A Polyhedral Approach to Sequence Alignment Problems

Dissertation
zur Erlangung des Grades
des Doktors der Ingenieurswissenschaften (Dr.-Ing.)
der Technischen Fakultät
der Universität des Saarlandes

von

Knut Reinert

Saarbrücken
5. August 1999

Datum des Kolloqiums: 5. August 1999

Dekan der technischen Fakultät:
Professor Dr. Wolfgang Paul

Gutachter:
Professor Dr. Kurt Mehlhorn, MPI für Informatik, Saarbrücken
Professor Dr. John Kececioglu, University of Georgia, Athens, USA

Abstract

We study two problems in sequence alignment both from a theoretical and a practical point of view. For the first time in sequence alignment, we use tools from combinatorial optimization to develop branch-and-cut algorithms that solve these problems efficiently. The Generalized Maximum Trace formulation captures several forms of multiple sequence alignment problems in a common framework, among them is the original formulation of Maximum Trace. The Structural Maximum Trace Problem captures the comparison of RNA molecules on the basis of their primary sequence and their secondary structure. For both problems we derive a characterization in terms of graphs which we use to reformulate the problems in terms of integer linear programs. We then study the polytopes (or convex hulls of all feasible solutions) associated with the integer linear program for both problems. For each polytope we derive several classes of facet-defining inequalities and show that for some of these classes the corresponding separation problem can be solved in polynomial time. This leads to a polynomial time algorithm for pairwise sequence alignment that is not based on dynamic programming. Moreover, for multiple sequences the branch-and-cut algorithms for both sequence alignment problems are able to solve to optimality instances that are beyond the range of present dynamic programming approaches.

Zusammenfassung

Wir betrachten zwei Sequenz-Alignment-Probleme von einem theoretischen und praktischen Standpunkt aus. Dabei nutzen wir Methoden der kombinatorischen Optimierung, um Branch-and-Cut-Algorithmen zu entwickeln, die diese Probleme effizient lösen. Das sogenannte Generalized-Maximum-Trace-Problem beinhaltet verschiedene Arten von multiplen Sequenz-Alignment in einem einheitlichen Rahmen, darunter auch das ursprüngliche Maximum-Trace-Problem. Das sogenannte Structural-Maximum-Trace-Problem beschreibt den Vergleich von RNA-Molekülen, basierend auf deren Primär- und Sekundärstruktur. Wir leiten für beide Probleme eine graphentheoretische Formulierung her, welche wir dann zur Definition ganzzahliger linearer Programme benutzen. Wir untersuchen die Polytope (d.h. die konvexen Hüllen aller zulässigen Lösungen), die mit den ganzzahligen, linearen Programmen assoziiert sind. Für jedes Polytop leiten wir mehrere Klassen facettendefinierender Ungleichungen her und zeigen, daß für einige dieser Klassen das entsprechende Separationsproblem in Polynomialzeit gelöst werden kann. Dies impliziert unter anderem einen Polynomialzeitalgorithmus zum paarweisen Sequenzvergleich, welcher nicht auf dem Prinzip der dynamischen Programmierung beruht. Darüber hinaus sind die vorgestellten Branch-and-Cut-Algorithmen in der Lage, Probleminstanzen einer Größe optimal zu lösen, die mit Verfahren, welche auf dynamischer Programmierung beruhen, nicht gelöst werden können.

Acknowledgments

The work on this thesis was carried out during the years 1994-1999 at the Max-Planck-Institut für Informatik in Prof. Dr. Kurt Mehlhorn's group. The MPI always provided a very pleasant working environment. The technical facilities were excellent and the large number of guests and researchers that visited our group created a stimulating research atmosphere. I was able to get to know many of these people during my stay at MPI. This is the place to say "thank you" to a lot of them. Foremost I would like to thank my advisor Dr. Hans-Peter Lenhof. He certainly was the person who piqued my interest in Computational Biology and during the past years he had the greatest influence on the directions of my research. We often had long discussions that taught me a lot, even if they sometimes were controversial. His enthusiasm and vision for the essential things make him a distinguished researcher. Apart from him, Kurt Mehlhorn and Dr. Petra Mutzel always had time to listen and to discuss problems. Petra taught me a lot about combinatorial optimization and Kurt is simply an inexhaustible source of information about any problem you can think of. Prof. Dr. John Kececioglu formulated the original version of one of the problems I address in this thesis. We invited him several times to the MPI, and each of his stays was valuable and fruitful for me, not only from a scientific point of view. Our way of thinking about problems is very much alike. Another person I had the honor to work with was Dr. Martin Vingron who has a very likeable personality and a great style of working on problems. In addition, he was "a living library" to me at the times we met or worked together. During the first year at MPI, Dr. Phil Bradford was my office mate and we had a lot of fun together (we actually also wrote a paper). Anybody who knows Phil knows what I mean. The contributions of some of my friends and fellow PhD students cannot be counted here, but nevertheless I shall try. They were room mates, office mates, correctors, discussion partners, and they were always willing to cheer me up when the going got tough. Thank you Michael, Gunnar, Rüdiger, Ralf, Oliver, René, Hannah, Volker, Susan and all the others. Also many researchers that stayed at the MPI or were editors for some of the papers that address the problems in the thesis made valuable remarks on some topics. Among all of them I am particulary thankful to Prof. Dr. Pavel Pevzner and Prof. Dr. Naveen Garg. Last, but by far not least, I want to thank my wife, Birgit, who always supported me during this time, proofread the thesis and — being the librarian at the MPI — always provided me with the papers and books I needed.

Contents

Chapter 1

Introduction

Motivation

In 1865 Gregor Mendel started to conduct experiments about the nature of inheritance. Based on his research with pea plants, Mendel proposed that the characteristics of an offspring are determined by discrete units that are inherited from its parents. The concept of the *gene* as the basic functional unit of heredity was born. However, it would take until the middle of the 20th century to understand this concept at a molecular level. At this time it was known that the nucleus of every living eukaryotic cell contains long, linear macromolecules – called DNA – that are composed of only four basic building blocks, the nucleotides adenine (A), cytosine (C), guanine (G), and thymine (T). Experiments by Frederick Griffith in 1928 and later by Oswald Avery and coworkers in 1944 pointed out that DNA plays an important role in heredity. Nevertheless, many scientists were still reluctant to accept the simple DNA molecule as the genetic material rather than the more complex proteins. Finally, in 1952, Alfred Hershey and Martha Chase used a radioactively marked virus to conclude incontrovertably that DNA is the hereditary material. After the central role of DNA in heredity became

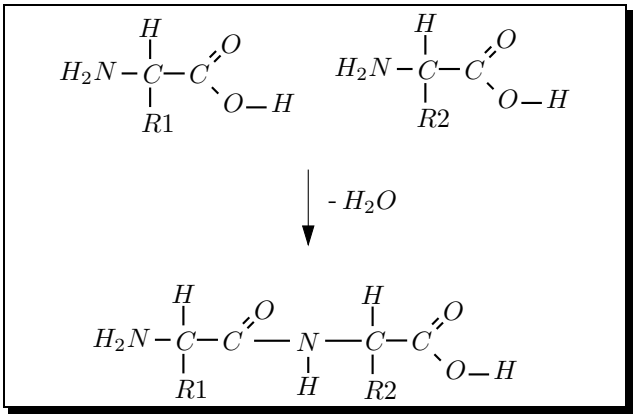

Figure 1.1: Peptide bond between two amino acid chains.

clear, many researchers tried to determine its exact structure. In 1953, one year after the Hershey-Chase experiment, Francis Crick and James Watson discovered the double-helix structure of DNA, which gave exciting new insights into how the genetic material is replicated. But still it remained unclear where the blueprints describing the formation of proteins are hidden. It would take another eight years to find out exactly how this information is encoded in the DNA. In 1961 several researchers revealed the nature of the code that is used in DNA to encode protein blueprints. The code consists of non-overlapping triples of nucleotides called *codons* that stand for different amino acids. For example, the triplet `GCT` stands for the amino acid alanine. The stretches of DNA that code for a protein – the *genes* – can basically be found in linear order in the DNA, interspersed with some non-coding regions.

Proteins are macromolecules composed of twenty different amino acids that are assembled in a complex organelle of a cell, the *ribosome*. In a complicated process, genetic information is transferred to the ribosome, which translates the triplet code into a chain of amino acids. The general formula for an amino acid is $H_2N–CHR–COOH$, where the side chain, or R group, uniquely determines the amino acid (*e.g.*, a hydrogen atom forms the side chain in the amino acid glycine). Amino acids are linked together in proteins by covalent bonds, called peptide bonds. A peptide bond is formed through a condensation reaction that involves the removal of a water molecule (see Figure 1.1 on the facing page). Proteins have a complex structure that is traditionally thought of as having three (sometimes four) levels. The linear sequence of the amino acids in a pro-

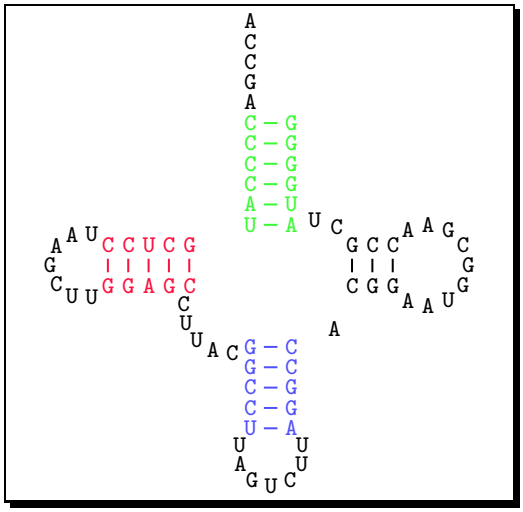

Figure 1.2: Secondary structure of a tRNA molecule.

tein is called the *primary* structure of the protein. The *secondary* structure of a protein refers to the spatial arrangements that result from interactions between amino acids that are close together in the linear sequence. Hydrogen bonds between the CO and NH groups of different residues often cause the polypeptides to bend into regular structures. Two such structures that frequently occur are the α-helix and the β-sheet. A protein also has a three-dimensional architecture, termed the *tertiary* structure, which is generated by hydrogen bonds and by electrostatic, hydrophobic, and Van-der-Waals interactions that cause the protein chain to fold back on itself (see also Figure 1.3 on the next page). The three-dimensional structure determines, to a large extent, the functionality of a protein. It is assumed that the primary sequence alone should be sufficient to determine a protein's tertiary structure and thus its functionality.

The terms primary, secondary, and tertiary structure are also used in the context of RNA sequences, which – like DNA – are chains constructed from four nucleotides. RNA

differs from DNA in that the RNA nucleotides contain the sugar deoxyribose instead of the sugar ribose, and RNA molecules contain the nucleotide uracil (U) instead of thymine. The primary structure describes – as for amino acids – the linear order of the nucleotides. Unlike DNA, however, an RNA molecule is generally a single-stranded

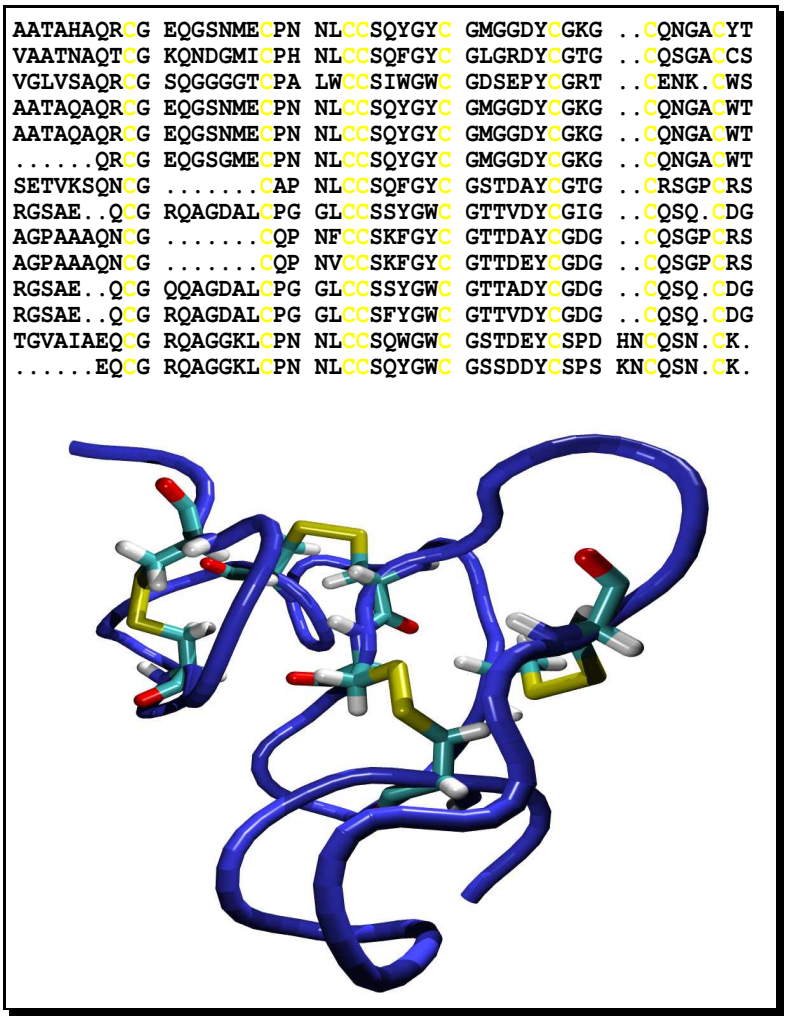

Figure 1.3: Multiple alignment of N-acetylglucosamine-binding proteins and tertiary structure of one of these proteins, the hevein.

nucleic acid molecule. Some RNA molecules such as rRNA or tRNA fold in space due to the formation of hydrogen bonds between their bases (see also Figure 1.2 on the preceding page). This base-paired structure is called the secondary structure of the RNA molecule, while the actual three-dimensional architecture of the RNA is referred to as its tertiary structure.

The discovery of the structure and function of DNA was truly a remarkable moment

in scientific history. Now, 45 years later at the end of the 20th century, we are at the brink of a new epoch when humans will soon know their own book of life and medicine will be revolutionized. *Time* magazine (Isaacson 1999) even compares the projected impact of Watson and Crick's discovery on the next century with the impact that the discovery of the electron had on the 20th century.

At the time this thesis is being written, huge efforts are being made to sequence and assemble the entire human genome, which can be represented as a string of ≈ 3.5 billion nucleotides. In parallel with the assembly efforts, the human genome (and others) is being searched for the location of genes that code for proteins or other functional units that are important for cell metabolism. Hence, huge databases are being built that contain information about thousands of proteins (for example, SwissProt (Bairoch and Apweiler 1999)), rRNA (for example, the Antwerpen database of ribosomal RNA (de Rijk *et al.* 1994)), and nucleotides (for example, the EMBL nucleotide sequence database (Stoesser *et al.* 1998)).

The ultimate goal obviously is to understand completely how the complex processes work that happen every second in an organism. If we know, for example, the function of a protein, where is this function encoded in its sequence? Can we spot genetic diseases by inspecting DNA sequences? There are many interesting questions akin to the ones above that can be answered using sequence comparison techniques.

Evolution reuses and duplicates "successful" patterns in DNA or protein sequences. It has become common knowledge that life in all organisms is based on a repertoire of building blocks that are shared by many organisms. What is the nature of these patterns or building blocks? Biological research has revealed that there are regions in sequences that are uncritical to evolutionary changes, while in other regions a single point mutation can cause the loss of functionality. For example, hemoglobin is a protein consisting of four chains of about 140 amino acids each. It binds and transports oxygen in many different organisms. While the sequence conservation within one family might be very high (for example, the hemoglobins of humans and chimpanzees are identical), there are examples of insect hemoglobin that reveal on average about 100 mutations in *each* of the four chains when compared to human hemoglobin, although the molecule serves the same purpose in both organisms. On the other hand, sometimes a single point mutation can cause serious diseases such as cystic fibrosis.

At this point (multiple) sequence alignment enters the scene. Alignments are used to exhibit the commonalities or differences between two or more sequences. An alignment of k sequences can be considered as an array of k rows. Each row contains a sequence that is interspersed with blank characters, such that "similar" parts of the sequences are in the same columns of the alignment (see Figure 1.3 on the preceding page). The only condition is that there may not be a column that contains only the blank character.

If we want to gain insight in how sequences diverge during evolution, the comparison of sequences can reveal this information. Correctly aligning a set of functionally related sequences allows the researcher to identify the regions that encode the functionality. He or she can see what changes in the sequence did not harm the function, or, by inspecting pathogenic sequences together with functional sequences, try to identify the harmful mutations.

As an example, consider Figure 1.3 on page 4. It shows a multiple alignment of subsequences of N-acetylglucosamine-binding proteins and the tertiary structure of one of these proteins, the hevein. The displayed part of the alignment shows a region of the sequences where they basically share the same tertiary structure. The yellow (or grey) columns of the alignment exhibit a part of the sequence that contains an important component for the formation of the tertiary structure. They contain eight cysteins that form four disulphid bridges that are essential for the formation of the three-dimensional structure of this protein.

Since Needleman and Wunsch first published their paper on two-sequence alignment in 1970 the diversity of alignment problems and their associated algorithms has grown tremendously. There are many applications ranging from fast database searching (Altschul *et al.* 1990) to computing consensus sequences during sequence assembly (Kececioglu and Myers 1995) to detecting subtle common patterns in a family of proteins (McClure, Vasi, and Fitch 1994). It is interesting to note that despite the variety of problem formulations most alignment problems that have been studied are solved by dynamic programming. This technique, while quite powerful, has the drawback that it generally yields an algorithm with a time and space complexity that is exponential in the number of sequences in the input. Even sophisticated implementations that use elaborate bounding techniques quickly reach their limits (Lermen and Reinert 1997; Gupta, Kececioglu, and Schaeffer 1995).

Branch-and-Cut for Sequence Alignment

In this thesis we study a new approach to solving sequence alignment problems based on an area of combinatorial optimization known as *polyhedral combinatorics* (Schrijver 1986; Nemhauser, Kan, and Todd 1989). We demonstrate how this approach, when applied to certain alignment problems, yields an algorithm for each problem that is not based on dynamic programming but is known as a *branch-and-cut algorithm* (see for example Jünger, Reinelt, and Thienel (1995b)). Branch-and-cut algorithms combine linear programming with the branch-and-bound paradigm, and are currently among the most successful algorithms for solving hard combinatorial problems. They were first successfully applied to the Linear Ordering Problem (Grötschel, Jünger, and Reinelt 1984) and to the Traveling Salesman Problem (Padberg and Rinaldi 1987). Since then they have been applied in many fields of Operations Research and the natural

sciences (*e.g.*, Christof *et al.* (1997), Jünger, Reinelt, and Rinaldi (1995a), Applegate *et al.* (1995); for an excellent bibliography see Chapter 4 in Dell'Amico, Maffioli, and Martello (1997)). In this work branch-and-cut algorithms are used for the first time in the field of sequence alignment.

As a prerequisite for designing a branch-and-cut algorithm we need to formulate the problem in terms of an integer linear program (ILP). In Figure 1.4 we illustrate some basic concepts of a branch-and-cut algorithm. Assume we are given the integer linear

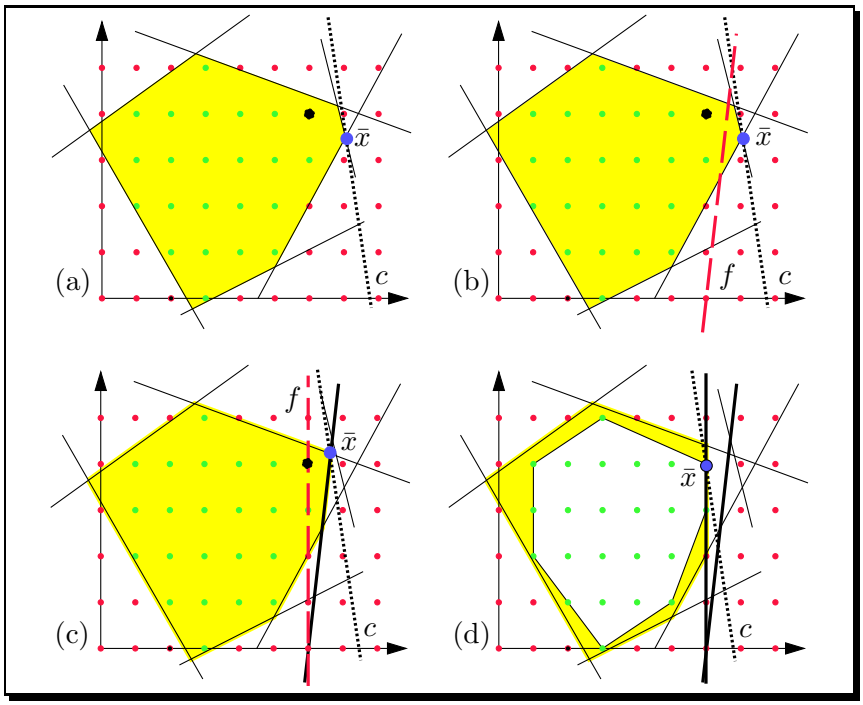

Figure 1.4: Adding cutting planes to the LP.

programming formulation of a problem. Then each solution of the problem can be represented by a high-dimensional vector, the so-called *incidence* vector. The convex hull of all feasible incidence vectors (the green points in Figure 1.4) forms the problem polytope P (the inner, white polytope in Figure 1.4 (d)). If one considers the inequalities of the ILP, they generally describe a larger polytope (the yellow polytope in Figure 1.4), although this polytope does not contain an infeasible integer point (the red points in Figure 1.4).

Unfortunately, solving an ILP is NP-hard (Garey and Johnson 1979). Hence, we relax the given integer linear program, for example by dropping the integer condition, and solve the resulting linear program. If the solution $\bar{x}$ of the linear program is integral, it corresponds to a feasible incidence vector that represents an optimal solution. Oth-

erwise we search for a valid inequality $fx \leq f_0$ that "cuts off" the solution $\bar{x}$, *i.e.*, $fy \leq f_0$ for all $y \in P$ and $f\bar{x} > f_0$; the set $\{x \mid fx = f_0\}$ is called a *cutting plane*. The search for a cutting plane is called the *separation problem*. Any cutting plane found is added to the linear program and the linear program is solved again. The generation of cutting planes is repeated until either an optimal solution is found or the search for a cutting plane fails. This procedure is illustrated in Figure 1.4 on the page before. The dotted line is the objective function c for which the large black circle symbolizes the optimal integer solution. In step (a) the optimal solution $\bar{x}$ of the linear program (symbolized by the blue point) is fractional. Hence, we add a cutting plane f (the red, dashed line) in step (b) to the LP and solve the LP again (thereby cutting off a piece of the yellow polytope). Again, the optimal solution of the LP is fractional and thus we search again for a cutting plane to cut off this infeasible solution. The cutting plane added in step (c) contains the facet of the problem polytope that contains the optimal integer solution. Hence, solving the resulting LP yields the optimal solution in the example.

If we fail to generate further cutting planes a branch step follows: We generate two subproblems by setting one fractional variable to zero in the first subproblem and to one in the second subproblem and solve these subproblems recursively. This gives rise to an enumeration tree of subproblems. In each node of the enumeration tree a branch-and-cut algorithm solves a number of linear programming relaxations and uses integer and fractional solutions computed during its execution as upper and lower bounds for the problem. The question arises as to which cutting planes one should add during the execution of the algorithm. Although there are classes of general cutting planes, it turns out that problem-specific cutting planes are much more effective. In particular, the facet-defining inequalities of the problem polytope seem to be useful, as they are not dominated by any other inequalities in an irredundant description of the polytope (see also Figure 1.4 (d) where the added cutting plane contains a facet of the problem polytope). Hence, a good knowledge of the problem polytope is crucial for the good performance of a branch-and-cut algorithm.

Graphs, traces, and multiple alignment

In this thesis we investigate two rather general alignment problems, the *Generalized Maximum Trace* (GMT) problem, and the *Structural Maximum Trace* (SMT) problem. To describe the GMT and SMT problem we first review a formulation of multiple alignment in terms of graphs introduced by Kececioglu (1991) that we extend to model the two new problems.

Let $S = \{S_1, S_2, \ldots, S_k\}$ be a set of k strings over an alphabet Σ and let $\hat{\Sigma} = \Sigma \cup \{-\}$, where "$-$" (dash) is a symbol to represent "gaps" in strings. An *alignment* of S is a set $\hat{S} = \{\hat{S}_1, \hat{S}_2, \cdots, \hat{S}_k\}$ of strings over the alphabet $\hat{\Sigma}$ that satisfies the following two

properties: (1) the strings in $\hat{S}$ all have the same length, and (2) ignoring any dashes, string $\hat{S}_i$ is identical to string S_i. An alignment in which each string $\hat{S}_i$ has length l can be interpreted as an array of k rows and l columns where row i corresponds to string $\hat{S}_i$ (see also Figure 1.3 on page 4). Two characters of distinct strings in S are said to be *aligned* under $\hat{S}$ if they are placed in the same column of the alignment array. We view the character positions of the k input strings in S as the vertex set V of a k-partite graph $G = (V, E)$ called the input *alignment graph*. The edge set E connects pairs of characters that one would like to have aligned in an alignment of the input strings. We call an edge in E an *alignment edge* and say that an alignment edge is *realized* by an alignment if the endpoints of the edge are placed in the same column of the alignment array. The subset of E realized by an alignment $\hat{S}$ is called the *trace* of $\hat{S}$. Figure 1.5

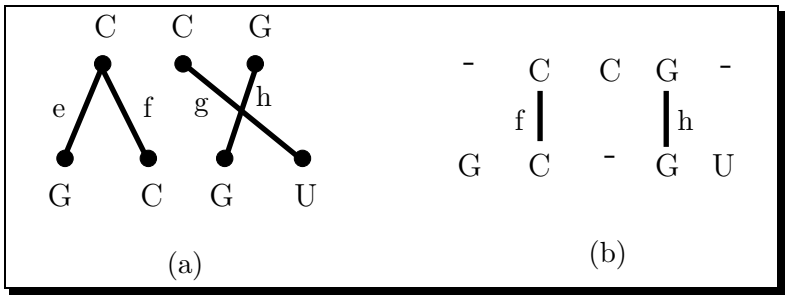

Figure 1.5: (a) An alignment graph of two sequences `CCG` and `GCGU`. Edges e, f and g, h are in conflict. (b) The trace $\{f, h\}$ is realized by the alignment shown.

shows an alignment graph of two strings containing four edges and an alignment that realizes two of the edges. Note that several alignments can have exactly the same trace, where such alignments differ only in their arrangement of unaligned regions.

The notion of a trace of two strings as illustrated in Figure 1.5 is a basic concept in sequence comparison (see, for instance, Sankoff and Kruskal (1983) pp. 10–18) which Kececioglu (1991) generalized to multiple sequence alignment with the notion of a trace of an alignment graph. The relationship between multiple alignment and multi-partite graphs was also examined by Vingron and Pevzner (1995) in the context of filtering pairwise dot-plots of a set of sequences.

The Generalized Maximum Trace Problem

In the *Maximum Trace Problem* (MT), introduced originally to model the final multiple alignment phase of DNA sequence assembly, every edge in the alignment graph has a positive weight representing the benefit of aligning the endpoints of the edge. The goal is to compute an alignment $\hat{S}$ whose trace has maximum weight. Kececioglu (1991) showed that MT is NP-complete and developed a branch-and-bound algorithm for the problem based on dynamic programming, with worst-case time complexity $O(k^2 2^k N)$

and space complexity $O(N)$, where $N = \prod_i |S_i|$. The algorithm is able to solve to optimality relatively small problem instances. In this thesis we show how to generalize the Maximum Trace Problem to accommodate different scoring schemes. In the *Generalized Maximum Trace Problem* (GMT) we allow multiple edges between two vertices in the alignment graph G and we partition the edge set E into a set D of so-called *blocks*. A block is a trace in which every edge is incident to nodes in the same pair of sequences. We regard a block $d \in D$ as realized if all the edges in d are realized.

Every block $d \in D$ has a weight w_d representing the benefit of realizing that block, and the weight of an alignment is the sum of the weights of the blocks it realizes. The goal is to compute an alignment $\hat{S}$ of maximum weight. Notice that this captures the construction of a multiple alignment out of local pairwise alignments.

Most commonly used scoring schemes are based on the similarity of single pairs of characters (for instance, Dayhoff, Schwartz, and Orcut (1979) or Henikoff and Henikoff (1992)). This corresponds to a partition of the edges into singleton sets as in Figure 1.6 and is equivalent to the original MT formulation. It is worth noting that the singleton case includes as a special case the well studied sum-of-pairs multiple alignment problem. GMT also allows more general scoring schemes based on the similarity of pairs

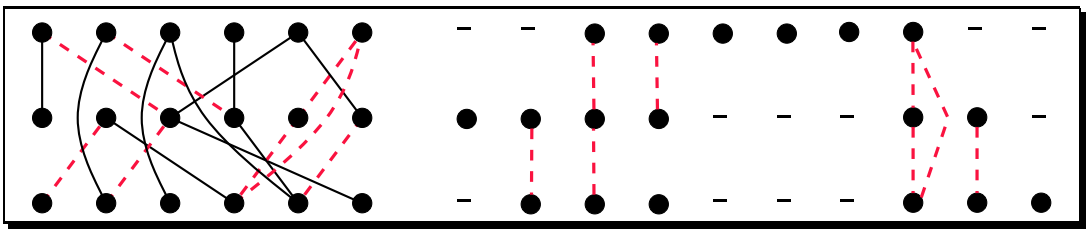

Figure 1.6: An alignment graph with 18 edges. $D = \{\{e_1\}, \{e_2\}, \ldots, \{e_{18}\}\}$ is a partition into singleton sets. The alignment on the right realizes 8 edges (and thus 8 singleton sets).

of whole segments of two sequences (see, for instance, Altschul and Erickson (1986), Morgenstern *et al.* (1998), and Wilbur and Lipman (1984)). To illustrate how this can be done, Figure 1.7 shows a partition into sets of edges that form consecutive runs of matches. Here the edges of a run form a block. Note that blocks d_5 and d_1 both contain a *different* edge that runs between the same pair of vertices. Hence, any alignment that realizes either d_1 or d_5 must match the corresponding characters.

The graph-theoretic formulation of the GMT enables us to give an ILP formulation for the GMT in which we associate with every block d in D a binary variable x_d that indicates whether a block is realized ($x_d = 1$) or not ($x_d = 0$). An integer solution is feasible if the alignment edges of the realized blocks form a trace. The goal is to find

the feasible solution that realizes a set of blocks with maximum overall weight.

As noted above, once the ILP is formulated, it is necessary to investigate the structure of the problem polytopes. Hence, we studied the GMT polytope thoroughly as a first essential step on the way to an efficient branch-and-cut algorithm. We were able to identify numerous classes of facet-defining inequalities, and for many of these classes we could devise efficient exact or heuristic separation algorithms that turn the theoretical knowledge about the polyhedra into practical routines for deriving upper bounds.

In the pairwise case we show that the *clique* inequalities together with the *trivial* inequalities form a complete description of the GMT polytope. Together with our ability to separate the clique inequalities in polynomial time, this implies the existence of a polynomial time algorithm for sequence alignment that is not based on dynamic programming. A similar observation was already made by Pevzner and Waterman (1993) who devised a primal-dual algorithm for different sequence alignment algorithms. For the multiple sequence case we give two more classes of valid inequalities (the *mixed cycle* inequalities and *ladder* inequalities) and show when they are facet-defining. We

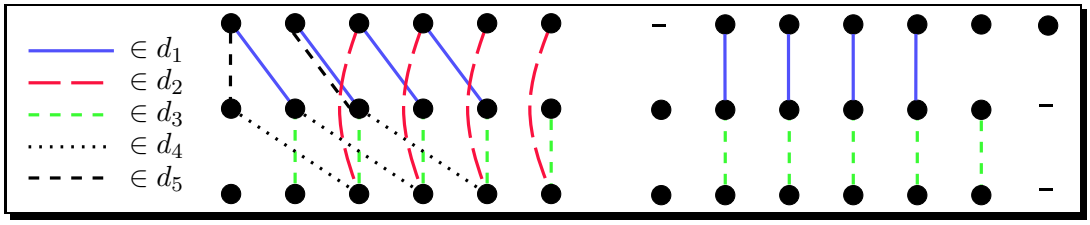

Figure 1.7: An alignment graph with 18 edges. D is a partition into five blocks d_1, d_2, d_3,d_4, and d_5. On the right, an alignment realizes 2 blocks consisting of a total of 9 edges.

describe how a data structure called a *pairgraph* can be used to represent an exponential number of clique inequalities in polynomial space and how one can use the pairgraph to devise an exact and efficient separation routine for the class of clique inequalities. For the class of *mixed cycle* inequalities we use the extended alignment graph itself to develop an efficient separation routine. Our implementation of the branch-and-cut algorithm for the GMT shows that the use of methods from combinatorial optimization in the field of sequence alignment leads to algorithms that are comparable or superior to existing algorithms that are based on dynamic programming. We can, for example, align up to 18 sequences of length ≈ 200, a problem size not tractable for dynamic programming based approaches.

The Structural Maximum Trace Problem

The second alignment problem we address is the *Structural Maximum Trace Problem* (SMT). The aim is to compute an alignment that maximizes sequence and structure

consensus simultaneously. To be more precise, the score that is optimized is a weighted sum of the sequence similarity and the structural similarity of the sequences under consideration. In this context structural similarity stands for the similarity of the *secondary structures* of the sequences, which are RNA sequences in our examples.

An RNA molecule is generally a single-stranded nucleic acid molecule that folds in space due to the formation of hydrogen bonds between its bases. Conventional sequence alignment algorithms can only account for the primary sequence and thus ignore structural aspects. Our aim is to align the sequences using the structural information given, thereby exhibiting not only sequence similarity but also structural similarity. In RNA molecules it is the secondary structure that carries the functionality and hence tends to be conserved through evolution. This was strikingly demonstrated by Levitt (1969) with the prediction of tRNA structure from a set of similar sequences. His work shows that sets of similar sequences can yield convincing evidence for how an RNA molecule folds. The computational problem of considering sequence and structure of an RNA molecule simultaneously was first addressed by Sankoff (1985) who proposed a dynamic programming algorithm that aligns a set of RNA sequences while at the same time predicting their common fold. Algorithms similar in spirit were later proposed for the problem of comparing one RNA sequence to one or more of known structure. Corpet and Michot (1994) align simultaneously a sequence with a number of other sequences using both primary and secondary structure. Their dynamic programming algorithm requires $O(n^5)$ running time and $O(n^4)$ space (n is the length of the sequences) and thus can handle only short sequences. Corpet and Michot propose an anchor-point heuristic to divide large alignment problems by fixed alignment regions into small subproblems so that the dynamic programming algorithm can then be applied. Bafna *et al.* (1995) improved the dynamic programming algorithm to a running time of $O(n^4)$, which still does not make it applicable to real-life problems. Gorodkin *et al.* (1997) iterate Sankoff's dynamic programming algorithm to find motifs among many RNA sequences. Instead of using dynamic programming the algorithm of Waterman (1989) searches for common motifs among several sequences. Eddy and Durbin (1994) describe probabilistic models for measuring the secondary structure and primary sequence consensus of RNA sequence families. They present algorithms for analyzing and comparing RNA sequences as well as database search techniques. Since the basic operation in their approach is an expensive dynamic programming procedure, their algorithms cannot analyze sequences longer than 150–200 nucleotides. Notredame *et al.* (1997) implemented a genetic algorithm for the optimization of both alignment and structure correspondence between two RNA molecules. Their procedure produces biologically good results although at the expense of considerable running time.

In the case of the SMT problem we were able to extend the GMT formulation in order to deal with structural information. That means that the input to the SMT problem can

be viewed as an alignment graph, where, for each sequence, we additionally are given a list of possible *interactions* or *base pairs* between character pairs of that sequence, *e.g.*, a list produced by some secondary structure prediction program or a list of all possible Watson-Crick base pairs (`A-U` or `C-G`). Figure 1.8 shows two sequences with these different kinds of possible interactions.

A structural alignment can realize not only an alignment edge, *i.e.*, the match of two characters of the sequences, but also an *interaction match*. A pair of interactions in two different sequences is said to be *aligned* or *matched* if the interacting characters in the two sequences are aligned. In Figure 1.8 you see two structural alignments of the sequences. In the upper alignment three pairs of interactions are matched while in the second only one pair is matched. We devised an ILP formulation for the SMT in which we associate with every alignment edge e in E a binary variable x_e that indicates whether the edge is realized ($x_e = 1$) or not ($x_e = 0$). For the same purpose we assign to each interaction match m a binary variable x_m. An integer solution is feasible if the realized alignment edges form a trace and if each character is involved in at most one realized interaction match. Each variable is assigned a weight that represents the benefit of realizing the alignment edge or the interaction match. The goal is to find a feasible solution of maximum overall weight.

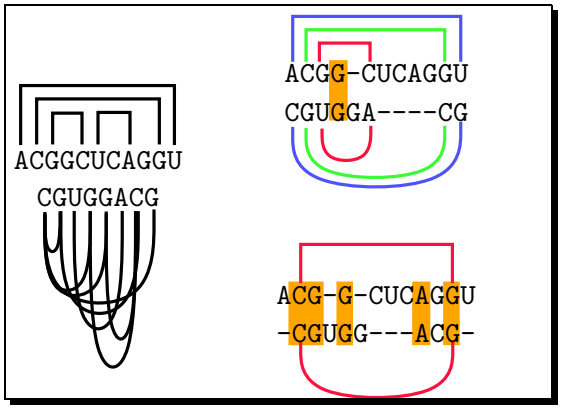

Figure 1.8: Two RNA sequences with interactions and two structural alignments of the sequences.

