excluded from being realigned itself to all other reads. We also included the reference sequence in the output to highlight differences of the newly computed consensus to the old reference.

The second extension we made is an option to substitute the basic dynamic programming scoring using position-dependent gap costs with an algorithm that

supports position-dependent gap costs *and* an additional gap-opening penalty. This option is useful for low coverage regions. In these regions the consensus tends to be unreliable and sequences might get disrupted by many interspersed gaps. To avoid such a behavior, we introduced this new scoring method.

7.2.2 Implementation

In Chapter 4 we introduced the FragmentStore data structure that is used throughout the realignment algorithm. Specifically, the algorithm makes use of the read store, the contig store and the aligned read store. The read store and the contig store are merely queried to retrieve the actual sequence data of the reads and the contig chosen for realignment. The aligned read store, however, is used multiple times and because of that, we present it once again here.

Aligned Read Store	
•Index:	none
•Members:	read id
	contig id
	pair match id
	begin pos
	end pos
	gaps
	unique id

The aligned read store is the only store within the fragment store that has no indexable field. This implies that the store can be arbitrarily sorted.

The input to the realignment algorithm is the fragment store, a scoring object and a contig id. The consensus score δ_c, the fractional score δ_f and the combined consensus and fractional score δ_{cf} have been subsumed under the common scoring interface introduced in Section 5.2. The corresponding tags are ConsensusScore, FractionalScore and WeightedConsensusScore. Hence, a consensus score using integers can be declared as Score<**int**, ConsensusScore>. The first step in the algorithm is to sort the aligned read store according to the contig id. We then select all the reads within the contig and sort that range according to the begin pos. Afterwards, we iterate over all the reads, create the profile $\mathcal{P}$ for the multi-read alignment and calculate the initial $Score(\mathcal{A})$. Finally, a simple loop shown in Algorithm 3 is entered and executed as long as the score of the newly calculated multi-read alignment is decreasing. The inner loop removes each read from the profile $\mathcal{P}$, realigns it to the banded profile $\mathcal{P}_{xy}$ using linear or constant gap costs (e.g. BandedNeedlemanWunsch

Algorithm 3 ReAlignment loop

1: $score = Score(\mathcal{A})$
2: **repeat**
3: $score\_old = score$
4: ReAlign all reads and create $\mathcal{A}'$
5: $score = Score(\mathcal{A}')$
6: **until** $score \geq score\_old$

or BandedGotoh) and reintegrates the read in the multi-read alignment. The actual bandwidth b and the used dynamic programming algorithm are configurable parameters. Optionally, the user can specify to include a reference sequence. This sequence is not realigned but used to create the profile string $\mathcal{P}$.

7.3 Graph-based Multi-Read Alignment

The key difference between the realignment algorithm and the graph-based multi-read alignment algorithm is that the former trusts and relies on the global layout of the reads whereas the latter builds such a reliable global layout from a possibly inaccurate initial alignment using pairwise overlap alignments. Hence, the realignment algorithm is only able to correct small local inconsistencies whereas our second method, the graph-based multi-read alignment algorithm, is more robust in that respect since it only requires a "rough" layout of the reads. How "rough" the layout can be depends on several factors, including the quality of the reads, the length of the reads and the coverage. The quality and the length of a read have great influence on the accuracy of the computed pairwise overlaps. The more false positive overlaps there are, the more difficult is the correct positioning of each and every read. Similarly, a higher coverage usually helps to differentiate random overlaps from true alignments. In Part III of this thesis we will see the strength and weaknesses of both methods on real and simulated data.

7.3.1 Algorithmic components

The multi-read alignment algorithm can be broadly subdivided into four processing steps: (1) computation of pairwise overlap alignments, (2) alignment graph construction, (3) consistency extension and (4) a graph-based progressive alignment.

Pairwise overlap alignments

The assumed input of the algorithm is a set of reads with their estimated begin and end positions. These estimated layout positions can, for example, stem from an assembler's layout module or they are inferred from mate pair information as shown previously in Figure 7.2 on page 91. Especially for the second case, the positions tend to deviate strongly from the true positions depending on the standard deviation of the used paired-end library. If the deviation is small the algorithm can be configured to use the positions of two given reads directly for estimating the alignment diagonal of a banded overlap alignment with affine gap costs (Gotoh, 1982). As mentioned previously, a banded overlap alignment initializes the dynamic programming matrix with zeros and computes only a band of size b around the estimated alignment diagonal where b is the bandwidth. Besides the accuracy of the layout positions, the choice of b depends on two parameters: the sequencing error rate and the length of the overlap. Hence, the baseline for b is a configurable parameter that is adjusted linearly based on the length of the overlap. Usually, it is unnecessary to compute all possible pairwise overlaps, especially for deep coverage sequencing projects. For that reason we provide a parameter that adjusts how many overlaps are computed per given read. The more error-prone the reads are, the more overlaps one should compute per read. If the deviation is large or in other words, the initial positions are unreliable then we also support a window based computation of pairwise overlaps. That is, all pairwise overlap alignments of reads lying in a user-defined window are computed with a standard dynamic programming algorithm.

Subsequent to the computation of overlap alignments, we select all overlaps of significant quality and length. Similar to the bandwidth, both parameters are adaptable from the command line. The selected overlaps are then subdivided into ungapped alignments (segment matches) as explained previously in Chapter 6.

Alignment graph construction

We reuse the multiple segment match refinement algorithm (Rausch et al., 2008a) introduced in Chapter 6 to subdivide overlapping segment matches into distinct submatches so that no segment match begins or ends within another segment match. We then construct the alignment graph by defining a vertex for each sequence segment and an edge for each segment match. The weight of this edge depends on the

Figure 7.5: The alignment on the left shows globally related sequences whereas the one on the right shows a simplified multi-read alignment. Note that the direction of the alignment is solely dependent on the edges.

quality of the segment match. Note that the alignment graph is also suitable to represent partially overlapping sequences as shown in Figure 7.5.

Consistency extension

We once again apply the triplet extension (Notredame et al., 2000) to incorporate multiple alignment information in the pairwise edges. However, we do not compute a full triplet extension because in case of hundreds or even thousands of reads this would be too expensive. We limit the triplet extension to a reweighting of the existing edges but we do not insert new edges as described before for the protein multiple alignment.

Graph-based progressive alignment

In the last step, the consistency-enhanced alignment graph is used to progressively align the reads according to a guide tree. This guide tree is constructed from the overlap alignment scores using a sparse distance matrix and the fast UPGMA algorithm (Sokal and Michener, 1958). A sparse distance matrix is used instead of an ordinary matrix because for each read only c other reads are expected to overlap where c is the assembly coverage. The guide tree ensures that the best quality overlaps are aligned first whereas the difficult and error-prone overlaps caused by reads with many sequencing errors come in late, when partial alignments along subtrees are already quite large and fixed. The progressive alignment itself is completely independent of the nature of the sequences. Given an input alignment graph, it builds a multiple alignment along the guide tree simply by aligning strings of ver-

tices as explained before. There is one notable exception: In case of a multi-read alignment the profiles will have only about c vertices at each position where c is again the coverage. Thus, the amount of required memory depends on the coverage and the source sequence length. It is, however, largely independent of the number of reads. This is a key distinction of our method to current multiple sequence alignment tools where the profiles grow linearly with the number of sequences. In a final post-processing step we compute the consensus sequence and convert the final profile into a multi-read alignment.

7.3.2 Implementation

The pairwise overlap computations make use of the BandedGotoh algorithm. During the traceback of each pairwise alignment we fill a string of segment matches, String<Fragment<> >. We append the newly found segment matches to the global segment match store if and only if the overlap alignment is of significant length and quality. We also store the pairwise alignment score in a sparse distance matrix. This sparse distance matrix is implemented as an undirected graph such that each sequence is a vertex and an edge between two sequences represents the alignment score. The sparse distance matrix is the input to the UPGMA tree reconstruction algorithm. The output is a guide tree that indicates when every read is added to the growing multi-read alignment. Because of possibly contradicting matches, the global set of segment matches is refined using the function matchRefinement and subsequently, the initial alignment graph is built. That graph is extended using the reduced triplet extension and finally, a progressive alignment computes a valid trace using the heaviest common subsequence algorithm (Jacobson and Vo, 1992) in each progressive step. The final multi-read alignment can be printed to a simple text file or written in AMOS message file format. The AMOS library provides a number of file conversion utilities that can be used to convert this message file to Arachne (Batzoglou et al., 2002) ace files or Celera Assembler (Myers et al., 2000) files, for instance. In addition, the AMOS library provides a contig viewer called Hawkeye (Schatz et al., 2007) that can be used to visualize the multi-read alignment.

Part III

Tools and Applications

SeqAn::T-Coffee

In Chapter 6 we described the MSA algorithm that is used in our tool SeqAn::T-Coffee. Part of the tool is also a meta-alignment method called SeqAn::M-Coffee. Both methods are influenced by a variety of parameters that can be set on the command line. In Section 8.2, we first evaluate the impact of the different parameter choices on the final alignment. As our standard of truth, we took the BAliBASE 3.0 (Thompson et al., 1999a, 2005) benchmark. The benchmark is subdivided into 6 standard reference sets RV11, RV12, RV20, RV30, RV40 and RV50. Each of these sets contains a number of alignment instances of full-length sequences. For each alignment instance the benchmark provides a corresponding reference alignment with an annotation of core block regions where the sequences can be unambiguously aligned. These core blocks are used to compare a computed alignment with the reference alignment. The benchmark also includes a scoring program that calculates the sum of pairs score (SP) and the total column score (TC) on the core block regions (see Section 8.1). In Section 8.3 we compare our algorithm to other state-of-the-art methods using the BAliBASE 3.0 and PREFAB 4.0 benchmark data sets. Finally, this chapter concludes with a brief overview of the command line of our tool.

8.1 SP and TC Score

The SP score (Thompson et al., 1999b) measures how many pairs of sequence character have been correctly aligned with respect to the BAliBASE reference alignment. Let $c_u^{i,j} \in \{0, 1\}$ be a boolean indicator variable with $i \neq j$. If $c_u^{i,j} = 1$ then the alignment character a_u^i of sequence $\tilde{S}^i$ in column u is correctly aligned (with respect to the reference alignment) to the alignment character a_u^j of sequence $\tilde{S}^j$ in column u.

If this is not the case $c_u^{i,j} = 0$. Hence, in an alignment $\mathcal{A}$ of n sequences, the score for column u is equal to

$$C_u(\mathcal{A}) = \sum_{0 \leq i < j < n} c_u^{i,j}$$

The SP score $Score_{SP}$ is then the fractional number of correctly aligned pairs of sequence characters.

$$Score_{SP}(\mathcal{A}) = \frac{\sum_{u=0}^{\tilde{l}-1} C_u(\mathcal{A})}{\sum_{u=0}^{\tilde{r}-1} C_u^r} = \frac{\sum_{u=0}^{\tilde{l}-1} \sum_{0 \leq i < j < n} c_u^{i,j}}{\sum_{u=0}^{\tilde{r}-1} C_u^r}$$

where $\tilde{l}$ and $\tilde{r}$ are the number of columns in the computed alignment and in the reference alignment, respectively. C_u^r is the score C_u of the u-th column in the *reference* alignment.

The TC score (Thompson et al., 1999b) measures how many total alignment columns have been correctly computed compared to the reference alignment. For the TC score, $C_u^*(\mathcal{A}) = 1$ if all the sequence characters in column u are also aligned in the reference alignment, otherwise $C_u^*(\mathcal{A}) = 0$. The TC score $Score_{TC}$ is thus the fractional number of correctly aligned columns.

$$Score_{TC}(\mathcal{A}) = \frac{\sum_{u=0}^{\tilde{l}-1} C_u^*(\mathcal{A})}{\tilde{l}}$$

8.2 Parameter Evaluation

Using progressive alignment, the MSA computation can be configured by a range of parameters. In this section, we explore the impact of these parameters on the final alignment quality. In particular, we analyze the effect of choosing appropriate gap penalties, scoring matrices, pairwise alignment algorithms and tree reconstruction methods.

8.2.1 Gap penalties

The initial input segment matches of our method can be generated using external tools such as BLAST or MUMmer or internal alignment algorithms. These alignment algorithms are either global or local algorithms. Both kinds of algorithms use a scoring matrix and gap penalties to compute the optimal pairwise alignment. Using the BAliBASE RV11 reference set, we computed a final multiple sequence alignment for all reference set alignment instances using various gap opening and gap extension

penalties. In Figure 8.1 and Figure 8.2 we show the SP and TC score for all final multiple alignments using only *global* pairwise alignments to generate segment matches. In Figure 8.3 and Figure 8.4 we show the SP and TC score for all final multiple alignments using only *local* pairwise alignments to generate segment matches. The contour lines in the bottom plane clearly show the range of reasonable gap penalty values to compute global and local pairwise alignments. For global alignments, the gap opening cost should be between -11 and -15 and the gap extension penalty should be equal to -1. For local alignments, a less stringent gap opening penalty should be applied, -8 or -9 seems to be a good choice. For local alignments, the best TC and SP score was achieved with $(gex, gop) = (-1, -8)$. For global alignments, the best TC score was achieved with $(gex, gop) = (-1, -11)$ whereas the best SP score was achieved with $(gex, gop) = (-1, -13)$. For all subsequent analyses, we fixed the gap penalties for local alignment algorithms to $(gex, gop) = (-1, -8)$ and for global alignment algorithms to $(gex, gop) = (-1, -13)$.

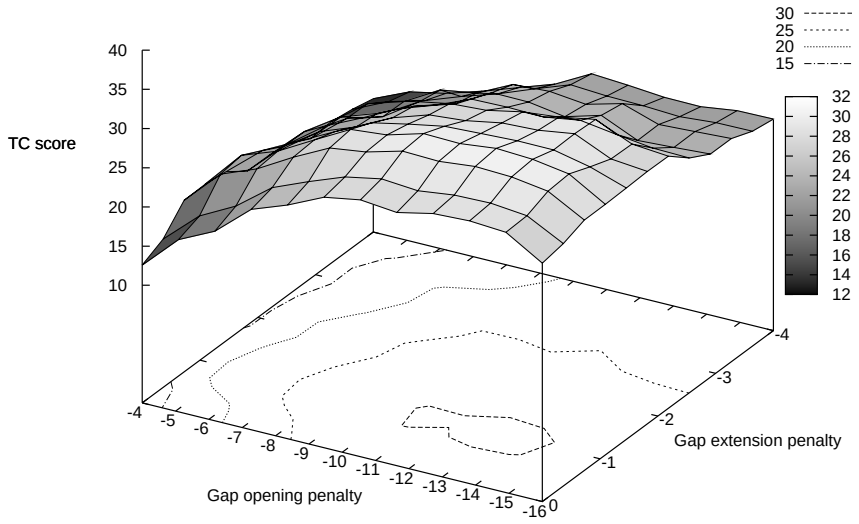


Figure 8.1: The average TC score on the first reference set of BAliBASE using different gap opening penalties (*gop*) and gap extension penalties (*gex*). Input segment matches were derived from global alignments only, using a BLOSUM62 scoring matrix. The contour lines at the bottom plane indicate that $-11 \geq gop \geq -15$ and $gex = -1$ are reasonable choices for global pairwise alignments according to the TC score.

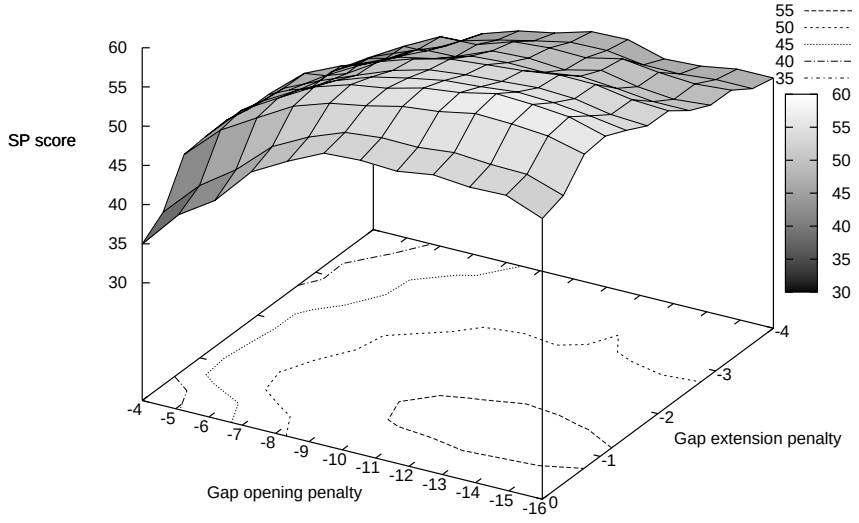


Figure 8.2: The average SP score on the first reference set of BAliBASE using different gap opening penalties (*gop*) and gap extension penalties (*gex*). Input segment matches were derived from global alignments only, using a BLOSUM62 scoring matrix. The contour lines at the bottom plane indicate that $-11 \geq gop \geq -16$ and $gex = -1$ are reasonable choices for global pairwise alignments according to the SP score.

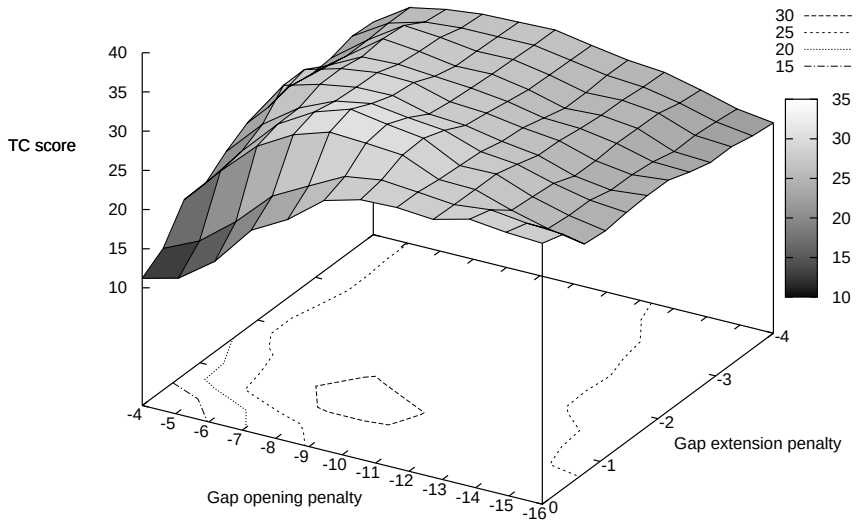


Figure 8.3: The average TC score on the first reference set of BAliBASE using different gap opening penalties (*gop*) and gap extension penalties (*gex*). Input segment matches were derived from local alignments only, using a BLOSUM62 scoring matrix. The contour lines at the bottom plane indicate that $-8 \geq gop \geq -11$ and $gex = -1$ are reasonable choices for local pairwise alignments according to the TC score.

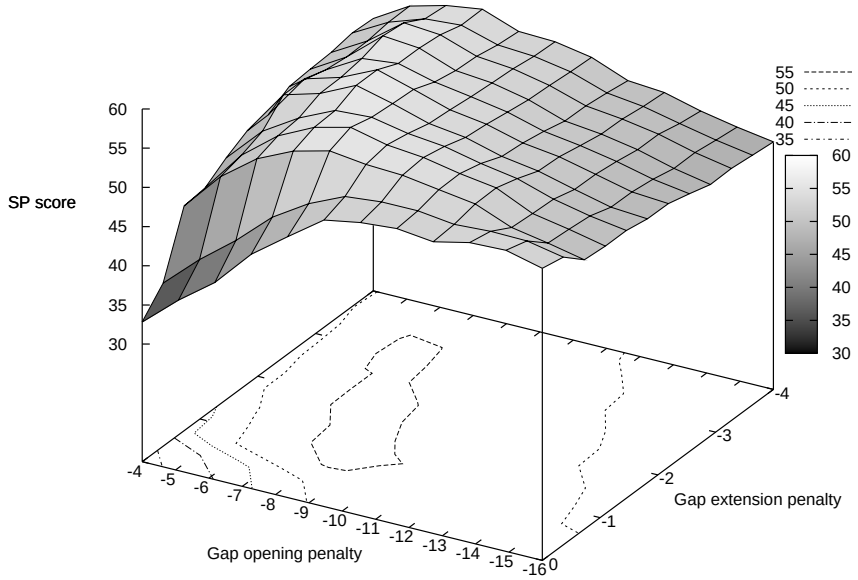


Figure 8.4: The average SP score on the first reference set of BAliBASE using different gap opening penalties (*gop*) and gap extension penalties (*gex*). Input segment matches were derived from local alignments only, using a BLOSUM62 scoring matrix. The contour lines at the bottom plane indicate that $-7 \geq gop \geq -9$ and $-1 \geq gex \geq -3$ are reasonable choices for local pairwise alignments according to the SP score.

8.2.2 Scoring matrix

The BLOSUM (Henikoff and Henikoff, 1992) and PAM (Dayhoff et al., 1979) matrices are frequently used amino acid substitution matrices. The BLOSUM matrices are based upon multiple local alignments of conserved blocks occurring in amino acid sequences. The BLOSUM62 matrix, for instance, has been constructed using an alignment of sequences with 62% identity. The observed changes in these alignments are then counted and converted to the specific scoring matrix. In contrast to the BLOSUM matrices that are all based upon observed amino acid changes, the PAM X matrices are all extrapolated from PAM1 by means of taking the x-th power of the initial PAM1 matrix. PAM stands for "percent accepted mutation" and the PAM1 matrix has been calculated from an alignment of sequences with 99% identity. Hence, BLOSUM matrices with small numbers such as BLOSUM45 assume distantly related sequences whereas BLOSUM80 is suitable for closely related proteins. For PAM matrices it is the opposite, PAM250 is used for distantly related proteins whereas PAM45 can be used for closely related sequences. In Figure 8.5 we show the average SP and TC score for all BAliBASE reference sets using different substitution matrices. The choice of the specific substitution matrix has surprisingly little influence on the final alignment quality, except for PAM45 that performs poorly on some of the alignment instances. Overall the BLOSUM matrices seem to be slightly better than the PAM matrices and BLOSUM62 seems to be the best choice on average.

8.2.3 Pairwise alignment algorithms

Most progressive alignment tools start with pairwise global and / or local alignments such as T-Coffee (Notredame et al., 2000), MAFFT (Katoh et al., 2005) or our own method. We analyzed the effect of the initial pairwise alignment method on the final SP and TC score. The results are shown in Figure 8.6. For the reference set RV40 with long terminal extensions, the overlap method performs significantly better than the global alignment method because it does not penalize leading and terminal gaps. The local alignment method, however, is equally suitable to align such sequences. For the reference set RV11 with sequences of similar length, the global alignment method outperforms the overlap method. What stands out is that adding additional pairwise alignment information in terms of combining global,

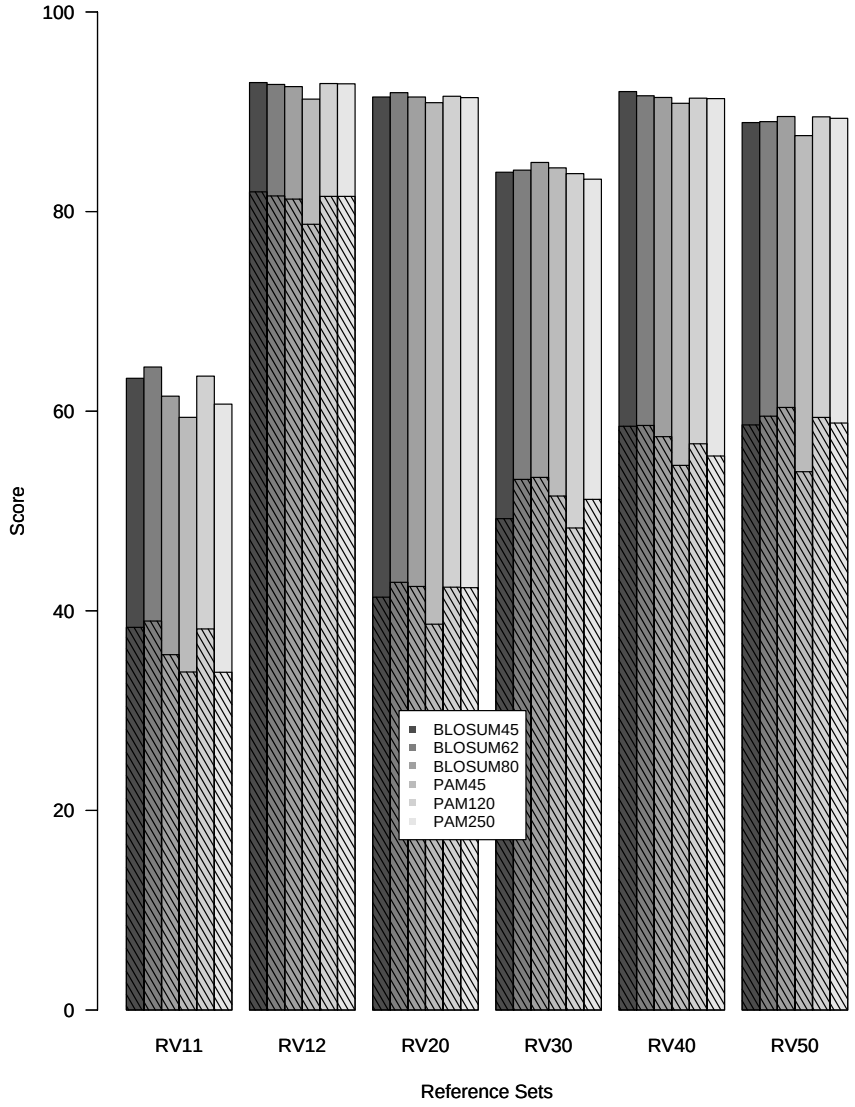


Figure 8.5: A comparison of scoring matrices used to compute the initial set of segment-matches. The SP scores (bars in gray) and the TC scores (shaded bars) on all BAliBASE reference sets RV11, RV12, $\cdots$, RV50 are shown.

local and overlap alignments does on average improve the final MSA quality. This argues in favor of the triplet extension that seems to be able to extract the true multiple conserved patterns. The scores increase largely for combining local and global alignments and to a lesser extent when adding overlap alignments. Because of that, the default alignment method uses only global and local alignments.

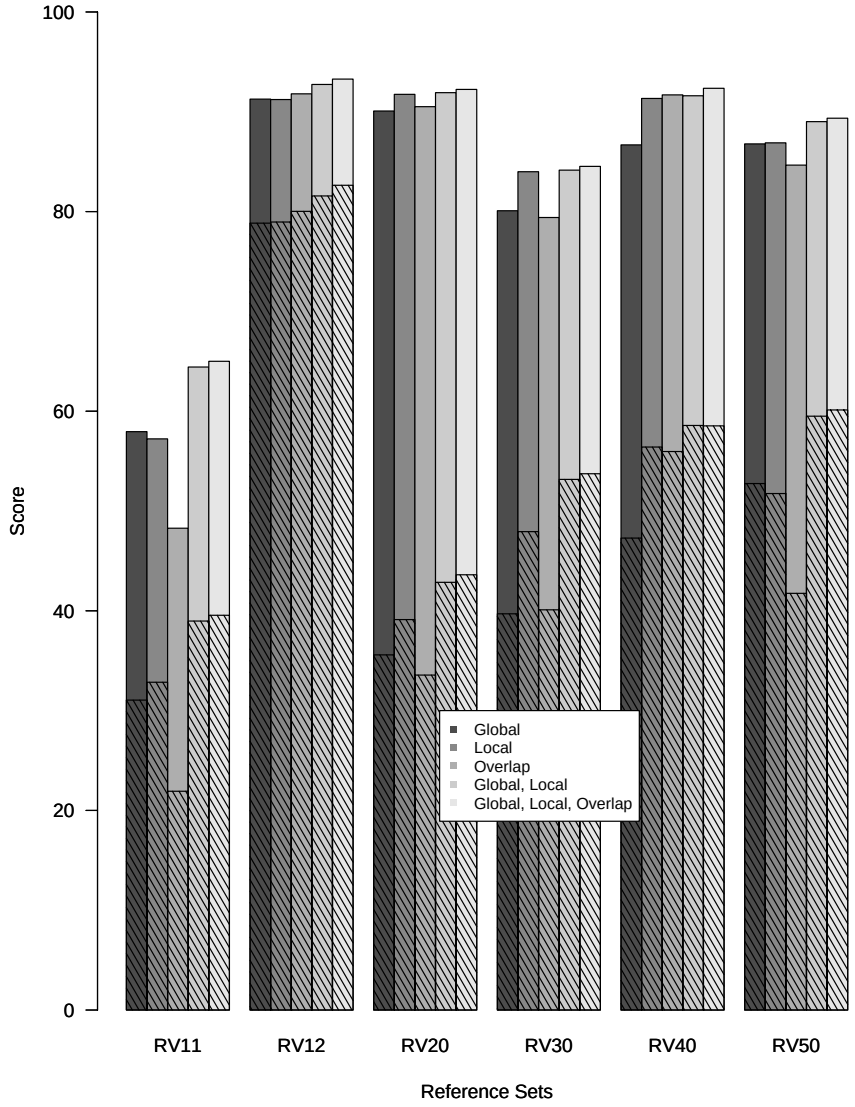


Figure 8.6: Different combinations of pairwise alignment algorithms can be used to compute the initial segment-matches. The SP scores (bars in gray) and the TC scores (shaded bars) on all BAliBASE reference sets RV11, RV12, $\cdots$, RV50 are shown.

8.2.4 Tree reconstruction

We implemented two different tree reconstruction algorithms, namely neighbor-joining and UPGMA. The UPGMA algorithm can either use single, complete, average or weighted average clustering. In Figure 8.7 and Figure 8.8 we show boxplots of the TC and SP scores from all implemented tree reconstruction algorithms on

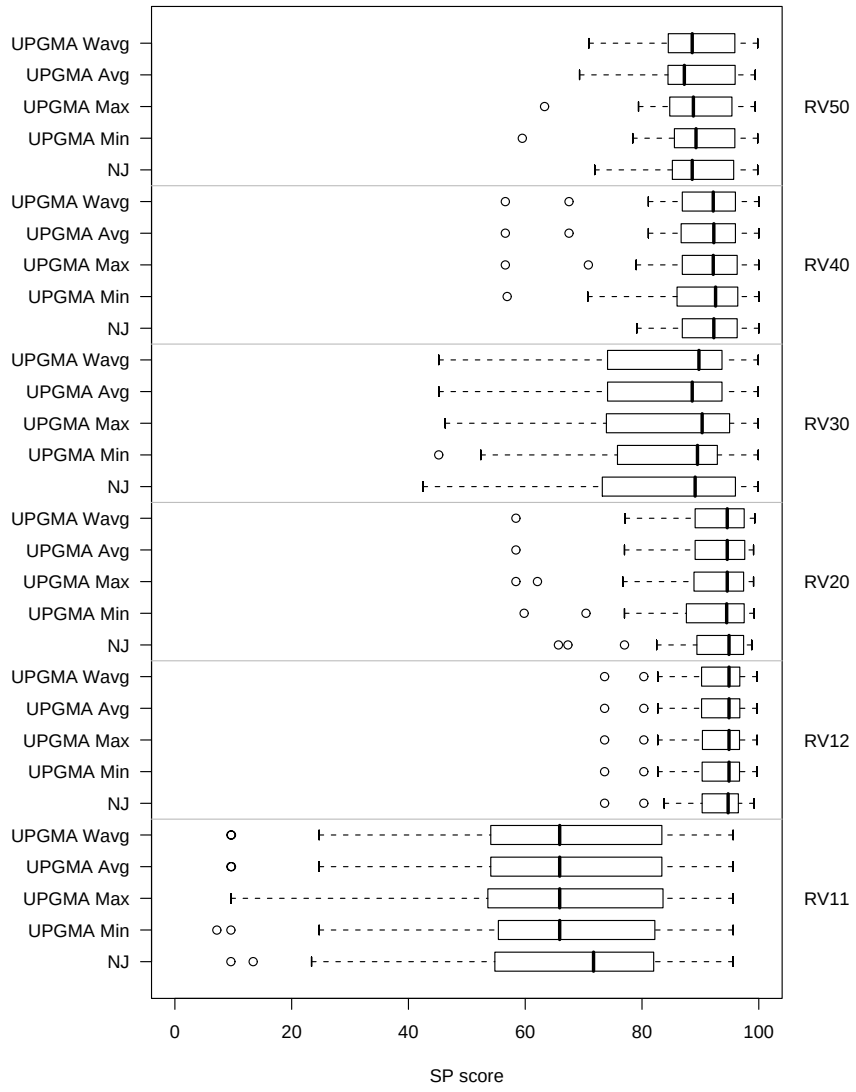


Figure 8.7: Boxplots showing for each reference set RV11, RV12, $\cdots$, RV50 the SP scores of each tree reconstruction method.

the full BAliBASE reference benchmark, subdivided according to the reference sets RV11, RV12, $\cdots$, RV50. The boxplot highlights the distribution of scores by means of showing the median (the vertical bar in the middle of each box), the lower and upper quartile (the left and right boundary of the box) and the extreme values (the end of each "whisker" to the left and right). Based on the above experiments, the differences among the tree reconstruction algorithms are minor. All methods have a similar median and variance although the neighbor-joining method slightly outperforms all other methods, in particular on the reference set RV11 and for the TC