The investigation of the SMT polytope shows that the *trivial* and *mixed cycle* inequalities are in essence the same as for the GMT polytope. We found three new classes of valid inequalities and showed under what conditions they are facet-defining: the *extended clique* inequalities, the *interaction* inequalities, and the *odd cycle* inequalities. We implemented a branch-and-cut algorithm for structurally aligning two RNA sequences and were able to align sequences of length ≈ 1400 provably better than con-

ventional algorithms for pairwise sequence alignment. Indeed, to our knowledge, there is no other algorithm that is able to structurally align sequences of this length to optimality. Algorithms based on dynamic programming cannot analyze sequences longer than a few hundred nucleotides. Moreover, our algorithm can easily be extended to handle multiple sequences.

Summary

We view as a main contribution of our work the introduction of the polyhedral approach to the area of sequence alignment. We formulated two rather general alignment problems that include numerous interesting problem variations. The formulation of the SMT and GMT problem in terms of graphs allows one to encode restrictions on the problems very conveniently. The alignment graph can, for example, contain only edges that either stem from matches occurring in (sub)optimal pairwise alignments (Kececioglu 1993, Reinert *et al.* 1997), or that fulfill certain context dependencies (Wilbur and Lipman 1984).

With a polyhedral approach, one formulates the alignment problem to be studied as an integer linear program; once such a formulation is found, variations of the problem can often be conveniently modeled through the addition of further constraints to the basic linear program. With dynamic programming, on the other hand, accommodating variations such as considering secondary structure in sequence alignment (Bafna, Muthukrishnan, and Ravi 1995), can cause at a minimum a significant restructuring of the basic recurrences. With the polyhedral approach, much of the code developed for the basic problem can be reused for the problem variations; for example, both the Generalized Maximum Trace and Structural Maximum Trace problems are based on the same integer linear programming formulation, and separation routines for their basic formulations are reused in the code for both problems. A polyhedral approach to a problem creates many research avenues for future investigators, as each researcher is able to build on prior theoretical work and practical software by discovering new classes of facet-defining inequalities and devising new separation routines for both known and newly-discovered classes. Our first implementations of branch-and-cut algorithms for the GMT and SMT problem already are superior or comparable to dynamic programming based approaches. This indicates that our new approach to sequence alignment is worthy of further investigation. We feel that this new approach has plenty of room for future development, while traditional methods based on dynamic programming are already thoroughly studied and hard to improve.

Guide to the thesis

In Chapter 2 we give basic definitions and notations. We also give the prerequisite mathematical background by reviewing fundamental theorems from some mathematical fields as for example polyhedral theory, linear programming, integer polytopes, and

polyhedral combinatorics. This chapter is to serve as a repository for the mathematical notations we use throughout the thesis.

In Chapter 3 we introduce the notion of pairwise and multiple alignments. We give a general framework for scoring alignments, which describes the most commonly used scoring functions in a comprehensive way. Then we extend this framework such that it incorporates structural information. Although we deal exclusively with RNA sequences in our experiments, the framework is designed to deal with any structural information that is based on the interaction of pairs of residues in the sequence.

In Chapter 4 we define pairwise and multiple traces. Compared to alignments, traces are a slightly different but more elegant means of analyzing sequence similarity. We elaborate on the relationship between traces and alignments and show how variations of alignment problems can be formulated in terms of certain weighted input graphs. An alignment corresponds to a subgraph in the input graph that fulfills certain restrictions. The problem of computing an alignment with optimal score reduces to the problem of finding a subgraph of maximal weight that fulfills these restrictions.

In Chapter 5 we formally define the GMT and SMT problem and review the currently best dynamic programming based algorithms for them.

In Chapter 6 we present the polyhedral approach we have taken to solve both problems. We start by introducing the main concepts of a generic branch-and-cut algorithm in Section 6.1. Then we describe first for the GMT problem (Section 6.2) and then for the SMT problem (Section 6.3) the problem specific details. First we give the ILP formulation for both problems in Section 6.2.1 and 6.3.1. The integer linear program builds the basis of the polyhedral approach we describe. It can naturally be associated with a high dimensional polytope (the convex hulls of the incidence vectors of all feasible solutions). In practice the facet-defining inequalities of the problem polytopes yield good cutting planes during the execution of a branch-and-cut algorithm. Hence, we investigate the facial structure of the problem polytope as a next step in Sections 6.2.2 and 6.3.2. In Sections 6.2.3 and 6.3.3 we turn the theoretical knowledge about the facial structure of the problem polytope into practical routines for separating the classes of inequalities we have found in the previous section. In Section 6.2.4 and 6.3.4 we present results of our implementations. We demonstrate that the use of methods from combinatorial optimization in the field of sequence alignment leads to implementations that are comparable or even superior to existing algorithms based on dynamic programming.

Finally, we discuss our results and open problems in Chapter 7.

Chapter 2

Mathematical Preliminaries

In this chapter we give a short and succinct overview of all the mathematical tools we are going to use. The reader familiar with the notions introduced here may skip this chapter and refer to it when necessary.

2.1 Graph Theory

For an *undirected graph* $G = (V, E)$ we denote its node set by $V = V(G)$ and its edge set by $E = E(G)$. Edges of G are denoted by the set of their endnodes, *i.e.*, we write $e = \{v, w\}$ for $e \in E(G)$. We say that e *joins* the nodes v and w. Two edges with a common endnode are called *adjacent*. For $e = \{v, w\} \in E(G)$, we say that v and w are *adjacent nodes*, whereas both nodes v and w are *incident* to the edge e and vice versa. An edge e is said to be *adjacent to the edge set* F if there is an edge $f \in F$ which is adjacent to e.

We denote the number of nodes and edges in a graph with $|V|$ and $|E|$ respectively. We say that $G' = (V', E')$ is a *subgraph* of G if $V' \subseteq V$ and $E' \subseteq E$. If $V' \subseteq V$, we define $E(V') := \{\{v, w\} \in E \mid v, w \in V'\}$ and call the subgraph $G' = (V', E(V'))$ *node-induced* by V'. Similarly if $E' \subseteq E$, we define $V(E') := \{v, w \in V \mid \{v, w\} \in E'\}$ and call the subgraph $G' = (V(E'), E')$ *edge-induced* by E'. In both cases we write $G[V']$ respectively $G[E']$ as shorthand.

A *path of length* k is an alternating sequence $(v_0, \{v_0, v_1\}, v_1, \{v_1, v_2\}, \ldots, \{v_{k-1}, v_k\}, v_k)$ of edges and nodes in G such that $\{v_i, v_{i+1}\} \in E$, $0 \leq i < k$ and all edges and all nodes are distinct. A path P together with an edge $\{v_k, v_{k+1}\}$ and a node v_{k+1} for $k \geq 2$ is called a *cycle* if $v_0 = v_{k+1}$. Since a path P (or a cycle C) is determined by the set of edges in P (respectively in C), we often identify paths and cycles by their edge set.

A *bipartite graph* $G = (V, E) = (V_1 \cup V_2, E)$ is a graph whose node set V is partitioned into two subsets V_1 and V_2 such that every edge joins a vertex from V_1 with a vertex from V_2.

The *complete graph* on n vertices is denoted by K_n, the *complete bipartite graph* with $|V_1| = p$ and $|V_2| = q$ as $K_{p,q}$.

A *clique* in a graph $G = (V, E)$ is a subset of nodes $V_C \subseteq V$ such that every pair of nodes $u, v \in V_C$ is adjacent in G, or, in other words, $G[V_C]$ is a complete graph.

We call a graph *connected* if for every pair $u, v \in V$ there exists a path in G connecting u and v. A graph consisting of a single node is connected by definition. For an arbitrary graph $G = (V, E)$ the maximal connected, node-induced subgraphs, *i.e.*, the connected subgraphs $G' = (V', E(V'))$ satisfying $V' \subseteq V$ and for all $v \in V \setminus V' : E(V' \cup \{v\})) =$

$E(V')$, are called the *connected components* of G.

For a *directed graph* or shortly *digraph* $D = (V, A)$ we denote the node set by $V = V(D)$ and the *arc* set by $A = A(D)$. An arc $a \subset A \times A$ is an ordered pair of elements of A. If $a = (u, v)$ is an arc in A then a is said to be *incident from* u and *incident to* v or u is the *source* and v is the *target* of a. Two nodes $u, v \in V$ are called *adjacent* in a digraph $D = (V, A)$ if $(u, v) \in A$ or $(v, u) \in A$. A digraph is *complete* if for any two nodes $u, v \in V$, $u \neq v$ the set A contains the arcs (u, v) and (v, u). The complete digraph with n nodes is denoted by D_n. For $D = (V, A)$ and $V' \subseteq V$ we define $A(V') := \{(u, v) \in A \mid u, v \in V'\}$ and for $A' \subseteq A$ we define $V(A') := \{u, v \in V \mid (u, v) \in A'\}$.

The definition of a *(node-induced, arc-induced) subdigraph* is analogous to the one for graphs. There are obvious directed counterparts for paths and cycles in graphs. $W = (v_0, (v_0, v_1), v_1, (v_1, v_2), \ldots, (v_{k-1}, v_k), v_k)$ is a *(di)path of length* k if $v_i \neq v_j$ for $i \neq j$. A path P together with an arc (v_k, v_{k+1}) and a node v_{k+1} for $k \geq 1$ is called a *(di)cycle* if $v_0 = v_{k+1}$. Since a path P (or a cycle C) is determined by the set of arcs in P (respectively in C), we often identify paths and cycles by their arc set.

We call a digraph *strongly connected* if for every pair $u, v \in V$ there exists a dipath in D from u and v and from v to u. A graph consisting of a single node is strongly connected by definition. For an arbitrary digraph $D = (V, A)$ the maximal strongly connected, node-induced subgraphs (maximal with respect to the cardinality of the node set) are called the *strongly connected components* of D.

A *mixed graph* is a tuple $G = (V, E, A)$, where V is a set of vertices, E is a set of edges and A is a set of arcs. A *path* in a mixed graph is an alternating sequence $w = (v_0, e_0, v_1, e_1, \ldots, v_k)$ of vertices and arcs or edges such that either $e_i = \{v_i, v_{i+1}\} \in E$ or $e_i = (v_i, v_{i+1}) \in A$, for all i, $0 \leq i < k$ and all vertices and all edges in the path are distinct. A path is called a *mixed path* if it contains at least one arc in A and one edge in E. A path P together with an edge (or arc) and a vertex v is called a *mixed cycle* if v and the first vertex on the path are the same and if P together with v and the edge (or arc) is a mixed path. Since a mixed path P (or a mixed cycle C) is determined by the set of arcs and edges in P (respectively in C), we often identify paths and cycles by their set of edges and arcs.

The *length of a mixed path* P *(cycle* C*)* is the number of edges and arcs it contains and is denoted by $|P|$ ($|C|$). The *size* of a mixed path P (cycle C) is the number of edges in E it contains.

We call a mixed graph *strongly connected* if for every pair $u, v \in V$ there exists a path in G from u to v and from v to u. A graph consisting of a single node is strongly connected by definition. For an arbitrary mixed graph $G = (V, E, A)$ the maximal strongly connected, node-induced subgraphs (maximal with respect to the cardinality

of the node set) are called the *strongly connected components* of G.

Note that all above notations and definitions also hold for *multigraphs* which are graphs where we allow multiple edges (arcs) between a pair of nodes. The only exception is that in the undirected case a cycle can have length two.

2.2 Linear Algebra

In this section we introduce some mathematical notation we use throughout the thesis. We denote the set of real (resp. rational, integer, natural) numbers by $\mathbb{R}$ (resp. $\mathbb{Q}, \mathbb{Z}, \mathbb{N}$). For an ordered finite set $E = \{e_1, e_2, \ldots, e_n\}$ and a field X we denote by X^E the set of vectors in which the components of each vector are indexed by the members of E. For simplicity, when $E = \{1, \ldots, n\}$ we write X^n. In particular, $\mathbb{R}^E$ denotes the $|E|$-dimensional vector space over the field $\mathbb{R}$.

By convention, the vectors are column vectors. For any field X a vector $x \in X^E$ is called *linear combination* of the vectors $x^1, x^2, \ldots, x^k \in X^E$ if there are scalars $\lambda_1, \ldots, \lambda_k \in X$ such that $x = \sum_{i=1}^k \lambda_i x^i$. If additionally $\lambda_i \geq 0$ (resp. $\sum_{i=1}^k \lambda_i = 1$, resp. $\sum_{i=1}^k \lambda_i = 1$ and $\lambda_i \geq 0$), then x is called a *conic* (resp. *affine*, *convex*)*combination* of the vectors $x^1, \ldots, x^k$. If $S \subseteq X^E$, we denote by *lin*(S) (resp. *cone*(S), *aff*(S), *conv*(S)) the *linear hull* (resp. *conic hull*, *affine hull*, *convex hull*) of the elements of S, defined as the set of all vectors which are linear (resp. conic, affine, convex) combinations of finitely many vectors of S. We define 0 as the zero vector in the respective vector space, *i.e.*, $\text{lin}(\emptyset) := \{0\}$, $\text{cone}(\emptyset) := \{0\}$, $\text{aff}(\emptyset) := \emptyset$, and $\text{conv}(\emptyset) := \emptyset$.

A nonempty finite set $S \subset X^E$ is called *linearly independent* (resp. *affinely independent*) if no vector $x \in S$ can be expressed as a linear (resp. affine) combination of the vectors in $S \setminus \{x\}$. Otherwise S is called *linearly dependent* (resp. *affinely dependent*).

If S is linearly independent, then S and $S \cup \{0\}$ are affinely independent. The *rank* (resp. *affine rank*) of a set $S \in X^E$ is the cardinality of the largest linearly (resp. affinely) independent subset of S and denoted by $\text{rank}(S)$ (resp. $\text{arank}(S)$). Observe that for any subset $S \in X^E$ with $0 \in \text{aff}(S)$ the following holds: $\text{arank}(S) = \text{rank}(S)+1$; whereas in the case $0 \notin \text{aff}(S)$ the following holds: $\text{arank}(S) = \text{rank}(S)$. The *dimension* of a set $S \subseteq X^E$ is defined as the maximum number of affinely independent points in S minus one and denoted by *dim*(S). S is called *full-dimensional* if its dimension is $|E|$.

2.3 Polyhedral Theory

In this section we present basic concepts of polyhedral theory. For a comprehensive treatment of this subject the reader should refer to the books by Pulleyblank (1989), by Grötschel and Padberg (1985), or by Cook *et al.* (1998).

Let E and I be two finite, ordered sets. Then a set $P \subseteq \mathbb{R}^E$ is called *polyhedron* if P is the solution of a system of linear inequalities, *i.e.*, $P = \{x \in \mathbb{R}^E \mid Ax \leq b\}$ for some $A \in \mathbb{R}^{I \times E}$ and $b \in \mathbb{R}^I$. A bounded polyhedron is called *polytope*. Another way to represent polyhedra that was first introduced by Farkas, Weyl, and Minkowski (Schrijver 1986) is the combination of the convex and conic hull of finite subsets of $\mathbb{R}^E$.

Theorem 2.3.1: Every polyhedron $P \subseteq \mathbb{R}^E$ has a representation of the form $P = \text{conv}(V) \cup \text{cone}(I)$ where V and I are finite subsets of $\mathbb{R}^E$.

Thus, polytopes are precisely those sets in $\mathbb{R}^E$ that are the convex hull of finitely many points in $\mathbb{R}^E$. If $a \in \mathbb{R}^E \setminus \{0\}$ and $a_0 \in \mathbb{R}$, then the polyhedron $\{x \in \mathbb{R}^E \mid a^T x \leq a_0\}$ is called a *halfspace*. Every polyhedron is the intersection of finitely many halfspaces. An inequality $a^T x \leq a_0$ is called *valid* with respect to a polyhedron P if $P \subseteq \{x \in \mathbb{R}^E \mid a^T x \leq a_0\}$. A set $F \subseteq P$ is called a *face* of P, if there exists a valid inequality $a^T x \leq a_0$ for P such that $F = \{x \in P \mid a^T x = a_0\}$. In this case we say that F is the face of P defined (resp. induced) by the valid inequality $a^T x \leq a_0$. Note that a face of a polytope is itself a lower-dimensional polytope. Especially important faces of polyhedra are those of minimum and maximum dimension.

The 0-dimensional faces of a polyhedron P are called *vertices* of P. Vertices are those points of a polyhedron that cannot be represented as a convex combination of other points.

The faces of dimension $dim(P) - 1$ are called *facets* of a polytope. We say that an inequality $a^T x \leq a_0$ *defines a facet* of P or *is facet-defining* for P, if $\{x \in P \mid a^T x = a_0\}$ is a facet of P.

The following well known theorem constitutes a basis for proving that a valid inequality for a polyhedron P defines a facet of P.

Theorem 2.3.2: (Nemhauser and Trotter 1973) Let $P \subseteq \mathbb{R}^n$ be a polyhedron and $D \in \mathbb{R}^{m \times n}$, $d \in \mathbb{R}^m$ such that $\text{aff}(P) = \{x \in \mathbb{R}^n \mid D \cdot x = d\}$. If F is a (nonempty) face of P then the following assertions are equivalent.

1. F is a facet of P.
2. $\dim(F) = \dim(P) - 1$.

3. There exists a valid inequality $a^T x \leq a_0$ with respect to P with the following three properties:

   (a) $F = \{x \in P \mid a^T x = a_0\}$

   (b) There exists a vector $\hat{x} \in P$ such that $a^T \hat{x} < a_0$

   (c) If $b^T x \leq b_0$ is a valid inequality for P such that $F \subseteq \bar{F} = \{x \in P \mid b^T x = b_0\}$ then there exists a vector $\lambda \in \mathbb{R}^m$ and a number $\mu \in \mathbb{R}$ such that

$$\begin{aligned} b^T &= \lambda^T D + \mu \cdot a^T \\ b_0 &= \lambda^T d + \mu \cdot a_0 \end{aligned}$$

Assertions 2 and 3 provide the two basic methods to prove that a given inequality $c^T x \leq c_0$ is facet-defining for a polyhedron P. The first method, called the direct method, consists of exhibiting a set of $d = dim(P)$ vectors $x_1, \ldots, x_d$ satisfying $c^T x_i = c_0$ and showing that these vectors are affinely independent. The indirect method is the following: We assume that $\{x \mid c^T x = c_0\} \subseteq \{x \mid a^T x = a_0\}$ for some valid inequality $a^T x \leq a_0$ and prove that there exists a $\lambda > 0$ such that $a^T = \lambda \cdot c^T$ and $a_0 = \lambda \cdot c_0$.

2.4 Linear Programming

In the *Linear Programming Problem* we are given a system $Ax \leq b$ of linear inequalities and a linear *objective function* $f(x) = c^T x$. The task is to find a *feasible* solution $\bar{x}$ (which means $A\bar{x} \leq b$) that maximizes (or minimizes) the objective function. In what follows we maximize the objective function without loss of generality.

Given a matrix $A \in \mathbb{R}^{m \times n}$, a vector $b \in \mathbb{R}^m$ and a vector $c \in \mathbb{R}^n$, the corresponding linear programming problem, or simply *linear program (LP)* , is denoted by

$$\max c^T x \quad \text{subject to } Ax \leq b$$

or shortly

$$\max\{c^T x \mid Ax \leq b\}.$$

A feasible solution x^* is called an *optimal solution* if $c^T x^* \geq c^T \bar{x}$ for all feasible solutions $\bar{x}$. If a linear program has no feasible solution, it is called *infeasible*.

An important theory whose development is closely related to linear programming is the *duality theory*. For every linear program

$$(P) : \max\{c^T x \mid Ax \leq b\}$$

another linear program, called the *dual problem* to (P), can be associated, which is defined as

$$(D) : \min\{y^T b \mid y^T A = c^T, y \geq 0\}.$$

The problem (P) is called the *primal problem*. Note that the dual of (D) is (P). A fundamental result in duality theory is stated in the following theorem.

Theorem 2.4.1: (Duality theorem of Linear Programming) Let $P = \max\{c^T x \mid Ax \leq b\}$ and $D = \min\{y^T b \mid y^T A = c^T, y \geq 0\}$. Then the following holds:

1. If (P) and (D) both have feasible solutions, then they have optimal solutions and the optimal values of the objective functions are equal.

2. If (P) (respectively (D)) is infeasible, then (D) (respectively (P)) is either infeasible or unbounded.

3. If (P) (respectively (D)) is unbounded, then (D) (respectively (P)) is infeasible.

It follows from the definition of optimality that the set of optimal solutions of a linear program over a polyhedron $Q = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ is a face of the polyhedron. In particular, if Q is a nonempty polytope then it has at least one optimal solution x^* that is a vertex of Q.

Although the Linear Programming Problem is only a special case of the more general problem of optimizing a multi-dimensional function under certain constraints, linear programming techniques play a prominent role as one of the most heavily and successfully used tools in mathematics. This is certainly due to an efficient solution method, the *simplex method*, that was developed by George Dantzig as early as in 1947. Although other methods such as interior point methods are meanwhile successfully applied to the Linear Programming problem as well, the simplex method is still amongst the state-of-the-art methods. The interested reader can refer to the books by Schrijver (1986) and Chvátal (1983).

The *Integer Linear Programming Problem*, or simply the *Integer Linear Program (ILP)* is defined as

$$\max c^T x \text{ subject to } Ax \leq b, x \text{ integral.}$$

Solving ILPs to optimality can be shown to be an NP-complete problem (Garey and Johnson 1979). Hence, there exist a lot of strategies to approximate or solve such an ILP. Some of these strategies rely on the *LP-relaxation* of the ILP. The LP-relaxation is the linear program, obtained from the above integer linear program, by dropping the integer condition. Substituting the condition (x is integral) by ($x \in \{0, 1\}$) we get a *(0/1)-integer linear programming problem.*

2.5 Integral Polytopes

Throughout the remainder of the thesis we will restrict ourselves to *rational polytopes* P, that means $P \subset \mathbb{Q}^E$. A rational polytope is called *integral* if all its vertices are integral. The applicability of polyhedral methods in combinatorial optimization (at least from a theoretical point of view) often comes down to our ability to prove that polyhedra are integral. An important tool in this effort is the following characterization by Hoffmann (1974), which allows us to restrict our attention to the optimal objective value rather than the actual solutions to linear programming problems over a polytope.

Theorem 2.5.1: (Hoffmann 1974) A rational polytope P is integral if and only if for all integral vectors w the optimal value of $\max\{w^T x \mid x \in P\}$ is an integer.

Proving that polyhedra are integral is often a difficult task that has led to the development of various sufficient conditions for integrality, stating that if A and b satisfy certain conditions, then we know that $\{x \mid Ax \leq b\}$ is integral. One of these sufficient conditions is the *total unimodularity* of the constraint matrix.

A matrix A is called *totally unimodular* if each subdeterminant of A is 0,+1 or -1. In particular, each entry in a totally unimodular matrix is 0,+1 or -1.

A link between total unimodularity and integer linear programming is given by the following fundamental theorem.

Theorem 2.5.2: (Schrijver 1986) Let A be a totally unimodular matrix and let b be an integral vector. Then the polyhedron $P := \{x \mid Ax \leq b\}$ is integral.

Hence, proving a constraint matrix to be totally unimodular gives an elegant way to prove the integrality of the associated polytope. The following theorem gives a list of useful characterizations of total unimodularity.

Theorem 2.5.3: (Schrijver 1986) Let A be a matrix with entries 0,+1 or -1. Then the following statements are equivalent:

1. A is totally unimodular, *i.e.*, each square submatrix of A has determinant 0,+1 or -1.
2. Each collection of columns of A can be split into two parts so that the sum of the columns in one part minus the sum of the columns in the other part is a vector with entries only 0,+1 or -1.

As we will see later, not all integral polytopes are described by a totally unimodular constraint matrix. In such a case one can check, whether another sufficient condition is satisfied, namely whether the primal system $Ax \leq b$ is *totally dual integral*. According

to the duality theorem (Theorem 2.4.1 on page 23) the following equation holds if both the primal and the dual program have feasible solutions:

$$\max\{c^T x \mid Ax \leq b\} = \min\{y^T b \mid y^T A = c^T, y \geq 0\}. \tag{2.1}$$

A rational system $Ax \leq b$ is defined as totally dual integral if the minimum of equation (2.1) can be achieved by an integral vector y for each integral c for which an optimum exists. A nice feature of this definition is that we get primal integrality for free, provided b is integral, as was shown by Hoffmann (1974).

Theorem 2.5.4: (Cook *et al.* 1998) Let $Ax \leq b$ be a totally dual integral system such that $P := \{x \mid Ax \leq b\}$ is a rational polytope and b is integral. Then the polytope P is integral.

Note that the condition that b is integral is essential for the integrality of P. The fact alone that a system is totally dual integral does not say anything about the structure of the polytope.

2.6 Polyhedral Combinatorics

In this section we combine the above introduced concepts of linear algebra, polyhedral theory and linear programming. A problem in the area of combinatorial optimization has the following characteristics:

1. a *groundset* E,
2. a family $\mathcal{I}$ of *feasible* subsets of E,
3. a weight or score w_e associated with each element $e \in E$.

The goal is to find the feasible set $F \in \mathcal{I}$ for which the linear objective function $w(F) := \sum_{e \in F} w_e$ is maximized. For any subset $B \subseteq E$ the *incidence vector* χ^B is defined as $\chi^B_e = 0$ if $e \notin B$ and $\chi^B_e = 1$ if $e \in B$. This yields a one-to-one correspondence of the feasible sets F with certain $(0,1)$-vectors in $\mathbb{Q}^E$. Hence, a combinatorial optimization problem can be associated in a natural way with a polytope P by defining P as the convex hull $conv\{\chi^F \mid F \in \mathcal{I}\}$ over all feasible incidence vectors χ^F. This polytope is also called the *problem polytope*. With the use of the problem polytope the combinatorial optimization problem can be written as the linear program

$$\max\{w^T x \mid x \in P\},$$

because the vertices of P are exactly the incidence vectors of the feasible sets $F \in \mathcal{I}$. In order to apply linear programming techniques to solve the above problem it is necessary

to characterize it as the solution of a linear inequality system. The question arises, under which circumstances can we hope to find such a description?

According to results of Grötschel, Lovász, and Schrijver (1981), Karp and Papadimitriou (1980), and Padberg and Rao (1981), we can optimize a linear objective function over a polytope in polynomial time if and only if we can solve the *separation problem* in polynomial time. The separation problem asks – given a vector $\bar{x} \in \mathbb{Q}^E$ – whether $\bar{x} \in P$, and if $\bar{x} \notin P$, to find a vector $c \in \mathbb{Q}^E$ and a scalar $c_0 \in \mathbb{Q}$ such that the inequality $c^T x \leq c_0$ *separates* $\bar{x}$ from P, *i.e.*, it is valid with respect to P and $c^T \bar{x} > c_0$.

The above result implies that it is very unlikely to find polynomial time separation routines for NP-hard problems. Indeed, for most NP-hard problems even a complete description in terms of linear inequalities is either not known or it has exponential size. Thus our objective can only be to identify a part of such a system. Although that might not appear very helpful at first sight, there are some good reasons to search for a partial description of the problem polytope. On the one hand, we only need to know at most $|E|$ many facet-defining inequalities that describe the optimal vertex of P, on the other hand, the partial description is a relaxation of the problem that can be solved in a branch-and-bound framework where cutting plane techniques and linear programming yield upper and lower bounds. Since an irredundant description of P by linear inequalities contains only facet-defining inequalities, we concentrate on finding those valid inequalities that are facet-defining. A more detailed reasoning is given in Chapter 6.1.

If the feasible sets of a groundset form an independence system, it turns out that a lot of results about the facial structure of the associated polytope follow easily. In the following chapter we introduce independence systems that define a special class of combinatorial optimization problems.

2.7 Independence Systems

A pair $I = (E, \mathcal{I})$ is called an *independence system on* E if $\mathcal{I}$ is a family of subsets of the finite set E with $\emptyset \in \mathcal{I}$ and the property that $F_1 \subseteq F_2$ and $F_2 \in \mathcal{I}$ implies $F_1 \in \mathcal{I}$. The members of $\mathcal{I}$ are called *independent* and those of $2^E \setminus \mathcal{I}$ *dependent* sets. As an example one might consider the problem of finding a spanning tree in a complete graph $K_n = (V, K)$. Every set of edges that does not form a cycle is independent, all other sets of edges are dependent. Hence, the pair $I = (K, \mathcal{I})$ with $\mathcal{I} := \{E \subset K \mid E \text{ does not induce a cycle in } G\}$ is an independence system.

Let $I = (E, \mathcal{I})$ be an independence system on E. A *basis* B of $F \subseteq E$ is a maximal (with respect to set inclusion) independent subset of F, *i.e.*, a set $B \subseteq F$ with the

property that $B \cup \{e\}$ is dependent for all $e \in F \setminus B$. In our example all spanning trees in the graph form a basis, because one cannot add another edge without introducing a cycle.

A *circuit* C is a minimal dependent subset of E, *i.e.*, a set $C \in 2^E \setminus \mathcal{I}$ satisfying $C \setminus \{e\} \in \mathcal{I}$ for all $e \in C$. In the example the cycles in the graph correspond to the circuits of the independence system, because removing any edge from a cycle results in an independent set.

With every subset $F \subseteq E$ we associate a number $r(F) := \max\{|B| \mid B \text{ is basis of } F\}$ called the *rank* of F. In the example the rank of the independence system is $n-1$, because every spanning tree – which forms a basis – has $n-1$ edges.

An independence system is called *k-regular* if each of its circuits is of size k. A set $F \subseteq E$ is a *clique* of a k-regular system $(E, \mathcal{I})$ if $|F| \geq k$ and all $\binom{|F|}{k}$ k-subsets of F are circuits of $(E, \mathcal{I})$. The polyhedron associated with $(E, \mathcal{I})$ is denoted by $P_{\mathcal{I}}$. The following theorems describe basic facts about independence systems.

Theorem 2.7.1: (Grötschel and Padberg 1985) Let $(E, \mathcal{I})$ be an independence system and let $F = E - \bigcup \mathcal{I}$. Then the dimension of $P_{\mathcal{I}}$ is $|E| - |F|$.

The next theorem for full-dimensional polytopes restricts the number of possible facet-defining inequalities.

Theorem 2.7.2: (Hammer, Johnson, and Peled 1975) If $P_{\mathcal{I}}$ is a full-dimensional polytope associated with the independence system $(E, \mathcal{I})$, then $x_i \geq 0$ for $i = 1, \ldots, |E|$ are the only facet-defining inequalities with right hand side 0. Moreover, all the nontrivial facets of $P_{\mathcal{I}}$ are defined by inequalities $a^T x \leq a_0$ with $a \geq 0$ and $a_0 > 0$.

Theorem 2.7.3: (Nemhauser and Trotter 1973) Suppose $F \subseteq E$ is a maximal clique in the k-regular independence system $(E, \mathcal{I})$. Then $\sum_{e \in F} x_e \leq k - 1$ is a facet of $P_{\mathcal{I}}$.

For $F \subseteq E$ we call $I' = (F, \mathcal{I}')$, where $\mathcal{I}' = \{B \in \mathcal{I} \mid B \subseteq F\}$, the *subsystem generated by* F. Given a facet-defining inequality $\sum_{e \in F} a_e x_e \leq a_0$ for the subsystem $(F, \mathcal{I}')$, one may ask whether there is a facet-defining inequality

$$\sum_{e \in F} a_e x_e + \sum_{e \in E \setminus F} a_e x_e \leq a_0$$

for the independence system $(E, \mathcal{I}) \supseteq (F, \mathcal{I}')$. The process of obtaining inequalities from inequalities of subsystems is called *lifting*. For every subset $F \subseteq E$ let $P_{\mathcal{I}}(F)$ denote the polytope $\{x \in P_{\mathcal{I}} \mid x_e = 0 \text{ for all } e \notin F\}$.

Theorem 2.7.4: (Nemhauser and Trotter 1973) Let $(E, \mathcal{I})$ be an independence system, let $F \subseteq E$ and $e \notin F$. Suppose $\sum_{k \in F} a_k x_k \leq a_0$ defines a facet of $P_{\mathcal{I}}(F)$ with $a_0 > 0$.

Set

$$a_e := a_0 - \max\{\sum_{k\in F} a_k \chi_k^I \mid I \subseteq F, \{e\} \cup I \in \mathcal{I}\}.$$

Then $a_e x_e + \sum_{k\in F} a_k x_k \leq a_0$ defines a facet of $P_{\mathcal{I}}(F \cup \{e\})$.

Thus, a facet-defining inequality $a^T x \leq a_0$ for $P_{\mathcal{I}}$ can be derived from a facet-defining inequality of $P_{\mathcal{I}}(F)$ by using the theorem above for all elements $e \in E \setminus F$. In the case that $a_e = 0$ for all $e \in E \setminus F$, we also say that the inequality $a^T x \leq a_0$ is derived by *zero-lifting* from an inequality of one of its subsystems.

2.8 Stringology

The notion of a *string* or *sequence* plays a central role in this work. For the sake of completeness we introduce the following notation about sequences.

Definition 2.8.1: A *sequence* of *length* l is a l-tuple $S = (s_1, s_2, \ldots, s_l)$ over a finite alphabet Σ. For brevity, we omit the parentheses and commata and write $s_1 s_2 \cdots s_l$. The length l is denoted by $|S|$. The set of all finite sequences is denoted by Σ^*. The *empty sequence* has length 0 and is denoted by ϵ. The *concatenation* $C = s_1 \cdots s_l t_1 \cdots t_m$ of two sequences $S = s_1 \cdots s_l$ and $T = t_1 \cdots t_m$ is denoted by the juxtaposition ST. The *reverse* sequence of $S = s_1 \cdots s_l$ is denoted by $S^r = s_l \cdots s_1$. A sequence $I = s_p \cdots s_q$ with $p \geq 1$, $q \leq l$ is called the *infix* from p to q of a sequence $S = s_1 \cdots s_l$ and denoted by $S_{[p:q]}$. An infix is called the i-th *prefix* of $S = s_1 \cdots s_l$ if $p = 1$ and $q = i$; it is called the i-th *suffix* of $S = s_1 \cdots s_l$ if $p = i$ and $q = l$.

Chapter 3

Detecting Similarity – Alignments

In this chapter we give an overview of one of the main concepts that is used for detecting similarity (or distance) between sequences, the concept of *alignments*. Alignments represent relations between sequences that should resemble functional relatedness. The significance of such a relation is quantified with *scoring functions* that assign numerical values to alignments. It is generally difficult to devise scoring functions in which extremely scored alignments indeed reveal biological relatedness of the sequences under consideration. Choosing a representation for similarity and a scoring scheme results in a well defined optimization problem for which efficient algorithms must be devised.

In the first part of this chapter we introduce the notion of *conventional alignment* as a representation of the relation between sequences. We define general scoring schemes in order to quantify the quality of an alignment for both, pairwise and multiple alignments. Although the pairwise case is included in the multiple case, we give it special attention, because scoring functions for multiple alignments are often based on pairwise scoring functions. For pairwise alignment we propose three classes of scoring schemes, the first encompassing scoring functions based on the comparison of two characters of the sequences, the second using the concept of gaps, while the third is based on the comparison of whole segments of the two sequences. For multiple alignments we give a general scoring scheme as well and describe some commonly used scoring functions.

In the second part we extend the framework for conventional alignment in order to deal with secondary structure information. We adapt the scoring schemes introduced in the first part in order to quantify similarity between sequences that is not only based on primary sequence information but also on secondary structure information.

3.1 Conventional Alignments

3.1.1 Pairwise Alignments

Definition 3.1.1: Let Σ be a finite alphabet without the blank character '-' and let $\hat{\Sigma} = \Sigma \cup \{\text{'-'}\}$. If S_1, S_2 are two sequences over Σ with lengths n_1 and n_2 then a *pairwise alignment* A of S_1 and S_2 are two strings $\hat{S}_1$, $\hat{S}_2 \in \hat{\Sigma}^*$ displayed in a $2 \times n$–dimensional matrix with the following properties:

- $a_{i,j} \in \hat{\Sigma} \quad \forall i = 1, 2,\ 1 \leq j \leq n.$
- Sequence $\hat{S}_i$ gives sequence S_i if the blanks are removed.
- There is no column consisting only of blank characters, implying $\max\{n_1, n_2\} \leq n \leq n_1 + n_2$.

We denote the infix of sequence S_i from position l to position k by $S_{i,[l:k]}$. Similarly we denote the infix of a row of the alignment matrix with $A_{i,[l:k]}$. Two characters in $\hat{\Sigma}$ are said to be *aligned* under A if they are placed in the same column of the alignment array. A pair of aligned characters is called a *substitution* or *mismatch* if $a \neq b$ and none of the two characters is the blank character. We call it a *match* if $a = b$, an *insertion* if $a =$ '-' or a *deletion* if $b =$ '-' (see also Figure 3.1). Since it is seldomly necessary to distinguish the case of an *in*sertion from the case of a *del*etion both cases are called *indel*.