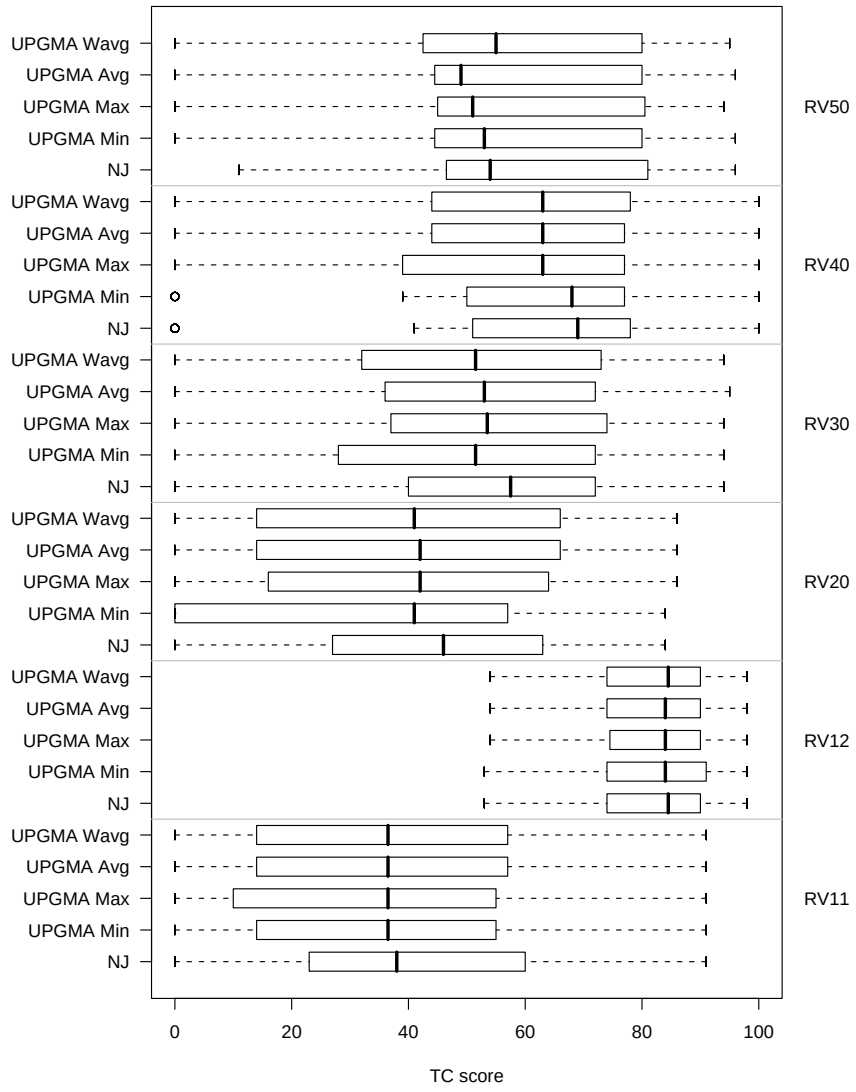


Figure 8.8: Boxplots showing for each reference set RV11, RV12, $\cdots$, RV50 the TC scores of each tree reconstruction method.

score also on the reference set RV20 and RV30. What stands out is that the TC score variation is huge compared to the SP score variation because a single misaligned sequence can cause the TC score to drop to zero. The SP score is in that sense more robust since it evaluates each pair of aligned characters independently of all other pairs. For all subsequent analyses, we used the neighbor-joining method as the default tree reconstruction algorithm.

8.3 Results

SeqAn::T-Coffee is our versatile multiple sequence alignment tool for amino acid and nucleotide sequences. The meta-method of the tool is called SeqAn::M-Coffee. As the original M-Coffee (Wallace et al., 2006), SeqAn::M-Coffee can be configured to use different subaligners such as MAFFT (Katoh et al., 2002), ProbCons (Do et al., 2004), MUSCLE (Edgar, 2004b) or any other method. We compared both algorithms, SeqAn::M-Coffee and SeqAn::T-Coffee, to a number of other tools on the BAliBASE 3.0 (Thompson et al., 1999a, 2005) and PREFAB 4.0 (Edgar, 2004b) benchmark alignment datasets (see Section 8.3.1 and Section 8.3.2). For nucleotide sequences, such benchmarks are still missing. Therefore we used the genomic sequences of six adenoviruses and four serotypes causing dengue fever to test all methods for aligning long DNA sequences (see Section 8.3.3). We downloaded for all tools the newest, stable version available. Our own method and all other methods were run with default parameters on all data sets. If input or output files needed to be converted to other formats we used the EMBOSS version 6.0.1 library (Rice et al., 2000). We compared our method with AMAP version 2.2 (Schwartz and Pachter, 2007), Clustal W version 2.0.11 (Larkin et al., 2007), DIALIGN-TX version 1.0.2 (Subramanian et al., 2005, 2008), Kalign version 2 (Lassmann and Sonnhammer, 2005), MAFFT version 6.71 (Katoh et al., 2002), MUSCLE version 3.7 (Edgar, 2004b,a), Opal version 1.0.3 (Wheeler and Kececioglu, 2007), POA version 2 (Grasso and Lee, 2004), ProbCons version 1.12 (Do et al., 2004), T-Coffee version 8.06 (Notredame et al., 2000) and M-Coffee version 8.06 (Wallace et al., 2006). We could not use ABA version 1.01 (Raphael et al., 2004) since the tool outputs a De Bruijn graph in DOT format instead of a true multiple alignment. Although this is beneficial to identify repeats or shuffled domains, it would also be highly interesting to see the method's capabilities to compute collinear alignments.

8.3.1 BAliBASE

BAliBASE 3.0 is the most widely used protein benchmark. We applied all tools to all available reference sets in BAliBASE and computed for all alignment instances the SP and TC score for each aligner. The results on BAliBASE are summarized in Table 8.1.

	RV11 (38)		RV12 (44)		RV20 (41)		RV30 (30)		RV40 (49)		RV50 (16)		CPU
Aligner	SP	TC	SP	TC	SP	TC	SP	TC	SP	TC	SP	TC	Time (s)
AMAP	50.25**	26.03**	90.66**	78.11**	86.25**	27.17**	70.71**	26.93**	77.05**	37.96**	79.20*	42.38*	19839
Clustal W	50.06**	22.74**	86.43**	71.16**	85.16**	21.98**	72.50**	27.23**	78.93**	39.55**	74.25**	30.75**	1539
DIALIGN-TX	51.52**	26.53**	89.18**	75.23**	87.87**	30.49**	76.18**	38.53**	83.64**	44.82**	82.28**	46.56*	5953
Kalign	60.53*	36.50*	91.21**	78.93**	90.03*	35.90*	81.26	47.60	88.39**	50.39**	82.01*	43.50*	31
MAFFT	66.19	43.79	93.36	83.36	92.70	45.12	**87.07**	58.50	92.19	**60.24**	90.25	59.06	2113
MUSCLE	57.16*	31.79*	91.54**	80.39**	88.91	35.00	81.44*	40.87	86.49**	45.02*	83.52	45.94	1224
Opal	65.78	40.18	93.68	84.02	90.77	34.85	82.62	44.40	84.20	55.22	87.33	49.38	25425
POA	37.96**	15.26**	83.19**	63.84**	85.28**	23.34**	71.93**	28.23**	76.69**	33.67*	71.49**	27.00**	1234
ProbCons	66.97	41.66	94.12	85.55	91.67	40.63	84.53	54.37	90.34	53.22	89.41	57.31	19739
SeqAn::T-Coffee	64.43	38.97	92.74	81.57	91.91	42.85	84.15	53.17	91.60	58.57	89.01	59.50	5687
T-Coffee	66.63	42.08	94.17	85.45	90.79	37.32	83.61	47.50	89.70	53.92	89.44	58.75	29266
M-Coffee [mp]	68.13	44.00	93.99	84.86	92.61	45.59	86.18	57.57	91.91	59.14	90.01	58.12	2493
M-Coffee [mps]	67.88	43.50	93.99	84.41	92.66	45.51	86.46	58.57	91.37	58.18	90.79	63.06	2873
M-Coffee [mpst]	68.38	42.71	94.04	84.48	92.36	43.90	86.12	57.50	91.59	59.92	90.57	**63.31**	3170
SeqAn::M-Coffee [mp]	69.00	44.87	94.32	85.93	**92.99**	47.07	86.66	**60.37**	92.49	57.08	90.34	60.94	550
SeqAn::M-Coffee [mps]	69.20	44.97	94.06	84.75	92.92	**47.29**	86.47	58.70	92.95	59.92	90.79	62.38	921
SeqAn::M-Coffee [mpst]	**69.83**	**45.97**	**94.37**	**86.00**	92.66	44.78	86.70	59.63	**93.01**	59.92	**91.03**	62.81	1180

Table 8.1: BAliBASE 3.0: Average sum-of-pairs scores (SP) and average total column scores (TC) for all reference sets RV11-RV50. The total running time is reported in the last column. The number of multiple alignment files in each reference set is given in parentheses. All scores have been multiplied by 100 and the best program is shown in bold face for each column. The wilcoxon rank test was used to assess significant differences from the best method, indicated by * ($p < 0.05$) or ** ($p < 0.01$). The submethods for all meta-methods (M-Coffee, SeqAn::M-Coffee) are given in brackets with m:MAFFT, p:ProbCons, s:SeqAn::T-Coffee and t:T-Coffee. For the meta-methods, we excluded the running time of the submethods to highlight the overhead induced by the meta-method.

The best single alignment methods are MAFFT, Opal, Probcons, SeqAn::T-Coffee and T-Coffee. Opal failed on three of the BAliBASE alignment instances due to insufficient memory. MAFFT and SeqAn::T-Coffee are more than three times as fast as Opal, Probcons and T-Coffee. All five methods improve largely over the ubiquitously used Clustal W (Larkin et al., 2007). The most accurate methods are the meta-alignment methods, M-Coffee and SeqAn::M-Coffee. SeqAn::M-Coffee using MAFFT, Probcons, SeqAn::T-Coffee and T-Coffee is the best method on the BAliBASE benchmark. In addition, SeqAn::M-Coffee tends to outperform the original M-Coffee on most instances. This suggests an improvement of SeqAn::M-Coffee resulting from the segment-based extension and the graph-based progressive alignment. This improvement in alignment quality compared to M-Coffee comes along with a significant improvement in performance, with SeqAn::M-Coffee only requiring about one third of the CPU time of the original M-Coffee. What stands out is that both meta-methods outperform their constituting methods. Once again, the inclusion of additional match information seems to be beneficial. This was shown before also for additional pairwise alignments in terms of combining global, local and overlap alignments. This also underlines the fact that none of the tools seems to be perfect for all scenarios covered by the BAliBASE reference sets. The other segment-based method DIALIGN-TX performs not as good as our method. This suggests that our segment-match refinement approach coupled with a consistency-based progressive alignment is more accurate than the selection of a consistent subset out of all *unrefined* segment matches. The only other graph-based aligner POA is also significantly worse than our own alignment method. The fastest method among all tools is Kalign. Despite the method's impressive speed, Kalign still outperforms frequently used methods such as Clustal W. It is, however, on some of the reference sets significantly worse than the best method.

Aligner	≤ 10% (240)	10% – 20% (620)	20% – 30% (499)	30% – 40% (114)	40% – 100% (209)	CPU Time (s)
AMAP	14.74**	45.71**	78.69**	90.41**	95.22**	53994
Clustal W	20.30**	49.21**	76.73**	89.55**	95.13	6835
DIALIGN-TX	20.40**	49.83**	77.93**	89.11**	96.82	35305
Kalign	25.04**	54.02**	78.81**	89.16**	96.58	106
MAFFT	33.96	64.00	85.46*	93.63	96.75	5881
MUSCLE	26.75**	57.62**	82.65**	92.04	95.91	3047
Opal	30.06**	60.23**	83.42**	91.85	95.45	37490
POA	11.27**	35.48**	63.10**	80.85**	95.38**	4422
ProbCons	32.78	63.21*	85.56	93.30	96.26**	115035
SeqAn::T-Coffee	31.02*	61.42**	83.40**	91.64*	96.72	51007
T-Coffee	33.34	62.21**	84.05**	91.91*	96.28**	124697
M-Coffee [mp]	34.74	64.62	85.80	93.52	96.73	17428
M-Coffee [mps]	34.29	64.30	85.35	92.90	96.42	20014
M-Coffee [mpst]	34.15	64.35	85.01*	92.64	96.41*	21366
SeqAn::M-Coffee [mp]	34.29	**65.53**	**86.48**	93.67	96.82	2754
SeqAn::M-Coffee [mps]	34.66	65.38	86.00	**93.71**	**96.97**	4664
SeqAn::M-Coffee [mpst]	**35.67**	65.33	86.32	93.66	96.68	5540

Table 8.2: PREFAB 4.0: Average Q scores for all PREFAB alignments. Alignments have been subdivided according to sequence identity ranges. The number of multiple alignment files in each subset is given in parentheses. All scores have been multiplied by 100 and the best program is shown in bold face for each column. The wilcoxon rank test was used to assess significant differences from the best method, indicated by * ($p < 0.05$) or ** ($p < 0.01$). The submethods for all meta-methods (M-Coffee, SeqAn::M-Coffee) are given in brackets with m:MAFFT, p:ProbCons, s:SeqAn::T-Coffee and t:T-Coffee. For the meta-methods we excluded the running time of the submethods to highlight the overhead induced by the meta-method.

8.3.2 PREFAB

For the PREFAB 4.0 (Edgar, 2004b) benchmark data set we clustered all reference alignments according to their sequence identity. The PREFAB benchmark also provides a scoring program that calculates a Q score for each alignment. The Q score is according to the authors of the benchmark equivalent to the BAliBASE SP score. As before, all methods were run with default parameters on all data sets and any file conversion was performed by means of the EMBOSS library (Rice et al., 2000). The results on PREFAB are summarized in Table 8.2. The PREFAB results confirm the results obtained on BAliBASE. MAFFT, Opal, Probcons, SeqAn::T-Coffee and T-Coffee seem to be on average the best stand-alone alignment tools. Kalign is once again the fastest method. The PREFAB benchmark highlights that Kalign performs the best on closely related sequences and is less suitable for divergent sequences. The difference between the meta-alignment methods and the single aligners is more pronounced on PREFAB than on BAliBASE. The best meta-alignment method is usually significantly better than most of the single alignment methods. SeqAn::M-Coffee once again outperforms the original M-Coffee in terms of alignment quality. At the same time, SeqAn::M-Coffee is several orders of magnitudes faster than M-Coffee.

8.3.3 DNA sequence alignment

By means of using an alignment graph of sequence segments, SeqAn::T-Coffee is suitable to align genomic nucleotide sequences. To test that purpose, we evaluated all available packages on a set of 6 adenoviruses of length 35KB obtained from the NCBI server (Accession: NC_001460, NC_004001, NC_001405, NC_002067, NC_003266, and NC_001454). Since no reference alignment is available for these long DNA sequences, we merely evaluated the ability of the programs to maximize the level of identity within the final multiple sequence alignment. Our quality measure is the level of sequence identity in each column. In Table 8.3 we report the number of columns with at least 6, 5, 4, and 3 identical characters together with the running times of the programs and the average identity. We first tried all the programs using the same default command line options as for the amino acid alignments, possibly turning on some kind of DNA switch. Using this setting all programs reported an allocation error, except Kalign, MAFFT and SeqAn::T-Coffee.

Aligner	= 6	≥ 5	≥ 4	≥ 3	Avg. identity	CPU Time (s)
Kalign	12133	17670	24540	32271	62%	190
SeqAn::T-Coffee	13057	18936	25569	32419	64%	1098
MAFFT	12594	17990	24458	31832	62%	720
DIALIGN-TX*	11993	16897	22956	30573	59%	3153
MUSCLE*	50	817	5257	21849	25%	1176
SeqAn::T-Coffee*	**13419**	**20108**	**26515**	**33143**	**65%**	336

Table 8.3: Running time and alignment quality of an alignment of 6 adenoviruses of average length 35027 nucleotides. The number of columns with at least 6, 5, 4, and 3 identical characters are reported together with the average identity. Methods marked with a * have been run with command line options that either improve the speed of the method or reduce the memory consumption.

Among these tools Kalign is again by far the fastest method whereas our approach delivers the best results. The results of Kalign, MAFFT and SeqAn::T-Coffee are summarized in Table 8.3. All other tools reporting an allocation error have been omitted. We then tried to adapt the other programs to this kind of alignment task using various command line options. In cases where we succeeded, we included the results of the best settings in Table 8.3 and added a * to the methods. For SeqAn::T-Coffee, we also included a second method marked with a * that does not use local and global alignments. This method's set of input matches consists of BLAST matches and matches retrieved from pairwise comparisons using the longest common subsequence algorithm. We included that method to highlight the ability of SeqAn::T-Coffee to use external and / or internal segment-match generation algorithms.