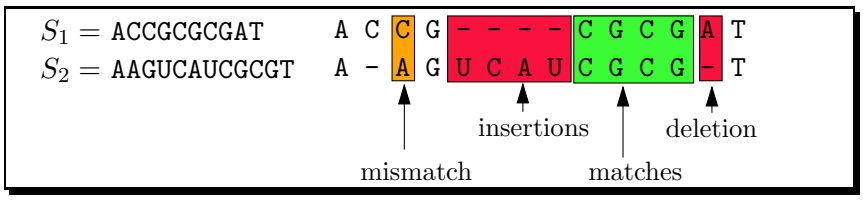

Figure 3.1: A pairwise alignment.

Scoring Pairwise Alignments

Alignments should reflect the biological relatedness (or difference) of the sequences under examination. In order to compute a result that is close to the one that would be produced manually by a biologist, it is necessary to assign some kind of score to each possible alignment. This should be done in such a way that the biologically meaningful results are scored extremely, *i.e.*, with high or low score. Then optimization algorithms can find the alignment(s) with extreme score. We write a pairwise alignment of S_1, S_2 as $A(S_1, S_2)$. The set of all alignments between S_1 and S_2 is denoted by $\mathcal{A}^2(S_1, S_2)$ and the set of all alignments between any two sequences by $\mathcal{A}^2$. Formally we define a pairwise *alignment score function* and the *optimal (pairwise) alignment score* as follows.

Definition 3.1.2 (pairwise alignment score): A function $sc : \mathcal{A}^2 \to \mathbb{R}$ is called *(pairwise) alignment score function*. If A is an alignment of two sequences, then $sc(A)$ is called *(pairwise) alignment score* of A. The *optimal (pairwise) alignment score* of two sequences S_1, S_2 is defined as $sc^{opt}(S_1, S_2) := \text{opt}_{A \in \mathcal{A}^2(S_1,S_2)} sc(A)$, where $\text{opt} \in \{\min, \max\}$ depends on the score function sc.

Score functions based on pairs of residues. Most common scoring schemes are based on the comparison of pairs of letters from $\hat{\Sigma}$. This makes sense on the assumption that the nucleotides respectively the amino acids have evolved independently on different sites of the sequence. A *mutation score function* quantifies for each pair (a, b)

of letters in $\hat{\Sigma}$ the score of aligning them. In general there are two ways to define mutation score functions:

- in form of a *similarity score* $sim : \hat{\Sigma} \times \hat{\Sigma} \to \mathbb{R}$, where pairs that are chemically similar are assigned a high score (similarity) and pairs that are chemically dissimilar are assigned a low score. Scores involving the blank character ($sim('\text{-}', a)$ and $sim(a, '\text{-}')$) are usually set to the same constant b.
- in form of a *distance score* $dist : \hat{\Sigma} \times \hat{\Sigma} \to \mathbb{R}_{\geq 0}$, where pairs that are chemically similar are assigned a low score (distance) and pairs that are chemically dissimilar are assigned a high score (distance). Scores involving the blank character ($dist('\text{-}', a)$ and $dist(a, '\text{-}')$) are usually set to the same constant c.

Normally a mutation score is given in form of a *mutation score matrix*. For the nucleic acids they are often rather simple whereas for amino acid sequences more elaborated matrices are used. They are based on differences in the genetic code (Fitch 1966), chemical properties of the amino acids (Grantham 1974), secondary structural properties (Niefind and Schomburg 1991), or they are derived from empirical data by counting true matches and mismatches in databases of structural alignments and by computing the most reasonable substitution probability that might have led to the observed changes (Risler, Delorme, Delacroix, and Henaut 1988; Jones, Taylor, and Thornton 1992; Henikoff and Henikoff 1992; Dayhoff, Schwartz, and Orcut 1979; Benner, Cohen, and Gonnet 1993).

Given such a score matrix, the most straightforward alignment score function is the sum of the individual distance or similarity scores, namely

$$sc_d : \mathcal{A}^2 \to \mathbb{R}_{\geq 0} \text{ with } sc_d(A) = \sum_{i=1}^{n} dist(a_{1,i}, a_{2,i}),$$

which is also known as *weighted edit distance* and

$$sc_s : \mathcal{A}^2 \to \mathbb{R} \text{ with } sc_s(A) = \sum_{i=1}^{n} sim(a_{1,i}, a_{2,i}).$$

Distance score functions are often required to be a *metric*. That means, that the function is symmetric, fulfills the *zero property* ($sc_d(a, b) = 0 \Leftrightarrow a = b$) and the *triangle inequality* ($sc_d(a, c) \leq sc_d(a, b) + sc_d(b, c)$). If the triangle inequality holds for a mutation score matrix, this means that the evolutionary event of substituting a by c is preferable to substituting a by b and then b by c. It is not clear, however, whether the triangle inequality has a biological justification. While some authors (Beyer, Stein, Smith, and Ulam 1974; Sankoff and Kruskal 1983) argue in favor of metrics, some of

the most commonly used mutation score matrices (Dayhoff, Schwartz, and Orcut 1979; Gonnet, Cohen, and Benner 1992; Henikoff and Henikoff 1992) do not fulfill the triangle inequality. The triangle inequality is important in proving performance guarantees for most approximation algorithms for multiple sequence alignment.

Score functions using gaps. If we use only mutation score matrices in score functions we regard a consecutive run of insertions or deletions as a number of independent events, each of which is assigned a score. On the other hand, if we assume that a single mutational event does not change the sequence at a single point (point mutation) but change longer parts of it, we need some means to score such an event. We call a consecutive run of insertions or deletions in an alignment a *gap* and define it formally as follows.

Definition 3.1.3: A *gap* of *length* l in an alignment A is a maximal, consecutive run of l blank characters in one of the rows of the alignment.

In Figure 3.1 on page 31 there is a gap of length four in the first sequence, whereas the second sequence contains two gaps of length one.

We define the similarity and distance score function for pairwise alignments with gaps as a combination of the summed score of matches, mismatches and indels in the alignment and a *gap cost function* $g_d(l) : \mathbb{N} \to \mathbb{R}_{\geq 0}$ (resp. $g_s(l) : \mathbb{N} \to \mathbb{R}_{\leq 0}$) that gives the cost of a gap of length l in the alignment. The distance score with gaps is normally defined as

$$sc_{d_g} : \mathcal{A}^2 \to \mathbb{R}_{\geq 0} \text{ with } sc_{d_g}(A) = \sum_{i=1}^{n} dist(a_{1,i}, a_{2,i}) + \sum_{l>0} g_d(l) \cdot \#\text{ gaps of length } l \text{ in } A,$$

and the similarity score with gaps is normally defined as

$$sc_{s_g} : \mathcal{A}^2 \to \mathbb{R} \text{ with } sc_{s_g}(A) = \sum_{i=1}^{n} sim(a_{1,i}, a_{2,i}) + \sum_{l>0} g_s(l) \cdot \#\text{ gaps of length } l \text{ in } A.$$

Note that in the literature there exists also a slightly different definition for scoring functions with gaps, because some authors do not sum the individual indel scores but solely let the gap function account for the gap (see Altschul 1989).

If insertions and deletions have the same score b then a gap of length l has score $b \cdot l$. Hence this way of scoring gaps is called *linear gap cost* (also *additive* or *homogeneous*). It is reasonable to assume that a gap cost function is *subadditive*, because the penalty for a gap of length l should always be less than the sum of the penalties for several gaps of total length l. Assume that a sequence $\hat{S}$ contains p gaps of lengths $l_1, l_2, \ldots, l_p$. In the case of a distance score we would require the gap cost function g_d to fulfill $g_d(l_1 + l_2 + \cdots + l_k) \leq g_d(l_1) + g_d(l_2) + \cdots + g_d(l_k)$.

Apart from the linear gap cost functions there are two other classes of subadditive functions that are used in practice, namely *affine functions* ($g(l) = a+b\cdot l$) and *concave functions* (they fulfill the *quadrangle inequality* $g(l+m)-g(l) \geq g(l+m+n)-g(l+n)$).

The subadditivity of these functions can be used to speed up the computation of optimal, pairwise alignments.

Score functions based on pairs of sequence segments. Another class of score functions is based on the comparison of whole segments of the sequences under consideration. The motivation for using such score functions is similar to the motivation for introducing gaps, namely the assumption that the single amino acids or nucleotides might not have evolved independently. Rather one assumes that a whole segment of a sequence was inserted, deleted, or replaced. So instead of scoring a pair of characters from $\hat{\Sigma}$ we now score a pair of segments of equal length from $\hat{\Sigma}^*$ which we call a *block*. A *block score function* quantifies for each block a score, just as a mutation score function assigns a score to each pair of characters. Although it is possible to define a block distance function, all block score functions used in practice are similarity scores based on a *block similarity score function* $bsim : \hat{\Sigma}^* \times \hat{\Sigma}^* \to \mathbb{R}$. The function $bsim$ assigns a high score (similarity) to blocks that contain segments that are similar and a low score to blocks that are dissimilar. Given a block score function the most

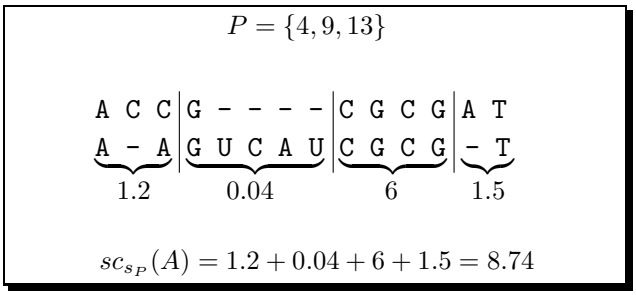

Figure 3.2: Scoring an alignment with a block score function.

straightforward alignment score function is the sum of the individual block similarity scores of consecutive blocks in the alignment. Of course the function needs to know at which position one block ends and the next starts. This information is represented by a sorted sequence of integers $P \subseteq \{2, \ldots, n\}$ that partitions the interval $[1..n]$ in consecutive, smaller intervals, *i.e.*, $P = \{p_1, p_2, \ldots, p_l\}$ with $1 < p_1 < p_2 < \cdots < p_l \leq n$ (see for example Figure 3.2). Setting $p_0 = 1$ and $p_{l+1} = n + 1$ the alignment score function can be defined as

$$sc_{s_P} : \mathcal{A}^2 \to \mathbb{R} \text{ with } sc_{s_P}(A) = \sum_{i=1}^{l+1} bsim(A_{1,[p_{i-1}:p_i-1]}, A_{2,[p_{i-1}:p_i-1]}).$$

3.1.2 Multiple Alignments

The definition of pairwise alignments generalizes easily to the case of more than two sequences.

Definition 3.1.4: Let Σ be a finite alphabet without the blank character '-' and let $\hat{\Sigma} = \Sigma \cup \{\text{'-'}\}$. If $S_1, \ldots, S_k$ are k sequences over Σ with lengths $n_1, \ldots, n_k$ then a *multiple alignment* A of $S_1, \ldots, S_k$ is a $k \times n$–dimensional matrix $A = (a_{i,j})$ consisting of k strings $\hat{S}_1, \ldots, \hat{S}_k \in \hat{\Sigma}^*$. A has the following properties:

- $a_{i,j} \in \hat{\Sigma} \quad \forall\ 1 \leq i \leq k,\ 1 \leq j \leq n.$
- Sequence $\hat{S}_i$ gives sequence S_i if the blanks are removed.
- There is no column consisting only of blank characters implying $\max\{n_1, \ldots, n_k\} \leq n \leq \sum_{i=1}^{k} n_i$.

We also need a notation to talk about the multiple alignment of a subset of the k strings in an alignment. Such a (sub)alignment is called a *projection*.

Definition 3.1.5: Let A be a multiple alignment of the k sequences $S_1, \ldots, S_k$ and $I \subseteq \{1, \ldots, k\}$ be a set of indices defining a subset of the k sequences. Let A_I be the alignment that results from first selecting all rows of A whose index i is in I and then deleting all columns that contain only the blank character. Then A_I is called the *projection* of A on $\{S_i \mid i \in I\}$. If the set I is given explicitly we simplify notation and drop the brackets, *e.g.*, $A_{\{i,j,k\}}$ becomes $A_{i,j,k}$.

Scoring Multiple Alignments

We write a multiple alignment of the k sequences $S_1, \ldots, S_k$ as $A(S_1, \ldots, S_k)$. The set of all alignments between the k sequences $S_1, \ldots, S_k$ is denoted by $\mathcal{A}^k(S_1, \ldots, S_k)$ and the set of all alignments between any k sequences by $\mathcal{A}^k$. Formally we define a multiple *alignment score function* as follows.

Definition 3.1.6 (multiple alignment score): A function $sc : \mathcal{A}^k \to \mathbb{R}$ is called *(multiple) alignment score function*. If A is an alignment of k sequences, then $sc(A)$ is called *(multiple) alignment score* of A. The *optimal (multiple) alignment score* of k sequences $S_1, \ldots, S_k$ is defined as $sc^{opt}(S_1, \ldots, S_k) := \text{opt}_{A \in \mathcal{A}^k(S_1, \ldots, S_k)} sc(A)$, where $\text{opt} \in \{\min, \max\}$ depends on the score function sc.

Score functions based on pairs of residues. Most score functions for multiple alignments are based on a combination of pairwise score functions for the projections

onto pairs of sequences. Three popular score functions are the *(weighted) sum of pairs ((W)SOP)* score, the *tree* score, and the *consensus* score.

Weighted Sum of Pairs Score. Given a pairwise alignment score function sc, the weighted sum of pairs score simply sums up a weighted sum of the scores of all pairwise projections of an alignment A, *i.e.*,

$$wsop : \mathcal{A}^k \to \mathbb{R} \quad \text{with} \quad wsop(A) = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} w_{i,j} \cdot sc(A_{i,j}).$$

There is no sound biological reasoning behind the WSOP score function, although it seems to work quite well in practice. The SOP score function was first introduced by Carrillo and Lipman (1988) and subsequently used in numerous publications. Altschul and Lipman (1989) first introduced the WSOP score function. The weights change the importance given to a specific pair of sequences and are often intended to reflect known evolutionary distance between the organisms from which the sequences are obtained. Using weights, one can try to induce the multiple alignment to more accurately reflect known evolutionary history. Normally this is done using a *evolutionary* or *phylogenetic* tree which is assumed to be given or has to be computed first (Gupta, Kececioglu, and Schaeffer 1995; Altschul, Carroll, and Lipman 1989). The weights are intended to compensate for the over-representation of certain sequence families that have little evolutionary distance.

Although the WSOP problem can be solved to optimality in exponential time by the same algorithm that solves the SOP problem, little is known about approximating the WSOP problem, whereas for the SOP problem a bounded error approximation with factor $2 - l/k$ is known (Bafna, Lawler, and Pevzner 1994), provided that the distance score is a metric ($l < k$ is the cardinality of optimally aligned subsets of the sequences).

Tree Score. When a phylogenetic tree T is available that depicts the ancestral relationships among an ensemble of sequences, one realistic approach is to reconstruct the ancestral sequences (if not given) and then align the input sequences together with the reconstructed sequences following the incidence relations of the tree. More formally, given a pairwise alignment score function sc the *tree score* of an alignment A is defined as

$$tree_T : \mathcal{A}^k \to \mathbb{R} \quad \text{with} \quad tree_T(A) = \sum_{(i,j) \in T} sc(A_{i,j}),$$

where (i, j) is an edge in the tree. Note that A contains the reconstructed sequences that may not be known in advance (see also Schwikowski 1998; Altschul and Lipman 1989). For example in Figure 3.3 on the next page you see a multiple alignment together with an evolutionary tree.

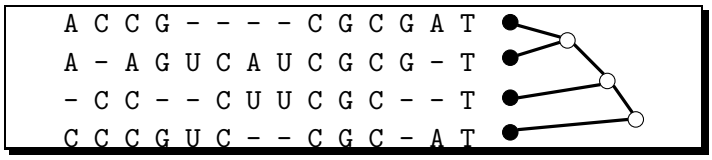

Figure 3.3: Multiple alignment with evolutionary tree.

Consensus Score. Let i be any column of a multiple alignment A. For a column j the character x_j is called the j-th *consensus character* if it yields an extremal score for the *consensus function* $\sum_{i=1}^{k} f(x_j, a_{i,j})$, where f is either similarity or distance score function. The concatenation of all consensus characters is called *consensus sequence*. The *consensus score* is then defined as the sum over the values of all consensus functions, *i.e.*,

$$cons_f : \mathcal{A}^k \to \mathbb{R} \quad \text{with} \quad cons_f(A) = \sum_{i=1}^{k} \sum_{j=1}^{n} f(x_j, a_{i,j}).$$

For example in Figure 3.4 the consensus function for each column scores 1 if $x_j = a_{i,j}$ and 0 otherwise. That means the character that appears most often in a column is the consensus character with ties broken arbitrarily. Then the consensus score of the alignment is 40.

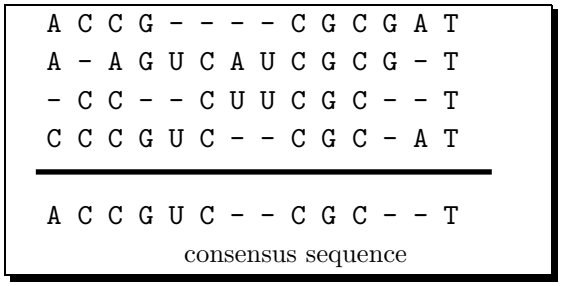

Figure 3.4: Multiple alignment with consensus sequence.

Score functions using gaps. In pairwise alignments scoring functions with affine gap costs help to build alignments that often model the biological truth quite well. For pairwise alignments an optimal alignment with affine gap costs can be computed in time $O(n_1 \cdot n_2)$ by a dynamic programming algorithm (Gotoh 1982) which uses three lookup tables.

For multiple alignment it is straightforward to extend the above mentioned dynamic programming algorithm such that its running time is $\Omega(n^k)$ per stored k-dimensional table. In addition most multiple alignment score functions could handle gaps in a

"natural" way (so-called by Altschul (1989)). However, the computational problems very quickly become intractable because the number of relevant information that must be stored and updated in each of the $\Omega(n^k)$ steps of the algorithm grows very quickly (Altschul (1989) showed that this number grows as $O(k^k \cdot \sqrt{k})$, that means for the WSOP alignment score function for two sequences we have to lookup 3 two-dimensional tables per step, whereas for nine sequences we have to lookup 7087261 nine-dimensional tables).

Therefore, in practice other gap models are used. Implementations of algorithms for exact multiple sequence alignment under the WSOP score either only use score function with linear gap costs (Lermen and Reinert 1997) or they use a gap cost function that do not need that amount of information (Altschul 1989; Kececioglu and Zang 1998), which yields acceptable results while not slowing down the computation too much.

Score functions based on pairs of segments. Similar to the case of two sequences, score functions based on the comparison of two segments are usually similarity functions. For multiple alignments one uses combinations of the pairwise block similarity functions like the WSOP score or the tree score. Since we have to know where a block starts and ends we now define for every pair S_i, S_j of sequences a sorted sequence of integers $P^{i,j} \subseteq \{2, \ldots, n\}$ with $P^{i,j} = \{p_1^{i,j}, \ldots, p_{l_{i,j}}^{i,j}\}$. The set $P = \{P^{i,j} \mid 1 \leq i < j \leq n\}$ completely describes where the blocks start and end. Then an alignment score function can be defined as

$$sc_{s_P} : \mathcal{A}^k \to \mathbb{R} \text{ with } sc_{s_P}(A) = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \sum_{m=1}^{l_{i,j}} bsim(A_{i,[p_{m-1}^{i,j}:p_m^{i,j}-1]}, A_{j,[p_{m-1}^{i,j}:p_m^{i,j}-1]}).$$

Figure 3.5 on the next page shows an alignment of three sequences and the corresponding blocks. In contrast to the pairwise case the alignment of two gap characters may occur in a block. This case can easily be accounted for by defining the block substitution function *bsim* appropriately.

3.2 Structural Alignment

In structural sequence alignment we not only take into account the primary but also the secondary structure of the sequences under consideration. RNA is typically a single stranded molecule which folds intramolecularly to form a number of hydrogen bonded base pairs, mostly `C-G` and `A-U`. These base pairs are called *complementary* and differ in the number of hydrogen bonds they form. `C-G` pairs form three hydrogen bonds and tend to be more stable than `A-U` pairs, which form only two. These so called *Watson-Crick* base pairs are approximately coplanar and are normally stacked onto other base

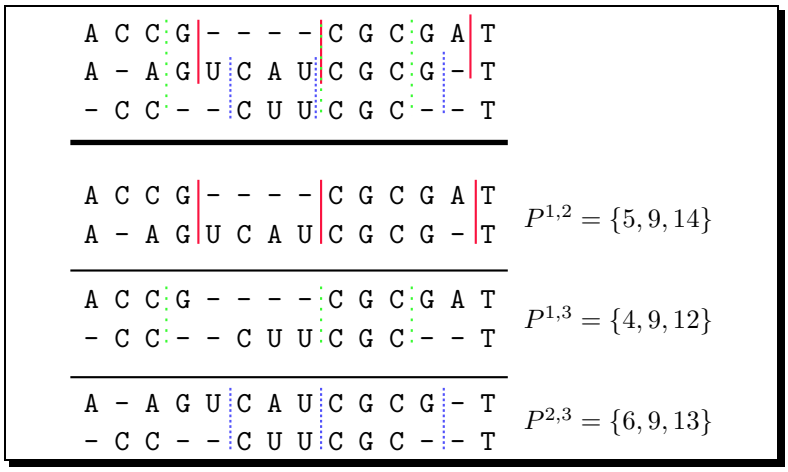

Figure 3.5: Scoring a multiple alignment with a block score function. Blocks may contain an aligned pair of gap characters.

pairs in an RNA structure. Contiguous stacked base pairs are called *stem* or *helix*,

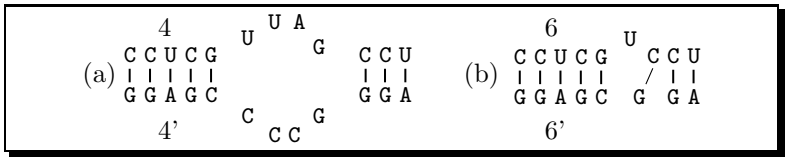

Figure 3.6: Stems with (a) interior loop and (b) bulge

because in three-dimensional space, RNA stems generally form a double helix. Each side of a stem is called a *strand* of the stem (or helix). If one strand is numbered x then the complementary strand is numbered x' and the whole stem is denoted by $x : x'$. Note that this notation has nothing to do with the $3'$-end and $5'$-end of a sequence. This whole base-paired structure is called the *secondary structure* of the RNA (see Figure 3.7 on the following page). It is worthwile noting that stems or helices may not be contiguous but rather lack the hydrogen bonding at some points. Single stranded bases (meaning not having a hydrogen bonded counterpart) occuring within a stem are called a *bulge* or *bulge loop* if the single stranded bases are only on one side of the stem (see Figure 3.6 (b)), or an *interior loop* if there are single stranded bases interrupting both sides of the stem (see Figure 3.6 (a)). In addition to the canonical Watson-Crick pairs other base pairs may occur as well, the most common of which is `G-U`. Non canonical pairs distort the regular form of RNA helices and seem to be the favored target of proteins specialized for recognizing RNA.

We can think of the *secondary structure* as a set of *interactions* between pairs of char-

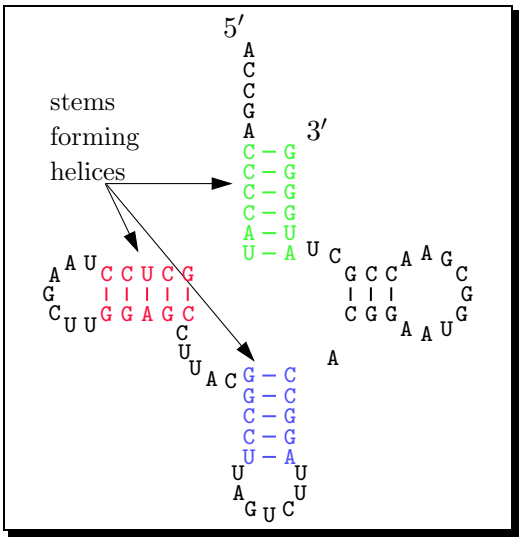

Figure 3.7: Representation of a structured sequence with base pairs grouped together.

acters in the primary structure. The only limitation we require is that any character is involved in at most one pairwise interaction. We represent an interaction between the i-th and the j-th character of a sequence S of length n by an ordered pair (i, j), $1 \leq i < j \leq n$. We say that an interaction (i, j) *encloses* another interaction (i', j') if $i < i' < j' < j$. Two interactions are called *nested* if one encloses the other. Interactions almost always occur in a nested fashion in RNA secondary structure. If two interactions (i, j), (i', j') are not nested they build a *pseudoknot* meaning $i < i' < j < j'$ or $i' < i < j' < j$. In Figure 3.7 and Figure 3.8 you see two possible representations of a

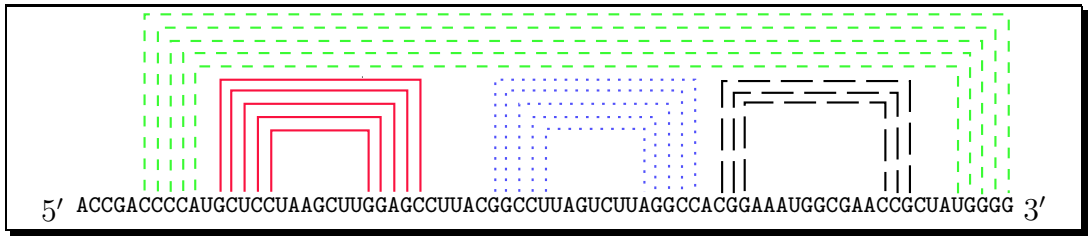

Figure 3.8: Linear representation of a structured sequence.

tRNA (transfer RNA) sequence together with its secondary structure. The first figure resembles the spatial distance of the nucleotides by grouping base pairs next to each other. The second figure shows a linear representation of the primary sequence. This drawing neatly displays the nested structure of a secondary structure without pseudoknots. This nested structure can also be represented as a tree in which the nodes

correspond to interactions. An edge between two nodes symbolizes the immediate enclosure of an interaction by another (see also Figure 3.9 for the tree corresponding to the structure of the tRNA in Figure 3.7 and Figure 3.8).

Note that the representations in Figure 3.7 and Figure 3.9 are not well suited for representing a secondary structure with pseudoknots while in the linear representation it appears as two "crossing edges". Therefore we will use this representation to display a sequence with its secondary structure. We now define the notions introduced above more formally.

Definition 3.2.1: Let S be a sequence over $\hat{\Sigma}$ with length m. A pair (i, j) with $1 \leq i < j \leq m$ is called *interaction* if $s_i \neq$ '-' and $s_j \neq$ '-'. A set P of interactions is called *annotation* of S. Two interactions (i, j), (k, l) in an annotation are in *conflict* if $i = k$ or $i = l$ or $j = k$ or $j = l$. A *(secondary) structure* is an annotation with no two interactions in conflict. If S is a sequence and P an annotation we call the pair (S, P) an *annotated sequence*. If P is a (secondary) structure we call it a *structured sequence*.

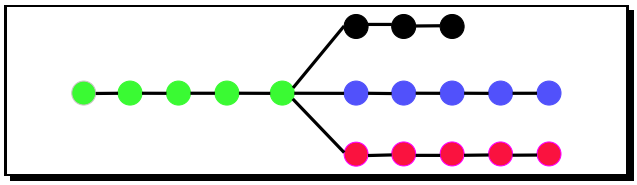

Figure 3.9: Tree representation of a structured sequence (must be without pseudoknots).

While in conventional sequence alignment the input consists of k (primary) sequences, in structural sequence alignment it consists of k annotated sequences $(S_1, P_1), \ldots, (S_k, P_k)$. Similar to conventional sequence alignment we represent the similarity among the k sequences by an alignment, the so-called *structural (multiple) alignment*.

Basically a structural alignment consists of k structured sequences $(\hat{S}_1, \hat{P}_1), \ldots, (\hat{S}_k, \hat{P}_k)$ such that the i-th sequence without blanks is equal to S_i and the pairs in $\hat{P}_i$ correspond to a subset of pairs in P_i that form a secondary structure. We introduce the function $gaps : \{1, \ldots, k\} \times \{1, \ldots, n\} \to \mathbb{N}$ with

$$gaps(i, j) = |\{l < j \mid a_{i,l} = \text{'-'}\}|.$$

The value $gaps(i, j)$ is the number of gap characters in the j-th prefix of sequence $\hat{S}_i$.

Definition 3.2.2: Let Σ be a finite alphabet without the blank character '-' and let $\hat{\Sigma} = \Sigma \cup \{\text{'-'}\}$. If $(S_1, P_1), \ldots, (S_k, P_k)$ are k annotated sequences $\in \Sigma^*$ with lengths $n_1, \ldots, n_k$ then a *multiple structural alignment* A of $(S_1, P_1), \ldots, (S_k, P_k)$ is a $k \times n$–

dimensional matrix consisting of k structured sequences $(\hat{S}_1, \hat{P}_1), \ldots, (\hat{S}_k, \hat{P}_k)$ with the following properties:

- $a_{i,j} \in \hat{\Sigma} \quad \forall\ 1 \leq i \leq k,\ 1 \leq j \leq n.$
- Sequence $\hat{S}_i$ gives sequence S_i if the blanks are removed.
- There is no column consisting only of blank characters implying $\max\{n_1, \ldots, n_k\} \leq n \leq \sum_{i=1}^{k} n_i$.
- $\forall (l, m) \in \hat{P}_i$ the following holds $(l - gaps(i, l), m - gaps(i, m)) \in P_i$.

The last condition in the above definition ensures that the secondary structures of the k structured sequences $(\hat{S}_1, \hat{P}_1), \ldots, (\hat{S}_k, \hat{P}_k)$ only contain interactions that are present in the annotations of the input sequences. For example in Figure 3.10 on the next page the interaction in the second sequence of the second structural alignment is $(2, 12)$. This is a valid interaction, because $(2 - gaps(2, 2), 12 - gaps(2, 12)) = (1, 8)$ is an interaction in the annotation of the second input sequence.

A pair of interactions (i, j), (k, l) in two different sequences is said to be *aligned* or *matched* if $i = k$ and $j = l$, that means the interacting characters in the two sequences are aligned. In Figure 3.10 on the facing page you see two structural alignments of the annotated sequences in the figure. In the upper alignment three pairs of interactions are matched while in the second, only one pair is matched. Note that a structural alignment can contain unmatched interactions in the secondary structures of its structured sequences.

Scoring Structural Alignments

Like in conventional alignment a structural alignment should reflect the biological relatedness of the sequences under examination. In contrast to the conventional case we have as information not only the primary but also the secondary structure of the sequences. As we have argued above, this additional information may be crucial for detecting similarity, because in some cases it is not the sequence itself that is conserved through evolution but the structure. Since it is not straightforward to define a distance score function for structural alignment and since all scoring functions used in practice are based on similarity scores for pairs of residues, we restrict ourselves to similarity score functions based on the comparison of two residues. We write a structural alignment of k annotated sequences $(S_1, P_1), \ldots, (S_k, P_k)$ as $A_s((S_1, P_1), \ldots, (S_k, P_k))$. The set of all structural alignments of $(S_1, P_1), \ldots, (S_k, P_k)$ is denoted by $\mathcal{A}_s^k((S_1, P_1), \ldots, (S_k, P_k))$ and the set of all alignments between any k annotated sequences by $\mathcal{A}_s^k$. Formally we

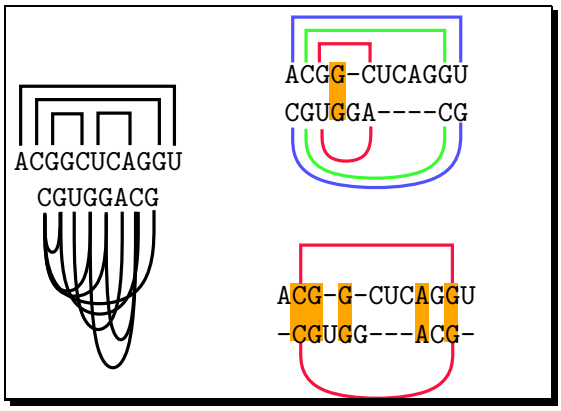

Figure 3.10: Two annotated RNA sequences and two structural alignments of them.

define a *structural alignment score function* and the *optimal structural alignment score* as follows.

Definition 3.2.3 (structural alignment score): A function $sc : \mathcal{A}_s^k \to \mathbb{R}$ is called *structural alignment score function*. If A_s is a structural alignment of k annotated sequences, then $sc(A_s)$ is called *structural alignment score* of A_s. The *optimal structural alignment score* of k annotated sequences $(S_1, P_1), \ldots, (S_k, P_k)$ is defined as $sc^{opt}((S_1, P_1), \ldots, (S_k, P_k)) := \max_{A_s \in \mathcal{A}_s^k((S_1,P_1),\ldots,(S_k,P_k))} sc(A_s)$.

Scoring functions for structural alignment are usually composed of two parts. One part that evaluates the alignment of the characters in the sequence, and another part that evaluates the alignment of the interactions in the secondary structures. Most scoring functions use a weighted sum to combine these two parts.

In the pairwise case one could for example use the similarity score *sim* defined as on page 32 in order to score the alignment of the characters in A_s and one could use an *interaction score isim* : $\Sigma^4 \to \mathbb{R}$ that assigns a score to two pairs of aligned characters that are defined by two matching interactions (note that an interaction never connects a gap character to another character).

We use annotated RNA sequences to illustrate how such an interaction score can be defined. The amount of information given may vary from little information (the annotation contains all possible Watson-Crick interactions) to specific information (the annotation is a secondary structure). These two possibilities for the annotation of an input RNA sequence give rise to three slightly different variations of the problem:

- We are given two annotated sequences (S_1, P_1) and (S_2, P_2), where the annotations consist in both sequences of all possible base pairs.

- We are given two annotated sequences (S_1, P_1) and (S_2, P_2), where the first annotation is a secondary structure and the second consist of all possible base pairs (see also Figure 3.10 on the preceding page).
- We are given two annotated sequences (S_1, P_1) and (S_2, P_2), where the annotations consist in both sequences of secondary structures.

Since secondary structure is highly conserved in RNA sequences, we assign a positive score to matching interactions. Among two alignments with almost the same conventional score we would prefer the one with more matching interactions.

The following structural alignment score function $rna : \mathcal{A}_s^2 \to \mathbb{R}$ could be used to identify alignments that have both, high sequence and high structure conservation:

$$rna(A_s) = \sum_{i=1}^{n} sim(a_{1,i}, a_{2,i}) + \sum_{\substack{(j,l)\in\hat{P}_1,(q,r)\in\hat{P}_2 \\ (q,r)=(j,l)}} isim(a_{1,j}, a_{1,l}, a_{2,q}, a_{2,r}).$$

The above function assigns a high score not only to alignments with many matching characters in the sequence but also to alignments that show few such matches but represent structural conservation. For example in Figure 3.10 on the page before one sees two structural alignments of two annotated sequences. The first has more matching interactions whereas the second has more matching characters. If we score a character match with 1 and an interaction match with 3, the first alignment would be scored higher (score 10) than the second (score 8). This shows how structural information encoded in the annotations can help to find biologically more relevant alignments.

For multiple structural alignment, we can combine pairwise structural scoring functions similar to conventional alignment, that means we could define a *weighted sum of pairs score* or a *tree alignment* score, just like in conventional alignment.

Chapter 4

Detecting Similarity – Traces

In this chapter we give an overview of another important concept that is used for detecting similarity between sequences, the *trace*. Similar to alignments traces represent relations between sequences that should resemble functional relatedness. In contrast to alignments they make no distinction between different arrangements of unaligned regions. We show that traces can be formulated in terms of graphs, which proves very useful in the formulation of integer linear programs for trace problems. The ILPs in turn are the basis for the polyhedral approach to solving these problems. In this chapter we proceed analogously to Chapter 3 on page 29 intending to illustrate the differences and similarities of traces and alignments. We start by introducing the notion of conventional traces as representation of the relation between two sequences. Then we define scoring schemes based on both, the comparison of two characters of the sequences and the comparison of whole segments of the sequences. Although the latter scheme includes the former as a special case, we nevertheless give it special treatment, because most of the commonly used scoring schemes are based on the comparison of two residues. The pairwise scoring schemes are then used to define scoring functions for multiple traces.

In the second part we extend the notion of a trace in order to deal with secondary structure information. We show how the scoring schemes introduced in the first two parts can be adapted in order to quantify a similarity between sequences that is not only based on primary but also on secondary sequence information.