Since most programs reported an allocation error on the set of adenovirus genomes, we also analyzed a smaller set of closely related virus serotypes causing dengue fever of length 10KB (Accession numbers: NC_001477, NC_001474, NC_001475, and NC_002640). All programs managed to align this set of sequences, except Opal and T-Coffee. The latter method, T-Coffee, managed to align this set using the quick-align command line option. Using the same notation and analysis as in Table 8.3 we report the results of all aligners on this smaller set in Table 8.4. Once again, Kalign outperformed all other methods in terms of speed at an acceptable level of quality. Probcons and AMAP have been designed to align protein sequences

Aligner	= 4	≥ 3	Avg. identity	CPU Time (s)
AMAP	517	768	6%	217
Clustal W	5454	7964	69%	101
DIALIGN-TX	5385	7947	68%	229
Kalign	5445	7956	69%	8
MAFFT	5454	7954	69%	80
MUSCLE	5470	7994	69%	350
POA	5584	8135	70%	41
Probcons	3009	5806	51%	1595
SeqAn::T-Coffee	5566	8103	69%	50
DIALIGN-TX*	5391	7949	68%	204
MUSCLE*	5481	8000	69%	54
SeqAn::T-Coffee*	**5881**	**8495**	**72**%	46
T-Coffee*	5497	8013	69%	48

Table 8.4: Running time and alignment quality of an alignment of four virus serotypes causing dengue fever of average length 10703 nucleotides. The number of columns with at least 4 and 3 identical characters are reported together with the average identity. Methods marked with a * have been run with command line options that either improve the speed of the method or reduce the memory consumption.

and hence, they cannot be recommended for alignments of DNA sequences. All other tools deliver almost equally good results with SeqAn::T-Coffee having a minor advantage in terms of quality over the remaining tools.

8.4 Command Line

The command line options of `./seqan_tcoffee` are given below.

```
Usage: seqan_tcoffee -s <FASTA sequence file> [Options]

  -h,  --help                              displays this help message
  -V,  --version                           print version information

Main Options:
  -s,  --seq <FASTA Sequence File>         file with sequences
  -a,  --alphabet [protein | dna | rna]    sequence alphabet (default protein)
  -o,  --outfile <Filename>                output filename (default out.fasta)
  -f,  --format [fasta | msf]              output format (default fasta)

Segment Match Generation Options:
  -m,  --method                            list of match generation methods
                                           global = Global alignments
                                           local = Local alignments
                                           overlap = Overlap alignments
                                           lcs = Longest common subsequence
                                           Default: global,local
                                           /*No spaces in-between.*/
  -bl, --blast <File1>,<File2>,...         list of BLAST match files
  -mu, --mummer <File1>,<File2>,...        list of MUMmer match files
  -al, --aln <File1>,<File2>,...           list of FASTA align files
  -li, --lib <File1>,<File2>,...           list of T-Coffee libraries

Scoring Options:
  -g,  --gop <Int>                         gap open penalty (default -11)
  -e,  --gex <Int>                         gap extension penalty (default -1)
  -ma, --matrix <Matrix file>              score matrix (default Blosum62)
  -ms, --msc <Int>                         match score (default 5)
  -mm, --mmsc <Int>                        mismatch penalty (default -4)

Guide Tree Options:
  -u,  --usetree <Newick guide tree>       tree filename
  -b,  --build [nj, min, max, avg, wavg]   tree building method (default nj)
                                           nj = Neighbor-joining
                                           min = UPGMA single linkage
                                           max = UPGMA complete linkage
                                           avg = UPGMA average linkage
                                           wavg = UPGMA weighted average linkage
                                           /*Neighbor-joining creates an
                                             unrooted tree. We root that tree
                                             at the last joined pair.*/

Alignment Evaluation Options:
  -i,  --infile <FASTA alignment file>     alignment file
```

The input of the algorithm is a multiple sequence file in FASTA format. The sequence alphabet can be DNA, RNA or protein. The initial segment matches can be

computed using pairwise global, local or overlap alignments. For DNA sequences the longest common subsequence method is also a suitable method. Alternatively, one can read segment matches from external sources such as BLAST or MUMmer match files, other alignment files or a T-Coffee library. If internal alignment algorithms are selected one can optionally set scoring parameters such as the scoring matrix or gap penalties on the command line. Similarly, the guide tree reconstruction method can be set by means of a command line argument. The computed multiple sequence alignment can be written to a file in FASTA or MSF format.

Sequence Consensus

In Chapter 7 we described two algorithms for computing multi-read alignments, SeqAn-ReAlign and SeqAn-Graph. Both methods are part of our sequence consensus command line tool `./seqcons` described and evaluated in this chapter. The former algorithm, SeqAn-ReAlign, was described in Section 7.2. It is a reengineered and adapted version of the original ReAligner algorithm proposed in 1997 (Anson and Myers, 1997). The latter algorithm, SeqAn-Graph, uses a modified pipeline of the graph-based multiple sequence alignment algorithm. This algorithm was described in Section 7.3. The two main applications for multi-read alignments are reference-guided genome assembly and de novo sequence assembly. In Section 9.1 and Section 9.2, we grouped the evaluation and analysis of the `./seqcons` program according to these two applications.

9.1 Multi-Read Alignment in De Novo Assembly

Classical assemblers follow a three phase protocol: overlap, layout and consensus phase. The output of the overlap and the subsequent layout module is a set of reads with their approximate layout positions. Using this positioned read set a consensus method computes a multi-read alignment and the final consensus sequence. The overlap, layout and consensus module are usually part of a monolithic fragment assembler. This lack of modularity (or a possible lack of documentation) made it very difficult for us to compare our method with other state-of-the-art algorithms.

Setting				0.5% Error Rate		1% Error Rate		2% Error Rate		4% Error Rate	
Source Length	Read Length	Coverage	Tool	Errors	Time	Errors	Time	Errors	Time	Errors	Time
50,000	800	10x	AMOS-Cons	0	0.20s	0	0.28s	2	0.42s	8	21.34s
			CA-Cons	1	1.76s	0	1.81s	5	2.19s	>20	5.80s
			SeqAn-ReAlign	0	5.42s	0	7.78s	1	21.26s	2	34.73s
			SeqAn-Graph	0	2.83s	1	3.32s	1	4.86s	3	8.46s
50,000	200	20x	AMOS-Cons	0	0.34s	0	0.48s	0	0.98s	0	17.04s
			CA-Cons	0	3.74s	0	4.44s	3	6.90s	>20	10.10s
			SeqAn-ReAlign	0	8.28s	0	10.52s	0	14.82s	0	29.38s
			SeqAn-Graph	0	18.05s	0	23.80s	0	34.03s	1	50.79s
50,000	35	30x	AMOS-Cons	0	1.21s	1	1.41s	0	3.15s	10	7.18s
			CA-Cons	0	7.91s	4	15.23s	-	-	-	-
			SeqAn-ReAlign	0	12.80s	0	24.94s	0	41.18s	0	52.60s
			SeqAn-Graph	0	464.25s	0	499.33s	0	558.26s	0	649.36s
100,000	800	10x	AMOS-Cons	0	0.44s	1	0.62s	3	0.82s	>20	38.81s
			CA-Cons	1	3.46s	1	3.66s	7	4.38s	>20	10.78s
			SeqAn-ReAlign	0	10.73s	1	15.98s	0	24.33s	11	55.23s
			SeqAn-Graph	0	5.76s	2	7.39s	0	10.78s	11	19.37s
100,000	200	20x	AMOS-Cons	0	0.86s	0	1.16s	0	2.13s	1	30.44s
			CA-Cons	0	9.10s	0	12.46s	10	14.33s	>20	24.45s
			SeqAn-ReAlign	0	17.50s	0	30.31s	0	33.23s	1	64.18s
			SeqAn-Graph	0	54.18s	0	68.40s	0	96.80s	2	153.28s
100,000	35	30x	AMOS-Cons	0	2.42s	0	3.12s	4	5.99s	8	16.10s
			CA-Cons	0	22.43s	7	48.06s	-	-	-	-
			SeqAn-ReAlign	0	58.32s	0	85.98s	0	137.02s	0	200.12s
			SeqAn-Graph	0	1962.36s	0	2043.05s	0	2284.96s	0	2562.45s

Table 9.1: AMOS-Cons is the consensus module of the AMOS library, CA-Cons is the consensus module of the Celera Assembler and SeqAn-ReAlign and SeqAn-Graph are the two proposed consensus methods. The number of consensus errors for all positions with coverage > 2 and the computation time are reported for each method. "-" values indicate that the consensus method did not produce a consensus.

With the help of the people from the Venter Institute and the AMOS consortium, we succeeded to run the consensus module of the Minimus assembler (Sommer et al., 2007) from the AMOS consortium (http://amos.sourceforge.net) and the consensus module of the Celera Assembler (Myers et al., 2000) as a stand-alone application. However, we did not succeed for the Atlas (Havlak et al., 2004), PCAP (Huang et al., 2003) and Phusion assembler (Mullikin and Ning, 2003). Since no multi-read alignment benchmarks are available yet, we used simulated data to compare all algorithms. We simulated a source sequence and randomly sampled reads from that source sequence using different error rate, read length and coverage assumptions. Since the true source sequence is known in advance the number of consensus errors can be counted and compared among the different tools. The different settings and all results are summarized in Table 9.1.

Since the simulated global structure of the multi-read alignment is correct there is almost no difference in the results of both SeqAn methods. For error rates below or equal to 2%, both methods make hardly any mistake. Additionally, the few discrepancies that did occur where in low coverage regions where a number of nucleotides appeared equally often. Then by chance, both methods either picked the true or the false nucleotide.The ReAlign method, however, scales significantly better than the graph-based multiple sequence alignment method. The ReAlign method's runtime approximately doubles if the source sequence length is doubled and it also scales well to high coverage scenarios. Due to the pairwise overlap computations, the graph-based multiple sequence alignment method is slowed down heavily by an increasing coverage. Hence, for accurate layout positions the ReAlign method should be preferred. The AMOS consensus program performs excellent on reads of length 200. For other read lengths and low error rates it also performs quite good. For high error rates, however, both SeqAn methods outperform the AMOS program. The quality of the consensus computed by the Celera Assembler heavily depends on the read length. For long reads, it can keep up with the AMOS and SeqAn consensus programs. For shorter reads, however, the quality of the Celera Assembler consensus tool degrades rapidly, in particular for high error rates. In some very difficult cases, the consensus module fails altogether.

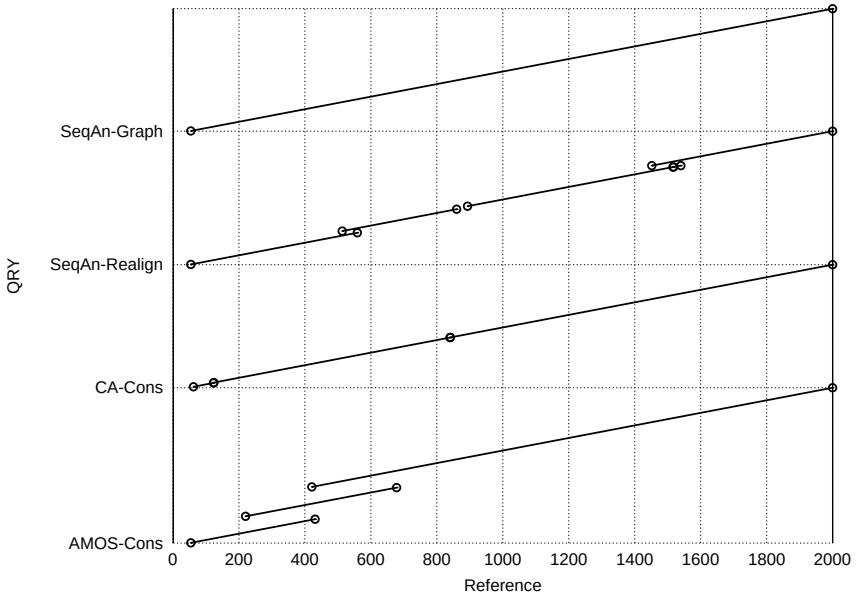
Setting					Normal distribution N(μ,σ)							
Insert Length	Read Length	Coverage	Error Rate	Tool	N(2000, 50)		N(2000, 100)		N(2000, 200)		N(2000, 400)	
					Errors	Time	Errors	Time	Errors	Time	Errors	Time
2,000	800	10x	2%	AMOS-Cons	0	1.02s	0	1.10s	>20	3.73s	>20	7.68s
				CA-Cons	1	0.31s	0	0.67s	>20	1.12s	-	-
				SeqAn-ReAlign	0	16.44s	0	16.53s	4	33.02s	>20	22.83s
				SeqAn-Graph	0	4.68s	0	4.76s	1	4.69s	1	4.44s
2,000	200	20x	2%	AMOS-Cons	>20	1.42s	>20	4.46s	>20	12.91s	>20	18.90s
				CA-Cons	6	0.98s	-	-	-	-	-	-
				SeqAn-ReAlign	>20	11.18s	>20	12.56s	>20	10.07s	>20	9.93s
				SeqAn-Graph	0	7.49s	0	8.12s	0	7.71s	>20	7.47s
2,000	35	30x	2%	AMOS-Cons	>20	5.46s	>20	9.09s	>20	17.82s	>20	13.54s
				CA-Cons	-	-	-	-	-	-	-	-
				SeqAn-ReAlign	>20	3.46s	>20	5.67s	>20	5.08s	>20	6.54s
				SeqAn-Graph	0	14.89s	0	14.42s	>20	13.46s	>20	13.04s

Table 9.2: Insert Sequencing: Given a set of mapped reads and a set of unmapped mate pairs we can approximate the positions of the unmapped mate pairs from the fragment library size. Given a Normal distribution N(μ,σ) describing the mean library size and its deviation, we simulated reads under the above settings. We report the number of errors in the consensus sequence with respect to the true insert sequence for all positions with coverage > 2. AMOS-Cons is the consensus module of the AMOS library, CA-Cons is the consensus module of the Celera Assembler and SeqAn-ReAlign and SeqAn-Graph are the two proposed consensus algorithms of the SeqAn library. "-" values indicate that the consensus module did not produce a consensus.

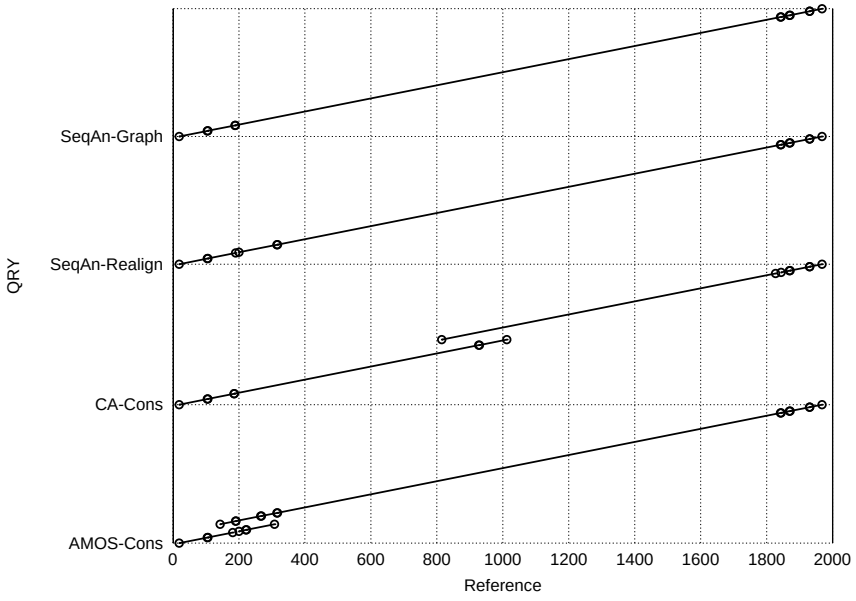
9.2 Multi-Read Alignment in Reference-Guided Sequence Assembly

For a reference-guided sequence assembly, we were particularly interested in the computation of a multi-read alignment given inaccurate layout positions derived from mate-pair information, such as the insert sequencing scenario described before (see Figure 7.2 on page 91). To simulate such insert sequencing data we assumed that all mate-pairs were sequenced from a fragment whose length was sampled from a Normal distribution N(μ,σ). Hence, the greater the σ of the Normal distribution is, the more imprecise are the final layout positions of the reads. In Table 9.2 we summarized the results for different parameters of such a Normal distribution. We report once again the number of errors in the consensus sequence with respect to the simulated insert sequence. The results strongly support our initial assumption that the graph-based multiple sequence alignment approach is more robust than classical consensus methods with respect to imprecise layout positions. This assumption could be further confirmed by aligning the consensus sequence of each tool with the simulated insert sequence using MUMmer (Kurtz et al., 2004) and the NUCmer program (Delcher et al., 2002). Two example alignments are shown in Figure 9.1. Our results indicate that the graph-based consistency extension might be a valid approach to resolve unmapped mate-pair data and retrieve the correct insert sequence. Remarkable is that the read length has great influence on the quality of such a multi-read alignment. The shorter the reads are, the less the method is able to find reliable overlaps and hence, the consensus quality degrades with decreasing read lengths. All other consensus methods are according to this study not suitable for insert sequencing, except if one uses very long reads and the library deviation is small.

A mere high-quality consensus without an accurate multi-read alignment, where all the sequencing errors and DNA polymorphisms can be readily identified, is insufficient for a sound variation analysis. Several applications such as separating haplotypes, calling SNPs, or repeat resolution rely on the multi-read alignment itself. The difficulty for the algorithms is that besides sequencing errors, source sequence variation might further complicate the problem. For the *Taeniopygia guttata* (zebra finch) mitochondrion the NCBI Genome database provides four submitted haplotypes (Accessions: DQ453512 - DQ453515). Although there is a reference genome



(a) Consensus to reference alignment: Read length 200, N(2000, 50)



(b) Consensus to reference alignment: Read length 800, N(2000, 200)

Figure 9.1: Alignment of each consensus with the source insert sequence where coverage > 2. Straight lines indicate matching segments and line endpoints are circled. Errors and gaps at the beginning and at the end of the source insert sequence are due to an insufficient sampling of reads at these positions (coverage ≤ 2).

for this species (Accession: NC_007897) we took the four haplotypes to sample reads in order to test our algorithm in case of sequence variation. To quantify the amount of variation we first aligned all four haplotypes. Because of high sequence similarity, this could be easily done with MAFFT (Katoh et al., 2002). The final multiple sequence alignment of the four haplotypes revealed 104 SNP locations and one insertion. We then simulated reads of length 200 with 2% error rate from each single haplotype at three-fold coverage. All reads were combined in a final testing set. On this set, all consensus tools computed a mixture of the four haplotypes as the consensus sequence. We then inspected the multi-read alignments manually to identify potential mis-alignments. In Table 9.3 we show a clipped view of a multi-read alignment of one of these SNP locations where the AMOS consensus module mis-aligned the reads. Note that both of our consensus methods allow a simple column-based SNP identification and consensus calling, since all alleles of the SNP ended up in the same column. In Table 9.4 we show a second example where all consensus methods computed a slightly different multi-read alignment. Nevertheless, the AMOS consensus method and our two methods computed the correct consensus sequence whereas the Celera Assembler missed one consensus letter. This example also highlights how difficult the identification of a polymorphism is when the minor allele is supported by only one haplotype. Presumably no automated method is going to be able to identify this polymorphic position in any of the given alignments. Besides a simple column-based SNP identification some variation detection algorithms use more sophisticated approaches. For instance, the Celera Assembler itself calls the consensus sequence using a window approach where the putative haplotype is the one that is confirmed by the largest number of ungapped reads (Denisov et al., 2008). This approach may have problems if the read error rate is very high or if some alleles are only supported by very few reads as shown in the example in Table 9.4.

Alignment of Haplotypes
DQ453512 ···gggtAattgtaact···
DQ453513 ···gggtAattgtaact···
DQ453514 ···gggtGattgtaact···
DQ453515 ···gggtGattgtaact···

AMOS-Cons	CA-Cons	Seq-ReAlign	Seq-Graph
ggg	ggg	ggg	ggg
gggt-a-ttgtaact	gggt-attgtaact	gggt-attgtaact	gggt-attgtaact
ggg	ggg	ggg	ggg
gggtga-ttgtaact	gggtGattgtaact	gggtGattgtaact	gggtGattgtaact
ggta-a-ttgtaact	-ggtAattgtaact	g-gtAattgtaact	g-gtAattgtaact
gggt-aattgtaact	gggtAattgtaact	gggtAattgtaact	gggtAattgtaact
gagtga-ttgtaact	gagtGattgtaact	gagtGattgtaact	gagtGattgtaact
gggtga-ttgtaa-t	gggtGattgtaa-t	gggtGattgtaa-t	gggtGattgtaa-t
gggtga-ttgtaact	gggtGattgtaact	gggtGattgtaact	gggtGattgtaact
gggt-aattgtaact	gggtAattgtaact	gggtAattgtaact	gggtAattgtaact
gggt-aattgtaact	gggtAattgtaact	gggtAattgtaact	gggtAattgtaact
gggtga-ttgtaact	gggtGattgtaact	gggtGattgtaact	gggtGattgtaact
gggt-aattgtaact	gggtAattgtaact	gggtAattgtaact	gggtAattgtaact
gggt-aattgtaact	gggtAattgtaact	gggtAattgtaact	gggtAattgtaact
ggt-aattgtaact	ggtAattgtaact	ggtAattgtaact	ggtAattgtaact
gggt-a-ttgtaact	gggtAattgtaact	gggtAattgtaact	gggtAattgtaact

Table 9.3: Multi-read alignment: A clipped view of a multi-read alignment with one SNP, highlighted in upper case letters. The alignment of the four haplotypes is shown at the top. Below we show for each consensus method the 15 reads spanning this region. The last row indicates the consensus sequence, which is incorrect for the AMOS consensus method.

Alignment of Haplotypes
DQ453512 ···aacAaccccg···
DQ453513 ···aacCaccccg···
DQ453514 ···aacAaccccg···
DQ453515 ···aacAaccccg···

AMOS-Cons	CA-Cons	Seq-ReAlign	Seq-Graph
aac-Aacc	aacaa-cc	aa-cAacc	aa-cAa-cc
aac-Aaccc-ccg	aacaa-cccccg	aa-cAacccccg	aa-cAacccc-cg
aac-Aaccc-c-g	aacaa-cccc-g	aa-cAaccc-cg	aa-cAa-ccc-cg
aac-Caccc-c-g	aacca-cccc-g	aa-cCaccc-cg	aa-cCa-ccc-cg
aac-Aaccc-c-g	aacaa-cccc-g	aa-cAaccc-cg	aa-cAa-ccc-cg
aac--cccc-c-g	aac---cccccg	aa-cC-ccc-cg	aa-c-c-ccc-cg
aac-Aaccc-c-g	aacaa-cccc-g	aa-cAaccc-cg	aa-cAa-ccc-cg
aac-Aaccc-c-g	aacaa-cccc-g	aa-cAaccc-cg	aa-cAa-ccc-cg
aac-Aaccc-c-g	aacaa-cccc-g	aa-cAaccc-cg	aa-cAa-ccc-cg
aac-Aaccc-c--	aacaa-cccc--	aa-cAaccc-c-	aa-cAa-ccc-c-
aac-Caccc-c-g	aacca-cccc-g	aa-cCaccc-cg	aa-cCa-ccc-cg
aaccAaccc-c-g	aaccaacccc-g	aaccAaccc-cg	aaccAa-ccc-cg
aac-Aacccgc-g	aa-caacccgcg	aa-cAacccgcg	aa-cAa-cccgcg
aac-Aaccc-c-g	aa-caacccc-g	aa-cAaccc-cg	aa-cAa-ccc-cg
aac-Aaccc-c-g	aa-caacccc-g	aa-cAaccc-cg	aa-cAa-ccc-cg
aac-Aaccc-c-g	aa-caacccc-g	aa-cAaccc-cg	aa-cAa-ccc-cg
aac-Aaccc-c-g	aa-caacccc-g	aa-cAaccc-cg	aa-cAa-ccc-cg
aac-Aaccc-c-g	aa-caacccc-g	aa-cAaccc-cg	aa-cAa-ccc-cg
aac-Aaccc-c-g	aa-caacccc-g	aa-cAaccc-cg	aa-cAa-ccc-cg
aac-Aaccc-c-g	aa-ca-cccc-g	aa-cAaccc-cg	aa-cAa-ccc-cg

Table 9.4: Multi-read alignment: A clipped view of a multi-read alignment with a polymorphic locus where one haplotype has a C instead of an A. This position is highlighted in upper case letters in the alignment of the four haplotypes shown at the top. Below we show for each consensus method the 19 reads spanning this region. The last row indicates the consensus sequence, which is incorrect for the Celera Assembler consensus method.

9.3 Command Line

The command line options of `./seqcons` are given below.

```
Usage: seqcons -r <FASTA file with reads> [Options]
       seqcons -a <AMOS message file> [Options]

  -h,  --help                      displays this help message
  -V,  --version                   print version information

Main Options:
  -r,  --reads <FASTA reads file>  file with reads
  -a,  --afg <AMOS afg file>       message file
  -o,  --outfile <Filename>        output filename (default align.txt)
  -f,  --format [seqan | afg]      output format (default afg)
  -m,  --method [realign | msa]    alignment method (default realign)
  -b,  --bandwidth <Int>           bandwidth (default 8)
  -n,  --noalign                   no align, only convert input

MSA Method Options:
  -ma, --matchlength <Int>         min. overlap length (default 15)
  -qu, --quality <Int>             min. overlap precent identity (default 80)
  -ov, --overlaps <Int>            min. number of overlaps per read (default 3)
  -wi, --window <Int>              window size (default 0)
                                   /*If this parameter is > 0 then all
                                     overlaps within a given window
                                     are computed.*/

ReAlign Method Options:
  -in, --include                   include contig sequence
  -rm, --rmethod [nw | gotoh]      realign method (default gotoh)
```

The input of the algorithm is a multiple sequence file in FASTA format where the sequence header contains the approximate begin and end position of each read. Alternatively, one can provide an AMOS afg message file with all the contig and aligned read information. We then use this message file to realign all the reads belonging to a given contig. Besides the usual input / output options we support the two alignment algorithms discussed before. One can either use the realignment method or the graph-based multiple sequence alignment method. For the realignment method, the only important parameter is the bandwidth that indicates how far a single read can move in a single step with respect to the initial global layout. In addition, it is possible to include the initial reference sequence to highlight the differences between the newly computed consensus and the former reference. For the graph-based method the most interesting parameter is probably the window option. Given such a window size all pairwise overlaps induced by that window are computed. Hence, this option allows reads to move largely and this option was, for instance, used in

the insert sequencing example discussed in this chapter. The overlaps option adjusts how many overlaps are on average computed per given read. Hence, this parameter provides a kind of switch to control the quality of the alignment versus the speed of computing such an alignment. The remaining parameters specify how good a found pairwise overlap alignment has to be so that it is used in the subsequent multiple sequence alignment. For most applications, the default parameters should be sufficient here.

PairAlign

PairAlign is a command line tool for pairwise sequence alignments. Pairwise alignments can be either global or local alignments and they can be computed in a banded or non-banded mode. In addition, the dynamic programming can use either linear or constant gap penalties. These different options are independent of one another and specialized algorithms for the different configurations do exist. So far, we have only implemented the most important global and local alignment algorithms, as shown in Table 10.1. For each available global and local alignment algorithm we show the corresponding algorithm tag in SeqAn. For all global alignment algorithms we support overlap and semi-global alignments by using a so-called AlignConfig object to initialize the dynamic programming matrix. The first row and column can either

	Non-banded	Banded
Global algorithms		
Constant gap costs	NeedlemanWunsch	BandedNeedlemanWunsch
Linear gap costs	Gotoh	BandedGotoh
Local algorithms		
Constant gap costs	SmithWaterman	–
Linear gap costs	SmithWaterman	–

Table 10.1: A listing of the available global and local alignment algorithms and their corresponding algorithm tag in SeqAn. The Smith-Waterman algorithm can be used for constant gap penalties. It uses, however, still three dynamic programming matrices instead of only one required for constant gap penalties.

be initialized with 0's or with normal gap costs. Similarly, the last row and column can be searched for the best value for a traceback or not. Hence, there are 2 options for each side of the dynamic programming matrix and each value can be set independently as shown below in the command line summary.

10.1 Command Line

The command line options of `./pair_align` are given below.

```
Usage: pair_align -s <FASTA sequence file> [Options]

  -h,  --help                            displays this help message
  -V,  --version                         print version information

Main Options:
  -s,  --seq <FASTA Sequence File>       file with 2 sequences
  -a,  --alphabet [protein | dna | rna]  sequence alphabet (default protein)
  -m,  --method [nw, gotoh, sw, lcs]     alignment method (default gotoh)
                                         nw = Needleman-Wunsch
                                         gotoh = Gotoh
                                         sw = Smith-Waterman
                                         lcs = Longest common subsequence
  -o,  --outfile <Filename>              output filename (default out.fasta)
  -f,  --format [fasta | msf]            output format (default fasta)

Scoring Options:
  -g,  --gop <Int>                       gap open penalty (default -11)
  -e,  --gex <Int>                       gap extension penalty (default -1)
  -ma, --matrix <Matrix file>            score matrix (default Blosum62)
  -ms, --msc <Int>                       match score (default 5)
  -mm, --mmsc <Int>                      mismatch penalty (default -4)

Banded Alignment Options:
  -lo, --low <Int>                       lower diagonal
  -hi, --high <Int>                      upper diagonal

DP Matrix Configuration Options:
  -c,  --config [ffff | ... | tttt]      alignment configuration (default ffff)
                                         tfff = First row with 0's
                                         ftff = First column with 0's
                                         fftf = Search last column for max
                                         ffft = Search last row for max
                                         All combinations are allowed.
```

The two sequences for a pairwise alignment must be present in a single FASTA file. It is important to note that the first sequence becomes the top of the dynamic programming matrix in case one wants to use the configuration options for the DP matrix. For DNA and RNA a simple match / mismatch scoring system is applied

whereas for protein sequences a scoring matrix is used. If two diagonals are specified the corresponding banded version of the algorithm is applied. The diagonals are enumerated from the negative length of the second sequence to the positive length of the first sequence. Hence, a lower diagonal of -2 and a upper diagonal of 2 would specify a banded alignment that includes the main diagonal starting in the upper left corner of the dynamic programming matrix. As for multiple alignments, the final computed alignment can be written to a file in FASTA or MSF format.

TreeRecon

TreeRecon is a command line tool providing distance-based tree reconstruction algorithms, namely the UPGMA and neighbor-joining algorithm. The input of the tool is a symmetric Phylip distance matrix. The output is a phylogenetic tree in DOT or Newick format. The DOT format is a description language for graph visualization whereas the Newick format is a very succinct textual description of a phylogenetic tree. Tools such as Graphviz (www.graphviz.org) can be used to render DOT graph files.

11.1 Command Line

The command line options of `./pair_align` are given below.

```
Usage: tree_recon -m <Phylip distance matrix> [Options]

  -h, --help                              displays this help message
  -V, --version                           print version information

Main Options:
  -m, --matrix <Phylip distance matrix>   file with distance matrix
                                          At least 3 species required.
  -b, --build [nj, min, max, avg, wavg]   tree building method (default nj)
                                          nj = Neighbor-joining
                                          min = UPGMA single linkage
                                          max = UPGMA complete linkage
                                          avg = UPGMA average linkage
                                          wavg = UPGMA weighted average linkage
                                          /*Neighbor-joining creates an
                                            unrooted tree. We root that tree
                                            at the last joined pair.*/
  -o, --outfile <Filename>                output filename (default tree.dot)
  -f, --format [dot | newick]             output format (default dot)
```

As noted previously, the UPGMA algorithm supports different options to merge

clusters, namely single linkage, complete linkage and (weighted) average linkage. For the neighbor-joining algorithm at least 3 species have to be present in the distance matrix.

Part IV

Outlook

Discussion

We presented in this thesis a new segment-based approach, called SeqAn::T-Coffee, to compute a multiple alignment of amino acid and DNA sequences. The results on protein benchmark data sets such as BAliBASE and PREFAB suggest that our method can compete with state-of-the-art alignment programs. The proposed meta-alignment algorithm is faster and more accurate than the so far best sequence-based method M-Coffee.

Our multi-read alignment method, called SeqCons, offers two different algorithms; a realignment algorithm derived from the original ReAligner program (Anson and Myers, 1997) and a novel graph-based algorithm. The realignment algorithm is able to compute consensus sequences in large resequencing projects whereas the graph-based algorithm is to our knowledge one of the first programs that can be readily used for insert sequencing.

In the following Section 12.1 we address the limitations of our algorithms to clearly delineate the use cases where they are applicable and where this is not the case. Both programs offer several opportunities for future research and we are going to present a few possible extensions in Section 12.2 and in the following Chapter 13. We conclude this chapter with a brief discussion of SeqAn in Section 12.3.