4.1 Conventional Traces

4.1.1 Pairwise Traces

Assume we are given a pairwise alignment $A(S_1, S_2)$ of two sequences S_1 and S_2. We can identify the characters of S_1 and S_2 as the vertices V of the complete bipartite graph $G = (V, E) = K_{n_1,n_2}$, *i.e.*, $V = S_1 \cup S_2$ where $S_i := \{s_{i,j} \mid 1 \leq j \leq n_i\}$. We call G the *input alignment graph*. The edges in G are called *alignment edges* and represent possible (mis)matches of the characters in the two sequences. We say that a (mis)match in the alignment A *realizes* the edge in E that joins the aligned characters. The set of all edges in E that is realized by an alignment is called a *trace*, a notion first introduced by Sankoff and Kruskal (1983). In Figure 4.1 on the next page an alignment and the corresponding trace are shown. Of course not all edges can be realized simultaneously. For an edge $e \in E$ we denote by $start(e)$ the position (or index) of the letter of S_1 where the edge e starts and by $end(e)$ the position of the letter of S_2 where the edge e ends. For two alignment edges e and f $(e \neq f)$ we define the irreflexive, transitive partial order '$\prec$' as follows:

Definition 4.1.1: Two alignment edges $e, f \in E$ are in relation $e \prec f$ if they both

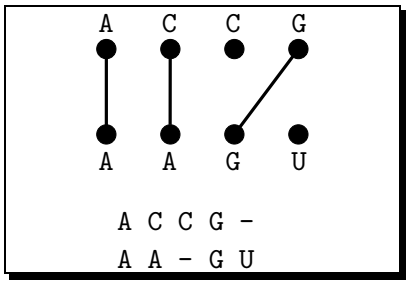

Figure 4.1: A pairwise trace and the alignment that realize that trace

start in S_i, end in S_j, and if

$$start(e) > start(f) \quad \text{and} \quad end(e) \leq end(f) \text{ or}$$
$$start(e) = start(f) \quad \text{and} \quad end(e) < end(f).$$

Two alignment edges e and f are *in conflict* if either $e \prec f$ or $f \prec e$.

In Figure 4.2 you see an alignment graph with four edges e, f, g and h. The two alignment edges e and f cannot be realized simultaneously, because they are both adjacent to the same node in the first sequence ($e \prec f$). Likewise the two alignment edges g and h cannot be both realized, because we are not allowed to change the order of the characters in the sequence ($h \prec g$). By choosing a proper subgraph of the

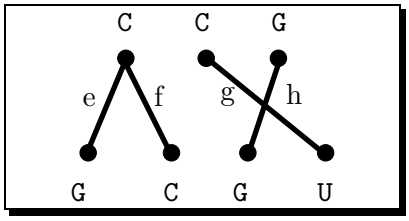

Figure 4.2: An alignment graph with four alignment edges e, f, g and h. The sets $\{e, f\}$ and $\{g, h\}$ contain conflicting alignment edges.

complete bipartite graph K_{n_1,n_2} rather than K_{n_1,n_2} itself, one can conveniently encode restrictions on the set of all possible traces. We will discuss this in Section 7 in more detail and define now a pairwise trace as follows.

Definition 4.1.2: Let Σ be a finite alphabet and S_1, S_2 be two sequences over Σ with lengths n_1 and n_2. Let $G = (V, E)$ be a bipartite graph where V corresponds to the characters of S_1 and S_2. A *pairwise trace* of G is a subset $T \subseteq E$ with the property that no two edges in T are in conflict.

At this point there are two things to note. Firstly, if the input alignment graph is

not the complete bipartite graph, then the alignment of two characters may not realize an alignment edge if it is not contained in the input alignment graph. For example in Figure 4.3 there is no alignment edge between the second character of the first sequence `C` and the second character of the second seuqence `A`. Hence the alignment of these two characters does not realize an alignment edge. Secondly, as a mode of analysis, pairwise alignments are richer than pairwise traces in the sense that an alignment makes some order distinctions between adjacent indels which a trace does not as Figure 4.3 illustrates. Note that there can be several alignments that realize all edges in the same

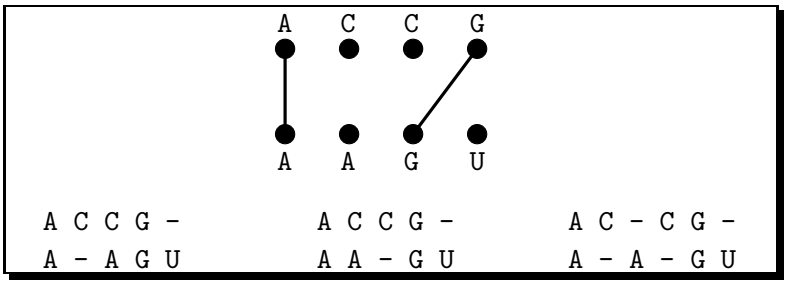

Figure 4.3: A pairwise trace and three pairwise alignments that realize that trace

trace. In a mathematical sense the mapping from alignments to traces is many-to-one and onto.

Scoring Pairwise Traces

In contrast to alignment scoring functions where the two concepts of distance and similarity occur, trace score functions are similarity scores. Given an alignment graph $G = (V, E)$, we denote the set of all pairwise traces in that graph by $\mathcal{T}^2(G)$ and the set of all traces in any pairwise alignment graph by $\mathcal{T}^2$. Formally we define a pairwise *trace score function* and an *optimal (pairwise) trace score* as follows.

Definition 4.1.3 (pairwise trace score): A function $sc : \mathcal{T}^2 \to \mathbb{R}$ is called *(pairwise) trace score function*. If $G = (V, E)$ is an alignment graph for two sequences S_1 and S_2 and $T \subseteq E$ is trace in G, then $sc(T)$ is called (pairwise) trace score of T. The *optimal pairwise trace score* of S_1, S_2 is defined as $sc^{opt}(S_1, S_2) := \max_{T \in \mathcal{T}^2(G)} sc(T)$.

Score functions based on pairs of residues. Traces neglect the order of unaligned residues and therefore only matches and mismatches are scored. Since the (mis)matches correspond to the alignment edges in the input alignment graph, we assign each alignment edge $e \in E$ a weight w_e representing the benefit of realizing that edge. Then a

pairwise trace score function can be defined as

$$sc : \mathcal{T}^2 \to \mathbb{R} \text{ with } sc(T) = \sum_{e \in T} w_e.$$

It is worthwile to note that the standard similarity and distance score functions for alignments can be used to define similarity trace score functions.

This can be done under the reasonable assumption that the score for an indel is a symbol-independent constant x and that for a similarity score $sim(a, b) \geq sim(a, \text{'-'}) + sim(\text{'-'}, b) = 2x$, which simply means that the single mutational event of substituting a by b should be scored higher than the two mutational events of deleting a and then inserting b. Analogously, we require for a distance score $dist(a, b) \leq dist(a, \text{'-'}) + dist(\text{'-'}, b) = 2x$ which is obviously true if $dist$ is a metric.

We show how to transform a similarity alignment score function to a trace score function assuming that the two above assumptions hold. It should be noted that for distance score functions that are a metric a similar construction is possible even for symbol-dependent indel costs (Kececioglu 1998).

If the above conditions hold and we are given a similarity alignment score function

$$sc_1 : \mathcal{A}^2 \to \mathbb{R} \quad \text{with} \quad sc_1(A) = \sum_{i=1}^{n} sim(a_{1,i}, a_{2,i}),$$

where n is the length of the alignment, then we can shift the values of the mutation score matrix in such a way that indels are scored zero and the optimal alignment(s) under the standard similarity stay(s) optimal. Define $sim'(a, b) = sim(a, b) - 2x$, $sim'(a, \text{'-'}) = 0 = sim(a, \text{'-'}) - x$, $sim'(\text{'-'}, b) = 0 = sim(\text{'-'}, b) - x$, and

$$sc_2 : \mathcal{A}^2 \to \mathbb{R} \quad \text{with} \quad sc_2(A) = \sum_{i=1}^{n} sim'(a_{1,i}, a_{2,i}).$$

Let A be any alignment of S_1 and S_2 and let m be the number of (mis)matches that occur at position $p_1, \ldots, p_m$ in that alignment. Then the following holds:

$$\begin{aligned} sc_1(A) &= \sum_{i=1}^{n} sim(a_{1,i}, a_{2,i}) \\ &= \sum_{i=1}^{m} (sim(s_{1,p_i}, s_{2,p_i}) - 2x) + 2mx + (n_1 + n_2 - 2m)x \\ &= \sum_{i=1}^{n} sim'(a_{1,i}, a_{2,i}) + (n_1 + n_2)x \\ &= sc_2(A) + (n_1 + n_2)x. \end{aligned}$$

That means the score of any alignment A under sc_1 differs only by a constant from its score under sc_2. Therefore an optimal alignment under sc_1 is also optimal under sc_2.

If we are given a complete alignment graph we can define the trace score function

$$sc_3 : \mathcal{T}^2 \to \mathbb{R} \text{ with } sc_3(T) = \sum_{e \in T} sim'(s_{1,start(e)}, s_{2,end(e)}),$$

that means we sum the similarity score of all pairs of characters that are incident to an edge in T. If we have an optimal alignment A^{opt} under sc_2 then this alignment realizes in G a trace T^{opt} with the property $sc_3(T^{opt}) = sc_2(A^{opt})$. Hence, if the above mild assumptions hold, the computation of an optimal trace and an optimal alignment yield alignments with the same score. Generally, we can use shifted mutation score matrices that assign positive scores to matches and mismatches.

Score functions based on pairs of sequence segments. In the case of traces we can incorporate segment-to-segment based scoring functions by introducing a partition D of the alignment edges in the alignment graph $G = (V, E)$ and by allowing multiple edges in G. We require that each element of the partition must be a trace and call such a set of edges a *block*. We define the surjective function $v : E \to D$ which maps each edge $e \in E$ to the block $d \in D$ in which e is contained. As a shorthand we write $v(T) := \{v(e) \mid e \in T\}$, $T \subseteq E$. We regard a block $d \in D$ as realized if all the edges in d are realized. Analogously to a block in an alignment we assign to each block $d \in D$

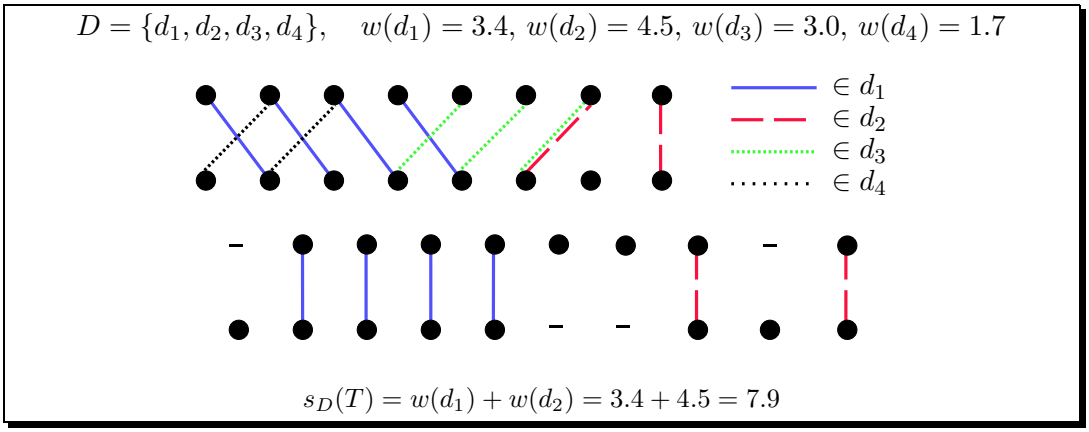

Figure 4.4: Scoring a trace with a block score function. The blocks d_1 and d_2 are realized by the shown alignment.

a positive score w_d representing the benefit of realizing that block. Then we can define

the score of a given trace T as the sum of the scores of the blocks it realizes

$$sc_D : \mathcal{T}^2 \to \mathbb{R} \quad \text{with} \quad sc_D(T) = \sum_{d \in v(T)} w_d.$$

In Figure 4.4 on the preceding page you see an alignment graph whose edges are partitioned into four blocks d_1, d_2, d_3 and d_4 (note that we allow multiple edges between a pair of vertices). The trace shown in the figure realizes the blocks d_1 and d_2.

4.1.2 Multiple Traces

Kececioglu (1991) first generalized Kruskal's notion of a trace in an alignment graph to more than two sequences. Similar to the pairwise case, we view the characters of the k input strings as the vertex set V of a k-partite graph. The edge set E represents pairs of characters that one would like to have aligned in an alignment of the input strings.

The notion of a conflict between two alignment edges, however, is no longer sufficient to test whether a set of alignment edges forms a trace or not. Figure 4.5 shows an input alignment graph for three sequences. The alignment edges e and f cannot be realized simultaneously, because they are in conflict ($e \prec f$). Neither can the three alignment edges g, h and i be realized simultaneously, although these three edges are not mutually in conflict. We can characterize such forbidden combinations of edges as mixed cycles

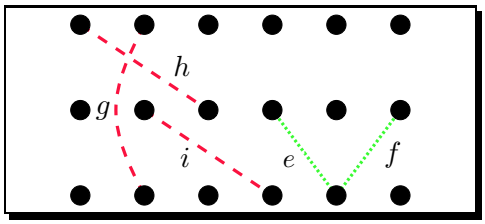

Figure 4.5: Alignment graph for three sequences.

in an extension of the input alignment graph $G = (V, E)$ (see Chapter 2.1 on page 18 for the definition of mixed graph and mixed cycle). We extend G to a mixed graph by adding a set of directed "horizontal" arcs

$$H = \{(s_{i,j}, s_{i,j+1}) \mid \ 1 \leq i \leq k, \ 1 \leq j < n_i\},$$

where $s_{i,j}$ is the vertex that corresponds to letter j in sequence i. We call this graph the *extended alignment graph (EAG)* (see Figure 4.6 on the following page for the two mixed cycles induced by the edges e, f, g, h and i from Figure 4.5).

We call a mixed cycle R in G *critical* if for all i, $1 \leq i \leq k$, all vertices in $R \cap S_i$ occur consecutively in R. Informally this means that a critical mixed cycle enters and leaves each sequences at most once.

The extended alignment graph gives us a simple way of testing whether an edge set may be realized by some alignment or not by simply looking for a critical mixed cycle in it. We give the formal statement in the following theorem.

Theorem 4.1.1: Let $G = (V, E, H)$ be an EAG, let $T \subseteq E$ and let $G' = (V, T, H)$ be the EAG induced by T. Then there exists a multiple alignment A that realizes all edges in T iff there is no critical mixed cycle in G'.

Proof. To prove one direction of the equivalence, we first assume that A is an alignment that realizes all edges in T. A arranges the vertices of G into columns such that all edges in T connect vertices in the same column and such that all arcs in H run from left to right. Thus G' contains no mixed cycle.

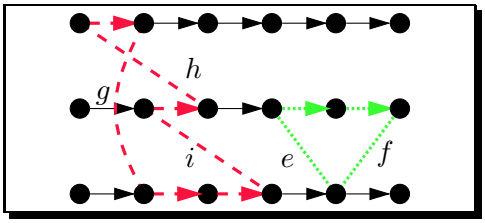

Figure 4.6: Extended alignment graph. Two critical mixed cycles (dashed) are shown.

To prove the converse direction, we assume next that G' contains no critical mixed cycle. We show first that G' contains no mixed cycle and then construct an alignment that realizes all edges in T. Assume first that G' contains a mixed cycle. Consider a mixed cycle R of smallest size (see Section 2.1 on page 18 for a definition of size) and assume that it is not critical. Then there is some i such that the vertices in $R \cap S_i$ are not consecutive in R. Let y be the rightmost vertex in $R \cap S_i$ and let Q be the subpath of R starting in y and ending in the next vertex x on $R \cap S_i$ (note that x is left of y or equal to y). If $x = y$, then either Q or R without the "loop" Q is a mixed cycle smaller than R. If $x \neq y$, then Q together with the path of arcs between x and y is a mixed cycle smaller than R. In both cases we have a contradiction. Thus G' contains no mixed cycle. Let $C_1, \ldots, C_m$ be the connected components of (V, T) (note that each connected component contains at most one vertex from each sequence). Define a directed graph with vertex set $\{C_1, \ldots, C_m\}$ and arc set $\{(C_i, C_j) \mid (x, y) \in H \text{ with } x \in C_i \text{ and } y \in C_j\}$. This graph is acyclic (because G' has no mixed cycle) and hence may be sorted topologically. We obtain an alignment that realizes T by making each component a column of the alignment and by ordering the columns as given by the topological ordering. □

Using Theorem 4.1.1 we can formally define a multiple trace.

Definition 4.1.4: Let Σ be a finite alphabet and $S_1, \ldots, S_k$ be k sequences over Σ with lengths $n_1, \ldots, n_k$. Let $G = (V, E, H)$ be a an extended alignment graph where V corresponds to the characters of $S_1, \ldots, S_k$. A *multiple trace* of G is a subset $T \subseteq E$ with the property that the extended alignment graph induced by $T \cup H$ contains no critical mixed cycle.

Scoring Multiple Traces

Given an extended alignment graph $G = (V, E, H)$ for k sequences we denote the set of all multiple traces in that graph by $\mathcal{T}^k(G)$ and the set of all traces in any extended alignment graph by $\mathcal{T}^k$. Formally we define a multiple *trace score function* and an *optimal (multiple) trace score* as follows.

Definition 4.1.5 (multiple trace score): A function $sc : \mathcal{T}^k \to \mathbb{R}$ is called *(multiple) trace score function.* If $G = (V, E, H)$ is an extended alignment graph for the k sequences $S_1, \ldots, S_k$ and $T \subseteq E$ a trace in G, then $sc(T)$ is called (multiple) trace score of T. The *optimal (multiple) trace score* of the k sequences $S_1, \ldots, S_k$ is defined as $sc^{opt}(S_1, \ldots, S_k) := \max_{T \in \mathcal{T}^k(G)} sc(T)$.

Score functions based on pairs of residues. Defining a multiple trace score is easy. We simply sum up the weights of the edges in the multiple trace:

$$sc : \mathcal{T}^k \to \mathbb{R} \quad \text{with} \quad sc(T) = \sum_{e \in T} w_e.$$

Note that the above trace score incorporates numerous scoring functions depending on how the input EAG is generated.

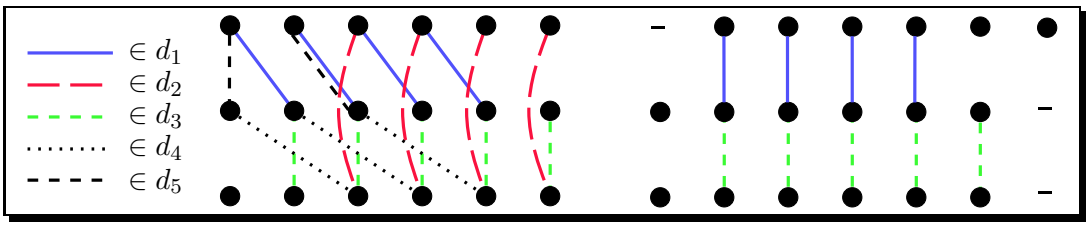

Figure 4.7: Scoring a trace with a block score function. The blocks d_1 and d_2 are realized by the shown alignment.

Score functions based on pairs of segments. Similar to the two sequence case, we define a partition D of the alignment edges into blocks. A block is a trace in which

every edge is incident to nodes in the same pair of sequences. We regard a block $d \in D$ as realized if exactly all edges in d are realized.

We extend the function v in order to handle sets of directed and undirected edges occurring in an extended alignment graph, namely $v : E \cup H \to D \cup \{\emptyset\}$ with

$$v(e) = \begin{cases} d & \text{if } e \in d, \\ \emptyset & \text{if } e \in H, \end{cases}$$

that means the function maps each edge $e \in E$ to the block $d \in D$ in which e is contained. As a shorthand we write $v(C) := \{v(e) \mid e \in C\}$, $C \subseteq E \cup H$.

Analogously to a block in an alignment we assign to each block $d \in D$ a positive score w_d representing the benefit of realizing that block. Since we want to score a block only if all alignment edges in the block are realized we define for each trace T $r(T) = \{d \in v(T) \mid |T \setminus d| = |T| - |d|\}$. Then we can define the score of a given trace T in an EAG $G = (V, E, H)$ as the sum of the scores of the blocks it realizes

$$sc_D : \mathcal{T}^k \to \mathbb{R} \quad \text{with} \quad sc_D(T) = \sum_{d \in r(T)} w_d.$$

In Figure 4.7 on the preceding page you see an alignment graph whose edges are partitioned into five blocks d_1, d_2, d_3, d_4 and d_5. The alignment shown in the figure realizes the blocks d_1 and d_3. Note that the two blocks d_5 and d_1 contain a *different* edge that joins the same pair of vertices. Hence, any alignment that realizes either d_1 or d_5 must match the corresponding characters.

4.2 Structural Traces

First we introduce an additional edge set I to the input EAG in order to reflect all possible interactions in an annotated sequence. We say that two *interaction edges* $i_p, i_q \in I$ are in *conflict* if they are adjacent. A subset $B \subseteq I$ is called *(secondary) structure* if no two edges in B are in conflict. We call $G = (V, E, H, I)$ the *structural extended alignment graph (SEAG)*. In Figure 4.8 on the next page there is a SEAG for two sequences. The interaction edges of the sequences form secondary structures where the second structure contains a pseudoknot (see Section 3.2 on page 38). Similar to structural alignment we define a *structural trace*.

Definition 4.2.1: Let Σ be a finite alphabet and $(S_1, P_1), \ldots, (S_k, P_k)$ be k annotated sequences over Σ. Let $G = (V, E, H, I)$ be a structural extended alignment graph. A *(multiple) structural trace* of G is a pair (T, B) with $T \subseteq E$, $B \subseteq I$ with the property

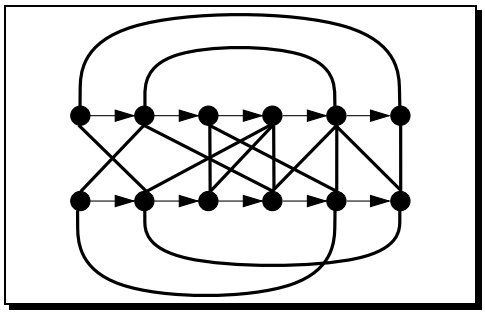

Figure 4.8: Structural extended alignment graph.

that the subgraph induced by $T \cup H$ does not contain a critical mixed cycle and there are no two conflicting interaction edges in B.

For example Figure 4.9 on the next page shows two possible structural traces (the green, dashed edges and the red, dotted edges).

Scoring Structural Traces

Let i_p, i_q be two interaction edges contained in secondary structures of two different sequences in a structural trace (T, B). We say that i_p *matches* i_q if the two alignment edges e_l, e_r joining the two left resp. two right nodes adjacent to i_p and i_q are realized, that means are contained in T. We call the set $\{i_p, i_q, e_l, e_r\}$ *interaction match* and e_l, e_r the *connecting edges* of the interaction match.

Given a structural extended alignment graph $G = (V, E, H, I)$ for k annotated sequences $(S_1, P_1), \ldots, (S_k, P_k)$ we denote the set of all structural traces in that graph with $\mathcal{T}_s^k(G)$ and the set of all structural traces in any structural extended alignment graph by $\mathcal{T}_s^k$. Formally we define a *structural trace score function* and the *optimal structural trace score* as follows.

Definition 4.2.2 (structural trace score): A function $sc : \mathcal{T}_s^k \to \mathbb{R}$ is called *structural trace score function*. If $G = (V, E, H, I)$ is a structural extended alignment graph of k annotated sequences $(S_1, P_1), \ldots, (S_k, P_k)$ and (T, B) with $T \subseteq E$, $B \subseteq I$ is a structural trace in G, then $sc((T, B))$ is called *structural trace score* of (T, B). The *optimal structural trace score* of k annotated sequences $(S_1, P_1), \ldots, (S_k, P_k)$ is defined as $sc^{opt}((S_1, P_1), \ldots, (S_k, P_k)) := \max_{(T,B) \in \mathcal{T}_s^k(G)} sc((T, B))$.

In conventional traces a pair of matching characters is represented by an alignment edge e. Therefore we assign a weight w_e to each alignment edge e, representing the benefit of matching these two characters. In structural traces the connecting edges e_l, e_r of

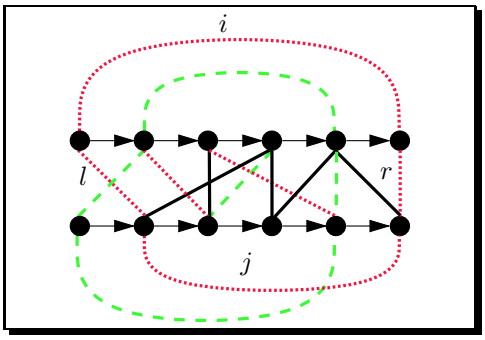

Figure 4.9: Two possible structural traces.

an interaction match $\{i_p, i_q, e_l, e_r\}$ completely describe a pair of matching interactions. Hence we denote an interaction match $\{i_p, i_q, e_l, e_r\}$ with $m_{l,r}$ and assign a weight $w_{l,r}$ to it. Let M be the set of all interaction matches $m_{l,r}$ of a given SEAG $G = (V, E, H, I)$, namely

$$M = \{m_{l,r} \mid m_{l,r} = \{i_p, i_q, e_l, e_r\} \text{ is an interaction match in } G\}.$$

We then can define a structural trace score for k annotated sequences as follows:

$$sc : \mathcal{T}_s^k \to \mathbb{R} \quad \text{with} \quad sc((T, B)) = \sum_{e \in T} w_e + \sum_{\substack{i_p, i_q \in B,\, e_l, e_r \in T \\ \{i_p, i_q, e_l, e_r\} \in M}} w_{l,r}$$

That means we score the alignment edges that are realized by (T, B) as well as the interaction matches that are realized.

4.3 Gapped Traces

It is possible to account for scoring functions with gaps by extending the EAG to a so called *gapped EAG (GEAG)*. A gapped EAG $G = (V, E, H, A)$ has an additional set $A = \bigcup A_{i,j}$ of so called *gap edges*. For each pair S_i, S_j of sequences with $i \neq j$ the set $A_{i,j}$ is defined as $\{g_{i,j,l,m} = \{s_{j,l}, s_{j,m}\}, 1 \leq l \leq n_j, l \leq m \leq n_j\}$. A gap edge $g_{i,j,l,m}$ represents a gap in row i of an alignment. The gap runs in sequence S_j from position l to position m inclusively and hence has length $m - l + 1$ (note that the edge joins two vertices in S_j but represents a gap in row i, see also Figure 4.10 on the facing page). Each gap edge $g_{i,j,l,m}$ is assigned a weight $w_{i,j,l,m}$ representing the penalty of the gap. We say that a gap edge $g_{i,j,l,m}$ *encloses* all nodes in $\{s_{j,q} \mid l \leq q \leq m\}$. For example in

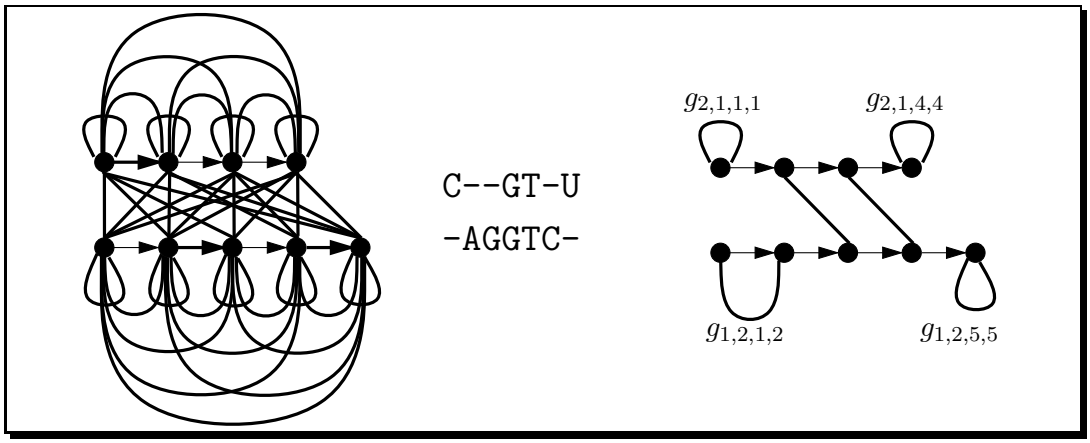

Figure 4.10: GEAG for two sequences. In the middle is an alignment that realizes the gapped trace on the right.

Figure 4.10 the lower left gap edge encloses the first two nodes of the second sequence and represents the gap of length two in the first row of the alignment.

In an alignment, for each pair of sequences a character is either aligned with a character in the other sequence or it is aligned to a gap character. In order to model this for gapped traces, for any pair of sequences a node must either be incident to an alignment edge or must be enclosed by exactly one gap edge. In addition we require that a consecutive run of gap characters is regarded as one gap rather than as the concatenation of two shorter gaps. This leads to the following definition of a *gapped trace*.

Definition 4.3.1: Let $G = (V, E, H, A)$ be an gapped alignment graph. A pair (T, C) with $T \subseteq E$ and $C \subseteq A$ is called *gapped trace* if

1. T is a trace.
2. For each vertex $s_{j,q}$ holds: For all $i = 1, \ldots, k$, $i \neq j$, $s_{j,q}$ is either
   - incident to an alignment edge adjacent to a node in S_i or
   - C contains exactly one gap edge $g_{i,j,l,m}$ that encloses $s_{j,q}$, that is, $l \leq q \leq m$.
3. C contains no two adjacent gap edges.

Given a gapped extended alignment graph $G = (V, E, H, A)$ for k sequences we denote the set of all gapped (multiple) traces in that graph by $\mathcal{G}^k(G)$ and the set of all gapped traces in any gapped extended alignment graph by $\mathcal{G}^k$. Formally we define a *gapped trace score function* and the *optimal gapped trace score* as follows.

Definition 4.3.2 (gapped trace score): A function $sc : \mathcal{G}^k \to \mathbb{R}$ is called *gapped trace score function.* If $G = (V, E, H, A)$ is a gapped extended alignment graph of k sequences $S_1, \ldots, S_k$ and (T, C) with $T \subseteq E$, $C \subseteq A$ is a gapped trace in G, then $sc((T, C))$ is called *gapped trace score* of (T, C). The *optimal gapped trace score* of k sequences $S_1, \ldots, S_k$ is defined as $sc^{opt}(S_1, \ldots, S_k) := \max_{(T,C)\in\mathcal{G}^k(G)} sc((T, C))$.

We define a gapped trace score function as follows:

$$sc : \mathcal{G}^k \to \mathbb{R} \text{ with } sc((T, C)) = \sum_{e\in T} w_e - \sum_{g\in C} w_g.$$

Note that in the gapped trace formulation it is trivial to encode affine, convex or any other reasonable gap cost function. We just have to precompute the weight of each gap edge in A, depending on the gap function we would like to use. Then we simply assign to each gap edge the gap cost depending on its length, position, etc.

Chapter 5

The GMT and SMT Problem

In this chapter we formally define the *Generalized Maximum Trace Problem* (GMT) and the *Structural Maximum Trace Problem* (SMT) using the notations from the previous chapter. Then we briefly describe the two algorithms of Kececioglu (1993) and Bafna, Muthukrishnan, and Ravi (1995) for the MT and SMT problem respectively. Both algorithms are based on dynamic programming. While the SMT algorithm is only designed for structurally aligning two sequences, the algorithm of Kececioglu can deal with $k \geq 2$ sequences.

5.1 Problem Definition

In the GMT formulation the input consists of an EAG $G = (V, E, H)$, a partition D of the alignments edges into blocks, and a block trace score function that assigns a positive weight w_d to each block $d \in D$. Using Theorem 4.1.1 we define the GMT problem as follows:

Generalized Maximum Trace Problem:
Given an EAG $G = (V, E, H)$ and a partition D into blocks with weights w_d ($\forall d \in D$). Find a set $M \subseteq D$ of maximum weight such that $\bigcup_{d \in M} d$ does not induce a critical mixed cycle on G.

In contrast to the original Maximum Trace (MT) formulation (Kececioglu 1991) that can only handle residue-to-residue based trace score functions, the GMT problem can deal with both, segment-to-segment based trace score functions and residue-to-residue based trace score functions. For the former the branch-and-cut algorithm proposed in this thesis is the first algorithm that is able to solve the GMT to optimality. For a restricted version Morgenstern *et al.* (1998) give a heuristic procedure that computes a trace using a greedy strategy.

For residue-to-residue based trace score functions Kececioglu (1993) proposed a dynamic programming based approach that solves the MT problem to optimality. Kececioglu's algorithm was introduced originally to model the final multiple alignment phase of DNA sequence assembly. Kececioglu showed that MT is NP-complete and developed a branch-and-bound algorithm for the problem based on dynamic programming, with worst-case time complexity $O(k^2 2^k N)$ and space complexity $O(N)$, where $N = \prod_i n_i$. The algorithm is able to solve to optimality relatively small problem instances. In Chapter 5.2.1 we review Kececioglu's algorithm, because it currently is the best dynamic programming approach for the MT problem.

The second problem we address is the *Structural Maximum Trace Problem* (SMT). The input for the SMT problem is an structural extended alignment graph (SEAG).

As structural trace score function we use a weighted sum of the weights of the alignment edges and the weights of interaction matches as described in Chapter 4.2 on page 54. Recall that M is the set of all interaction matches in G, namely

$$M = \{m_{l,r} = \{i_p, i_q, e_l, e_r\} \mid m_{l,r} \text{ is an interaction match in } G\}.$$

Given a subset M' of M let $I(M')$ be all interaction edges of the interaction matches in M', that is

$$I(M') = I \cap \bigcup_{m_{l,r} \in M'} m_{l,r}.$$

Then we define the SMT problem as follows:

The Structural Maximum Trace Problem
Given an SEAG $G = (V, E, H, I)$ with weights $w_e(\forall e \in E)$ and $w_{l,r}(\forall m_{l,r} \in M)$, find the structural trace $(E', I(M'))$, $E' \subseteq E$, $M' \subseteq M$ with maximum weight.

Bafna, Muthukrishnan, and Ravi (1995) introduced a number of problem formulations for computing similarity between two annotated sequences among them two variations of the SMT problem. In the first variation they align two structured sequences whereas in the second variation they infer the secondary structure of the first sequence to the second by allowing any Watson-Crick base pair in the annotation of the second sequence and then aligning it with the first structured sequence. In their paper they gave two different recurrences for the two above mentioned variations of the problem with a time bound of $O(n_1^2 \cdot n_2^2)$ in the case that P_1 and P_2 are both secondary structures and $O(n_1^2 \cdot n_2^2 + n_1 \cdot n_2^3)$ in the case that P_2 is no secondary structure. We will show that a slight modification of their recurrence can solve any variation of pairwise structural alignment in time $O(n_1^2 \cdot n_2^2)$. In Chapter 5.2.2 we review their algorithm, because it currently is the best dynamic programming approach for the SMT problem.

5.2 Dynamic Programming based Algorithms

5.2.1 The MT Algorithm

Kececioglu (1993) showed that the MT problem, like most multiple sequence alignment problems, can be solved by dynamic programming, and is equivalent to finding a longest path from a designated source to a designated sink in a k-dimensional acyclic mesh-shaped digraph, the so-called dynamic programming graph.

For simplicity assume that the input alignment graph is the complete k-partite graph. In this case the dynamic programming graph $DP = (V, A)$ has the node set $V =$

$\{v = (v_1, v_2, \ldots, v_k) \mid v_i \in \{0, \ldots, n_i\}\}$ and the arc set $A = \{(p, q) \mid p, q \in V,\ p \neq q \text{ and } q - p \in \{0, 1\}^k\}$.

If $S_{i,[1:j]}$ is the j-th prefix of sequence S_i, then each node in DP is a k-dimensional vector symbolizing the subproblem of finding a maximum trace for the subgraph of the alignment graph that is induced by the set of the respective prefixes $S_{i,[1:v_i]}$ of the k input sequences.

Each arc in A represents a possible column of the alignment realized by a trace. Moving from node $p = (p_1, \ldots, p_k)$ to node $q = (q_1, \ldots, q_k)$ using arc (p, q) means that the alignment of the prefixes induced by p is extended by a column c. The i-th entry of c consists of the next character of S_i if $q_i - p_i = 1$ or of the blank character if $q_i - p_i = 0$. Hence the set of all source-to-sink paths codes all possible alignments. The weight of an arc is the sum of the weights of the edges in the input alignment graph that are realized by the corresponding column and the weight of the alignment is the sum of the weights of the arcs on a source-to-sink path in DP.

Therefore finding a maximum trace corresponds to finding a longest source-to-sink path in DP. This can be accomplished by a breadth-first search (BFS) starting from the source node of the graph and visiting the nodes in lexicographical order. The search maintains a queue of nodes with the property that the length of a longest path is known from the source node to the node at the head of this queue. The generic step removes node v from the head of the queue, examines all arcs (v, w) leaving v, and updates the longest path information (length and predecessor) on w if the longest path to v followed by (v, w) is longer than the current longest path to w. Then w is added to the queue in lexicographical order, if it is not already queued.

The running time of this algorithms is easily analyzed. For each node $O(2^k - 1)$ arcs need to be checked in order to update the longest path information on the target nodes. For each arc the score function has to be computed which accounts to $s(k)$ time (*e.g.*, $O(k^2)$ for the WSOP scoring function). Since there are $N = \prod_i n_i$ nodes in V this yields an algorithm with worst case time complexity of $O(s(k) \cdot 2^k \cdot N)$ and space complexity $O(N)$, which is feasible only for very small problem instances.