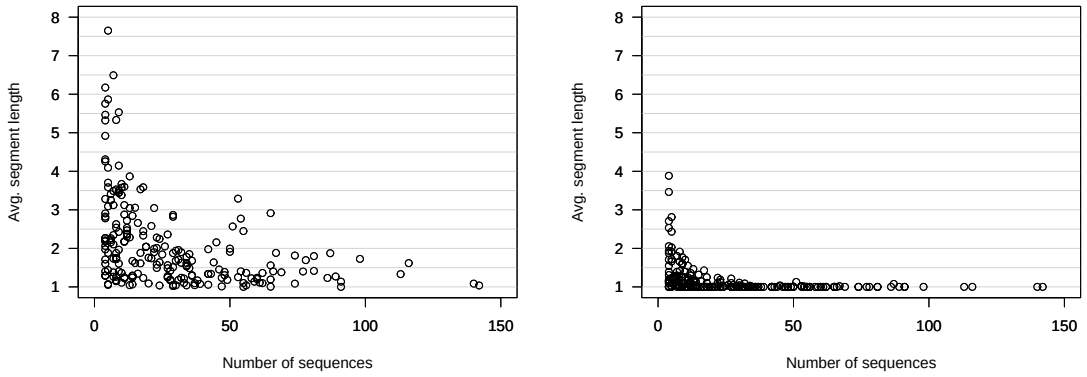
12.1 Limitations

Throughout this thesis a k-partite alignment graph was used to represent and build a multiple sequence alignment of k sequences. The graph as an additional layer of abstraction allowed us to compute alignments using sequence segments instead of single characters. Similarly, the graph could be used to align abstract entities such as

genes. With respect to sequence segments, we presented a multiple segment-match refinement algorithm that resolves overlapping and contradicting matches so that all parts of a match can be used. Through the course of this thesis, this segment-match refinement algorithm turned out to be both, a blessing and a curse. It is a blessing since it allowed us to compute very accurate alignments. In particular, in comparison to the other segment-based algorithm, called DIALIGN, our refinement approach seems to be more apt to compute accurate alignments according to the benchmark results. The refinement is, however, also a curse since the required, recursive cutting of matches shortens the length of the initial matches. Especially for divergent protein sequences this cutting has a strong detrimental influence and leads to a graph whose vertices represent on average only very few or just a single character. For meta-alignments, however, we usually encounter well-preserved regions in all subalignments. Hence, the shortening of matches is less pronounced. This can be seen in Figure 12.1 where we plotted for each alignment instance of BAliBASE the average number of sequence characters per vertex over the number of sequences in the given instance. There is a clear trend in the data that for an increasing number of sequences the average segment length is decreasing. We observed a similar trend with respect to the average sequence identity. The more related the sequences are, the longer is the average segment length.

The SeqAn::M-Coffee protocol clearly benefits from using segments. The figure underlines that probably one of the main reasons for the faster meta-alignment of SeqAn::M-Coffee compared to the original M-Coffee is the use of segments. However, one can also observe that in the stand-alone SeqAn::T-Coffee version an accurate protein alignment usually demands an alignment granularity at the level of a single character, unless one aligns less than 20 sequences. In that respect, the graph is inferior to other alignment models since it requires additional memory for storing the alignment edges, the segment information and the graph topology. Because of this graph segmentation issue, our method cannot be recommended yet for deep protein and DNA alignments of more than 200 sequences.

Likewise, a true genome alignment certainly requires the identification of non-collinear features such as transpositions, translocations, duplications or inversions. The current SeqAn alignment algorithms all require collinearity. Thus, they cannot detect these important features. However, we discuss possible adaptations of our method in Chapter 13 when we highlight some future challenges in the area of



(a) SeqAn::M-Coffee

(b) SeqAn::T-Coffee

Figure 12.1: Each circle represents one alignment instance on BAliBASE. On the left, we plotted the average number of sequence characters per vertex using SeqAn::M-Coffee. On the right, we plotted the average number of sequence characters per vertex using SeqAn::T-Coffee.

multiple sequence alignments.

12.2 Possible Extensions

The default SeqAn::T-Coffee algorithm computes all possible pairwise alignments. This simple procedure could be improved by restricting the computation of global and local alignments to a few informative sequence pairs. In addition, we have not taken into account yet the level of segmentation induced by the data. Especially in the meta-alignment method, the different levels of segmentation observed in different windows of the alignment could be a good indicator for the alignment accuracy and the sequence conservation. Similarly, one could group the sequences according to the level of fragmentation they cause if their matches are included. One could then delay the integration of the sequences causing the highest level of fragmentation using, for instance, a double progressive algorithm (Pei and Grishin, 2006, 2007).

So far, SeqAn::T-Coffee does not employ an iterative refinement loop of the initial multiple alignment. A recent publication (Wheeler and Kececioglu, 2007) suggests that well-designed iterative refinement loops can improve the quality of a MSA and a number of successful tools such as MAFFT (Katoh et al., 2005) and MUS-

CLE (Edgar, 2004b) employ refinement schemes. Hence, such a refinement procedure might be a useful addition to SeqAn::T-Coffee. Possible further extensions and applications of the proposed segment-based alignment include the comparison of different assemblies, an improved alignment of closely related, genomic sequences or an identification of conserved blocks. In addition, the alignment graph might be a suitable model for comparing genomic sequences as outlined in the next Chapter 13.

Our sequence consensus program SeqCons provides very accurate multi-read alignments and hence, the method lends itself for an improved detection of genetic variation such as SNP calling, haplotype separation or repeat resolution. The graph-based approach could also prove to be useful to bridge the gaps between contigs or to close small repeat regions. The bottleneck of the current method is the computation of the pairwise alignments among all overlapping reads using a dynamic programming solution with quadratic runtime. One possible extension is to replace this dynamic programming approach with an index based all-against-all comparison, which is significantly faster in practice. The index construction takes $\mathcal{O}(n)$ time where n is the total length of all reads. Using then an index-based filter algorithm such as SWIFT (Rasmussen et al., 2005) we can efficiently identify potential overlaps. The realignment algorithm could be further improved by parallelizing the realignment of different windows of the input multi-read alignment. Since each read is only allowed to move in a region delimited by the read's length and a user-defined bandwidth, the local realignment modifications could be computed in a distributed manner.

12.3 SeqAn

Throughout this thesis we emphasized the highly modular design of our own multiple sequence alignment programs and the SeqAn library in general. Although the amount of core data structures and algorithms available in the library is quite comprehensive by now, it is by no means complete. Solely in the field of multiple sequence alignments we have not touched yet, algorithms using terminal gap penalties, statistical algorithms for pairwise alignments, methods for sequence weighting or techniques that iteratively compute an alignment and reestimate a guide tree from the alignment for the next iteration. We do, however, provide basic pairwise alignment algorithms, alignment and tree data structures, methods to reconstruct

guide trees, basic file input and output routines and a comprehensive set of graph algorithms and data structures. In short, quite a number of *core* components are available that allow the rapid development of a new algorithm or application if and only if the programmer is willing to contribute the missing piece. Hence, SeqAn is not a plug-and-play software package but rather a get-you-started library.

The biggest strength of the SeqAn library is that the ready-to-use components are by no means limited to alignment applications. The library comprises a multitude of index, string and graph data structures and algorithmic components that range from simple string matching algorithms to sophisticated suffix array construction algorithms. We put special emphasis on a generic implementation of these core components and thus, the users are free to combine these core building blocks in their own way. Hence, the library eases the development of new applications and helps to save time and costs for creating sequence analysis software. In addition, a library helps to improve the software quality since each component error is corrected only once, while all component users benefit. Moreover, SeqAn aims at providing the most efficient algorithms currently available, which are quite often not the fastest one to implement. Hence, SeqAn might be quite often a better choice than an ad hoc implementation of a given algorithm. Finally, the user-base of the library helps to disseminate new algorithms.

Besides our attempts to promote SeqAn as an experimental platform that allows the development and testing of new applications we also recently started to create some prefabricated applications that can be used out of the box in the lab. The first published applications are a read mapper (Weese et al., 2009; Emde et al., 2010), a tool for constructing variable order Markov chains (Schulz et al., 2008) and the discussed alignment applications (Rausch et al., 2008b, 2009). In summary, we envision two kinds of SeqAn users. The biologists using one of the applications and the bioinformaticians adapting, extending and customizing the library to solve their own research problems.

Future Challenges

The defining property of an alignment is collinearity; an alignment preserves the order of sequence characters. Almost all published methods, tools and benchmarks have been developed using this specific collinearity assumption. The increasing amount of genomic sequences and the comparison of huge protein families with possibly shuffled domains demand a more generic sequence comparison model that takes into account sequence rearrangements.

Similarly, numerous methods, tools and benchmarks focused on structurally correct alignments. The benchmarks either used manually refined alignments with respect to structure or a structure alignment tool as a gold standard. Methods and tools then competed for the best scores on one of these benchmarks. However, structure prediction is only one multiple sequence alignment application. As a result, only little is known about the accuracy of alignment tools for phylogenetic studies, genomic alignments comprising mega base sequences, deep alignments comprising hundreds of protein or DNA sequences and multi-read alignments comprising thousands of small reads that overlap by only a few bases. Especially for the last problem, we showed how much a multi-read alignment differs from a classical alignment of less than a hundred protein sequences. Besides having to cope with the massive amount of data, one has to consider different alignment data structures, fast methods to compute pairwise alignments using banded and overlap dynamic programming approaches and methods that speed-up the de facto standard consistency-based progressive alignment paradigm.

In this chapter, we highlight some of the key challenges evoked by the rapidly growing sequence databases and the new high-throughput sequencing technologies. Novel techniques such as RNA-Seq and ChIP-Seq offer the potential to study gene

regulation, disease mechanisms and all kinds of cellular processes at an unprecedented accuracy. Similarly, whole genome sequencing offers the potential to elucidate phylogenetic relations and mechanisms of genome evolution. These novel applications require, however, flexible and scalable methods. For non-collinear protein alignments, genome comparison and deep alignments we point out some of the challenges and first attempts to address them in Section 13.1, Section 13.2 and Section 13.3, respectively.

13.1 Non-Collinear Protein Alignments

POA (Lee et al., 2002) and ABA (Raphael et al., 2004) were the first tools that began to questionize whether a multiple sequence alignment should be collinear. The ABA representation can handle multi-domain protein sequences where a single domain of one sequence is repeated or shuffled in another sequence. Similarly, our k-partite alignment graph can represent such information. As an example, we show in Figure 13.2 the alignment graph corresponding to the ABA graph shown in Figure 13.1. Extending the graph to handle gapped sequence segments is a non-trivial technical problem but a more serious matter is that the initial alignment graph derived from pairwise segment-matches almost never resembles the one depicted in Figure 13.2. Given such an ideal graph, one could simply identify the connected components and these components would highlight all the relevant features. In practice, however, the components are not clearly separated but rather entwined because of contradicting initial matches. Some of the very large alignment instances on BAliBASE, for instance, resulted in a fully connected graph. One could, of course, select a consistent subset of the initial input matches (without segment-match refinement) but the results of DIALIGN have shown that such an approach is at the expense of quality according to the benchmarks. For ABA, we unfortunately lack this crucial benchmarking information.

In short, there seems to be no definite answer yet on how to solve such non-collinear alignments in a satisfactory manner. The most promising approach is probably an iterative method that starts with the best multiple local alignment and then adds further local alignment components one by one. For the alignment graph, one possible objective is to exploit the pattern of alignment edges. Ideally conserved regions manifest themselves in the k-partite alignment graph as a clique (Bron and

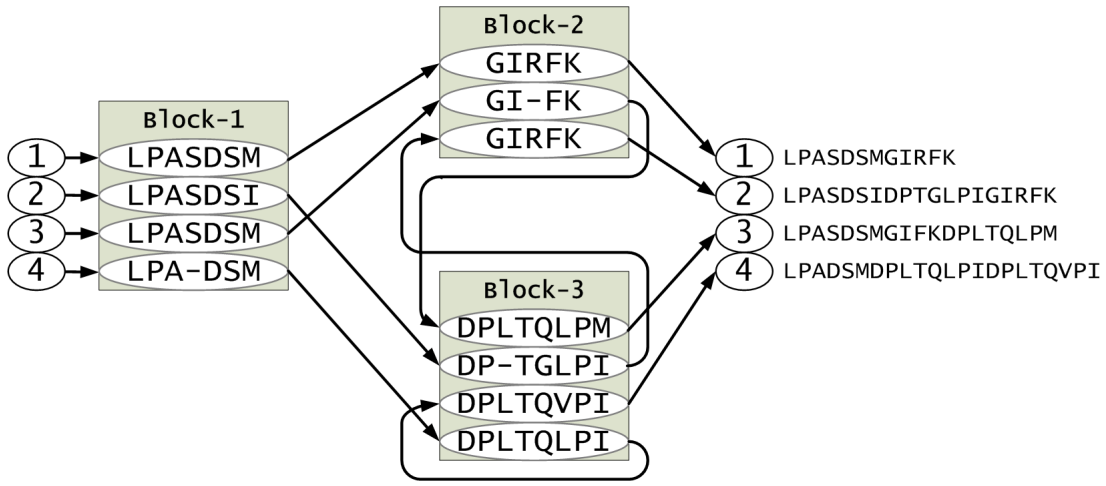


Figure 13.1: An alignment of shuffled and repeated domains of four sequences shown on the right using a De Bruijn graph (Raphael et al., 2004).

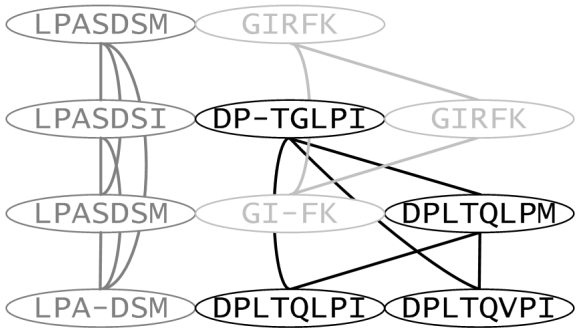


Figure 13.2: An alignment graph of shuffled and repeated domains of four sequences. The three components of the graph are highlighted in black, dark gray and light gray.

Kerbosch, 1973). Depending on the degree of conservation, some clique edges might, however, be absent. If one excludes duplicated protein domains, a non-collinear alignment resembles the well-studied assignment problem (Burkard, 2002). In our case, it is a "partial assignment" problem on a k-partite graph since a given protein domain does not have to be conserved in all of the sequences.

13.2 Genome Comparison

Similar to multi-domain protein alignments, a comparison of whole genomes demands an extension of the classical alignment operations beyond substitutions, deletions and insertions. For genomic sequences one has to take into account more complex operations such as transpositions, translocations, duplications or inver-

sions. Unfortunately, suitable comparison models, algorithms and implementations are rare for such genomic sequences.

Applications of genome comparisons are numerous. On the one hand, cross-species genome comparisons may help to reveal conserved regions, phylogenetic relations and patterns of genomic evolution. On the other hand, a comparison of closely related organisms or different members of the same species may elucidate the essential mechanisms for an organisms' phenotypic complexity. Especially the latter comparisons of closely related sequences have gained enormous practical importance with the new high-throughput sequencing technologies such as 454 Life Sciences (www.454.com), Illumina (www.illumina.com) and Applied Biosystems SOLiD Sequencing (www.appliedbiosystems.com). Using these technologies, the basic method is to align a set of sequenced reads derived from one organism to the target genome of a closely related organism. This method offers the potential to clearly delineate the extent of genomic variability (e.g. SNPs), to compare patterns of alternative splicing and to reveal the full range of genomic rearrangements through the use of mate pairs. Stretched, contracted or reversed mate-pairs can be used to identify deletions, insertions or inversions (Korbel et al., 2007). For instance, an insertion in the newly sequenced genome causes a mate-pair spanning this insertion to be contracted when mapped to the reference genome. Similarly, a deletion in the newly sequenced genome causes a mate-pair spanning this deletion to be stretched if it is mapped to the reference. An inversion affecting only one read of a mate-pair can be detected by an altered orientation of the affected read. These three cases are summarized in Figure 13.3.

In addition, the growing databases of structural variants and SNPs such as dbSNP (Smigielski et al., 2000) or the Database of Genomic Variants (Iafrate et al., 2004) can be readily used to map reads directly against the different variants as exemplarily shown in Figure 13.4 for a potential deletion or insertion. Likewise, the raw read depth can be directly used to detect copy number variations.

The 1000 Genomes Project aims at cataloging the amount of naturally occurring human variation (www.1000genomes.org). With the help of such a catalog and with all of the techniques mentioned above, researchers have the great potential to characterize human diseases on the DNA level (Pleasance et al., 2009b). These disease-specific genomic variants ranging from coding and non-coding substitutions, insertions and deletions to genomic rearrangements and copy number changes will

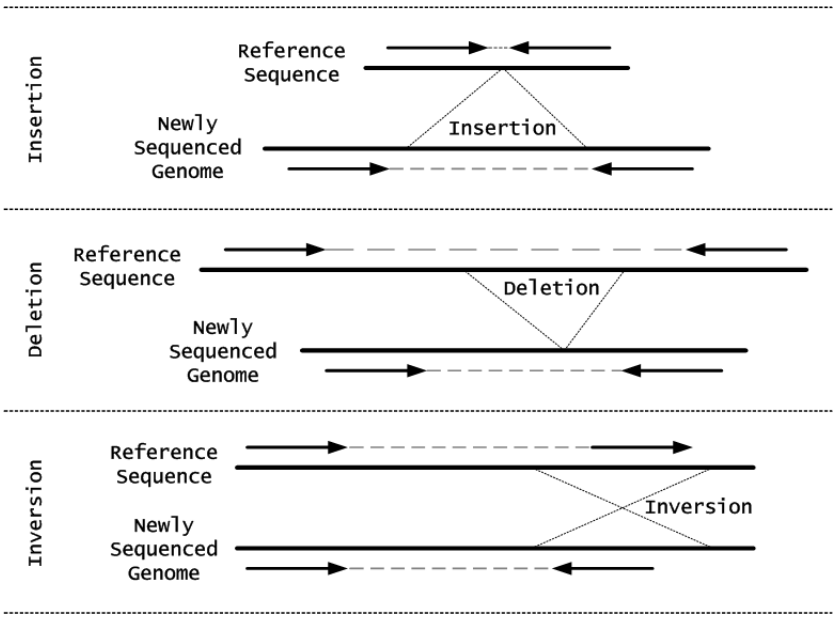


Figure 13.3: Deviations from the expected mate-pair distance indicate possible insertion or deletions. Inversions can be detected if the order of the two mate-pair reads is preserved but one of them changed its orientation.

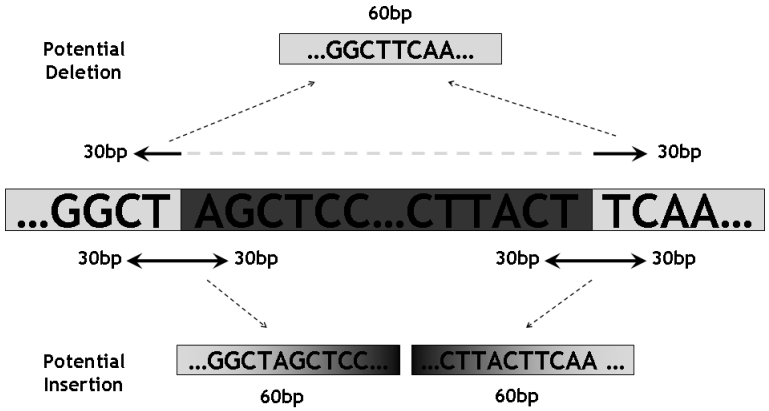


Figure 13.4: A reference sequence (middle) with a known insertion / deletion shown in dark gray. The three possible junction sequences can be directly used to map the sequenced reads in order to test if the structural variant is present or not. To avoid misaligned reads one could take only 30bp to the left and right of the breakpoint for reads of length 36.

soon be indispensable to advance clinical therapy of prevalent diseases.

Large-scale cross-species genome comparison usually try to identify conserved blocks first. Mauve, for instance, termed such regions locally collinear blocks (Darling et al., 2004). Unfortunately, the identification of such conserved blocks becomes

the more difficult the more divergent the sequences are. Another complicating factor is the support for each rearranged region. For example, the first version of Mauve could identify only rearranged region that were shared among *all* input genomes. Besides Mauve, a number of other tools for computing and visualizing genomic comparisons appeared recently (Blanchette, 2007). We listed the most important ones already in Table 2.3 on page 47 of the introduction.

In summary, there are two types of genome comparison. The first type actually compares a set of genomic sequences whereas the second one uses a reference sequence to align a set of sequenced reads. In the latter case, the final multi-read alignment is subsequently analyzed to call structural variants, single nucleotide polymorphisms or genomic rearrangements and to determine gene expression levels or alternative splicing patterns.

13.3 Deep Alignments

Today, we are starting to align thousands of protein, DNA and RNA sequences of relatively short length. Given the unprecedented pace and throughput of the new sequencing technologies it is only a matter of time until we need to take care of deep genomic alignments, that is, a comparison of dozens of genomes. It is quite clear that no standard consistency-based progressive aligner is going to be able to handle such data sets using current protocols. MUMmer and many other tools have already shown that index-based methods are certainly the most scalable methods. Hence, one of the main questions is how established and working techniques of protein aligners can be adapted and transferred to deep alignments.

A popular approach to handle deep alignments is based on the notion of a hierarchical alignment or template alignment. Instead of aligning all sequences simultaneously, this approach is two-staged. In the first stage, sequences are clustered and each cluster is multiple aligned. In the second step, the profiles of every single cluster alignment are multiple aligned. This profile alignment then serves as a template for merging all cluster alignments into a single global multiple sequence alignment.

13.4 Concluding Remarks

Throughout the past years, a number of interesting review articles have covered certain aspects of multiple sequence alignments. Among many others, we want to point out a recent review of computational methods for genomic alignments (Blanchette, 2007), an in-depth review of accurate protein sequence alignments for divergent protein sequences (Pei, 2008), an evaluation of parameter choices in progressive alignment methods (Wheeler and Kececioglu, 2007) and two program-centered multiple sequence alignment review articles (Edgar and Batzoglou, 2006; Pirovano and Heringa, 2008). The impact of next generation sequencing technologies is covered by a huge number of review articles but probably the most fascinating papers in this area are those at the edge of research. Two recent Nature papers highlight, for instance, the power of these techniques to identify genomic variants present in a cancer genome (Pleasance et al., 2009b,a).

Bibliography

M. I. Abouelhoda and E. Ohlebusch. Multiple genome alignment: Chaining algorithms revisited. In *Proc. 14th Annual Symposium on Combinatorial Pattern Matching, Lect. Notes Comput. Sci.*, pages 1–16, 2003.

E. Althaus and S. Canzar. *Bioinformatics research and development*, chapter LASA: A tool for non-heuristic alignment of multiple sequences, pages 489–498. Springer, 2008.

E. Althaus, A. Caprara, H. P. Lenhof, and K. Reinert. Multiple sequence alignment with arbitrary gap costs: Computing an optimal solution using polyhedral combinatorics. *Bioinformatics*, 18 Suppl 2:S4–S16, 2002.

E. Althaus, A. Caprara, H.-P. Lenhof, and K. Reinert. A branch-and-cut algorithm for multiple sequence alignment. *Math. Programm.*, 105:387–425, 2006.

S. F. Altschul, W. Gish, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J. Mol. Biol.*, 215(3):403–410, Oct. 1990.

S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucleic Acids Res.*, 25(17):3389–3402, 1997.

E. L. Anson and E. W. Myers. Realigner: A program for refining DNA sequence multi-alignments. In *Proc. 1st Annual International Conference on Research in Computational Molecular Biology, RECOMB*, pages 9–16, New York, NY, USA, 1997. ACM.

F. Armougom, S. Moretti, O. Poirot, S. Audic, P. Dumas, B. Schaeli, V. Keduas, and C. Notredame. Expresso: Automatic incorporation of structural information in

multiple sequence alignments using 3D-Coffee. *Nucleic Acids Res.*, 34:W604–608, Jul 2006.

S. Batzoglou, D. B. Jaffe, K. Stanley, J. Butler, S. Gnerre, E. Mauceli, B. Berger, J. P. Mesirov, and E. S. Lander. ARACHNE: A whole-genome shotgun assembler. *Genome Res.*, 12(1):177–189, 2002.

M. Blanchette. Computation and analysis of genomic multi-sequence alignments. *Annu. Rev. Genomics Hum. Genet.*, 8(1):193–213, 2007.

M. Blanchette, W. J. Kent, C. Riemer, L. Elnitski, A. F. Smit, K. M. Roskin, R. Baertsch, K. Rosenbloom, H. Clawson, E. D. Green, D. Haussler, and W. Miller. Aligning multiple genomic sequences with the threaded blockset aligner. *Genome Res.*, 14(4):708–715, 2004.

C. Bron and J. Kerbosch. Finding all cliques of an undirected graph. *Commun. ACM*, 16(9):575–577, 9 1973.

M. Brudno, C. B. Do, G. M. Cooper, M. F. Kim, E. Davydov, E. D. Green, A. Sidow, and S. Batzoglou. LAGAN and Multi-LAGAN: Efficient tools for large-scale multiple alignment of genomic DNA. *Genome Res.*, 13:721–731, Apr 2003.

J. Buhler. Efficient large-scale sequence comparison by locality-sensitive hashing. *Bioinformatics*, 17(5):419–428, 2001.

R. E. Burkard. Selected topics on assignment problems. *Discrete Appl. Math.*, 123 (1-3):257–302, 2002.

S. Burkhardt, A. Crauser, P. Ferragina, H.-P. Lenhof, E. Rivals, and M. Vingron. q-gram based database searching using a suffix array (QUASAR). In *RECOMB '99: Proceedings of the third annual international conference on Computational molecular biology*, pages 77–83, New York, NY, USA, 1999. ACM.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms.* MIT Press, Cambridge, MA, 2001.

G. E. Crooks, G. Hon, J.-M. Chandonia, and S. E. Brenner. WebLogo: A sequence logo generator. *Genome Res.*, 14(6):1188–1190, 2004.

A. C. Darling, B. Mau, F. R. Blattner, and N. T. Perna. Mauve: Multiple alignment of conserved genomic sequence with rearrangements. *Genome Res.*, 14(7):1394–1403, 2004.

M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. A model of evolutionary change in proteins. In *Atlas of Protein Structure*, pages 345–352. National Biomedical Reasearch Foundataion, 1979.

A. L. Delcher, S. Kasif, R. D. Fleischmann, J. Peterson, O. White, and S. L. Salzberg. Alignment of whole genomes. *Nucleic Acids Res.*, 27(11):2369–2376, 1999.

A. L. Delcher, A. Phillippy, J. Carlton, and S. L. Salzberg. Fast algorithms for large-scale genome alignment and comparison. *Nucleic Acids Res.*, 30(11):2478–2483, 2002.

G. Denisov, B. Walenz, A. L. Halpern, J. Miller, N. Axelrod, S. Levy, and G. Sutton. Consensus generation and variant detection by Celera Assembler. *Bioinformatics*, 24(8):1035–1040, 2008.

C. B. Do, M. Brudno, and S. Batzoglou. ProbCons: Probabilistic consistency-based multiple alignment of amino acid sequences. In D. L. McGuinness and G. Ferguson, editors, *AAAI*, pages 703–708. AAAI Press / The MIT Press, 2004.

C. B. Do, M. S. Mahabhashyam, M. Brudno, and S. Batzoglou. ProbCons: Probabilistic consistency-based multiple sequence alignment. *Genome Res.*, 15:330–340, Feb 2005.

A. Döring, D. Weese, T. Rausch, and K. Reinert. SeqAn - An efficient, generic C++ library for sequence analysis. *BMC Bioinformatics*, 9:11, Jan 2008.

H. Edelsbrunner. *Dynamic Data Structures for Orthogonal Intersection Queries.* Tech. Univ. Graz, Technical Report Rep. F59, 1980.

R. Edgar. MUSCLE: A multiple sequence alignment method with reduced time and space complexity. *BMC Bioinformatics*, 5(1):113, 2004a.

R. C. Edgar. MUSCLE: Multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res.*, 32(5):1792–1797, 2004b.

R. C. Edgar. Local homology recognition and distance measures in linear time using compressed amino acid alphabets. *Nucleic Acids Res.*, 32(1):380–385, 2004c.

R. C. Edgar and S. Batzoglou. Multiple sequence alignment. *Curr. Opin. Struct. Biol.*, 16(3):368 – 373, 2006.

R. C. Edgar and K. Sjolander. A comparison of scoring functions for protein sequence profile alignment. *Bioinformatics*, 20(8):1301–1308, 2004.

A.-K. Emde. Progressive alignment of multiple genomic sequences. Master's thesis, Freie Universität Berlin, 2007.

A.-K. Emde, M. Grunert, D. Weese, K. Reinert, and S. R. Sperling. MicroRazerS: Rapid alignment of small RNA reads. *Bioinformatics*, 26(1):123–124, 2010.

J. Felsenstein. Evolutionary trees from DNA sequences: A maximum likelihood approach. *J. Mol. Evol.*, 17:368–376, 1981.

J. Felsenstein. PHYLIP - Phylogeny inference package (Version 3.2). *Cladistics*, 5: 164–166, 1989.

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

R. D. Finn, J. Tate, J. Mistry, P. C. Coggill, S. J. Sammut, H.-R. Hotz, G. Ceric, K. Forslund, S. R. Eddy, E. L. L. Sonnhammer, and A. Bateman. The Pfam protein families database. *Nucl. Acids Res.*, 36(suppl 1):D281–288, 2008.

W. M. Fitch. Toward defining the course of evolution: Minimum change for a specific tree topology. *Systematic Zoology*, 20:406–416, 1971.

R. Giegerich, S. Kurtz, and J. Stoye. Efficient implementation of lazy suffix trees. *Software: Practice and Experience*, 33(11):1035–1049, 2003.

A. Gogol-Döring and K. Reinert. *Biological sequence analysis using the SeqAn C++ library.* CRC Press Inc., 2009.

O. Gotoh. An improved algorithm for matching biological sequences. *J. Mol. Biol.*, 162(3):705–708, Dec 1982.

O. Gotoh. Alignment of three biological sequences with an efficient traceback procedure. *J. Theor. Biol.*, 121(3):327–37, 1986.

O. Gotoh. Consistency of optimal sequence alignments. *Bull. Math. Biol.*, 52:509–525, 1990.

O. Gotoh. Multiple sequence alignment: Algorithms and applications. *Adv. Biophys.*, 36:159–206, 1999.

C. Grasso and C. Lee. Combining partial order alignment and progressive multiple sequence alignment increases alignment speed and scalability to very large alignment problems. *Bioinformatics*, 20(10):1546–1556, 2004.

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. Comput. Biol.*, 2:459–472, 1995.

A. L. Halpern, D. H. Huson, and K. Reinert. Segment match refinement and applications. In *WABI '02: Proceedings of the Second International Workshop on Algorithms in Bioinformatics*, pages 126–139, London, UK, 2002. Springer-Verlag.

P. Havlak, R. Chen, K. J. Durbin, A. Egan, Y. Ren, X.-Z. Song, G. M. Weinstock, and R. A. Gibbs. The atlas genome assembly system. *Genome Res.*, 14(4):721–732, 2004.

S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. U.S.A.*, 89(22):10915–10919, 1992.

D. G. Higgins and P. M. Sharp. CLUSTAL: A package for performing multiple sequence alignment on a microcomputer. *Gene*, 73(1):237–244, 1988.

M. Hohl, S. Kurtz, and E. Ohlebusch. Efficient multiple genome alignment. *Bioinformatics*, 18(suppl 1):S312–320, 2002.

X. Huang, J. Wang, S. Aluru, S.-P. Yang, and L. Hillier. PCAP: A whole-genome assembly program. *Genome Res.*, 13(9):2164–2170, 2003.

A. J. Iafrate, L. Feuk, M. N. Rivera, M. L. Listewnik, P. K. Donahoe, Y. Qi, S. W. Scherer, and C. Lee. Detection of large-scale variation in the human genome. *Nat. Genet.*, 36:949–951, Sep 2004.

G. Jacobson and K.-P. Vo. Heaviest increasing/common subsequence problems. In *CPM '92: Proceedings of the Third Annual Symposium on Combinatorial Pattern Matching*, pages 52–66, London, UK, 1992. Springer-Verlag.

K. Katoh, K. Misawa, K. Kuma, and T. Miyata. MAFFT: A novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nucleic Acids Res.*, 30:3059–3066, Jul 2002.

K. Katoh, K. Kuma, H. Toh, and T. Miyata. MAFFT version 5: Improvement in accuracy of multiple sequence alignment. *Nucleic Acids Res.*, 33(2):511–518, 2005.

J. D. Kececioglu. *Exact and approximation algorithms for DNA sequence reconstruction.* PhD thesis, University of Arizona, Tucson, AZ, USA, 1992.

J. D. Kececioglu. The maximum weight trace problem in multiple sequence alignment. In *Proc. 4th Annual Symposium on Combinatorial Pattern Matching, Lect. Notes Comput. Sci.*, pages 106–119, London, UK, 1993. Springer-Verlag.

J. D. Kececioglu and D. Starrett. Aligning alignments exactly. In *Proc. 8th Annual International Conference on Research in Computational Molecular Biology, RECOMB*, pages 85–96, New York, NY, USA, 2004. ACM.

J. D. Kececioglu and W. Zhang. Aligning alignments. In *Proc. 9th Annual Symposium on Combinatorial Pattern Matching, Lect. Notes Comput. Sci.*, pages 189–208. Springer Verlag, 1998.

J. D. Kececioglu, H.-P. Lenhof, K. Mehlhorn, P. Mutzel, K. Reinert, and M. Vingron. A polyhedral approach to sequence alignment problems. *Discrete Appl. Math.*, 104(1-3):143–186, 2000.