However, the above algorithm does not need to construct DP before conducting the BFS. Rather the dynamic programming graph is constructed on the fly. Starting with nothing but the start node $s = (0, 0, \ldots, 0)$ all neighboring arcs and nodes are generated during the execution of the algorithm. If the input alignment graph is not complete this does not necessarily comprise all $2^k - 1$ possible nodes. Thus we can view this step as a *branch* step within a branch-and-bound framework (see also Section 6.1.2 for a general description of the branch-and-bound paradigm for integer linear programming). A bounding step can be added before putting a node w into the queue. Suppose we

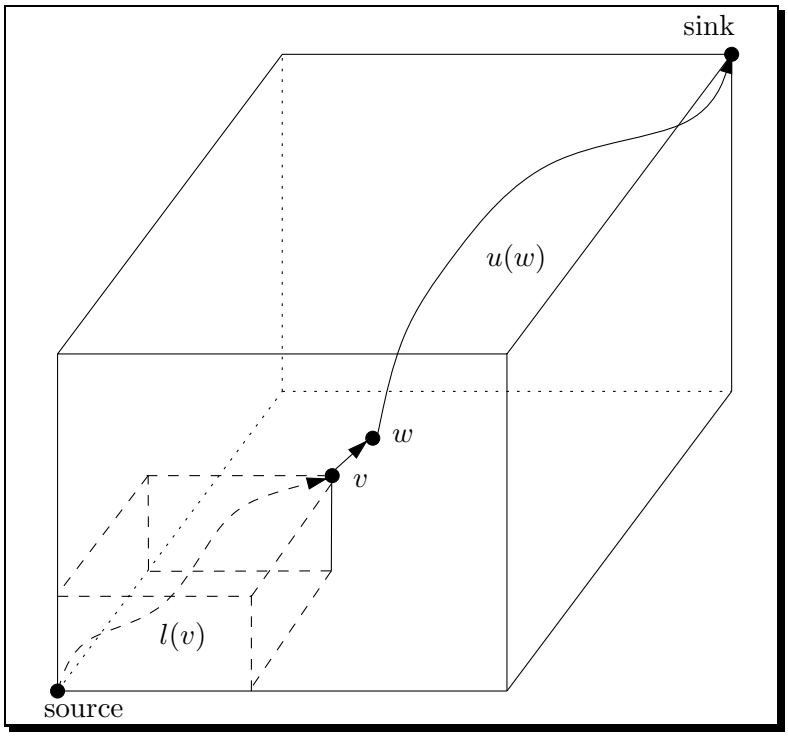

Figure 5.1: Branch and bound step in a three-dimensional DP graph

have a lower bound l on the length of the longest source to sink path, for example the score of multiple trace computed by some heuristic. If we have in addition an upper bound $u(w)$ on the length of the longest path from w to the sink, then we can avoid putting w into the queue if

$$l(v) + sc(v, w) + u(w) \leq l,$$

where $l(v)$ is the length of the longest path from the source to v and $sc(v, w)$ is the weight of arc (v, w) (see also Figure 5.1).

As mentioned above, only in the case of the complete k-partite graph the branch step has to consider k neighboring nodes. If the input alignment graph is sparse, generally fewer neighboring nodes have to be examined.

Kececioglu (1993) gave an elaborate procedure to further reduce the number of nodes to be examined. His method is based on the computation of a minimum cut which eliminates some nodes from consideration (Kececioglu 1999). This sparsification of the problem inherent in the input alignment graph speeds up the computation of the longest path in this dynamic programming algorithm.

5.2.2 The SMT Algorithm

Bafna, Muthukrishnan, and Ravi (1995) introduced a number of problem formulations for computing similarity between two annotated sequences (S_1, P_1) and (S_2, P_2), among them two variations of the SMT problem. In the first variation they align two structured sequences whereas in the second variation they infer the secondary structure of the first sequence to the second by allowing any Watson-Crick interaction in the annotation of the second sequence and then aligning it with the first structured sequence. Although they claim that their algorithms can be extended to handle pseudoknots in the secondary structure they do not explain how this could be done.

In their paper they gave two different recurrences for the two above mentioned variations of the problem with time bound of $O(n_1^2 \cdot n_2^2)$ in the case that P_1 and P_2 are both secondary structures and $O(n_1^2 \cdot n_2^2 + n_1 \cdot n_2^3)$ for the case that P_2 is no secondary structure.

We will show that a slight modification of their second recurrence leads to a dynamic programming algorithm with running time $O(n_1^2 \cdot n_2^2)$ for *all* variations of pairwise structural alignment, that means P_1 and P_2 may be any annotation and may contain pseudoknots.

Since they do not use a gap cost functions the two problems they address can be regarded as instances of the SMT problem. Since Bafna et al. presented their algorithm as an alignment algorithm we describe the modified recurrence in terms of structural alignments (see Chapter 3) rather than using the structural trace terminology (see Chapter 4).

They use a weighted sum of a sequence similarity score $sim : \hat{\Sigma}^2 \to \mathbb{R}$ and an interaction score $isim : \Sigma^4 \to \mathbb{R}$ as structural alignment score function, namely, in our notation, $rna : \mathcal{A}_s^2 \to \mathbb{R}$ with

$$rna(A_s) \mapsto \sum_{i=1}^{n} sim(a_{1,i}, a_{2,i}) + \sum_{\substack{(j,l)\in\hat{P}_1,(q,r)\in\hat{P}_2 \\ (q,r)=(j,l)}} isim(a_{1,j}, a_{1,l}, a_{2,q}, a_{2,r}).$$

Assume that we are given two structured sequences (S_1, P_1) and (S_2, P_2) together with the structural alignment score function rna. The goal is to compute an optimal structural alignment with respect to rna. We use a four-dimensional array A to record the value of solutions of subproblems. The entry $A[i_1, j_1, i_2, j_2]$ contains the score of an optimal structural alignment between the annotated sequences $(S_{1,[i_1:j_1]}, P_{1,[i_1:j_1]})$ and $(S_{2,[i_2:j_2]}, P_{2,[i_2:j_2]})$ where $P_{i,[j:k]}$ is the set of all interaction $(x, y) \in P$ with $j \leq x < y \leq k$. Hence $A[1, n_1, 1, n_2]$ contains the score of the optimal structural alignment between (S_1, P_1) and (S_2, P_2). The base cases for this recurrence are as

follows:

1. The score of aligning an empty infix of (S_1, P_1) with an empty infix of (S_2, P_2) is zero. Hence $A[i_1, j_1, i_2, j_2] = 0$ for all quadruples (i_1, j_1, i_2, j_2) with $j_1 < i_1$ and $j_2 < i_2$.

2. Aligning an empty infix of (S_1, P_1) with an nonempty infix of (S_2, P_2) yields as score the sum of all insertion scores for the infix of the second sequence, *i.e.*, $A[i_1, j_1, i_2, j_2] = \sum_{i_2 \leq k \leq j_2} sim(\text{'-'}, s_{2,k})$ for all quadruples (i_1, j_1, i_2, j_2) with $j_1 < i_1$ and $j_2 \geq i_2$.

3. Aligning an empty infix of (S_2, P_2) with an nonempty infix of (S_1, P_1) yields as score the sum of all deletion scores for the infix of the second sequence, *i.e.*, $A[i_1, j_1, i_2, j_2] = \sum_{i_1 \leq k \leq j_1} sim(s_{1,k}, \text{'-'})$ for all quadruples (i_1, j_1, i_2, j_2) with $j_2 < i_2$ and $j_1 \geq i_1$.

For the case that both infixes are nonempty, consider the following recurrence for all $1 \leq i_1 < j_1 \leq n_1$ and $1 \leq i_2 < j_2 \leq n_2$.

$$\begin{aligned}\hat{A}[i_1, j_1, i_2, j_2] = \max\big\{&\hat{A}[i_1, j_1 - 1, i_2, j_2] + sim(s_{1,j_1}, \text{'-'}),\\ &\hat{A}[i_1, j_1, i_2, j_2 - 1] + sim(\text{'-'}, s_{2,j_2}),\\ &\hat{A}[i_1, j_1 - 1, i_2, j_2 - 1] + sim(s_{1,j_1}, s_{2,j_2}),\\ &\hat{A}[i_1 + 1, j_1, i_2, j_2] + sim(s_{1,i_1}, \text{'-'}),\\ &\hat{A}[i_1, j_1, i_2 + 1, j_2] + sim(\text{'-'}, s_{2,i_2}),\\ &\hat{A}[i_1 + 1, j_1, i_2 + 1, j_2] + sim(s_{1,i_1}, s_{2,i_2})\big\}\end{aligned} \tag{5.1}$$

This recurrence describes all possible ways to optimally align the infixes $(S_{1,[i_1:j_1]}, P_{1,[i_1:j_1]})$ and $(S_{2,[i_2:j_2]}, P_{2,[i_2:j_2]})$ provided that there are no two interactions $(i_1, j_1) \in P_1$ and $(i_2, j_2) \in P_2$, and provided we know the score of the optimal structural alignments of all smaller infixes. The optimal alignment of the two infixes is obtained by extending an optimal alignment of smaller infixes either to the left or to the right with either a deletion, an insertion, or a match. For each of this cases we add the similarity score of the (mis)match or indel to the known optimal score of structurally aligning the smaller infixes.

However, if $(i_1, j_1) \in P_1$ and $(i_2, j_2) \in P_2$, then matching the two character pairs at the end of the infix yields an interaction match with positive score and hence may change the optimal structural alignment. This case is accounted for in the second part of the

recurrence.

$$\begin{aligned} A[i_1, j_1, i_2, j_2] = \max\{ & \hat{A}[i_1, j_1, i_2, j_2], \\ & A[i_1 + 1, j_1 - 1, i_2 + 1, j_2 - 1] \\ & + sim(s_{1,j_1}, s_{2,j_2}) \\ & + sim(s_{1,i_1}, s_{2,i_2}) \\ & + isim(s_{1,i_1}, s_{1,j_1}, s_{2,i_2}, s_{2,j_2})\} \end{aligned} \tag{5.2}$$

Filling the four-dimensional table requires time $O(n_1^2 \cdot n_2^2)$, because for each quadruple the maximum is taken over a constant number of terms. The dynamic programming table can be filled bottom up. The table entries are visited in lexicographical order of the length of the infixes, *i.e.*, we first compute the entries for all infixes of length one in the first sequence and all infixes of length one in the second sequence. Then the entries for all infixes of length one in the first sequence and all infixes of length two in the second sequence and so on. In contrast to the second recurrence of Bafna et al. our recurrence can cope with pseudoknots and is less involved.

Chapter 6

The Combinatorial Optimization Approach

In this chapter we describe in detail the polyhedral approach we take in order to develop efficient branch-and-cut algorithms for the GMT and SMT problem. In the previous chapter we have formulated both problems in terms of graphs, *i.e.*, as combinatorial optimization problems. We start this chapter by describing a general branch-and-cut framework in Section 6.1. Then we give specific details for the GMT problem in Section 6.2 and for the SMT problem in Section 6.3. Each of the two sections contains problem-specific details for the general branch-and-cut algorithm and is divided in four parts:

(1) Definition of the ILP and of the polytope (or polyhedron) P associated with the problem (Sections 6.2.1 and 6.3.1).

(2) Investigation of the facial structure of P; this yields a partial description (also called relaxation), defining a polytope $R \supseteq P$ (Sections 6.2.2 and 6.3.2).

(3) Devising separation routines over the relaxed polytope R (Sections 6.2.3 and 6.3.3).

(4) Presenting results of our implementations (Sections 6.2.4 and 6.3.4).

6.1 The Generic Branch-and-Cut Algorithm

In this section we describe the algorithmic technique we used to solve the GMT and SMT problem to optimality. This so-called *branch-and-cut* algorithm is a combination of the cutting plane technique and the well known branch-and-bound paradigm. While the enumerative frame involved basically stays the same for every combinatorial optimization problem, the main work of computing good lower and upper bounds is rather problem specific.

In Section 6.1.1 we outline the general *cutting plane algorithm* which yields upper bounds for a given combinatorial optimization problem like the SMT and GMT problem. If we are interested in an exact solution of a combinatorial optimization problem, generally a cutting plane approach will not be sufficient. Rather we have to combine the cutting plane approach with the branch-and-bound paradigm which is reviewed in Section 6.1.2. This combination is called *branch-and-cut* approach and is described in Section 6.1.3 in a general way.

6.1.1 A Cutting Plane Approach

Suppose we are given a combinatorial optimization problem P over a ground set of cardinality n (see also Chapter 2). Let P be described by the (0/1)-integer linear

programming problem $I = \max\{c^T x \text{ subject to } Ax \leq b,\ x \in \{0,1\}\}$. Also assume that we are given a heuristic solution to this problem with objective value l, which is hopefully not to far away from the optimal solution of P. One way to derive a statement about the quality of the heuristic solution is to compute an upper bound u on the optimal solution of P. Then we know that the heuristic solution is at most a fraction of $(u - l)/u$ away from the optimal solution of P. An upper bound can be computed using a *cutting plane algorithm*. In a cutting plane algorithm linear programming techniques are used to solve a sequence $P_0, P_1, \ldots, P_k$ of *relaxations* of P, where the solutions of these relaxations provide a series of non-increasing upper bounds to the original problem.

A relaxation of a combinatorial optimization problem P is another optimization problem Q, whose set of feasible solutions properly contains all feasible solutions of the original problem. The objective function of Q is an extension of the objective function of P. Hence, the value of the optimal solution of the relaxation of P is at least as high as the value of the optimal solution of the original problem. Consequently, solving a relaxation of P yields an upper bound for P itself.

Since we use linear programming techniques in a cutting plane algorithm, the relaxations of the original problem P are expressed in terms of systems of linear inequalities. They differ from each other in the sense that during the algorithm inequalities are added to the system describing the current relaxation. To be more specific, if we consider relaxation P_i described by $A_i x \leq b_i$, we add at least one or more inequalities to $A_i x \leq b_i$, which yields relaxation P_{i+1}. As noted above, the algorithm guarantees that the solutions of the relaxations provide a series of decreasing upper bounds for the original problem.

The inequalities we add stem from the solution of the *general separation problem* which is defined as follows:

The general separation problem

Given a class of valid inequalities for a combinatorial optimization problem,
and a vector $y \in \mathbb{Q}^n$, either prove that y satisfies all inequalities of this class,
or find an inequality of this class which is violated by y.

The solution of the general separation problem gives the cutting plane algorithm its name, because it "cuts" off the current optimal solution which is not feasible for the original ILP I, or proves that the solution is feasible for the examined class of inequalities.

The hyperplane defined by the set of points that fulfills the separating inequality with equality is therefore called *cutting plane*. An algorithm which solves the general separation problem is called *exact separation algorithm* in contrast to a *heuristic separation*

algorithm, which may find violated inequalities, but if it fails, it is not guaranteed that no inequality of the class is violated.

If we want to apply a cutting plane algorithm to a combinatorial optimization problem P we generally face two problems, each of which can be circumvented by using appropriate relaxations of the ILP I that describes P.

1. Solving a (0/1)-ILP is NP-hard, meaning that no efficient algorithm for solving an ILP is known and that it is very unlikely that such an algorithm exists. Therefore we relax the ILP I by dropping the integrality constraint and adding the two trivial inequalities $x_i \geq 0$ and $x_i \leq 1$ for each variable x_i. The resulting linear programming problem $L = \max\{c^T x \text{ subject to } Ax \leq b,\ 0 \leq x_i \leq 1\}$ can be solved efficiently, that means in polynomial time in the size of the LP.

2. Unfortunately, the linear program L can contain an exponential number of inequalities in its constraint matrix. In this case solving L would take polynomial time in the size of L but result in an exponential algorithm in the size of the original problem. We circumvent this by further relaxing L. We start with an initial linear program L' containing only a polynomial number of constraints. L' can then be solved in time polynomial in the size of the original problem. (Note that this relaxation even is applied in cases when a class of inequalities does not contain an exponential, but a large number of inequalities.)

Applying these two relaxations to the original problem, we start the cutting plane algorithm with the linear program L' containing a polynomial number of inequalities in its constraint matrix.

Let x' be the optimal solution of L'. Since the system of inequalities does not contain all inequalities from the original system $Ax \leq b$, we first have to solve the general separation problem for the classes of inequalities occurring in the original ILP description of the problem. We call this set of inequalities *ILP inequalities*.

If we find a violated inequality from $Ax \leq b$, we add it to the current set of inequalities and solve the problem again. Note that we need to employ *exact* separation algorithms, no matter whether x' is integral or fractional. Otherwise we would not know whether an integral solution is optimal or whether only the heuristic separation routine failed to identify a violated inequality.

If the exact separation routine confirms that no ILP inequalities are violated we have to consider two cases.

1. x' is integral.
   Then x' is an optimal solution to the original problem, because no ILP inequality

is violated and because x' is the optimal solution of a relaxation of the original problem, yielding an upper bound on the value of an optimal solution of the original problem.

2. x' is fractional.
   In this case we have to keep on solving the separation problem for other classes of valid inequalities. At this point we can employ heuristic separation methods, as long as we use exact methods for the ILP inequalities.

We now rise the question how further cutting planes can be found, if the optimal solution of the current LP is not integral and no ILP inequality is violated.

One way is to consider classes of cutting planes that can be applied to any combinatorial optimization problem, such as, for example, *Gomory cuts* (see Gomory 1958). We call such cutting planes *general purpose cutting planes*, because they are not problem specific and can be employed to solve any combinatorial optimization problem. Although there has been intensive research in finding good general purpose cutting planes, they seem to be of limited use for solving combinatorial optimization problems. Successful computational work relies on specific cutting planes for a particular problem.

A very successful method for deriving problem specific cutting planes is the investigation of the polytope associated with the combinatorial optimization problem under consideration. Especially the facet-defining inequalities are useful, because they are valid for the integer linear programming formulation and are not dominated by any other valid inequality. Hence, they are in some sense the best cutting planes. In general, however, it is not simple to identify classes of facet-defining inequalities for the problem polytope (see Chapter 6 for facet-defining inequalities for the SMT and GMT polytope). Even if one has identified such a class, this is algorithmically only useful if their separation problem can be solved exactly or at least be attacked with heuristic separation routines.

Cutting plane algorithms using problem specific cutting planes, *e.g.*, facet-defining inequalities, often have to stop without finding an optimal solution. This can have two different reasons. Firstly, for no NP-hard combinatorial optimization problem a complete linear description is known. Secondly, even if a big class of facets is known, no efficient algorithm may be available for the solution of the exact separation problem for this class. Nevertheless, large instances of NP-hard combinatorial optimization problems can be solved with the help of facet-defining cutting planes in combination with the branch-and-bound paradigm.

6.1.2 Branch-and-Bound

The cutting plane approach outlined so far does not necessarily solve a problem instance to optimality for various reasons discussed above. We can end up in a solution which is not a feasible incidence vector of the combinatorial optimization problem. If we then are not able to find a cutting plane that would allow us to create a new relaxation we can employ another basic algorithmic paradigm for solving hard combinatorial problems, *branch-and-bound*.

Branch-and-bound is a divide-and-conquer approach trying to solve the original problem by dividing it into smaller problems for which lower and upper bounds are computed. This gives rise to an enumeration tree of subproblems. In the context of (0/1)-integer linear programs the root of this tree corresponds to the original problem and each internal node has two children representing the two subproblems generated by setting one variable to zero in the one subproblem and to one in the other subproblem. Each path from the root to another node in the tree exactly describes the constraints

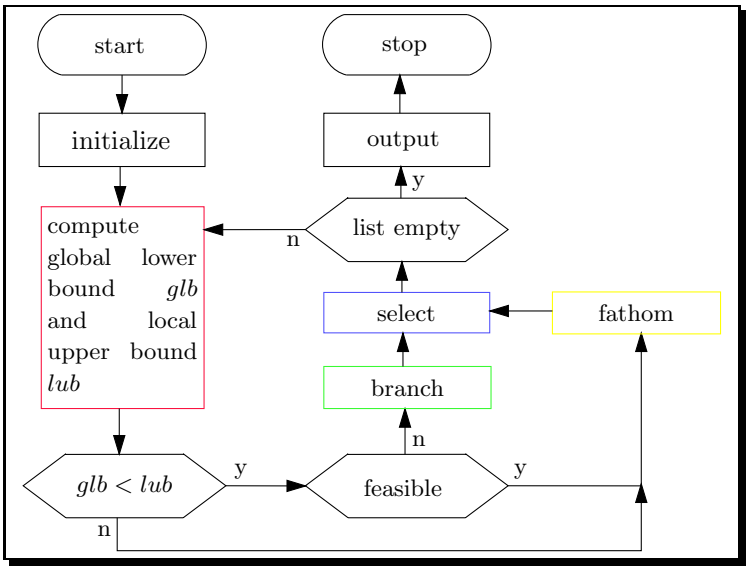

Figure 6.1: Flowchart of a branch-and-bound algorithm.

on the subproblem, namely the variables which are set and the values to which they are set during a branch step. The crucial part of a successful branch-and-bound algorithm is the computation of upper bounds for these subproblems. We call an upper bound *lub local*, if it is computed for a subproblem in the enumeration tree. If the solution found for the subproblem happens to be feasible for the original problem (*i.e.*, there are no contradictions to the variable settings imposed by the path in the enumeration tree leading to the node of the subproblem) and has a higher objective value than any

feasible solution found so far, it is memorized and the *global lower bound* (*glb*) for the objective function is increased accordingly.

The whole algorithm is depicted in Figure 6.1 on the facing page. A branch-and-bound algorithm maintains a list of subproblems of the original problem, which is **initialized** with the original problem itself. In each major iteration step, the algorithm **selects** a subproblem from this list, computes a local upper bound for this problem and tries to improve the global lower bound. If the local upper bound does not exceed the global lower bound, the current subproblem is **fathomed**, because its solution cannot possibly be better than the best known feasible solution.

If the local upper bound exceeds the global lower bound and no feasible solution was found for the current problem, we perform a **branching** step by dividing the current problem into two subproblems by selecting one *branching variable* which is set to zero in the one subproblem and to one in the other.

If the **list** of subproblems becomes **empty**, then the memorized feasible solution whose objective function value is equal to the global upper bound can be **output** as the optimum solution.

6.1.3 Branch-and-Cut

We will now discuss the general branch-and-cut algorithm which is the combination of the cutting plane approach and the branch-and-bound paradigm, both explained above.

Its main difference from the classical branch-and-bound method is the use of LP relaxations and the employment of problem specific cutting planes at every node of the enumeration tree. This feature incurs several technical details that make the design and implementation of a branch-and-cut algorithm a nontrivial task.

Figure 6.2 on the next page shows a flowchart of a branch-and-cut algorithm. The four dashed boxes resemble the corresponding boxes of the normal branch-and-bound algorithm depicted in Figure 6.1 on the facing page. They basically can be identified with the four rectangular boxes in Figure 6.1 that stand for **bounding, branching, selecting** and **fathoming**. The most important part is the problem-specific bounding which we will describe in more detail in Sections 6.3.3 and 6.2.3 for the SMT and GMT problem respectively. The main difference to a branch-and-bound algorithm lies in the bounding part. Here cutting planes are iteratively added to the relaxed program thereby making the current solution infeasible and (normally) improving the upper bound for the problem. If no further cutting planes can be found the bounding part is left and the execution of the branch-and-bound algorithm continues.

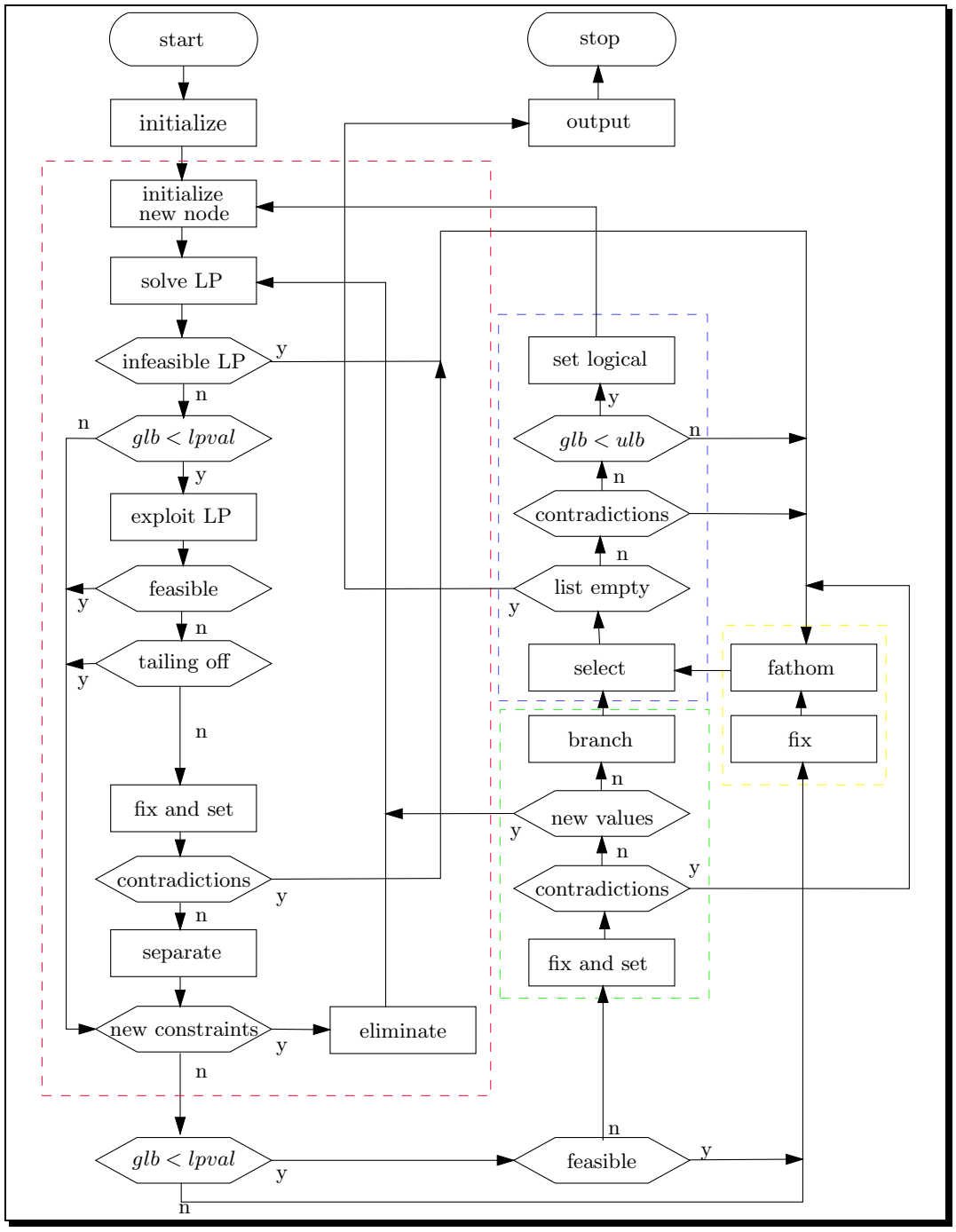

Figure 6.2: Flowchart of a branch-and-cut algorithm.

Already a short inspection of Figure 6.2 shows that this explanation is a gross simplification of a branch-and-cut algorithm. Nevertheless it describes the main idea. In the

next part we describe the additional details of the branch-and-cut algorithm depicted in Figure 6.2. The reader who is content with the simplified explanation above may skip this part. It is intended to reflect the complexity of the general branch-and-cut-algorithm without eluding technicalities too much. A very profound treatment of this topic can be found in Jünger, Reinelt, and Thienel (1995b).

Let *lpval* be the value of the current relaxed LP and *glb* be the current global lower bound of the problem. In the **bounding** (or cutting plane) part (the leftmost dashed box in Figure 6.2 on the preceding page), there is an additional step after the first test of $glb < lpval$ called **exploit LP**. In this step the lower bounding of Figure 6.1 on page 72 is done. Before proceeding to the computation of new upper bounds (called **separate**) the current fractional solution is exploited to improve the lower bound. This is done by trying to generate a feasible (0/1) solution, either by rounding fractional variables or by employing other problem specific procedures.

If we succeed in finding a new feasible solution we leave the bounding part. If not, we check whether we have made a significant decrease in *lpval* during the last few cutting plane iterations. If not, it is reasonable to abort the cutting plane generation. This strategy is called **tailing off**, where the decision to abort is made according to two parameters k and p. If, during the last k iterations in the bounding part, *lpval* did not decrease at least by p percent, then new subproblems are generated instead of generating further cutting planes. Good choices for the parameters p and k are rather problem specific and dependent on the quality of the available cutting plane generation procedures. Finally, before entering the **separation**, the values of some variables can be **fixed** to zero or one, according to their reduced costs in the current solution. For details we again refer to Jünger, Reinelt, and Thienel (1995b). Once we have fixed some variables by reduced costs, it may occur that we can fix even more variables by **logical implications**, that means, for example, if one alignment edge variable in the GMT is fixed to one by its reduced costs we can fix all conflicting alignment edge variables to zero. We can also **set** variables using the same criteria as for fixing. However, in contrast to fixing, the setting is only locally valid, *i.e.*, only for the current branch-and-cut node and its descendants. Fixing and setting of variables may result in contradictions to variables previously set or fixed. If such contradictions are detected the current node is **fathomed**.

The upper bounding is done in the part **separate**. If we find here new cutting planes we add them to the current LP. However, this could result in a large number of inequalities. As a remedy for this, the step **eliminate** deletes inequalities that are no longer needed according to some specific criterion. One criterion could be, for example, that the inequality is non-binding in the current solution.

This concludes the bounding part. In the **branching** part (the lower middle dashed

box in Figure 6.2 on page 74) some variable is chosen as the *branching variable* and two new branch-and-cut nodes are created and added to the set of *active* branch-and-cut nodes. An active branch-and-cut node is a leaf in the enumeration tree that is not yet fathomed. In the first node the variable is set to one, in the second to zero. If one can separate the ILP inequalities exactly, then there is at least one fractional variable which is a reasonable candidate for a branching variable. There are different strategies of choosing among fractional variables, for example,

- Choose the variable closest to 0.5.
- Choose the variable closest to 0.5 and among those the one with biggest objective function coefficient.
- Choose the fractional variable with the biggest objective function coefficient.

In the **selecting** part a branch-and-cut node is selected (if possible) and removed from the list of active branch-and-cut nodes. If the list of active branch-and-cut nodes is empty, the best known feasible solution is also the optimal solution. Otherwise the selected node is processed. After a selection the set variables (including the branching variables) must be adjusted. If it turns out that some variable must be set to 0 or 1, yet has been fixed to the opposite value in the meantime, we have a contradiction and hence fathom the current node. If the local upper bound *lub* of the selected node does not exceed the global lower bound *glb*, the node is fathomed immediately and the selection process is continued. Similar to the branching part there are different strategies of how to select a branch-and-cut node.

- In the case of *depth-first* search a branch-and-cut node with maximum depth in the branch-and-cut tree is selected.
- In the case of *breadth-first* search a branch-and-cut node with minimum depth in the branch-and-cut tree is selected.
- In the case of *best-first* search the most "promising" branch-and-cut node is selected, *i.e.*, in the case of a maximization problem, the node with the maximum local upper bound.

Finally, the last dashed box describes the **fathoming** part. A branch-and-cut node does not need to be considered any longer as soon as one of the following applies:

- the objective value of the current best feasible solution *glb* is at least the value of the (local) upper bound obtained in this branch-and-cut node,

- the branch-and-cut node is infeasible, that means, the linear program does not contain a point in the solution space,

- a contradiction occurs, for example, we have to fix a variable to one which has been set to zero before.

6.2 The GMT Problem

6.2.1 A Characterization of the GMT Problem as ILP

Recall the definition of the GMT problem on page 60. The input consists of an EAG $G = (V, E, H)$, a partition D of the alignments edges into blocks, and a block trace score function that assigns a positive weight w_d to each block $d \in D$. The goal is to find the set of blocks with maximum overall weight. Note that the only conditions on D are (1) every $d \in D$ is a trace between a pair of sequences, *i.e.*, it does not contain two conflicting edges, and (2) D is a partition, *i.e.*, any edge in E is contained in exactly one block of D. A feasible set over D is a set M of blocks such that the union $U = \bigcup_{d \in M} d$ does not induce a mixed cycle on G. In that case U is a trace according to Theorem 4.1.1. Let $\mathcal{T} := \{M \subset D \mid \bigcup_{d \in M} d \text{ is a trace}\}$ be the set of all feasible solutions. We define the *GMT polytope* as the convex hull of all incidence vectors of D that are feasible, *i.e.*,

$$P_{\mathcal{T}}(G) := conv\{\chi^M \in \{0,1\}^{|D|} \mid M \in \mathcal{T}\},$$

where the *incidence vector* χ^F for a subset $F \subseteq D$ is defined by setting $\chi^F_d = 1$ if $d \in F$ and setting $\chi^F_d = 0$ if $d \notin F$. For reasons of clarification we speak in the singleton case also of the *MT polytope*.

Recall the definition of the surjective function $v : E \cup H \to D \cup \{\emptyset\}$ (see also on page 54). The function v maps each edge $e \in E$ to the block $d \in D$ in which e is contained.

It is now easy to formulate GMT as an integer linear program. For every $d \in D$ we have a binary variable $x_d \in \{0, 1\}$ indicating whether d is in the solution or not. In view of Theorem 4.1.1 the GMT-problem

$$\max \sum_{d \in D} w_d \cdot x_d \text{ subject to } x \in P_{\mathcal{T}}(G)$$

is equivalent to

$$\begin{aligned} \text{maximize} \quad & \sum_{d \in D} w_d \cdot x_d \\ \text{subject to} \quad & \sum_{d \in v(C)} x_d \leq |v(C)| - 1, \ \forall \text{ critical mixed cycles } C \text{ in } G \\ & x_d \in \{0,1\}, \ \forall d \in D \end{aligned} \tag{6.1}$$

We call the inequalities in Equation (6.1) *mixed-cycle inequalities*. All mixed cycle inequalities are valid inequalities for the GMT polytope of G. In Section 6.2.2 we derive conditions under which they are facet-defining.

6.2.2 The Structure of the GMT Polytope

In this section we investigate the structure of the GMT polytope. First we consider the case of two sequences and then the case of multiple sequences. Recall that D is the set of all blocks that might be realized by an alignment. A subset of D is called *feasible* if the alignment edges in the block do not induce a mixed cycle in the input EAG.

Since every subset of a feasible set of blocks is also feasible and the empty set is feasible as well, the pair $I_{\mathcal{T}}(G) = (D, \mathcal{T})$ forms an independence system on D. We call the set of blocks which have non-zero coefficients in an inequality $c^T x \leq c_0$ the *support* of the inequality. According to the definition of circuits in an independence system we observe the following:

Observation 1: Let R be any critical mixed cycle in an extended alignment graph $G = (V, E, H)$. Then the incidence vectors of $v(R)$ form a circuit of the independence system $I_{\mathcal{T}}(G)$.

Lemma 6.2.1: Let $G = (V, E, H)$ be an extended alignment graph. $P_{\mathcal{T}}(G)$ is full-dimensional and the inequalities $x_d \geq 0$, $d \in D$ are facet-defining for $P_{\mathcal{T}}(G)$. Further let d be any block in D. Then the inequality $x_d \leq 1$ is facet-defining iff no edge of d is in conflict with another edge.

Proof. Since every $d \in D$ is independent, it follows by Theorem 2.7.1 that $P_{\mathcal{T}}(G)$ is full-dimensional. From Theorem 2.7.2 follows that the inequality $x_d \geq 0$ is facet-defining for $P_{\mathcal{T}}(G)$ for all $d \in D$.

To prove the last statement, let us assume that no edge of d is in conflict with another edge. Then $A = \{\{s, d\} \subseteq D \mid s \in D \setminus d\} \cup \{\{d\}\}$ is a set of $|D|$ many feasible solutions whose incidence vectors are affinely independent. According to Theorem 2.3.2 on page 21 the inequality $x_d \leq 1$ defines a facet of $P_{\mathcal{T}}(G)$.

On the other hand, assume that there is an edge $e \in d$ in conflict with another edge f. Then each incidence vector χ^M of a feasible solution $M \subseteq D$ satisfying $\chi^M_d = 1$ has to satisfy $\chi^M_{v(f)} = 0$, so $\dim\{x \in P_{\mathcal{T}}(G) \mid x_d = 1\} \leq |D| - 2$. Thus $x_d \leq 1$ is not facet-defining. □

We call the inequalities defined in the lemma above the *trivial* inequalities for the GMT problem.

For two sequences all circuits of $I_{\mathcal{T}}(G)$ (recall that $I_{\mathcal{T}}(G) = (D, \mathcal{T})$) are of cardinality two because a critical mixed cycle visits every sequence at most once (see also Theorem 4.1.1 on page 52). Hence, the independence system is 2-regular, which means that in a clique of $I_{\mathcal{T}}$ each pair of blocks contains edges e_1, e_2 with $e_1 \prec e_2$. Theorem 2.7.3 implies that the inequalities

$$\sum_{d \in C} x_d \leq 1, \quad C \text{ is a maximal clique of } I_{\mathcal{T}}(G)$$

are facet-defining for $P_{\mathcal{T}}(G)$. We call these inequalities *clique inequalities*.

It is known that the two-sequence case of MT problem can be reduced to the problem of computing the heaviest increasing subsequence of an integer sequence. Therefore the question arises whether the trivial and clique inequalities already give a complete description of the (G)MT polytope. In fact it can be shown that for the MT the clique inequalities together with the trivial inequalities build a complete description of the MT polytope.