W. J. Kent. BLAT – The BLAST-like alignment tool. *Genome Res.*, 12:656–664, Apr 2002.

J. O. Korbel, A. E. Urban, J. P. Affourtit, B. Godwin, F. Grubert, J. F. Simons, P. M. Kim, D. Palejev, N. J. Carriero, L. Du, B. E. Taillon, Z. Chen, A. Tanzer, A. C. E. Saunders, J. Chi, F. Yang, N. P. Carter, M. E. Hurles, S. M. Weissman, T. T. Harkins, M. B. Gerstein, M. Egholm, and M. Snyder. Paired-end mapping

reveals extensive structural variation in the human genome. *Science*, 318(5849): 420–426, 2007.

S. Kurtz, A. Phillippy, A. Delcher, M. Smoot, M. Shumway, C. Antonescu, and S. Salzberg. Versatile and open software for comparing large genomes. *Genome Biol.*, 5(2):R12, 2004. ISSN 1465-6906.

B. Langmead, C. Trapnell, M. Pop, and S. Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology*, 10 (3):R25, 2009.

M. A. Larkin, G. Blackshields, N. P. Brown, R. Chenna, P. A. McGettigan, H. McWilliam, F. Valentin, I. M. Wallace, A. Wilm, R. Lopez, J. D. Thompson, T. J. Gibson, and D. G. Higgins. Clustal W and Clustal X version 2.0. *Bioinformatics*, 23(21):2947–2948, 2007.

T. Lassmann and E. Sonnhammer. Kalign - An accurate and fast multiple sequence alignment algorithm. *BMC Bioinformatics*, 6(1):298, 2005. ISSN 1471-2105.

C. Lee, C. Grasso, and M. F. Sharlow. Multiple sequence alignment using partial order graphs. *Bioinformatics*, 18(3):452–464, 2002.

M. Lermen and K. Reinert. The practical use of the A* algorithm for exact multiple sequence alignment. *J. Comput. Biol.*, 7:655–671, 2000.

H. Li, J. Ruan, and R. Durbin. Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome Res.*, 18:1851–1858, Nov 2008a.

M. Li, B. Ma, D. Kisman, and J. Tromp. PatternHunter II: Highly sensitive and fast homology search. *Genome Inform.*, 14:164–175, 2003.

R. Li, Y. Li, K. Kristiansen, and J. Wang. SOAP: Short oligonucleotide alignment program. *Bioinformatics*, 24(5):713–714, 2008b.

D. J. Lipman, S. F. Altschul, and J. D. Kececioglu. A tool for multiple sequence alignment. *Proc. Natl. Acad. Sci. U.S.A.*, 86:4412–4415, 1989.

B. Ma, J. Tromp, and M. Li. PatternHunter: Faster and more sensitive homology search. *Bioinformatics*, 18(3):440–445, March 2002.

B. Ma, Z. Wang, and K. Zhang. Alignment between two multiple alignments. In *Proc. 14th Annual Symposium on Combinatorial Pattern Matching, Lect. Notes Comput. Sci.*, volume 2676 of *Lect. Notes Comput. Sci.*, pages 254–265. Springer, 2003.

L. J. McGuffin, K. Bryson, and D. T. Jones. The PSIPRED protein structure prediction server. *Bioinformatics*, 16(4):404–405, 2000.

K. Mehlhorn, S. Näher, and C. Uhrig. *LEDA: A platform for combinatorial and geometric computing*. Springer, 1999.

B. Morgenstern, K. Frech, A. Dress, and T. Werner. DIALIGN: Finding local similarities by multiple sequence alignment. *Bioinformatics*, 14(3):290–294, 1998.

J. C. Mullikin and Z. Ning. The Phusion assembler. *Genome Res.*, 13(1):81–90, 2003.

M. Murata, J. S. Richardson, and J. L. Sussman. Simultaneous comparison of three protein sequences. *Proc. Natl. Acad. Sci. U.S.A.*, 82(10):3073–3077, 1985.

E. W. Myers. A fast bit-vector algorithm for approximate string matching based on dynamic programming. *Journal of the ACM*, 46(3):395–415, 1999.

E. W. Myers, G. G. Sutton, A. L. Delcher, I. M. Dew, D. P. Fasulo, M. J. Flanigan, S. A. Kravitz, C. M. Mobarry, K. Reinert, K. A. Remington, E. L. Anson, R. A. Bolanos, H.-H. Chou, C. M. Jordan, A. L. Halpern, S. Lonardi, E. M. Beasley, R. C. Brandon, L. Chen, P. J. Dunn, Z. Lai, Y. Liang, D. R. Nusskern, M. Zhan, Q. Zhang, X. Zheng, G. M. Rubin, M. D. Adams, and J. C. Venter. A whole-genome assembly of drosophila. *Science*, 287(5461):2196–2204, 2000.

G. Myers and W. Miller. Chaining multiple-alignment fragments in sub-quadratic time. In *Proc. 6th Annual ACM-SIAM Symposium*, pages 38–47, Philadelphia, PA, USA, 1995. Soc. Ind. Appl. Math.

G. Navarro and M. Raffinot. *Flexible pattern matching in strings*. Cambridge University Press, 2002.

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.

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

R. Nussinov, G. Pieczenik, J. R. Griggs, and D. J. Kleitman. Algorithms for loop matchings. *SIAM Journal on Applied Mathematics*, 35(1):68–82, 1978.

O. O'Sullivan, K. Suhre, C. Abergel, D. G. Higgins, and C. Notredame. 3DCoffee: Combining protein sequences and structures within multiple sequence alignments. *J. Mol. Biol.*, 340(2):385 – 395, 2004.

I. Ovcharenko, G. G. Loots, B. M. Giardine, M. Hou, J. Ma, R. C. Hardison, L. Stubbs, and W. Miller. Mulan: Multiple-sequence local alignment and visualization for studying function and evolution. *Genome Res.*, 15(1):184–194, 2005.

M. H. Overmars. Designing the computational geometry algorithms library CGAL. In *FCRC '96/WACG '96: Selected papers from the workshop on applied computational geormetry*, pages 53–58, London, UK, 1996. Springer.

J. Pei. Multiple protein sequence alignment. *Curr. Opin. Struct. Biol.*, 18(3):382 – 386, 2008.

J. Pei and N. V. Grishin. MUMMALS: Multiple sequence alignment improved by using hidden Markov models with local structural information. *Nucleic Acids Res.*, 34:4364–4374, 2006.

J. Pei and N. V. Grishin. PROMALS: Towards accurate multiple sequence alignments of distantly related proteins. *Bioinformatics*, 23:802–808, Apr 2007.

J. Pei, B.-H. Kim, and N. V. Grishin. PROMALS3D: A tool for multiple protein sequence and structure alignments. *Nucleic Acids Res.*, 36(7):2295–2300, 2008.

P. A. Pevzner, H. Tang, and G. Tesler. De novo repeat classification and fragment assembly. *Genome Research*, 14(9):1786–1796, 2004.

W. Pirovano and J. Heringa. Multiple sequence alignment. *Methods Mol. Biol.*, 452: 143–61, 2008.

E. D. Pleasance, R. K. Cheetham, P. J. Stephens, D. J. McBride, S. J. Humphray, C. D. Greenman, I. Varela, M. L. Lin, G. R. Ordóñez, G. R. Bignell, K. Ye, J. Alipaz, M. J. Bauer, D. Beare, A. Butler, R. J. Carter, L. Chen, A. J. Cox, S. Edkins, P. I. Kokko-Gonzales, N. A. Gormley, R. J. Grocock, C. D. Haudenschild, M. M. Hims, T. James, M. Jia, Z. Kingsbury, C. Leroy, J. Marshall, A. Menzies, L. J. Mudie, Z. Ning, T. Royce, O. B. Schulz-Trieglaff, A. Spiridou, L. A. Stebbings, L. Szajkowski, J. Teague, D. Williamson, L. Chin, M. T. Ross, P. J. Campbell, D. R. Bentley, P. A. Futreal, and M. R. Stratton. A comprehensive catalogue of somatic mutations from a human cancer genome. *Nature*, Dec 2009a.

E. D. Pleasance, P. J. Stephens, S. O'Meara, D. J. McBride, A. Meynert, D. Jones, M. L. Lin, D. Beare, K. W. Lau, C. Greenman, I. Varela, S. Nik-Zainal, H. R. Davies, G. R. Ordoñez, L. J. Mudie, C. Latimer, S. Edkins, L. Stebbings, L. Chen, M. Jia, C. Leroy, J. Marshall, A. Menzies, A. Butler, J. W. Teague, J. Mangion, Y. A. Sun, S. F. McLaughlin, H. E. Peckham, E. F. Tsung, G. L. Costa, C. C. Lee, J. D. Minna, A. Gazdar, E. Birney, M. D. Rhodes, K. J. McKernan, M. R. Stratton, P. A. Futreal, and P. J. Campbell. A small-cell lung cancer genome with complex signatures of tobacco exposure. *Nature*, Dec 2009b.

G. P. Raghava, S. Searle, P. Audley, J. Barber, and G. Barton. OXBench: A benchmark for evaluation of protein multiple sequence alignment accuracy. *BMC Bioinformatics*, 4(1):47, 2003.

B. Raphael, D. Zhi, H. Tang, and P. Pevzner. A novel method for multiple alignment of sequences with repeated and shuffled elements. *Genome Res.*, 14(11):2336–2346, 2004.

K. Rasmussen, J. Stoye, and G. Myers. Efficient q-gram filters for finding all epsilon-matches over a given length. In *RECOMB*, pages 189–203, 2005.

T. Rausch and K. Reinert. *The problem solving handbook for computational biology and bioinformatics*, chapter Practical multiple sequence alignment. (Accepted for publication) Springer, 2010.

T. Rausch, A.-K. Emde, and K. Reinert. Robust consensus computation. *BMC Bioinformatics*, 9(Suppl 10):P4, 2008a.

T. Rausch, A.-K. Emde, D. Weese, A. Döring, C. Notredame, and K. Reinert. Segment-based multiple sequence alignment. *Bioinformatics*, 24(16):i187–192, 2008b.

T. Rausch, S. Koren, G. Denisov, D. Weese, A.-K. Emde, A. Doring, and K. Reinert. A consistency-based consensus algorithm for de novo and reference-guided sequence assembly of short reads. *Bioinformatics*, 25(9):1118–1124, 2009.

K. Reinert. *A polyhedral approach to sequence alignment problems.* PhD thesis, Universität Saarbrücken, 1999.

K. Reinert, H.-P. Lenhof, P. Mutzel, K. Mehlhorn, and J. Kececioglu. A branch-and-cut algorithm for multiple sequence alignment. In *Proc. 1st Annual International Conference on Research in Computational Molecular Biology, RECOMB*, pages 241–249, 1997.

K. Reinert, J. Stoye, and T. Will. An iterative method for faster sum-of-pairs multiple sequence alignment. *Bioinformatics*, 16(9):808–814, 2000.

P. Rice, I. Longden, and A. Bleasby. EMBOSS: The european molecular biology open software suite. *Trends Genet.*, 16(6):276 – 277, 2000.

B. Rost. Review: Protein secondary structure prediction continues to rise. *J. Struct. Biol.*, 134(2-3):204 – 218, 2001. ISSN 1047-8477.

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

D. Sankoff. Simultaneous solution of the RNA folding, alignment and protosequence problems. *SIAM Journal on Applied Mathematics*, 45(5):810–825, 1985.

D. Sankoff and J. B. Kruskal. *Time warps, string edits, and macromolecules: The theory and practice of sequence comparison.* Addison-Wesley, Reading, MA, 1983.

M. Schatz, A. Phillippy, B. Shneiderman, and S. Salzberg. Hawkeye: An interactive visual analytics tool for genome assemblies. *Genome Biology*, 8(3):R34, 2007.

T. D. Schneider and R. M. Stephens. Sequence logos: A new way to display consensus sequences. *Nucleic Acids Res.*, 18:6097–6100, 1990.

M. H. Schulz, D. Weese, T. Rausch, A. Döring, K. Reinert, and M. Vingron. Fast and adaptive variable order Markov Chain construction. In *Proc. 8th International Workshop on Algorithms in Bioinformatics*, pages 306–317. Springer-Verlag, 2008.

A. S. Schwartz and L. Pachter. Multiple alignment by sequence annealing. *Bioinformatics*, 23:e24–29, Jan 2007.

V. A. Simossis and J. Heringa. PRALINE: A multiple sequence alignment toolbox that integrates homology-extended and secondary structure information. *Nucleic Acids Res.*, 33:W289, 2005.

V. A. Simossis, J. Kleinjung, and J. Heringa. Homology-extended sequence alignment. *Nucleic Acids Res.*, 33(3):816–824, 2005.

E. M. Smigielski, K. Sirotkin, M. Ward, and S. T. Sherry. dbSNP: A database of single nucleotide polymorphisms. *Nucl. Acids Res.*, 28(1):352–355, 2000.

T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *J. Mol. Biol.*, 147(1):195–197, 1981.

R. R. Sokal and C. D. Michener. A statistical method for evaluating systematic relationships. *Univ. Kansas Sci. Bull.*, 38:1409–1438, 1958.

D. Sommer, A. Delcher, S. Salzberg, and M. Pop. Minimus: A fast, lightweight genome assembler. *BMC Bioinformatics*, 8(1):64, 2007.

A. Subramanian, J. Weyer-Menkhoff, M. Kaufmann, and B. Morgenstern. DIALIGN-T: An improved algorithm for segment-based multiple sequence alignment. *BMC Bioinformatics*, 6(1):66, 2005.

A. Subramanian, M. Kaufmann, and B. Morgenstern. DIALIGN-TX: Greedy and progressive approaches for segment-based multiple sequence alignment. *Algorithms Mol. Biol.*, 3(1):6, 2008.

W. Taylor. Protein structure comparison using iterated double dynamic programming. *Protein Sci.*, 8(3):654–665, 1999.

The UniProt Consortium. The Universal Protein Resource (UniProt). *Nucl. Acids Res.*, 35(suppl 1):D193–197, 2007.

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

J. Thompson, F. Plewniak, and O. Poch. A comprehensive comparison of multiple sequence alignment programs. *Nucl. Acids Res.*, 27(13):2682–2690, 1999b.

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

J. D. Thompson, P. Koehl, R. Ripp, and O. Poch. BAliBASE 3.0: Latest developments of the multiple sequence alignment benchmark. *Proteins*, 61:127–136, Oct 2005.

T. Treangen and X. Messeguer. M-GCAT: Interactively and efficiently constructing large-scale multiple genome comparison frameworks in closely related species. *BMC Bioinformatics*, 7(1):433, 2006.

G. Venter et al. The sequence of the human genome. *Science*, 291(5507):1304–1351, 2001.

M. Vingron and P. Argos. A fast and sensitive multiple sequence alignment algorithm. *Comput. Appl. Biosci.*, 5(2):115–121, 1989.

M. Vingron and P. Argos. Motif recognition and alignment for many sequences by comparison of dot-matrices. *Journal of Molecular Biology*, 218(1):33 – 43, 1991.

I. M. Wallace, O. O'Sullivan, D. G. Higgins, and C. Notredame. M-Coffee: Combining multiple sequence alignment methods with T-Coffee. *Nucleic Acids Res.*, 34:1692–1699, 2006.

I. V. Walle, I. Lasters, and L. Wyns. SABmark - A benchmark for sequence alignment that covers the entire known fold space. *Bioinformatics*, 21(7):1267–1268, 2005.

L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *J. Comput. Biol.*, 1:337–348, 1994.

M. S. Waterman and M. Eggert. A new algorithm for best subsequence alignments with application to tRNA-rRNA comparisons. *Journal of Molecular Biology*, 197 (4):723–728, 1987.

D. Weese and M. H. Schulz. Efficient string mining under constraints via the deferred frequency index. In P. Perner, editor, *Proc. 8th Industrial Conference on Data Mining*, pages 374–388. Springer Verlag, Jul 2008.

D. Weese, A.-K. Emde, T. Rausch, A. Döring, and K. Reinert. RazerS - fast read mapping with sensitivity control. *Genome Research*, 19(9):1646–1654, 2009.

T. J. Wheeler and J. D. Kececioglu. Multiple alignment by aligning alignments. *Bioinformatics*, 23:559–568, Jul 2007.

D. R. Zerbino and E. Birney. Velvet: Algorithms for de novo short read assembly using de bruijn graphs. *Genome Research*, 18(5):821–829, 2008.

Y. Zhang and M. S. Waterman. An eulerian path approach to global multiple alignment for DNA sequences. *Journal of Computational Biology*, 10(6):803–819, 2003.

H. Zhou and Y. Zhou. SPEM: Improving multiple sequence alignment with sequence profiles and predicted secondary structures. *Bioinformatics*, 21(18):3615–3621, 2005.

M. Zuker and P. Stiegler. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucl. Acids Res.*, 9(1):133–148, 1981.

Curriculum Vitae

Der Lebenslauf ist in der Online-Version
aus Gründen des Datenschutzes nicht enthalten

Index

R

S

T

W