To prove this, we need a more intuitive understanding of cliques in the independence system $I_{\mathcal{T}}(G)$. Observe that (V, E) is a subgraph of the complete bipartite graph $K_{p,q}$ with nodes $x_1, \ldots, x_p$ and $y_1, \ldots, y_q$.

Definition 6.2.1: Let $PG(K_{p,q})$ be the $p \times q$ directed grid graph, such that the arcs go from right to left and from bottom to top. Row r, $1 \leq r \leq p$ of $PG(K_{p,q})$ contains q nodes which correspond from left to right to the q edges that go between node x_r and node $y_1, \ldots, y_q$ in $K_{p,q}$. We call $PG(K_{p,q})$ the pairgraph of $K_{p,q}$ (see Figure 6.3 on the following page) and a node in the pairgraph *essential* if it corresponds to an edge in E.

The graph $PG(K_{p,q})$ has exactly one source and one sink and there is a path from node n_2 to node n_1 in $PG(K_{p,q})$ iff $e_1 \prec e_2$ for the corresponding edges e_1, e_2 in $K_{p,q}$. For example in Figure 6.3, e_3 is the source and $e_5 \prec e_3$, because there is a path from e_3 to e_5. Recall that the two singleton sets $\{e_1\}$ and $\{e_2\}$ form a circuit in $I_{\mathcal{T}}(G)$ iff e_1 and e_2 are in conflict, *i.e.*, if either $e_1 \prec e_2$ or $e_2 \prec e_1$.

Lemma 6.2.2: Let $p = n_1, \ldots, n_{p+q}$ be a source-to-sink path (n_1 is the source) in $PG(K_{p,q})$ and let $e_1, \ldots, e_l$, $l \leq p + q$ be the edges in E that correspond to essential

nodes in p. Then $C := \{\{e_1\}, \ldots, \{e_l\}\}$ is a clique of $I_{\mathcal{T}}(G)$ if $|C| \geq 2$. Moreover, every maximal clique of $I_{\mathcal{T}}(G)$ is represented by a source-to-sink path in $PG(K_{p,q})$.

Proof. For any two nodes n_i and n_j in p with $i > j$ the corresponding edges e_i and e_j are in relation $e_i \prec e_j$ and hence $\{e_i\}$ and $\{e_j\}$ form a circuit of $I_{\mathcal{T}}(G)$. Thus $\{\{e_1\}, \ldots, \{e_l\}\}$ is a clique of $I_{\mathcal{T}}(G)$. Conversely, the edges in the singleton sets of any clique $C = \{\{e_1\}, \ldots, \{e_l\}\}$ of $I_{\mathcal{T}}(G)$ can be totally ordered with respect to $\prec$ because $\prec$ is transitive and the edges in any two singleton sets of C are in relation $\prec$. We assume w.l.o.g. $e_1 \prec e_2 \prec \cdots \prec e_l$. As noted above that means that there exists a path from n_i to n_{i+1} for $1 \leq i < l$. This implies the existence of a source-to-sink path containing the essential nodes $n_1, \ldots, n_l$. This path cannot contain another essential node, because otherwise C would not be maximal. □

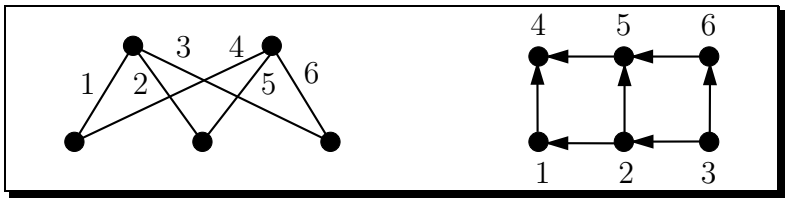

Figure 6.3: The $K_{2,3}$ and the corresponding pairgraph.

The pairgraph is a powerful data structure. It represents $\frac{(2n-2)!}{(n-1)!^2} = \Omega(2^n)$ clique inequalities where n is the number of edges in $K_{p,q}$. If G is sparse, however, it is unnecessary to store nonessential nodes. In this case the space consumption can be reduced using a *sparse pairgraph*. In a sparse pairgraph there are only essential nodes and paths consisting of nonessential nodes are replaced by arcs (see also Figure 6.4). In practice, the space consumption of a sparse pairgraph is linear in the number of edges in G, although there are examples, in which the sparse pairgraph needs more space, because of a high number of arcs.

To prove the integrality of the MT polytope we could prove that the constraint matrix formed by the trivial and clique inequalities is totally unimodular. Unfortunately this is not the case. We can construct a counterexample in which we can identify a set of columns in the constraint matrix with the following property: There is no partition of the set in sets S_+ and S_- such that the sum of the column vectors in S_+ minus the sum of the column vectors in S_- yields a vector with entries 1,0 or -1. According to Theorem 2.5.3 on page 24 this cannot be for a totally unimodular matrix.

Lemma 6.2.3: There are instances of the MT problem for two sequences such that the constraint matrix formed by the trivial and clique inequalities is not totally unimodular.

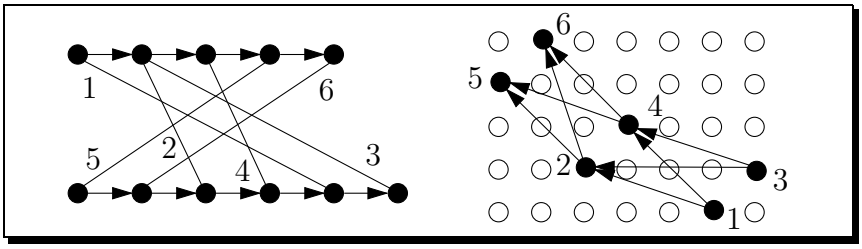

Figure 6.4: Instance of MT with corresponding sparse pairgraph (in black) that has no totally unimodular constraint matrix (white nodes are nonessential).

Proof. The proof is conducted by exhibiting an instance of the MT problem which gives rise to a constraint matrix that is not totally unimodular. Figure 6.4 shows an instance of MT and the corresponding sparse pairgraph. It is easy to verify that the matrix C in Figure 6.5 gives the coefficients for all clique inequalities: If we choose the columns 2, 3 and 6 there are exactly three ways to partition them w.l.o.g. into S_+ and S_-, namely

1. $S_+ = \{2\}$ and $S_- = \{3, 6\}$.
2. $S_+ = \{3\}$ and $S_- = \{2, 6\}$.
3. $S_+ = \{6\}$ and $S_- = \{2, 3\}$.

Subtracting the vectors in S_- from the vector in S_+ yields the following three vectors:

1. $(1, 0, 0, -1, 0, -1, -1, -2)^T$
2. $(-1, -2, 0, -1, 0, -1, 1, 0)^T$
3. $(-1, 0, 0, 1, -2, -1, -1, 0)^T$

Each of these vectors has an entry different from 0,1 and -1. According to Theorem 2.5.3 this is not possible for a totally unimodular matrix. □

Deprived of this convenient way of showing that the trivial and cliques inequalities form a complete description of the MT polytope we try a more direct way by using the pairgraph in a constructive proof.

Theorem 6.2.1: In the two-sequence case of the MT problem the trivial and the clique inequalities together form a complete description of the MT polytope.

Proof. Let P be the polytope defined by the trivial and the clique inequalities. Then certainly $P_T(G) \subseteq P$. If we could prove that P is integral, *i.e.*, has only integral vertices,

$$C = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

Figure 6.5: Coefficient matrix C.

we would have equality since a full-dimensional polytope has – up to a multiplicative factor – a unique description.

Lemma 6.2.4: P is integral.

Proof. Assume that P has a fractional vertex $\hat{x}$. Let w be a vector of weights such that $\hat{x}$ is the unique optimum solution of $\max\{w^T x \mid x \in P\}$; any w lying in the cone generated by supporting hyperplanes of $\hat{x}$ is suitable.

Assign to each node n_e in $PG(K_{p,q})$ that corresponds to an edge e in E the value $\hat{x}_{\{e\}}$ of the singleton set $\{e\}$ and assign zero to all other nodes. Now let PG' be the subgraph of $PG(K_{p,q})$ that consists of tight paths, *i.e.*, the subgraph that is induced by the edges that are contained in some source-to-sink path, where the values of the nodes on that path sum up exactly to one. Note that such a tight path exists, because otherwise $\hat{x}$ would not be optimal. Moreover, all paths in PG' are tight because for any node n_e in PG' holds that all paths from the source to n_e have the same value and all paths from n_e to the sink have the same value. This follows, because otherwise there would exist at least one source-to-sink path that has a value greater than one. This in turn would imply a violated clique inequality which would contradict the feasibility of $\hat{x}$.

Let s be the source of PG'. We construct node sets of PG', such that every source-to-sink path goes exactly once through each node set. Let C_1 be the set of nodes with nonzero value such that the nodes in C_1 are the first nodes with nonzero value on a source-to-sink path. Such a set exists, as we have only tight paths in PG'. Let m be the minimal value of the nodes in C_1. Clearly $m < 1$, because we assume a fractional solution. Let $M \subseteq C_1$ be the set of all nodes of C_1 with value m. Further let $N(M)$ be the set of the first nodes with nonzero value reachable from M and let $C_2 = (C_1 \setminus M) \cup N(M)$. This leads to the following observations (see also Figure 6.6):

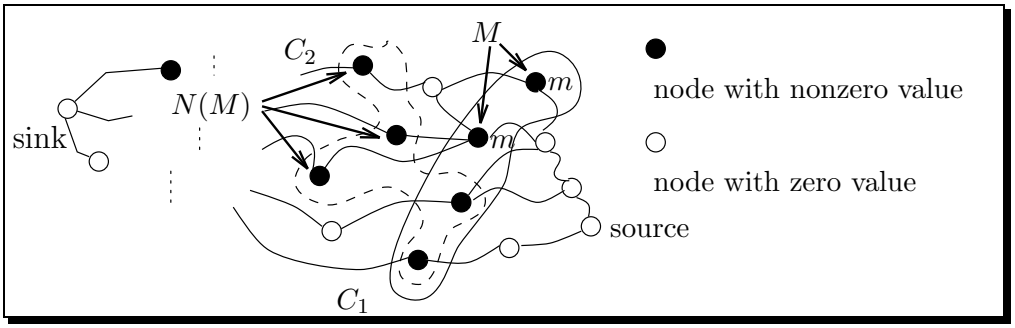

Figure 6.6: $PG'(K_{p,q})$ with the node sets C_1 and C_2.

1. There is no arc from n_e to n_f between any two nodes $n_e, n_f \in C_1$. Otherwise there would be two paths with different value from s to n_f, one with value $x_{\{f\}}$ and one with value $x_{\{f\}} + x_{\{e\}}$ which is impossible.

2. There is no arc from n_e to n_f between any two nodes $n_e, n_f \in N(M)$. Otherwise there would be two paths with different values from s to n_f, namely one with value $m + x_{\{f\}}$ and one with value $m + x_{\{f\}} + x_{\{e\}}$.

3. The nodes in $C_1 \setminus M$ cannot have an edge to a node in $N(M)$. Again, this would result in two paths of different value from the source to an edge in $N(M)$.

From the above observations it follows that every source-to-sink path visits C_1 and C_2 exactly once. Define $S_1 = \sum_{\{e|n_e \in M\}} w_{\{e\}}$ and $S_2 = \sum_{\{e|n_e \in N(M)\}} w_{\{e\}}$. Here $w_{\{e\}}$ is the weight (in the weight vector w) of the singleton set $\{e\}$. Assume $S_1 \leq S_2$. We then decrease the value of the nodes in M by m and increase the value of the nodes in $N(M)$ by this amount. Then all tight paths are still tight, as by our invariant every tight path goes once through C_1 and once through C_2. However, we have a new fractional solution which achieves at least the optimum weight. This is a contradiction to the assumption that we have a unique optimal solution. Therefore the solution must be integral. The case $S_1 > S_2$ can be handled analogously by increasing the value in M and decreasing the value in $N(M)$ by m', the minimum value of nodes in $N(M)$. □

The proof of the integrality of P concludes the proof of Theorem 6.2.1. □

The above lemma can also be proved in a different way which we sketch briefly. Pevzner and Waterman (1993) showed that for two conjugate partial orders $\sqsubset$ and $\sqsubset^*$ over a set Q and a weight function w over Q the length of a heaviest $\sqsubset$-sequence equals the size of a minimum cover of w by $\sqsubset^*$ sequences. Here two partial orders $\sqsubset$ and $\sqsubset^*$ are

called conjugate, if for any two $q_1, q_2 \in Q$ holds:

$$p_1 \text{ and } p_2 \text{ are } \sqsubset\text{-comparable} \Leftrightarrow p_1 \text{ and } p_2 \text{ are } \sqsubset^*\text{-incomparable.}$$

A family $\mathcal{C} = \{C\}$ of subsets of Q is called a *cover* of a function w if $\forall q \in Q$ there exist at least $w(q)$ subsets in $\mathcal{C}$ containing q. The proof is conducted by an easy application of Dilworth's theorem.

Since the relation $\prec$ which we defined on the alignment edges E is a partial order, we can naturally define a conjugate partial order $\prec^*$ which sets two edges in relation if they are not in relation $\prec$. Then it follows that the weight of a maximum trace equals the size of a minimal clique cover for the trace score function. In other words, if $Ax \leq b$ is the system of the clique inequalities from the lemma above, then for each integral vector c holds $\max\{cx \mid Ax \leq b\} = \min\{y^T b \mid y^T A = c, y \geq 0\}$. The equality of the solution of the primal and dual problem for each integral c implies that $Ax \leq b$ is *totally dual integral*. Together with the integrality of b this implies that P is integral (see Theorem 2.5.4 on page 25).

It is not clear whether the clique inequalities and the trivial inequalities always form a complete description of the GMT polytope.

We now switch to the case of multiple sequences. For more than three sequences Kececioglu (1991) showed that the MT is NP-hard. Hence, we cannot expect to find a complete description of the GMT polytope in this case.

First we will show that the facet-defining inequalities of the two-sequence case of the GMT are also facet-defining in the multiple-sequence case. If an inequality is facet-defining for a polytope P_1 associated with some subgraph G_1 of the EAG, then it is still facet-defining for a polytope P_2, if the EAG G_2 associated with P_2 is augmented only by edges that do not induce a mixed cycle with edges in G_1. An application of the lifting theorem (see Theorem 2.7.4) yields that the coefficients of all blocks whose edges do not induce a mixed cycle with the edges in G_1 are zero. This reads formally as follows:

Lemma 6.2.5: (Zero lifting) Let $G = (V, E, H)$ be an extended alignment graph, $U \subseteq D$ and $c^T x \leq c_0$ be a facet-defining inequality for $P_{\mathcal{T}}(G[E \setminus \bigcup_{s \in U} s])$ (where $G[A]$ with $A \subseteq E$ is the subgraph of G induced by A). Choose any $d \in U$ whose edges do not induce a mixed cycle with an edge in the support of $c^T x \leq c_0$. Then $c^T x \leq c_0$ defines a facet of $P_{\mathcal{T}}(G[(E \setminus \bigcup_{s \in U} s) \cup d])$.

If we apply Lemma 6.2.5 to the clique inequalities, we get the following theorem:

Theorem 6.2.2: Let $G = (V, E, H)$ be the extended alignment graph for $k > 2$ sequences and D be a partition into blocks. Let $D_{i,j}$ be the set of blocks between sequences S_i and S_j, and let $P_{\mathcal{T}}(G_{i,j})$ be the GMT polytope for the subgraph

$G_{i,j} = (V_{i,j}, E_{i,j}, H_{i,j})$ induced by the edges in $\bigcup d$, $d \in D_{i,j}$. Then every facet-defining inequality of $P_{\mathcal{T}}(G_{i,j})$ is also facet-defining for $P_{\mathcal{T}}(G)$.

Proof. Lemma 6.2.5 implies that a facet-defining inequality $c^T x \leq c_0$ is also facet-defining for $P_{\mathcal{T}}(G)$, because no edge in a block in $D \setminus D_{i,j}$ can form a mixed cycle with an edge in the support of $c^T x \leq c_0$. □

We now turn our attention to the next class of inequalities, the mixed cycle inequalities. This is an important class of inequalities, because they appear in the formulation of the problem as an integer linear program. We need some more notation.

Definition 6.2.2: Let C be a critical mixed cycle in an extended alignment graph. We call an edge $e = \{v, w\} \in E$ a *chord* of C if $C_1 \cup \{e\}$ and $C_2 \cup \{e\}$ are critical mixed cycles where C_1 and C_2 are obtained by splitting C at v and w.

For reasons of convenience we write $x(F) = \sum_{f \in F} x_f$.

Lemma 6.2.6: Let $G = (V, E, H)$ be an extended alignment graph, D be a partition into blocks and C be a critical mixed cycle of size ℓ. Then the inequality

$$x(v(C)) \leq \ell - 1$$

defines a facet of $P_{\mathcal{T}}(G)$ if and only if C has no chord.

Proof. Assume that C is a critical mixed cycle of size ℓ without a chord. Let $e_1, \ldots, e_\ell$ be the ℓ edges on C. Note that by definition of D, $v(e_1) \neq v(e_2) \neq \cdots \neq v(e_\ell)$. We obtain ℓ different feasible solutions by taking only the edges in $v(C)$ and removing the edges in $v(e_i)$, $1 \leq i \leq \ell$. The incidence vectors of these solutions are linearly independent and satisfy $x(v(C)) = \ell - 1$. Since C has no chord, we can add any block from $D \setminus v(C)$ to one of the above solutions without inducing a mixed cycle on G. This yields another $|D| - \ell$ vectors that fulfill $x(v(C)) = \ell - 1$.

Moreover, the incidence vectors of all sets of blocks constructed above are linearly independent. Thus $x(v(C)) \leq \ell - 1$ is a facet-defining inequality.

On the other hand, if C has a chord e then each incidence vector χ^M of a solution $M \subseteq D$ satisfying $x(v(C)) = \ell - 1$ has to satisfy $\chi^M_{v(e)} = 0$, so $\dim\{x \in P_{\mathcal{T}}(G) | x(v(C)) = \ell - 1\} \leq |D| - 2$. Thus $x(v(C)) \leq \ell - 1$ is not a facet-defining inequality. □

The next lemma addresses the case in which we have a mixed cycle with a chord.

Lemma 6.2.7: Let $G = (V, E, H)$ be an extended alignment graph consisting of a critical mixed cycle C of size ℓ with a chord e and D be a partition into blocks. Then

the inequality

$$x(v(C \cup \{e\})) \leq \ell - 1$$

defines a facet of $P_{\mathcal{T}}(G)$.

Proof. From Lemma 6.2.6 we know that $x(v(C)) \leq \ell - 1$ is a facet-defining inequality for $P_{\mathcal{T}}(G[E \setminus v(e)])$. Since $I_{\mathcal{T}}$ is an independence system we can use the lifting theorem to obtain the coefficient of the block $v(e)$ in the above facet-defining inequality. First we observe that there are solutions that satisfy $x(v(C)) = \ell - 1$. In order to construct a feasible solution containing the block $v(e)$, we have to remove a block from any feasible solution of satisfying $x(v(C)) = \ell - 1$, because e induces two critical mixed cycles together with the edges in C. Theorem 2.7.4 implies that the coefficient of $x_{v(e)}$ is 1 which proves the lemma. □

There is in fact an even stronger version of Lemma 6.2.7 for the case that there are several chords in a critical mixed cycle such that they form an *r-ladder* of size ℓ. A r-ladder of size ℓ is a mixed cycle that contains r chords, such that each pair of chords together with at least one arc in C induce a mixed cycle in G. We define r-ladder of size ℓ inductively as follows:

Definition 6.2.3: Let C be a critical mixed cycle of size ℓ and $e_1, \ldots, e_r$ be r chords in C. The chords split C into C_{i_1} and C_{i_2} for $1 \leq i \leq r$. The set $C \cup \{e_1, \ldots, e_r\}$ is called r-ladder of size ℓ if for each e_i, $1 \leq i \leq r$ holds: $\{e_1, \ldots, e_r\} \setminus \{e_i\}$ can be partitioned into two (possibly empty) sets $U = \{e_{u_1}, \ldots, e_{u_j}\}$ and $L = \{e_{l_1}, \ldots, e_{l_k}\}$ with $j, k \geq 0$ and $j + k = r - 1$ such that $C_{i_1} \cup U \cup \{e_i\}$ is a j-ladder of size $|C_{i_1}| + 1$ and $C_{i_2} \cup L \cup \{e_i\}$ is a k-ladder of size $|C_{i_2}| + 1$.

In terms of Definition 6.2.3 a critical mixed cycle without a chord is a 0-ladder and a critical mixed cycle with one chord is a 1-ladder. Figure 6.7 on the next page shows two different drawings of an EAG consisting of a 3-ladder of size 4 (we ommited the arrows of the arcs in the right part). We can use the lifting theorem in an inductive proof to show that in an EAG consisting of an r-ladder of size ℓ the *ladder inequality* $x(v(C \cup \{e_1, \ldots, e_r\})) \leq \ell - 1$ is facet-defining.

Lemma 6.2.8: Let $G = (V, E, H)$ be an extended alignment graph and D be a partition into blocks. If G consists of a r-ladder $C \cup \{e_1, \ldots, e_r\}$ of size ℓ then the inequality

$$x(v(C \cup \{e_1, \ldots, e_r\})) \leq \ell - 1$$

defines a facet of $P_{\mathcal{T}}(G)$.

Proof. The proof follows an inductive argument over the number of chords. The base cases are given by Lemma 6.2.7 and Lemma 6.2.6. Assume inductively that for a

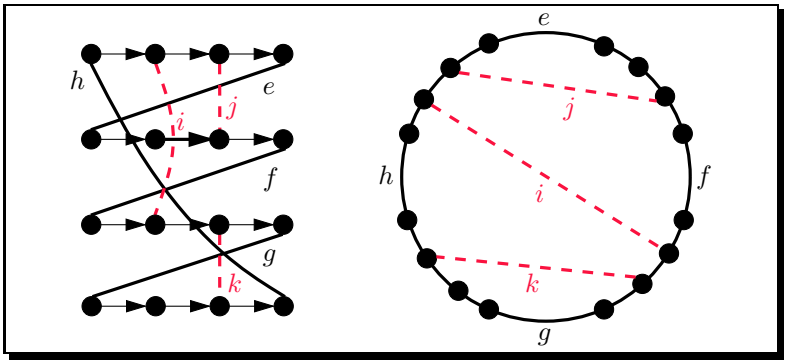

Figure 6.7: A 3-ladder of size 4.

k-ladder with $k < r$ the lemma holds, *i.e.*, the ladder inequality for the k-ladder is facet-defining for the EAG induced by the edges in the blocks of the k-ladder.

Let e_i be any of the r chords in C. Since e_i splits C it follows that $C_{i_1} \cup U \cup \{e_i\}$ is a j-ladder of size $|C_{i_1}|+1$ and $C_{i_2} \cup L \cup \{e_i\}$ is a k-ladder of size $|C_{i_2}|+1$ for some $j, k \geq 0$ with $j + k = r - 1$ and $|C_{i_1}| + |C_{i_2}| = \ell$. By the induction hypothesis the inequalities

- $x(v(C_{i_1} \cup U \cup \{e_i\})) \leq |C_{i_1}|$
- $x(v(C_{i_2} \cup L \cup \{e_i\})) \leq |C_{i_2}|$
- $x(v(C \cup \{e_1, \ldots, e_r\} \setminus \{e_i\})) \leq \ell - 1$

are facet-defining for the corresponding EAGs. This implies that there are solutions satisfying the above inequalities with equality. Surely these inequalities are still valid in G. That means that in each solution containing $v(e_i)$ we can choose at most $|C_{i_1}|-1$ additional blocks from $v(C_{i_1} \cup U)$ and at most $|C_{i_2}|-1$ additional blocks from $v(C_{i_2} \cup L)$ which sums up to $|C_{i_1}|-1+|C_{i_2}|-1 = \ell-2$ additional blocks from $v(C \cup \{e_1, \ldots, e_r\} \setminus \{e_i\})$.

On the other hand, a feasible solution satisfying $x(v(C \cup \{e_1, \ldots, e_r\} \setminus \{e_i\})) = \ell - 1$ may contain $\ell-1$ blocks. Since we just argued that we can have at most $\ell-2$ additional blocks in any solution containing $v(e_i)$, we have to remove a block from each solution satisfying $x(v(C \cup \{e_1, \ldots, e_r\} \setminus \{e_i\})) = \ell-1$. Theorem 2.7.4 implies that the coefficient of $x_{v(e_i)}$ is 1 which proves the lemma. □

We call the inequalities defined in the two preceding lemmas *mixed-cycle inequalities*, *chorded-mixed-cycle inequalities* and *ladder inequalities* respectively.

6.2.3 Bounds for the GMT Problem

Computation of Lower Bounds

For the computation of lower bounds we employed two heuristics. In the singleton partition case of the GMT problem we used John Kececioglu's package PRIMAL which implements an iterative alignment heuristic in order to compute a heuristic solution to the MT problem (Kececioglu 1993).

In the general case, it is still an open question to devise a good heuristic for the GMT problem. In our experiments we used a simple greedy strategy which sorts the blocks according to their score and then tries to incorporate as many as possible blocks from that list as long as a newly considered block does not induce a mixed cycle in the EAG.

Computation of Upper Bounds

In order to specialize the generic branch-and-cut algorithm we need to describe separation algorithms for our various classes of inequalities.

Mixed Cycle Inequalities. First we describe how to solve the separation problem for the mixed cycle inequalities. Assume the solution $\bar{x}$ of the linear program is fractional. Our problem is to find a critical mixed cycle C in the extended alignment graph $G = (V, E, H)$ which violates the mixed-cycle inequality $\sum_{d \in v(C)} x_d \leq |v(C)| - 1$.

First assign for each block d the cost $1 - \bar{x}_d$ to each edge $e \in d$ and 0 to all $a \in H$. Then compute for each arc $a = (u, v) = (s_{i,j}, s_{i,j+1})$, $1 \leq i \leq k,\ 1 \leq j < n_i$ the shortest path from v to u. Together with the arc a this path forms a mixed cycle. During this computation we have to take care that we compute the shortest path with the fewest edges. This can be done by ordering paths lexicographically according to their costs and then according to the number of edges. Then the lexicographically shortest mixed cycle is also critical (see also definition of critical mixed cycle on page 51).

If a shortest path P from v to u is found, it must contain $l \geq 2$ edges $e_1, \ldots, e_l$ from different blocks. If the cost of P is less than 1, *i.e.*, $\sum_{d \in v(P)}(1 - \bar{x}_d) < 1$, a violated inequality is found, namely $\sum_{d \in v(P)} \bar{x}_d > |v(P)| - 1$.

Theorem 6.2.3: The separation problem for the mixed-cycle inequalities in an extended alignment graph $G = (V, E, H)$ can be solved in polynomial time by computing at most $|H|$ shortest paths in G.

Unfortunately this might still result in a big number of shortest path computations. This is particularly annoying for partitions with large blocks, because there is a lot

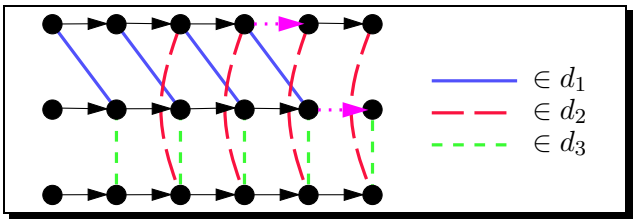

Figure 6.8: Extended alignment graph with 13 edges partitioned into 3 blocks d_1, d_2, d_3. Only the dotted purple arcs need to be checked.

of different paths resulting in the same inequality. For example Figure 6.2.3 shows an EAG with a partition into three blocks. The only mixed cycle inequality that can be found is $x_{d_1} + x_{d_2} + x_{d_3} \leq 2$. With the naive approach, we would have to make 15 shortest path computations. We will show that in this example it is safe to make only two such computations (for the dotted purple arcs).

We call two paths P and P' *equivalent* if $v(P) = v(P')$. The set of all paths forms equivalence classes under the above relation. We will now show how to pick a subset $A \subseteq H$ of arcs, such that we only have to compute a shortest path from v to u for each $a = (u, v) \in A$. We do that by excluding certain arcs from consideration.

We say that an alignment edge e is *right of* an alignment edge f in a block $d \in D$ if $start(e) > start(f)$ and $end(e) > end(f)$. Let $D(u, i)$ be the set of all blocks that have an edge incident to u and to a node in sequence S_i, *i.e.*,

$$D(u, i) := \{d \in D \mid \exists e = \{u, s_{i,x}\} \in d \text{ for some } 1 \leq x \leq n_i\}.$$

Then the following lemma holds:

Lemma 6.2.9: Let $a = (u, v)$ be an arc in sequence S_i with $D(u, j) \subseteq D(v, j)$ for some $1 \leq j \neq i \leq k$. Then for any critical mixed cycle C that contains a and that enters S_i from S_j through an edge e incident to u, there is an equivalent critical mixed cycle C' using an edge $f \in v(e)$ which is right of e.

Proof. Since $a = (u, v)$ is an arc and $D(u, j) \subseteq D(v, j)$ both nodes u and v must be incident to edges in $v(e)$, namely to e and f. Since the block $v(e)$ is a trace, f must be right of e. Let w be the node in S_j that is incident to e and w' be the node in S_j incident to f. We can construct C' from C by deleting e and a from C and replacing it by the path consisting of the arcs running from w to w' followed by f (see Figure 6.9). □

We can therefore discard an arc $a = (u, v)$ in sequence S_i from consideration if for all $j = 1, \ldots, k$, $j \neq i$ holds that $D(u, j) \subseteq D(v, j)$, because

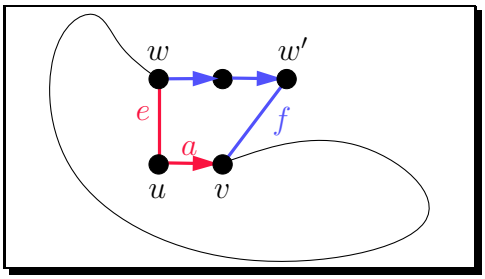

Figure 6.9: Two equivalent critical mixed cycles

1. for any critical mixed cycle C that enters at node u there is an equivalent critical mixed cycle C', and

2. any critical mixed cycle that contains a and enters S_i before u must contain an additional arc in S_i. Thus we do not need to check arc a.

Theorem 6.2.4: The separation problem for the mixed-cycle inequalities in an extended alignment graph $G = (V, E, H)$ can be solved in polynomial time by computing at most $|A|$ shortest paths in G, where $A \subseteq H$ is defined as $\{(u, v) \in H \mid \exists i \in \{1, \ldots, k\}$ such that $D(u, i) \subsetneq D(v, i)\}$.

It turns out that the application of Lemma 6.2.4 can reduce the running time for the mixed cycle separation considerably. There are examples where the number of shortest path computations is reduced by a factor of ten.

Clique Inequalities. In the separation algorithm for the class of clique inequalities we make use of the *pairgraph* $P_{\mathcal{T}}(G_{i,j})$ (see Definition 6.2.1 on page 79) for the sequences S_i and S_j for $1 \leq i < j \leq k$. Again, assume the solution $\bar{x}$ of the linear program is fractional. Our problem is to find a clique C which violates the clique inequality $\sum_{d \in v(C)} \bar{x}_d \leq 1$. For each edge $e \in d$ assign the cost $\bar{x}_d$ to the node v_e in $P_{\mathcal{T}}(G_{i,j})$. Recall that no two edges in the same block can lie on a source-to-sink path, and that all maximal cliques in the independence system are represented by some source-to-sink path.

We compute the longest source-to-sink path C in $P_{\mathcal{T}}(G_{i,j})$. If the cost of C is greater than 1, *i.e.*, $\sum_{d \in v(C)} \bar{x}_d > 1$ we have found a violated clique inequality. Since $P_{\mathcal{T}}(G_{i,j})$ is acyclic, such a path can be found in time polynomial in the size of the EAG by simply exploring the graph in topological order and examining the arcs incident to each node. The longest path to a certain node is computed by taking the maximum over the value of all incident arcs plus the length of the longest path to the source of that arc.

Theorem 6.2.5: The separation problem for the clique inequalities in an extended

alignment graph $G = (V, E, H)$ can be solved in polynomial time by computing a longest source-to-sink path in the $\binom{k}{2}$ pairgraphs $P_{\mathcal{T}}(G_{i,j})$ for $1 \leq i < j \leq k$, where k is the number of sequences.

The separation of clique inequalities is fast and efficient. Using sparse pairgraphs as explained on page 80 make this separation routine the backbone of our branch-and-cut algorithms.

Separation and Branching Strategy. In our computational experiments we tried several separation strategies, that means different orders in which we test a class for violated inequalities. The following strategy turned out to be the best. In the branch-and-cut algorithm we first separate the clique inequalities as described above. If we cannot find a violated clique inequality we check whether the EAG contains a mixed cycle by computing a number of shortest paths from v to u for each arc $a = (u, v)$ in a set A as defined in Theorem 6.2.4. If we find one or more such paths we add the corresponding mixed cycle inequalities to the LP and resolve it. Finally, if we do not find any violated inequalities or if the solution value of the LP does not improve significantly over a number of iterations, we branch. In the branching phase we choose the fractional base pair variable which is closest to 0.5 and among those the one with the highest objective function coefficient. After the branching we iterate the process on the two subproblems.

6.2.4 Computational Results for the GMT Problem

In this section we report on the results computed with our program. The implementation is coded in C++ using the **L**ibrary of **E**fficient **D**ata Types and **A**lgorithms LEDA (Mehlhorn and Näher 1995) and the branch-and-cut framework ABACUS (Jünger and Thienel 1997).

We tested three different approaches to generate the extended alignment graph, each with a different trace score function. We used the following data sets in order to test our algorithms:

- Two sets of 38 and 18 protein sequences that were found in SwissProt (Bairoch and Apweiler 1999) by conducting a similarity search with hevein, a protein that binds N-acetylglucosamin (a sugar). We call the two test sets `hevein1` and `hevein2`. These data sets stem from the GELENA project in which the MPI für Informatik participates together with other partners. In this project new gen transfer methods based on nanoparticles are investigated. The goal is to repair genetic defects of certain cells. These cells have N-acetylglucosamin oligomers on

their surface. Hence, peptides that have a high binding affinity to these sugar molecules can be used in docking reactions to identify these cells and to insert genetic material into them.

- 12 Globin sequences from the data set of (McClure, Vasi, and Fitch 1994). McClure used this test sets to compare the performance of alignment methods.
- 18 Prion proteins from the SwissProt (Bairoch and Apweiler 1999) protein database. This set contains highly similar but long prion proteins from different species.
- 9 Tyrosine kinases used as example by Kececioglu (1993).

We present different alignments for each of the three approaches, because a test set that took minutes in the one approach would not finish within a day in the other approach. The difficulty of a problem instance in each of the three approaches is not easily measured. It depends on the number of sequences in the test set and their degree of homology. The more sequences we consider and the less homology they have, the more complex the input EAG and the longer the optimization takes.

If space allows, we present an alignment as in Figure 6.10 together with the global lower bound we used in the branch-and-cut algorithm, the optimal score, and the running time in seconds that the main optimization phase took. Small letters indicate that the respective residue is not aligned with another residue in the optimal solution. The alignment in Figure 6.10 contains the same sequences as in Kececioglu (1993) but the input alignment graph uses a different mutation score matrix.

Generating the Input

We tested three different approaches to generate the extended alignment graph.

- As an example of a scoring scheme based on the comparison of two residues (MT) we adapted the PRIMAL package by Kececioglu (1993). The value of the approximate solution of this program is used as a lower bound in our branch-and-cut algorithm.
- As an example for a scoring scheme based on the comparison of segment pairs we adopted two ways to generate the input for the branch-and-cut algorithm. The first takes the set of blocks that are computed by Burkhard Morgenstern's DIALIGN package (Morgenstern, Dress, and Werner 1996). The weight of DIALIGN's greedy heuristic is used as a lower bound for the branch-and-cut algorithm.

```
Exact       value: 45769
Approximate value: 45693
Total time       : 261 sec.

v-src   0 -----GLAKDAWEIPRESLRLEAKLGQGCFGEVWMGTWN-DTTRVAIKTLKPGTMSP---
v-yes   0 -----GLAKDAWEIPRESLRLEVKLGQGCFGEVWMGTWNG-TTKVAIKTLKLGTMMP---
v-abl   0 tIYGVSPNYDKWEMERTDITMKHKLGGGQYGEVYEGVWKKYSLTVAVKTLKEDTM---EV
v-fes   0 -VLNRAVPKDKWVLNHEDLVLGEQIGRGNFGEVFSGRLRADNTLVAVKSCRE-TLPPDIK
v-fps   0 -VLTRAVLKDKWVLNHEDVLLGERIGRGNFGEVFSGRLRADNTPVAVKSCRE-TLPPELK
v-raf   0 -------SSYYWKMEASEVMLSTRIGSGSFGTVYKGKWHGDVAVKILKVVDP-T--PEQL

v-src  60 EAFLQEAQVMKKLRHEKLVQLYAVVS-EEPIYIVIEYMSKGSLLDFLKG-EmG-KYLRLP
v-yes  60 EAFLQEAQIMKKLRHDKLVPLYAVVS-EEPIYIVTEFMTKGSLLDFLKEGE-G-KFLKLP
v-abl  60 EEFLKEAAVMKEIKHPNLVQLLGVCTREPPFYIITEFMTYGNLLDYLRECN-R-QEVSAV
v-fes  60 AKFLQEAKILKQYSHPNIVRLIGVCTQKQPIYIVMELVQGGDFLTFLRT-E-GAR-LRMK
v-fps  60 AKFLQEARILKQCNHPNIVRLIGVCTQKQPIYIVMELVQGGDFLSFLRS-K-GPR-LKMK
v-raf  60 QAFRNEVAVLRKTRHVNILLFMGYMT-KDNLAIVTQWCEGSSLYKHLHV-Q-ETK-FQMF

v-src 120 QLVDMAAQIASGMAYVERMNYVHRDLRAANILVGENLVCKVADFGLARLIEDNEYTARQG
v-yes 120 QLVDMAAQIADGMAYIERMNYIHRDLRAANILVGDNLVCKIADFGLARLIEDNEYTARQG
v-abl 120 VLLYMATQISSAMEYLEKKNFIHRDLAARNCLVGENHLVKVADFGLSRLMTGDTYTAHAG
v-fes 120 TLLQMVGDAAAGMEYLESKCCIHRDLAARNCLVTEKNVLKISDFGMSREAADGIYAASGG
v-fps 120 KLIKMMENAAAGMEYLESKHCIHRDLAARNCLVTEKNTLKISDFGMSRQEEDGVYASTGG
v-raf 120 QLIDIARQTAQGMDYLHAKNIIHRDMKSNNIFLHEGLTVKIGDFGLATVKSRWSGSQQVE

v-src 180 AK-FPIKWTAPEAALY---GRFTIKSDVWSFGILLTELTTKGRVPYPGMVNR-EVLDQVE
v-yes 180 AK-FPIKWTAPEAALY---GRFTIKSDVWSFGILLTELVTKGRVPYPGMVNR-EVLEQVE
v-abl 180 AK-FPIKWTAPESLAY---NKFSIKSDVWAFGVLLWEIATYGMSPYPGIDLS-QVYELLE
v-fes 180 LRQVPVKWTAPEALNY---GRYSSESDVWSFGILLWETFSLGASPYPNLSNQ-QTREFVE
v-fps 180 MKQIPVKWTAPEALNY---GWYSSESDVWSFGILLWEAFSLGAVPYANLSNQ-QTREAIE
v-raf 180 QPTGSVLWMAPEVIRMqddNPFSFQSDVYSYGIVLYELMA-GELPYAHINNRdQIIFMVG

v-src 240 RGYRMP----CPPECPESLHDLMCQCWRKDPEERPTFKYLQAQLLPACVLEVAE------
v-yes 240 RGYRMP----CPQGCPESLHELMKLCWKKDPDERPTFEYIQSFLEDYFTAAEPSGY----
v-abl 240 KDYRME----RPEGCPEKVYELMRACWQWNPSDRPSFAEIHQAFETMFQESSIS------
v-fes 240 KGGRLP----CPELCPDAVFRLMEQCWAYEPGQRPSFSAIYQELQSIRKRHR--------
v-fps 240 QGVRLE----PPEQCPEDVYRLMQRCWEYDPHRRPSFGAVHQDLIAIRKRHR--------
v-raf 240 RGYASPdlsrLYKNCPKAIKRLVADCVKKVKEERPLFPQILSSIELLQHSLPKINRsape
```

Figure 6.10: Optimal trace of six tyrosine kinase sequences. Input was generated using PRIMAL.

- In the second approach we compute (sub)optimal local alignments between two sequences that do not share (mis)matches. We call the procedure that produces the blocks LOCAL. Here we employed a simple greedy strategy to compute lower bounds for the branch-and-cut algorithm.

In the following we describe the three approaches in more detail and present multiple alignments we have computed with our branch-and-cut algorithm.

Results

All tests were conducted on a single 333 MHz processor of a Sun Enterprise 10000 with 12 gigabyte of main memory, where a single process can use up to 2 gigabyte.

```
Exact       value: 38239
Approximate value: 37787
Total time       : 1507 sec.

HEVE_HEVBR 0 ---------------MNIFIVVLLCLTGVAIAEQCGRQAGGKLCPNNLCC
HEVP_HEVBR 0 --------------------------------EQCGRQAGGKLCPNNLCC
WIN2_SOLTU 0 -------MVKLSCGPILLALVLCISLTSVANAQQCGRQRGGALCGNNLCC
WIN1_SOLTU 0 -------MVKLISNSTILLSLFLFSIAAIANAQQCGRQKGGALCSGNLCC
S18750     0 ----------MSVW-AFAFFSLFLSLSVRGSAEQCGQQAGDALCPGGLCC
CHIB_POPTR 0 ----------MSVW-AFAFFSLFLSLSVRGSAEQCGQQAGDALCPGGLCC
CHI6_POPTR 0 ----------MSVWALFAFFSLFLSLSVRGSAEQCGRQAGDALCPGGLCC
CHIC_POPTR 0 ----------MSVWAFFAFFSLFLSLSVRGSAEQCGRQAGDALCPGGLCC
S40414     0 MSTPRAAASLAKKAALVALAVLAAALATAARAEQCGAQAGGARCPNCLCC
S39979     0 MSTPRAAASLAKKALAVALAVLAAALATAARAEQCGAQAGGARCPNCLCC

HEVE_HEVBR 50 SQWGWCGSTDEYCSPDHNCQSNCKDSGEGVGGGSAS--NVLATYHL
HEVP_HEVBR 50 SQYGWCGSSDDYCSPSKNCQSNCKGGG-------------------
WIN2_SOLTU 50 SQYGWCGSSDEYCSPSQGCQSQCTGSGPDPGQGGSA-QNVRATYHI
WIN1_SOLTU 50 SQFGWCGSTPEFCSPSQGCQSRCTGTGGSTPTPSGSaQNVRATYHI
S18750     50 SSYGWCGTTADYCG--DGCQSQCDGGGGGGGGGGGG-GG-------
CHIB_POPTR 50 SSYGWCGTTADYCG--DGCQSQCDGGGGGGGGGGGG-GG-------
CHI6_POPTR 50 SSYGWCGTTVDYCGI--GCQSQCDGGGGGDGGDDGC-DGG-DD---
CHIC_POPTR 50 SFYGWCGTTVDYCG--DGCQSQCDGGDGCDGGGG------------
S40414     50 SRWGWCGTTSDFCG--DGCQSQCSGCGPTPTPTPPS-P--------
S39979     50 SRWGWCGTTSDFCG--DGCQSQCSGCGPTTPTPPSP----------
```

Figure 6.11: Optimal trace of 10 proteins from `hevein2`. Input was generated using PRIMAL.

Blocks computed by PRIMAL. To generate an extended alignment graph PRIMAL computes all pairwise alignments of the sequences whose score is within a fixed difference of the optimum.

(As parameters for PRIMAL we chose the `blosum80` mutation score matrix, shifted to make all similarity values positive and in the range 0 to 24, a gap penalty of 40, and collected all pairwise alignments that scored within 10 of optimum, unless stated otherwise.)

```
Exact       value: 82166
Approximate value: 81822
Total time       : 345 sec.

AGI1_WHEAT   0 AQRCGEQGSNMEC-PNNLCCSQYGYCGMGGDYCGK--G--CQNGAC
AGI_ORYSA    0 AQTCGKQNDGMIC-PHNLCCSQFGYCGLGRDYCGT--G--CQSGAC
CHI4_BRANA   0 -------SQNCGC-APNLCCSQFGYCGSTDAYCGT--G--CRSGPC
CHIA_MAIZE   0 -------AQNCGC-QPNFCCSKFGYCGTTDAYCGD--G--CQSGPC
CHIP_BETVU   0 -------AQNCGC-APNLCCSNFGFCGTGTPYCGV--GN-CQSGPC
CHI4_PHAVU   0 -------AQNCGC-AEGLCCSQYGYCGTGEDYCGT--G--CQQGPC
CHIT_DIOJA   0 -------Q-NCQCdTTIYCCSQHGYCGNSYDYCGP--G--CQAGPC
AGI_URTDI    0 AQRCGSQGGGGTC-PALWCCSIWGWCGDSEPYCGR--T--CENK-C
CHI1_ORYSA   0 GEQCGSQAGGALC-PNCLCCSQYGWCGSTSDYCGA--G--CQSQ-C
CHI2_ORYSA   0 AEQCGSQAGGAVC-PNCLCCSQFGWCGSTSDYCGA--G--CQSQ-C
CHIX_PEA     0 AEQCGSQAGGAVC-PNGLCCSKFGFCGSTDPYCGD--G--CQSQ-C
CHI1_TOBAC   0 AEQCGSQAGGARC-PSGLCCSKFGWCGNTNDYCGP--GN-CQSQ-C
CHI5_PHAVU   0 GEQCGRQAGGALC-PGGNCCSQFGWCGSTTDYCGK--D--CQSQ-C
CHIB_POPTR   0 AEQCGQQAGDALC-PGGLCCSSYGWCGTTADYCGD--G--CQSQ-C
CHI2_BRANA   0 AEQCGRQAGGALC-PNGLCCSEFGWCGDTEAYCKQP-G--CQSQ-C
HEVE_HEVBR   0 AEQCGRQAGGKLC-PNNLCCSQWGWCGSTDEYCSP--DHNCQSN-C
WIN1_SOLTU   0 AQQCGRQKGGALC-SGNLCCSQFGWCGSTPEFCSPSqG--CQSR-C
HEVL_ARATH   0 GQQCGRQGGGRTC-PGNICCSQYGYCGTTADYCSP--TNNCQSN-C
CHI8_POPTR   0 TAQCGSQAGNATC-PNDLCCSSGGYCGLTVAYCCA--G--CVSQ-C
```

Figure 6.12: Optimal trace of the set `hevein1`. Input was generated using PRIMAL.

PRIMAL then superimposes all alignment edges corresponding to the (mis)matches in these pairwise alignments to form an alignment graph. Our input is the corresponding extended alignment graph. When the input is generated with a residue-to-residue based scoring scheme, the problem can be divided into independent subproblems by splitting the input EAG into its strongly connected components. The optimal trace for the original input EAG is simply the union of the optimal traces for its components.

This procedure normally produces a lot of small components if the sequences under consideration have high homology and only a few components (sometimes only one) if the sequences have little homology. Note that this was already observed by Kececioglu, and that his code takes also advantage of this fact.

In Figure 6.10 on page 93 we already showed an optimal trace for the six tyrosine kinases sequences used by Kececioglu (1993). The next example shows the quality of the traces we computed. For the `hevein1` data set and a subset of the `hevein2` data set we generated input EAGs using PRIMAL.

The optimal traces in Figure 6.12 on the page before and Figure 6.11 on page 94 exhibit the four disulphid bridges that are essential for the formation of the three-dimensional structure of this protein. These bridges are built by the eight cysteins (shown in yellow) that are perfectly aligned. Also the N-acetylglucosamin-binding sites (the main part is shown in red) are well aligned. The next example shows a trace for 18 relatively similar prion sequences. Despite the high homology, PRIMAL could not optimally align this dataset.

For this number of sequences the bottleneck is normally the space consumption which is not the case for our approach. It is not so sensitive to the number of sequences but to the structure and size of the extended alignment graph. The branch-and-cut algorithm produced the alignment shown in Figures 6.13 and 6.14 in 7178 seconds.

Blocks computed by DIALIGN. In the DIALIGN program (Morgenstern, Dress, and Werner 1996) the blocks are called *diagonals*, because a block represents a gapless alignment which is a diagonal run in the corresponding dynamic programming matrix. The algorithm greedily picks the best diagonal from *all* possible diagonals which is consistent with previously chosen diagonals. Although this input could be modeled in the GMT formulation it is far too big. Therefore we input solely diagonals stemming from optimal pairwise alignments that are not in conflict.

The weight w_d of a diagonal d is defined as follows: Let l_d be the length of the diagonal and s_d be the sum of the individual similarity values of residue pairs within this diagonal. Let $P(l_d, s_d)$ be the probability that a random diagonal of the same length l_d has at least the same sum s_d of similarity values. Then w_d is defined to be $-\log P(l_d, s_d)$. For a more in depth treatment see Morgenstern *et al.* (1998).

Unfortunately, we cannot divide the problem into smaller subproblems by splitting the input EAG in its strongly connected components. This might put alignment edges of the same block (or diagonal) into different components. Hence, we have to consider the whole graph. On the positive side, we have fewer variables in the ILP formulation, because the blocks contain many alignment edges.

```
prio_night   0 ---------MLVLFVATWSDLGLCKKRPKP-GGWNTGGSRYPGQSSPGGNRYPPQSGG-W
prion_amer   0 MVKSHIGSWLLVLFVATWSDIGFCKKRPKPGGGWNTGGSRYPGQGSPGGNRYPPQGGGGW
prion_blac   0 ---------MLVLFVATWSDLGLCKKRPKP-GGWNTGGSRYPGQGSPGGNRYPPQGG---
prion_mand   0 ---------MLVLFVATWSDLGLCKKRPKP-GGWNTGGSRYPGQGSPGGNRYPPQGGGGW
prion_pres   0 --MANLGCWMLVLFVATWSDLGLCKKRPKP-GGWNTGGSRYPGQGSPGGNRYPPQGGGGW
prion_crab   0 --MANLGCWMLVLFVATWSDLGLCKKRPKP-GGWNTGGSRYPGQGSPGGNRYPPQGGGGW
prion_gree   0 --MANLGCWMLVVFVATWSDLGLCKKRPKP-GGWNTGGSRYPGQGSPGGNRYPPQGGG--
prion_brow   0 --MANLGCWMLVLFVATWSDLGLCKKRPKP-GGWNTGGSRYPGQGSPGGNLYPPQGGG-W
prion_chim   0 --MANLGCWMLVLFVATWSDLGLCKKRPKP-GGWNTGGSRYPGQGSPGGNRYPPQGGGGW
prion_oran   0 --MANLGCWMLVLFVATWSNLGLCKKRPKP-GGWNTGGSRYPGQGSPGGNRYPPQGGGGW
prion_gori   0 --MANLGCWMLVLFVATWSDLGLCKKRPKP-GGWNTGGSRYPGQGSPGGNRYPPQGGGGW
prion_huma   0 --MANLGCWMLVLFVATWSDLGLCKKRPKP-GGWNTGGSRYPGQGSPGGNRYPPQGGGGW
prion_bovi   0 MVKSHIGSWILVLFVAMWSDVGLCKKRPKPGGGWNTGGSRYPGQGSPGGNRYPPQGGGGW
prion_shee   0 MVKSHIGSWILVLFVAMWSDVGLCKKRPKPGGGWNTGGSRYPGQGSPGGNRYPPQGGGGW
prion_mule   0 MVKSHIGSWILVLFVAMWSDVGLCKKRPKPGGGWNTGGSRYPGQGSPGGNRYPPQGGGGW
 prion_rat   0 -------------------------------GGWNTGGSRYPGQGSPGGNRYPPQSGGTW
prion_gold   0 --MANLSYWLLALFVAMWTDVGLCKKRPKP-GGWNTGGSRYPGQGSPGGNRYPPQGGGTW
prion_mous   0 --MANLGYWLLALFVTMWTDVGLCKKRPKP-GGWNTGGSRYPGQGSPGGNRYPPQGG-TW

prio_night  60 GQPHGGGWGQPHGGGWGQPHGGGWGQPHGGG-WGQ--------GGGTHNQWNKPSKPKTN
prion_amer  60 GQPHGGGWGQPHGGGWGQPHGGGWGQPHGGGgWGQ--------GGGSHGQWGKPSKPKTN
prion_blac  60 ------GWGQPHGGGWGQPHGGGWGQPHGGG-WGQ--------GGGTHNQWNKPSKPKTN
prion_mand  60 GQPHGGGWGQPHGGGWGQPHGGGWGQPHGGG-WGQ--------GGGTHNQWHKPNKPKTS
prion_pres  60 GQPHGGGWGQPHGGGWGQPHGGGWGQPHGGG-WGQ--------GGGTHSQWNKPSKPKSN
prion_crab  60 GQPHGGGWGQPHGGGWGQPHGGGWGQPHGGG-WGQ--------GGGTHNQWHKPSKPKTS
prion_gree  60 ------GWGQPHGGGWGQPHGGGWGQPHGGG-WGQ--------GGGTHNQWHKPSKPKTS
prion_brow  60 GQPHGGGWGQPHGGGWGQPHGGSWGQPHGGG-WGQ--------GGGTHNQWNKPSKPKTS
prion_chim  60 GQPHGGGWGQPHGGGWGQPHGGGWGQPHGGG-WGQ--------GGGTHSQWNKPSKPKTN
prion_oran  60 GQPHGGGWGQPHGGGWGQPHGGGWGQPHGGG-WGQ--------GGGTHSQWNKPSKPKTN
prion_gori  60 GQPHGGGWGQPHGGGWGQPHGGGWGQPHGGG-WGQ--------GGGTHSQWNKPSKPKTN
prion_huma  60 GQPHGGGWGQPHGGGWGQPHGGGWGQPHGGG-WGQ--------GGGTHSQWNKPSKPKTN
prion_bovi  60 GQPHGGGWGQPHGGGWGQPHGGGWGQPHGGG-WGQphggggwgQGGTHGQWNKPSKPKTN
prion_shee  60 GQPHGGGWGQPHGGGWGQPHGGGWGQPHGGG-GWG--------QGGSHSQWNKPSKPKTN
prion_mule  60 GQPHGGGWGQPHGGGWGQPHGGGWGQPHGGG-GWG--------QGGTHSQWNKPSKPKTN
 prion_rat  60 GQPHGGGWGQPHGGGWGQPHGGGWGQPHGGG-WSQ--------GGGTHNQWNKPSKPKTN
prion_gold  60 GQPHGGGWGQPHGGGWGQPHGGGWGQPHGGG-WGQ--------GGGTHNQWNKPSKPKTN
prion_mous  60 GQPHGGGWGQPHGGSWGQPHGGSWGQPHGGG-WGQ--------GGGTHNQWNKPSKPKTN

prio_night 120 MKHMAGAAAAGAVVGGLGGYMLGSAMSRPLIHFGNDYEDRYYRENMYRYPNQVYYRPVDQ
prion_amer 120 MKHVAGAAAAGAVVGGLGGYMLGSAMSRPLIHFGNDYEDRYYRENMYRYPNQVYYKPVDQ
prion_blac 120 MKHMAGAAAAGAVVGGLGGYMLGSAMSRPLIHFGNDYEDRYYRENMYRYPNQVYYRPVDQ
prion_mand 120 MKHMAGAAAAGAVVGGLGGYMLGSAMSRPLIHFGNDYEDRYYRENMYRYPNQVYYRPVDQ
prion_pres 120 MKHMAGAAAAGAVVGGLGGYMLGSAMSRPLIHFGNDYEDRYYRENMYRYPNQVYYRPVDQ
prion_crab 120 MKHMAGAAAAGAVVGGLGGYMLGSAMSRPLIHFGNDYEDRYYRENMYRYPNQVYYRPVDQ
prion_gree 120 MKHMAGAAAAGAVVGGLGGYMLGSAMSRPLIHFGNDYEDRYYRENMYRYPNQVYYRPVDQ
prion_brow 120 MKHVAGAAAAGAVVGGLGGYMLGSAMSRPLIHFGNDYEDRYYRENMYRYPNQVYYRPVDQ
prion_chim 120 MKHMAGAAAAGAVVGGLGGYMLGSAMSRPIIHFGSDYEDRYYRENMHRYPNQVYYRPMDQ
prion_oran 120 MKHMAGAAAAGAVVGGLGGYMLGSAMSRPIIHFGNDYEDRYYRENMYRYPNQVYYRPVDQ
prion_gori 120 MKHMAGAAAAGAVVGGLGGYMLGSAMSRPIIHFGSDYEDRYYRENMHRYPNQVYYRPMDQ
prion_huma 120 MKHMAGAAAAGAVVGGLGGYMLGSAMSRPIIHFGSDYEDRYYRENMHRYPNQVYYRPMDE
prion_bovi 120 MKHVAGAAAAGAVVGGLGGYMLGSAMSRPLIHFGSDYEDRYYRENMHRYPNQVYYRPVDQ
prion_shee 120 MKHVAGAAAAGAVVGGLGGYMLGSAMSRPLIHFGNDYEDRYYRENMYRYPNQVYYRPVDR
prion_mule 120 MKHVAGAAAAGAVVGGLGGYMLGSAMNRPLIHFGNDYEDRYYRENMYRYPNQVYYRPVDQ
 prion_rat 120 LKHVAGAAAAGAVVGGLGGYMLGSAMSRPMLHFGNDWEDRYYRENMYRYPNQVYYRPVDQ
prion_gold 120 MKHMAGAAAAGAVVGGLGGYMLGSAMSRPMMHFGNDWEDRYYRENMNRYPNQVYYRPVDQ
prion_mous 120 LKHVAGAAAAGAVVGGLGGYMLGSAMSRPMIHFGNDWEDRYYRENMYRYPNQVYYRPVDQ
```

Figure 6.13: Optimal trace of 18 prion protein sequences (part 1). Input was generated using PRIMAL.

```
prio_night 180 YSNQNNFVHDCVNITIKQHTVTTTTKGENFTETDVKIMERVVEQMCITQYEKESQAYYQ-
prion_amer 180 YSNQNNFVHDCVNITVKQHTVTTTTKGENFTETDMKIMERVVEQMCVTQYQRESEAYYQ-
prion_blac 180 YNNQNNFVHDCVNITIKQHTVTTTTKGENFTETDVKMMERVVEQMCITQYERESQAYYQ-
prion_mand 180 YSNQNNFVHDCVNITIKQHTVTTTTKGENFTETDVKMMERVVEQMCITQYEKESQAYYQ-
prion_pres 180 YSNQNNFVHDCVNITIKQHTVTTTTKGENFTETDVKMMERVVEQMCITQYEKESQAYYQ-
prion_crab 180 YSNQNNFVHDCVNITIKQHTVTTTTKGENFTETDVKMMERVVEQMCITQYEKESQAYYQ-
prion_gree 180 YSNQNNFVHDCVNITIKQHTVTTTTKGENFTETDVKMMERVVEQMCITQYEKESQAYYQ-
prion_brow 180 YSNQNNFVHDCVNITIKQHTVTTTTKGENFTETDVKMMERVVEQMCITQYERESQAYYQ-
prion_chim 180 YSSQNNFVHDCVNITIKQHTVTTTTKGENFTETDVKMMERVVEQMCITQYERESQAYYQ-
prion_oran 180 YSNQNNFVHDCVNITIKQHTVTTTTKGENFTETDVKMMERVVEQMCITQYERESQAYYQ-
prion_gori 180 YSNQNNFVHDCVNITIKQHTVTTTTKGENFTETDVKMMERVVEQMCITQYERESQAYYQ-
prion_huma 180 YSNQNNFVHDCVNITIKQHTVTTTTKGENFTETDVKMMERVVEQMCITQYERESQAYYQ-
prion_bovi 180 YSNQNNFVHDCVNITVKEHTVTTTTKGENFTETDIKMMERVVEQMCITQYQRESQAYYQ-
prion_shee 180 YSNQNNFVHDCVNITVKQHTVTTTTKGENFTETDIKIMERVVEQMCITQYQRESQAYYQ-
prion_mule 180 YNNQNTFVHDCVNITVKQHTVTTTTKGENFTETDIKMMERVVEQMCITQYQRESQAYYQ-
 prion_rat 180 YSNQNNFVHDCVNITIKQHTVTTTTKGENFTETDVKMMERVVEQMCVTQYQKESQAYYDG
prion_gold 180 YNNQNNFVHDCVNITIKQHTVTTTTKGENFTETDIKIMERVVEQMCTTQYQKESQAYYDG
prion_mous 180 YSNQNNFVHDCVNITIKQHTVTTTTKGENFTETDVKMMERVVEQMCVTQYQKESQAYYDG

prio_night 240 -RGSSMVLFSSPPVILLISFL------
prion_amer 240 -RGASAILFSPPPVILLISLLILLIVG
prion_blac 240 -RGSSMVLFSSPPVILLISFLI-----
prion_mand 240 -RGSSMVLFSSPPVILLISFLI-----
prion_pres 240 -RGSSMVFFSSPPVILLISFLIFLIVG
prion_crab 240 -RGSSMVLFSSPPVILLISFLIFLIVG
prion_gree 240 -RGSSMVLFSSPPVILLISFLIFLIVG
prion_brow 240 -RGSSMVLFSSPPVILLISFLIFLIVG
prion_chim 240 -RGSSMVLFSSPPVILLISFLIFLIVG
prion_oran 240 -RGSSMVLFSSPPVILLISFLIFLIVG
prion_gori 240 -RGSSMVLFSSPPVILLISFLIFLIVG
prion_huma 240 -RGSSMVLFSSPPVILLISFLIFLIVG
prion_bovi 240 -RGASVILFSSPPVILLISFLIFLIVG
prion_shee 240 -RGASVILFSSPPVILLISFLIFLIVG
prion_mule 240 -RGASVILFSSPPVILLISFLIFLIVG
 prion_rat 240 -RRSSAVLFSSPPVILLISFLIFLIVG
prion_gold 240 -RRSSAVLFSSPPVILLISFLIFLMVG
prion_mous 240 rRSSSTVLFSSPPVILLISFLIFLIVG
```

Figure 6.14: Optimal trace of 18 prion protein sequences (part 2). Input was generated using PRIMAL.

Figure 6.15 on the next page shows an optimal trace of 10 globin sequences from McClure's data set. All five motifs except the first one (shown in red) are correctly aligned. Inspecting the red motif shows that in the first 8 sequences gaps are inserted before the motif, whereas the last two sequences contain a gap right of the motif. Simply removing these gaps yields the correct alignment of the first block.

These gaps have no "biological" meaning. They are rather introduced by the output procedure of an optimal trace, which sorts the strongly connected components in topological order and then outputs the components in this order.

Hence, the strange gaps in the first motif stem obviously from the fact, that the input

EAG contained no alignment edges between the aligned blocks in the last two sequences and the aligned blocks in the first eight sequences.

```
HUMA      0 ----VLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTK--------TYFPHF--
HAOR      0 ----MLTDAEKKEVTALWGKAAGHGEEYGAEALERLFQAFPTTK--------TYFSHF--
HADK      0 ----VLSAADKTNVKGVFSKIGGHAEEYGAETLERMFIAYPQTK--------TYFPHF--
HBHU      0 --VH-LTPEEKSAVTALWGKV--NVDEVGGEALGRLLVVYPWTQ--------RFFESFGD
HBOR      0 --VH-LSGGEKSAVTNLWGKV--NINELGGEALGRLLVVYPWTQ--------RFFEAFGD
HBDK      0 --VH-WTAEEKQLITGLWGKV--NVADCGAEALARLLIVYPWTQ--------RFFASFGN
MYHU      0 -G---LSDGEWQLVLNVWGKVEADIPGHGQEVLIRLFKGHPETL--------EKFDKFKH
MYOR      0 -G---LSDGEWQLVLKVWGKVEGDLPGHGQEVLIRLFKTHPETL--------EKFDKFKG
GPYL      0 g---VLTDVQVALVKSSFEEFNANIPKNTHRFFTLVLEIAPGAKd-LFSFLK--------
GPUGNI    0 ----ALTEKQEALLKQSWEVLKQNIPAHSLRLFALIIEAAPESK-yVFSFLK--------

HUMA     60 ----DLSHGSAQVKGHGKKVADALTNAVAHVD--------DMPNALSALSDLHAHKLRVD
HAOR     60 ----DLSHGSAQIKAHGKKVADALSTAAGHFD--------DMDSALSALSDLHAHKLRVD
HADK     60 ----DLSHGSAQIKAHGKKVAAALVEAVNHVD--------DIAGALSKLSDLHAQKLRVD
HBHU     60 LSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLD--------NLKGTFATLSELHCDKLHVD
HBOR     60 LSSAGAVMGNPKVKAHGAKVLTSFGDALKNLD--------DLKGTFAKLSELHCDKLHVD
HBDK     60 LSSPTAILGNPMVRAHGKKVLTSFGDAVKNLD--------NIKNTFAQLSELHCDKLHVD
MYHU     60 LKSEDEMKASEDLKKHGATVLTALGGILKKKG--------HHEAEIKPLAQSHATKHKIP
MYOR     60 LKTEDEMKASADLKKHGGTVLTALGNILKKKG--------QHEAELKPLAQSHATKHKIS
GPYL     60 -GSSEVPQNNPDLQAHAGKVFKLTYEAAIQLEVNG-AVAS--DATLKSLGSVHVSKGVVD
GPUGNI   60 -DSNEIPENNPKLKAHAAVIFKTICESATELRQKGhAVWD--NNTLKRLGSIHLKNKITD

HUMA    120 PVNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVL--------TSKYR------
HAOR    120 PVNFKLLAHCILVVLARHCPGEFTPSAHAAMDKFLSKVATVL--------TSKYR------
HADK    120 PVNFKFLGHCFLVVVAIHHPAALTPEVHASLDKFMCAVGAVL--------TAKYR------
HBHU    120 PENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANAL--------AHKYH------
HBOR    120 PENFNRLGNVLIVVLARHFSKDFSPEVQAAWQKLVSGVAHAL--------GHKYH------
HBDK    120 PENFRLLGDILIIVLAAHFTKDFTPECQAAWQKLVRVVAHAL--------ARKYH------
MYHU    120 VKYLEFISECIIQVLQSKHPGDFGADAQGAMNKALELFRKDM--------ASNYKELGFQG
MYOR    120 IKFLEYISEAIIHVLQSKHSADFGADAQAAMGKALELFRNDM--------AAKYKEFGFQG
GPYL    120 A-HFPVVKEAILKTIKEVVGDKWSEELNTAWTIAYDELAIIIKKEMKDaa-----------
GPUGNI  120 P-HFEVMKGALLGTIKEAIKENWSDEMGQAWTEAYNQLVATIKAEMKE-------------
```

Figure 6.15: Optimal trace of 10 globin protein sequences from McClure's data set. Input was generated using DIALIGN.

One could compress these gaps, but then information would be lost. On the other

hand, one can distinguish the "artifical" gaps from biological relevant gaps by a simple inspection of the alignment.

In the next example we use DIALIGN to generate the input EAG for the 18 prion sequences. The maximum trace was computed in 4296 seconds with a lower bound of 53474.2 and an optimal solution of 59862.5. This shows that we were able to improve the heuristic solution of DIALIGN considerably.

Blocks computed by LOCAL. In this approach we proceed as follows for all pairs of sequences. First we compute an optimal local alignment with affine gap costs.

```
HUMA    0 V-LSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHF-
HAOR    0 M-LTDAEKKEVTALWGKAAGHGEEYGAEALERLFQAFPTTKTYFSHF-
HADK    0 V-LSAADKTNVKGVFSKIGGHAEEYGAETLERMFIAYPQTKTYFPHF-
HBHU    0 VHLTPEEKSAVTALWGKV--NVDEVGGEALGRLLVVYPWTQRFFESFG
HBOR    0 VHLSGGEKSAVTNLWGKV--NINELGGEALGRLLVVYPWTQRFFEAFG
HBDK    0 VHWTAEEKQLITGLWGKV--NVADCGAEALARLLIVYPWTQRFFASFG

HUMA :48 DLS-----HGSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAH
HAOR :48 DLS-----HGSAQIKAHGKKVADALSTAAGHFDDMDSALSALSDLHAH
HADK :48 DLS-----HGSAQIKAHGKKVAAALVEAVNHVDDIAGALSKLSDLHAQ
HBHU :48 DLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCD
HBOR :48 DLSSAGAVMGNPKVKAHGAKVLTSFGDALKNLDDLKGTFAKLSELHCD
HBDK :48 NLSSPTAILGNPMVRAHGKKVLTSFGDAVKNLDNIKNTFAQLSELHCD

HUMA :96 KLRVDPVNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLT
HAOR :96 KLRVDPVNFKLLAHCILVVLARHCPGEFTPSAHAAMDKFLSKVATVLT
HADK :96 KLRVDPVNFKFLGHCFLVVVAIHHPAALTPEVHASLDKFMCAVGAVLT
HBHU :96 KLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALA
HBOR :96 KLHVDPENFNRLGNVLIVVLARHFSKDFSPEVQAAWQKLVSGVAHALG
HBDK :96 KLHVDPENFRLLGDILIIVLAAHFTKDFTPECQAAWQKLVRVVAHALA

HUMA :144 SKYR
HAOR :144 SKYR
HADK :144 AKYR
HBHU :144 HKYH
HBOR :144 HKYH
HBDK :144 ARYH
```

Figure 6.16: Optimal alignment of 6 globin sequences. Input was generated using LOCAL.

```
----------MLVLFVATWSDLGLCKKRPKP-GGWNTGGSRYPGQSSPGGNRYPPQs--GGWGQPHGGGWGQPHGGGWGQPHGGGWGQPHG
MVK-SHIGSWLLVLFVATWSDIGFCKKRPKPGGGWNTGGSRYPGQGSPGGNRYPPQ-GGGGWGQPHGGGWGQPHGGGWGQPHGGGWGQPHG
----------MLVLFVATWSDLGLCKKRPKP-GGWNTGGSRYPGQGSPGGNRYP----------PQGGGWGQPHGGGWGQPHGGGWGQPHG
----------MLVLFVATWSDLGLCKKRPKP-GGWNTGGSRYPGQGSPGGNRYPPQ-GGGGWGQPHGGGWGQPHGGGWGQPHGGGWGQPHG
---MANLGCWMLVLFVATWSDLGLCKKRPKP-GGWNTGGSRYPGQGSPGGNRYPPQ-GGGGWGQPHGGGWGQPHGGGWGQPHGGGWGQPHG
---MANLGCWMLVLFVATWSDLGLCKKRPKP-GGWNTGGSRYPGQGSPGGNRYPPQ-GGGGWGQPHGGGWGQPHGGGWGQPHGGGWGQPHG
---MANLGCWMLVVFVATWSDLGLCKKRPKP-GGWNTGGSRYPGQGSPGGNRYPPQ-GGGGWGQPHGGGWGQPHGGGWGQPHGGGWGQ---
---MANLGCWMLVLFVATWSDLGLCKKRPKP-GGWNTGGSRYPGQGSPGGNLYPPQ-GGG-WGQPHGGGWGQPHGGGWGQPHGGSWGQPHG
---MANLGCWMLVLFVATWSDLGLCKKRPKP-GGWNTGGSRYPGQGSPGGNRYPPQ-GGGGWGQPHGGGWGQPHGGGWGQPHGGGWGQPHG
---MANLGCWMLVLFVATWSNLGLCKKRPKP-GGWNTGGSRYPGQGSPGGNRYPPQ-GGGGWGQPHGGGWGQPHGGGWGQPHGGGWGQPHG
---MANLGCWMLVLFVATWSDLGLCKKRPKP-GGWNTGGSRYPGQGSPGGNRYPPQ-GGGGWGQPHGGGWGQPHGGGWGQPHGGGWGQPHG
---MANLGCWMLVLFVATWSDLGLCKKRPKP-GGWNTGGSRYPGQGSPGGNRYPPQ-GGGGWGQPHGGGWGQPHGGGWGQPHGGGWGQPHG
MVK-SHIGSWILVLFVAMWSDVGLCKKRPKPGGGWNTGGSRYPGQGSPGGNRYPPQ-GGGGWGQPHGGGWGQPHGGGWGQPHGGGWGQPHG
MVK-SHIGSWILVLFVAMWSDVGLCKKRPKPGGGWNTGGSRYPGQGSPGGNRYPPQ-GGGGWGQPHGGGWGQPHGGGWGQPHGGGWGQPHG
MVK-SHIGSWILVLFVAMWSDVGLCKKRPKPGGGWNTGGSRYPGQGSPGGNRYPPQ-GGGGWGQPHGGGWGQPHGGGWGQPHGGGWGQPHG
--------------------------------GGWNTGGSRYPGQGSPGGNRYPPQ-SGGTWGQPHGGGWGQPHGGGWGQPHGGGWGQPHG
---MANLSYWLLALFVAMWTDVGLCKKRPKP-GGWNTGGSRYPGQGSPGGNRYPPQ-GGGTWGQPHGGGWGQPHGGGWGQPHGGGWGQPHG
---MANLGYWLLALFVTMWTDVGLCKKRPKP-GGWNTGGSRYPGQGSPGGNRYPPQ-GGT-WGQPHGGGWGQPHGGSWGQPHGGSWGQPHG

GG--WGQ---------GGGTHNQWNKPSKPKTNMKHMAGAAAAGAVVGGLGGYMLGSAMSRPLIHFGNDYEDRYYRENMYRYPNQVYYRPV
GGg-WGQ---------GGGSHGQWGKPSKPKTNMKHVAGAAAAGAVVGGLGGYMLGSAMSRPLIHFGNDYEDRYYRENMYRYPNQVYYKPV
GG--WGQ---------GGGTHNQWNKPSKPKTNMKHMAGAAAAGAVVGGLGGYMLGSAMSRPLIHFGNDYEDRYYRENMYRYPNQVYYRPV
GG--WGQ---------GGGTHNQWHKPNKPKTSMKHMAGAAAAGAVVGGLGGYMLGSAMSRPLIHFGNDYEDRYYRENMYRYPNQVYYRPV
GG--WGQ---------GGGTHSQWNKPSKPKSNMKHMAGAAAAGAVVGGLGGYMLGSAMSRPLIHFGNDYEDRYYRENMYRYPNQVYYRPV
GG--WGQ---------GGGTHNQWHKPSKPKTSMKHMAGAAAAGAVVGGLGGYMLGSAMSRPLIHFGNDYEDRYYRENMYRYPNQVYYRPV
----------------GGGTHNQWHKPSKPKTSMKHMAGAAAAGAVVGGLGGYMLGSAMSRPLIHFGNDYEDRYYRENMYRYPNQVYYRPV
GG--WGQ---------GGGTHNQWNKPSKPKTSMKHVAGAAAAGAVVGGLGGYMLGSAMSRPLIHFGNDYEDRYYRENMYRYPNQVYYRPV
GG--WGQ---------GGGTHSQWNKPSKPKTNMKHMAGAAAAGAVVGGLGGYMLGSAMSRPIIHFGSDYEDRYYRENMHRYPNQVYYRPM
GG--WGQ---------GGGTHSQWNKPSKPKTNMKHMAGAAAAGAVVGGLGGYMLGSAMSRPIIHFGNDYEDRYYRENMYRYPNQVYYRPV
GG--WGQ---------GGGTHSQWNKPSKPKTNMKHMAGAAAAGAVVGGLGGYMLGSAMSRPIIHFGSDYEDRYYRENMHRYPNQVYYRPM
GG--WGQ---------GGGTHSQWNKPSKPKTNMKHMAGAAAAGAVVGGLGGYMLGSAMSRPIIHFGSDYEDRYYRENMHRYPNQVYYRPM
GG--WGQphggggwgq-GGTHGQWNKPSKPKTNMKHVAGAAAAGAVVGGLGGYMLGSAMSRPLIHFGSDYEDRYYRENMHRYPNQVYYRPV
GG-GWGQ----------GGSHSQWNKPSKPKTNMKHVAGAAAAGAVVGGLGGYMLGSAMSRPLIHFGNDYEDRYYRENMYRYPNQVYYRPV
GG-GWGQ----------GGTHSQWNKPSKPKTNMKHVAGAAAAGAVVGGLGGYMLGSAMNRPLIHFGNDYEDRYYRENMYRYPNQVYYRPV
GG--WSQ---------GGGTHNQWNKPSKPKTNLKHVAGAAAAGAVVGGLGGYMLGSAMSRPMLHFGNDWEDRYYRENMYRYPNQVYYRPV
GG--WGQ---------GGGTHNQWNKPSKPKTNMKHMAGAAAAGAVVGGLGGYMLGSAMSRPMMHFGNDWEDRYYRENMNRYPNQVYYRPV
GG--WGQ---------GGGTHNQWNKPSKPKTNLKHVAGAAAAGAVVGGLGGYMLGSAMSRPMIHFGNDWEDRYYRENMYRYPNQVYYRPV

DQYSNQNNFVHDCVNITIKQHTVTTTTKGENFTETDVKIMERVVEQMCITQYEKESQAYY----QRGSSMVLFSSPPVILLISFL------
DQYNNQNNFVHDCVNITVKQHTVTTTTKGENFTETDMKIMERVVEQMCVTQYQRESEAYY----QRGASAILFSPPPVILLISLLILLIVG
DQYSNQNNFVHDCVNITIKQHTVTTTTKGENFTETDVKMMERVVEQMCITQYERESQAYY----QRGSSMVLFSSPPVILLISFLI-----
DQYSNQNNFVHDCVNITIKQHTVTTTTKGENFTETDVKMMERVVEQMCITQYEKESQAYY----QRGSSMVLFSSPPVILLISFLI-----
DQYSNQNNFVHDCVNITIKQHTVTTTTKGENFTETDVKMMERVVEQMCITQYEKESQAYY----QRGSSMVFFSSPPVILLISFLIFLIVG
DQYSNQNNFVHDCVNITIKQHTVTTTTKGENFTETDVKMMERVVEQMCITQYEKESQAYY----QRGSSMVLFSSPPVILLISFLIFLIVG
DQYSNQNNFVHDCVNITIKQHTVTTTTKGENFTETDVKMMERVVEQMCITQYEKESQAYY----QRGSSMVLFSSPPVILLISFLIFLIVG
DQYSNQNNFVHDCVNITIKQHTVTTTTKGENFTETDVKMMERVVEQMCITQYERESQAYY----QRGSSMVLFSSPPVILLISFLIFLIVG
DQYSNQNNFVHDCVNITIKQHTVTTTTKGENFTETDVKMMERVVEQMCITQYERESQAYY----QRGSSMVLFSSPPVILLISFLIFLIVG
DQYSNQNNFVHDCVNITIKQHTVTTTTKGENFTETDVKMMERVVEQMCITQYERESQAYY----QRGSSMVLFSSPPVILLISFLIFLIVG
DQYSNQNNFVHDCVNITIKQHTVTTTTKGENFTETDVKMMERVVEQMCITQYERESQAYY----QRGSSMVLFSSPPVILLISFLIFLIVG
DEYSNQNNFVHDCVNITIKQHTVTTTTKGENFTETDVKMMERVVEQMCITQYERESQAYY----QRGSSMVLFSSPPVILLISFLIFLIVG
DQYSNQNNFVHDCVNITVKEHTVTTTTKGENFTETDIKMMERVVEQMCITQYQRESQAYY----QRGASVILFSSPPVILLISFLIFLIVG
DRYSNQNNFVHDCVNITVKQHTVTTTTKGENFTETDIKIMERVVEQMCITQYQRESQAYY----QRGASVILFSSPPVILLISFLIFLIVG
DQYNNQNTFVHDCVNITVKQHTVTTTTKGENFTETDIKMMERVVEQMCITQYQRESQAYY----QRGASVILFSSPPVILLISFLIFLIVG
DQYSNQNNFVHDCVNITIKQHTVTTTTKGENFTETDVKMMERVVEQMCVTQYQKESQAYY--DG-RRSSAVLFSSPPVILLISFLIFLIVG
DQYNNQNNFVHDCVNITIKQHTVTTTTKGENFTETDIKIMERVVEQMCTTQYQKESQAYY--DG-RRSSAVLFSSPPVILLISFLIFLMVG
DQYSNQNNFVHDCVNITIKQHTVTTTTKGENFTETDVKMMERVVEQMCVTQYQKESQAYYdg--RRSSSTVLFSSPPVILLISFLIFLIVG
```

Figure 6.17: Optimal alignment of 18 prion protein sequences. Input was generated using DIALIGN.

This naturally gives rise to a number of blocks by cutting the alignment at the gapped positions and taking the consecutive runs of (mis)matches as a block.

Then we continue to compute the next best local alignment between these two sequences that shares no matches or mismatches with alignments already output. We stop this procedure when the length of the local alignments falls below a given value. For a pair of sequences we now have a collection of diagonals stemming from "good" local alignments not sharing a common (mis)match. The score w_d of a diagonal d is defined as follows: Let l_d be the length of the diagonal and m_d be the number of matching residue pairs within this diagonal. Let $P'(l_d, m_d)$ be the probability that a random diagonal of the same length l_d has at least the m_d matches. Then w_d is defined to be $-\log P'(l_d, m_d)$.

Since the greedy heuristic that we used does not yield very good lower bounds, we could not solve very large problem instances to optimality. Nevertheless the procedure produced blocks of high quality as the alignment of six globin sequences in Figure 6.16 on page 100 indicates.

6.3 The SMT Problem

6.3.1 A Characterization of the SMT Problem as ILP

Recall the definition of the SMT problem on page 61. The input for the SMT problem is a structural extended alignment graph (SEAG). As structural trace score function we use a weighted sum of the weights of the alignment edges and the weights of interaction matches as described in Chapter 4.2 on page 54. Recall that M is the set of all interaction matches in G, namely

$$M = \{m_{l,r} = \{i_p, i_q, e_l, e_r\} \mid m_{l,r} \text{ is an interaction match in } G\}.$$

Given a subset M' of M let $I(M')$ be all interaction edges of the interaction matches in M', that is

$$I(M') = I \cap \bigcup_{m_{l,r} \in M'} m_{l,r}.$$

The goal is to find the structural trace $(E', I(M'))$, $E' \subseteq E$, $M' \subseteq M$ with maximal weight.

For every alignment edge $e_i \in E$ we define a binary *alignment* variable x_i indicating whether e_i is realized by the structural alignment $(E', I(M'))$ or not. Likewise, for every $m_{l,r} \in M$ we define a binary *interaction match* variable $x_{l,r}$ that indicates whether the interaction match $m_{l,r}$ is realized. The set of realized alignment edges and interaction

matches can be represented by a $|E \cup M|$-dimensional incidence vector χ^A. Let

$$\mathcal{R} := \{A = (E', M'), E' \subseteq E, M' \subseteq M \mid (E', I(M')) \text{ is a structural trace of } G\}$$

be the set of all feasible solutions. We define the *SMT polytope* of G as the convex hull of the incidence vectors of all feasible solutions, *i.e.*,

$$P_{\mathcal{R}}(G) := conv\{\chi^A \in \{0,1\}^{|E \cup M|} \mid A \in \mathcal{R}\}.$$

It is now easy to formulate the SMT problem as an integer linear program. Let w_i and $w_{l,r}$ be the score of realizing the alignment edge e_i, respectively the interaction match $m_{l,r}$. The problem

$$\max \sum_{a \in E \cup M} w_a \cdot x_a \quad \text{subject to} \quad x \in P_{\mathcal{R}}(G)$$

can then be formulated as follows:

$$\begin{aligned}
\text{maximize} \quad & \sum_{e_i \in E} w_i \cdot x_i + \sum_{m_{i,j} \in M} w_{i,j} \cdot x_{i,j} \\
\text{subject to} \quad & \sum_{e \in C} x_e \leq |C \cap E| - 1, \ \forall \text{ critical mixed cycles } C \text{ in } G \qquad (6.2) \\
& \sum_{j} x_{i,j} \leq x_i, \\
& \sum_{i} x_{i,j} \leq x_j, \ \forall \text{ interaction match variables } x_{i,j} \qquad (6.3) \\
& x_i, x_{i,j} \in \{0,1\}
\end{aligned}$$

Any solution of the above ILP corresponds to a structural trace (see Definition 4.2.1 on page 54). Given an incidence vector χ^A for $A = (E', M')$, we have to ensure that T is a trace and that the interaction edges in $I(M')$ do not conflict. These two requirements are taken care of by the mixed cycle inequalities (Equations 6.2) and the *interaction inequalities* (Equations 6.3).

The former ensure that the chosen alignment edges form a trace in G while the latter guarantee the following two properties:

1. An interaction match $m_{l,r}$ is only realized if e_l and e_r are both realized.

2. There can be no conflict between two interaction edges, because only one interaction match can "use" a specific alignment edge e_i as its left or right connecting edge.

6.3.2 The Structure of the SMT Polytope

In this section we investigate the structure of the SMT polytope. Unfortunately the pair $I_{\mathcal{R}}(G) = (E \cup M, \mathcal{R})$ does not form an independence system on $E \cup M$, because the interaction matches are dependent on the alignment edges. This deprives us of an elegant way of proving results about the SMT polytope and forces us to prove even trivial properties in a different manner. First we state some basic results about the SMT polytope and then define four non-trivial classes of valid inequalities and show in which case they are facet-defining.

Lemma 6.3.1: Let $G = (V, E, H, I)$ be a SEAG with n alignment edges and m interaction matches. Then

- $P_{\mathcal{R}}(G)$ is full-dimensional and
- the inequality $x_i \leq 1$ is facet-defining iff there is no $e_j \in E$ in conflict with e_i.

Proof. The first part of the lemma is proven by exhibiting $n + m + 1$ affinely independent incidence vectors of feasible solutions. We can easily do that by constructing n feasible solutions consisting of one alignment edge and m feasible solutions consisting of one interaction match. Together with the zero vector this yields $n + m + 1$ affinely independent incidence vectors.

To prove the second part, we assume that there is no $e_j \in E$ which is in conflict with e_i. We define $n - 1$ sets $\{e_j, e_i\}$, $\forall e_j \in E \setminus \{e_i\}$ and m sets $\{e_i\} \cup m_{l,r}$, $\forall m_{l,r} \in M$. Together with the set $\{e_i\}$ this yields $n + m$ feasible solutions A_k, $k = 1, \ldots, n + m$ whose incidence vectors χ^{A_k} are affinely independent and satisfy $\chi_i^{A_k} = 1$.

On the other hand, if there is a $e_j \in E$ which is in conflict with e_i, then for every incidence vector χ of a feasible solution, $\chi_i = 1$ would imply $\chi_j = 0$. Therefore $\dim\{x \in P_{\mathcal{R}}(G) \mid x_i = 1\} \leq n + m - 1$ and hence $x_i \leq 1$ is not a facet-defining inequality. □

Lemma 6.3.2: Let $G = (V, E, H, I)$ be a SEAG with n alignment edges and m interaction matches.

1. The inequality $x_i \geq 0$ is facet-defining iff e_i is not contained in an interaction match.
2. For each interaction match $m_{i,j}$ the inequality $x_{i,j} \geq 0$ is facet-defining.

Proof. (1) Let $e_i \in E$ be an alignment edge which is not contained in an interaction match. Then the $n - 1$ feasible solutions consisting of one alignment edge other than

e_i and the m feasible solutions consisting of one interaction match form a collection of $n+m-1$ feasible solutions A_k, $k = 1, \ldots, n+m-1$ whose incidence vectors χ^{A_k} are affinely independent. Together with the zero vector this yields $n+m$ affinely independent incidence vectors with $\chi_i^{A_k} = 0$. Therefore $x_i \geq 0$ is a facet-defining inequality. On the other hand, if $e_i \in E$ is contained in an interaction match $m_{i,j}$, then for every incidence vector χ, $\chi_i = 0$ would imply $\chi_{i,j} = 0$. Therefore $\dim\{x \in P_{\mathcal{R}}(G) \mid x_i = 0\} \leq n+m-1$ and hence $x_i \geq 0$ is not a facet-defining inequality.

(2) Let $m_{i,j} \in M$ be an interaction match. The n feasible solutions consisting of one alignment edge and the $m-1$ feasible solutions consisting of one interaction match other than $m_{i,j}$ form $n+m-1$ feasible solutions A_k, $k = 1, \ldots, n+m-1$ whose incidence vectors χ^{A_k} are affinely independent. Together with the zero vector this yields $n+m$ affinely independent incidence vectors satisfying $\chi_{i,j}^{A_k} = 0$. Therefore $x_{i,j} \geq 0$ is a facet-defining inequality. □

We now show that the interaction inequalities of the ILP formulation are facet-defining.

Theorem 6.3.1: Let $G = (V, E, H, I)$ be a SEAG. Let e_i be an alignment edge and let M_i be the set of interaction matches that contain e_i. Then the interaction inequality $\sum_{m_{i,j} \in M_i} x_{i,j} - x_i \leq 0$ is facet-defining for $P_{\mathcal{R}}(G)$.

Proof. Denote the interaction inequality by $c^T x \leq c_0$. Obviously condition 3 (b) of Theorem 2.3.2 holds because for the incidence vector χ_i of the set $\{e_i\}$ holds $c^T \chi_i < c_0$. Therefore it is sufficient to show that every valid inequality $a^T x \leq a_0$ with $\{x \mid c^T x = c_0\} \subseteq \{x \mid a^T x = a_0\}$ is – up to a multiplicative factor – equal to $c^T x \leq c_0$.

Assume that $\{x \mid c^T x = c_0\} \subseteq \{x \mid a^T x = a_0\}$. Since $c_0 = 0$ it follows that $a_0 = 0$. The incidence vectors $\chi^{\{e\}}$ of the $|E|-1$ sets $\{e\}$, $\forall e \in E \setminus \{e_i\}$ fulfill $c^T \chi^{\{e\}} = a^T \chi^{\{e\}} = 0$. Hence, $a_e = 0$, $\forall e \in E \setminus \{e_i\}$. Similarly the $|M| - |M_i|$ incidence vectors $\chi^{A_{m_{l,r}}}$ of the sets $A_{m_{l,r}} = m_{l,r}$, $\forall m_{l,r} \in M \setminus M_i$ fulfill $c^T \chi^{A_{m_{l,r}}} = a^T \chi^{A_{m_{l,r}}} = 0$. Hence, $a_{m_{l,r}} = 0$, $\forall m_{l,r} \in M \setminus M_i$.

The sets $A_{m_{i,j}} = m_{i,j}$, $\forall m_{i,j} \in M_i$ form feasible solutions whose incidence vectors $\chi^{A_{m_{i,j}}}$ satisfy $c^T \chi^{A_{m_{i,j}}} = 0$ and therefore $a^T \chi^{A_{m_{i,j}}} = 0$. If one subtracts $a^T \chi^{A_{m_{i,j}}} = 0$ from $a^T \chi^{A_{m_{i,j'}}} = 0$, $\forall m_{i,j'} \in M_i \setminus \{m_{i,j}\}$ this yields $a_{m_{i,j}} = \cdots = a_{m_{i,j'}}$, because we have just shown that all other coefficients except a_{e_i} are zero. Since $a_{m_{i,j}} = -a_{e_i}$ we can choose $\lambda = \frac{c_{m_{i,j}}}{a_{m_{i,j}}}$ which yields $\lambda \cdot a^T = c^T$. □

In the ILP formulation the mixed cycle inequalities for two sequences are of the form $x_l + x_k \leq 1$, $\forall l, k$ with e_l in conflict with e_k. They ensure that only one of the conflicting alignment edges can be realized. We can tighten these inequalities by augmenting them to sets of alignment edges and interaction matches, the members of which are mutually

in conflict. We say that an interaction match $m_{l,r}$ is in *conflict* with an alignment edge e_i, if one of its connecting edges e_l, e_r is in conflict with e_i. Two interaction matches $m_{l,r}$ and $m_{s,t}$ are in conflict if e_l or e_r is in conflict with e_s or e_t.

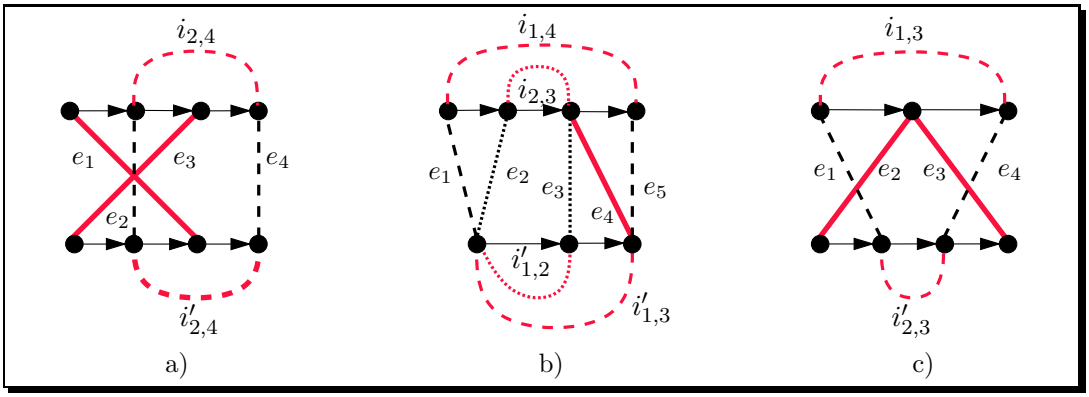

Figure 6.18: One redundant (a) and two contributing (b,c) extended cliques.

Definition 6.3.1: Let $G = (V, E, H, I)$ be a SEAG, $M' \subseteq M$ be a set of interaction matches and $E(M')$ be the set of connecting alignment edges in M'. Let $E' \subseteq E \backslash E(M')$ be a set of alignment edges in G. If each pair of elements of the set $C = E' \cup M'$ is in conflict then C is called an extended clique.

It is clear that only one element from an extended clique can be realized by a structural trace. Therefore the *extended clique* inequality $x(C) = \sum_{e_i \in E'} x_i + \sum_{m_{i,j} \in M'} x_{i,j} \leq 1$ is valid. We will prove that an extended clique inequality is facet-defining unless it is *redundant*, *i.e.*, if one can replace an interaction match by one of its connecting edges such that the resulting set is still an extended clique. If an extended clique is not redundant, we call it *contributing*. For example in Figure 6.18 (a) the set $\{e_1, e_3\} \cup m_{2,4} = \{e_2, e_4, i_{2,4}, i'_{2,4}\}\}$ builds an extended clique. However, when replacing the interaction match $m_{2,4}$ by the connecting edge e_2 this also yields an extended clique $\{e_1, e_2, e_3\}$. In Figure 6.18 (b) and (c) there is no way of replacing one of the interaction matches by a connecting alignment edge such that the resulting set is still an extended clique. In case (b) the set $m_{1,5} \cup m_{2,3} \cup \{e_4\}$ is a contributing extended clique, in case (c) the set $m_{1,4} \cup \{e_2, e_3\}$.

Theorem 6.3.2: Let $G = (V, E, H, I)$ be a SEAG. Let $C = E' \cup M'$ be a maximal contributing extended clique in G. Then the inequality $x(C) \leq 1$ is facet-defining for $P_{\mathcal{R}}(G)$.

Proof. Denote the extended clique inequality by $c^T x \leq c_0$. Condition 3 (b) of Theorem 2.3.2 holds, because for the zero incidence vector $c^T \chi_i < c_0$. Therefore it is sufficient

to show that every valid inequality $a^T x \leq a_0$ with $\{x \mid c^T x = c_0\} \subseteq \{x \mid a^T x = a_0\}$ is – up to a multiplicative factor – equal to $c^T x \leq c_0$.

Assume that $\{x \mid c^T x = c_0\} \subseteq \{x \mid a^T x = a_0\}$. All coefficients a_e of alignment edges $e \in E'$ are equal to a_0 because the set $\{e\}$ is a feasible solution. Let e be any alignment edge not in C. Then there must be an element in C such that this element and e are not in conflict; otherwise C would not be maximal or it would be redundant. There are two cases:

1. There is an alignment edge $e' \in E'$ such that e' and e are not in conflict. In that case the two sets $A = \{e'\}$ and $A' = \{e, e'\}$ build feasible solutions which satisfy $c^T \chi = c_0$ and therefore $a^T \chi = a_0$ for $\chi = \chi^A$ and $\chi = \chi^{A'}$. Subtracting $a_{e'} = a_0$ from $a_e + a_{e'} = a_0$ yields $a_e = 0$.

2. There is no alignment edge e' with the above mentioned property. Consequently there must exist an interaction match $m'_{l,r} \in M'$ that is not in conflict with e and whose connecting alignment edges e'_l and e'_r are different from e. Otherwise C would not be a maximal contributing extended clique. In this case the two sets $A = m'_{l,r}$ and $A' = m'_{l,r} \cup \{e\}$ build feasible solutions which satisfy $c^T \chi = c_0$ and therefore $a^T \chi = a_0$ for $\chi = \chi^A$ and $\chi = \chi^{A'}$. Subtracting $a_{m'_{l,r}} + a_{e'_l} + a_{e'_r} = a_0$ from $a_{m'_{l,r}} + a_{e'_l} + a_{e'_r} + a_e = a_0$ yields $a_e = 0$.

Since the coefficients of all alignment edges in $E \setminus E'$ are zero, the coefficients of interaction matches $M'_{l,r} \in M'$ are equal to a_0. Let $m_{l,r}$ be an interaction match not in M'. Then $m_{l,r}$ is a feasible solution. If one of its connecting edges is in C then the other connecting alignment edge is in $E \setminus E'$, because connecting edges cannot be in conflict. Since one of the coefficients of the connecting edges is a_0 and the other is 0 the coefficient $a_{m_{l,r}}$ has to be 0. If both connecting edges of $m_{l,r}$ are in $E \setminus E'$ then their coefficients are 0 and there are again two cases:

1. There is an alignment edge $e' \in E'$ such that e' and $m_{l,r}$ are not in conflict. In that case the two sets $A = \{e', e_l, e_r\}$ and $A' = m_{l,r} \cup \{e'\}$ build feasible solutions which satisfy $c^T \chi = c_0$ and therefore $a^T \chi = a_0$ for $\chi = \chi^A$ and $\chi = \chi^{A'}$. Subtracting $a_{e'} + a_{e_l} + a_{e_r} = a_0$ from $a_{e'} + a_{m_{l,r}} + a_{e_l} + a_{e_r} = a_0$ yields $a_{m_{l,r}} = 0$.

2. There is no alignment edge e' with the above mentioned property. Consequently there must exist an interaction match $m_{l',r'} \in M'$ that is not in conflict with $m_{l,r}$ and whose connecting alignment edges are different from e_l and e_r. In this case the two sets $A = m_{l',r'} \cup \{e_l, e_r\}$ and $A' = m_{l',r'} \cup m_{l,r}$ build feasible solutions which satisfy $c^T \chi = c_0$ and therefore $a^T \chi = a_0$ for $\chi = \chi^A$ and $\chi = \chi^{A'}$. Subtracting $a_{m_{l',r'}} + a_{e_{l'}} + a_{e_{r'}} + a_{e_l} + a_{e_r} = a_0$ from $a_{m_{l',r'}} + a_{e_{l'}} + a_{e_{r'}} + a_{e_l} + a_{e_r} + a_{m_{l,r}} = a_0$ yields $a_{m_{l,r}} = 0$.

This completes the proof that the coefficients of edges not in C are 0. The rest of the coefficients is equal to a_0. Choosing $\lambda = \frac{a_0}{c_0}$ we have $a_0 = \lambda \cdot c_0$ and $a^T = \lambda \cdot c^T$. Therefore $c^T x \leq c_0$ is a facet-defining inequality for $P_{\mathcal{R}}(G)$. □

The above mentioned classes of facet-defining inequalities do not form a complete description of the SMT polytope for two sequences. It is indeed an open question to find such a complete description. We were able to identify another class of inequalities that is not always facet-defining, the so called *odd cycle inequalities*. In the following we characterize this class and prove that it is facet-defining for certain SEAGs. All indices are to be read modulo $2i + 2$.

Definition 6.3.2: Let $G = (V, E, H, I)$ be a SEAG containing $2i + 1$ contributing extended cliques $C_1, \ldots, C_{2i+1}$. The set $C := C_1 \cup C_2 \cup \cdots \cup C_{2i+1}$ is called odd cycle of length i if for all $f \in C_j$, $1 \leq j \leq 2i + 1$ holds: f is in conflict with each $g \in C_{j-1} \cup C_j \setminus \{f\} \cup C_{j+1}$ and not in conflict with some $g \in C_k$, $k \neq j - 1, j, j + 1$.

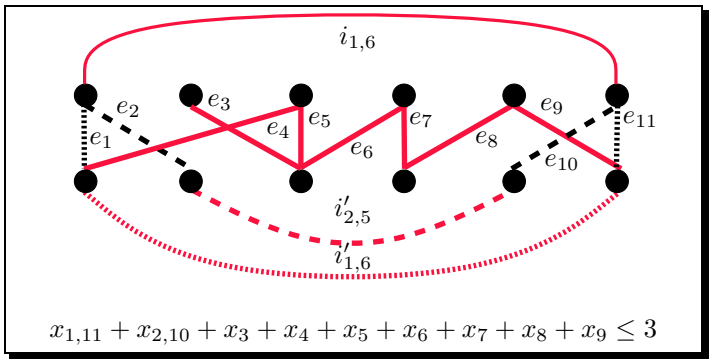

Figure 6.19: A SEAG forming an odd cycle of length 3.

Given a SEAG consisting of an odd cycle C of length i, only i elements in C can be realized simultaneously, namely one out of every other extended clique. Therefore for any odd cycle C the *odd cycle inequality*

$$x(C) = \sum_{m_{l,r} \in C \cap M} x_{l,r} + \sum_{e_j \in C \cap E} x_j \leq i$$

is valid. Note that an odd cycle must contain at least one interaction match.

In Figure 6.19 an odd cycle of length 3 is shown. More specifically $C_1 = \{m_{1,10}, m_{2,9}\}$, $C_2 = \{e_3\}$, $C_3 = \{e_4\}$,..., $C_6 = \{e_7\}$ and $C_7 = \{e_8\}$.

The odd cycle inequality is indeed facet-defining, if the input SEAG G consists of an odd cycle together with its connecting alignment edges. We prove this in the following theorem using the notations from Definition 6.3.2.

Theorem 6.3.3: Let $G = (V, E, H, I)$ be a SEAG consisting of an odd cycle C. Then the odd cycle inequality

$$x(C) = \sum_{m_{l,r} \in C \cap M} x_{l,r} + \sum_{e_j \in C \cap E} x_j \leq i$$

is facet-defining for $P_{\mathcal{R}}(G)$.

Proof. Denote the odd cycle inequality by $c^T x \leq c_0$. Clearly condition 3 (b) of Theorem 2.3.2 holds. If we realize any element contained in one of the extended cliques of an odd cycle, we have a feasible solution, the incidence vector of which fulfills $c^T \chi_i < c_0$. Hence, it is sufficient to show that every valid inequality $a^T x \leq a_0$ with $\{x \mid c^T x = c_0\} \subseteq \{x \mid a^T x = a_0\}$ is – up to a multiplicative factor – equal to $c^T x \leq c_0$.

Assume that $\{x \mid c^T x = c_0\} \subseteq \{x \mid a^T x = a_0\}$. Throughout the proof d_j denotes either an interaction match in C_j or an alignment edge in C_j.

First we show that the coefficients of all connecting alignment edges in G are zero. Let $m_{l,r}$ be an interaction match contained in some contributing extended clique C_j. Since $m_{l,r}$ is in conflict with all elements in C_{j-1} (resp. C_{j+1}) and with all elements in C_{j+1} (resp. C_{j-1}) it follows from the definition of conflict that e_l is in conflict with all elements in C_{j-1} and e_r is in conflict with all elements in C_{j+1}. Additionally there must exist an element in C_{j+1} that is not in conflict with e_l and an element in C_{j-1} that is not in conflict with e_r. If this was not true, C_j would be a redundant extended clique in the odd cycle, because one could replace $m_{l,r}$ by its left or right connecting edge.

Therefore there exist sets $L_j := d_{j-2} \cup d_{j+1} \cup d_{j+3} \cup \cdots \cup d_{j-4}$ and $R_j := d_{j-1} \cup d_{j+2} \cup \cdots \cup d_{j-3}$ that form feasible solutions satisfying $c^T \chi = c_0$ and hence $a^T \chi = a_0$. Also the sets $L_j \cup \{e_l\}$ and $R_j \cup \{e_r\}$ form feasible solutions satisfying $c^T \chi = c_0$ and hence $a^T \chi = a_0$. The subtraction of the two equalities yields $a_{e_l} = a_{e_r} = 0$. Since the above argument holds for any interaction match in C it follows that the coefficients of all connecting alignment edges in G are zero.

Next we show that the coefficients of all variables in an odd cycle inequality are equal. Define for $k = 1, \ldots, i+1$ the sets

$$M_k := d_2 \cup d_4 \cup \cdots \cup d_{2k-2} \cup d_{2k+1} \cup d_{2k+3} \cup \cdots \cup d_{2i+1}$$

that means M_k contains one element from each extended clique with even indices from 2 to $2k-2$ and one element from each extended clique with odd indices from $2k+1$ to $2i+1$. Every M_k forms a feasible solution and its incidence vector χ^{M_k} satisfies $c^T \chi^{M_k} = c_0$ and hence $a^T \chi^{M_k} = a_0$. Subtracting $a^T \chi^{M_{k+1}} = a_0$ from $a^T \chi^{M_k} = a_0$ yields $a_{d_{2k+1}} = a_{d_{2k}}$ for $k = 1, \ldots, i$.

Next we define for $k = 0, \ldots, i$ the sets

$$N_k := d_1 \cup d_3 \cup \cdots \cup d_{2k-1} \cup d_{2k+2} \cup d_{2k+2} \cup \cdots \cup d_{2i}$$

that means N_k contains one element from each extended clique with odd indices from 1 to $2k-1$ and one element from each extended clique with even indices from $2k+2$ to $2i$. Every N_k forms a feasible solution and its incidence vector χ^{N_k} satisfies $c^T\chi^{N_k} = c_0$ and hence $a^T\chi^{N_k} = a_0$. Subtracting $a^T\chi^{N_{k+1}} = a_0$ from $a^T\chi^{N_k} = a_0$ yields $a_{d_{2k+2}} = a_{d_{2k+1}}$ for $k = 0, \ldots, i-1$.

Putting the above arguments together we have $a_{d_1} = a_{d_2} = \cdots = a_{d_{2i+1}}$. Since the argument holds for any alignment edge $e \in C_i$ and any interaction match $m_{l,r} \in C_i$ the coefficients in the odd cycle C are equal. Choosing $\lambda = \frac{a_0}{c_0}$ we have $a_0 = \lambda \cdot c_0$ and $a^T = \lambda \cdot c^T$. Therefore $c^T x \leq c_0$ is a facet-defining inequality for $P_{\mathcal{R}}(G)$. □

In the multiple sequence case we show that the *mixed cycle* inequalities are facet-defining for $P_{\mathcal{R}}(G)$ if and only if they contain no chord.

Lemma 6.3.3: Let $G = (V, E, H, I)$ be a SEAG and C be a critical mixed cycle with $|C \cap E| = \ell$. Then the inequality

$$x(C \cap E) \leq \ell - 1$$

defines a facet of $P_{\mathcal{R}}(G)$ if and only if C has no chord.

Proof. Assume that C is a critical mixed cycle with $|C \cap E| = \ell$ and without chord. Let $e_1, \ldots, e_\ell$ be ℓ edges on C. We obtain ℓ different feasible solutions by removing the edge e_i, $1 \leq i \leq \ell$ from C. The incidence vectors of these solutions are linearly independent and satisfy $x(C \cap E) = \ell - 1$. Since C has no chord and is a critical mixed cycle we can add either an interaction match $m_{l,r} \in M$ or an alignment edge $e \in E \setminus C$ to one of the above solutions without introducing a mixed cycle in G. This yields another $m + n - \ell$ vectors that fulfill $x(C \cap E) = \ell - 1$. Moreover, the incidence vectors of all sets constructed above are linearly independent. Thus $x(C \cap E) \leq \ell - 1$ is a facet-defining inequality.

On the other hand, if C has a chord e then each incidence vector χ^A of a solution $A \subseteq E \cup B$ satisfying $x(C \cap E) = \ell - 1$ has to satisfy $\chi^A_e = 0$, so $\dim\{x \in P_{\mathcal{R}}(G) | x(C \cap E) = \ell - 1\} \leq |E| + |M| - 2$. Thus $x(C \cap E) \leq \ell - 1$ is not a facet-defining inequality. □

6.3.3 Bounds for the SMT Problem

Computation of Lower Bounds

In order to compute a lower bound for the SMT problem we compute an optimal conventional alignment with affine gap costs without considering the annotations of both sequences. This alignment realizes some interaction matches and can be considered as a structural alignment. The score of the structural alignment is taken as a lower bound in the branch-and-cut algorithm.

Computation of Upper Bounds

In order to specialize the generic branch-and-cut algorithm for the SMT problem we need to describe separation algorithms for the classes of inequalities described in Chapter 6.

Mixed Cycle Inequalities. The computation of mixed cycle inequalities is done in the same way as for the GMT. The only thing to note is that in the SMT formulation we deal with a singleton partition of the alignment edges.

Extended Clique Inequalities. Recall that all maximal extended cliques C that do not contain an interaction match are contributing and therefore the extended clique inequality of C is facet-defining for the SMT polytope. For the GMT problem we showed that those inequalities can be separated in polynomial time by computing a longest path in a pairgraph.

The other maximal extended cliques contain at least one interaction match and are by Theorem 6.3.2 facet-defining if and only if they are contributing. Unfortunately we have not found an efficient way of separating this class of inequalities. We chose two ways of handling this situation.

1. If the SEAG is not too dense, we enumerate all maximal extended cliques containing a fractional interaction match variable $x_{i,j}$. With the use of bit vectors and an adaption of an algorithm by Tsukiyama et al. (Tsukiyama, Ide, Ariyoshi, and Shirawaka 1977) this can be done in reasonable time for up to 20000-30000 cliques. Adding these clique inequalities to the current linear program cuts off the infeasible solution and considerably shrinks the enumeration tree in the branching phase.

2. If the SEAG is too dense and therefore the number of cliques in the above mentioned enumeration explodes, we refrain from enumerating the extended cliques (which we do in most cases).

Odd Cycle Inequalities. For separating the odd cycle inequalities we developed a heuristic procedure that works in two phases. In the first phase we solve the separation problem for a subset of all possible odd cycle inequalities by a dynamic programming procedure. If we find a violated inequality in that subset we try to strengthen the inequality by a heuristic procedure in the second phase of the heuristic.

In the first phase we look for a fractional interaction match variable $x_{l,r}$ representing that interaction match $\{i_p, i_q, e_l, e_r\}$. Then we compute a *maximal value linking chain* for that interaction match. A linking chain for a given interaction match is an ordered list of alignment edges, such that the first edge in the chain is in conflict with the left connecting edge e_l and the last edge with the right connecting edge e_r. We then say that the linking chain for e_l and e_r *starts* with the first edge. Every other edge in the chain is exactly in conflict with its two neighboring edges in the chain. The value of a linking chain is the sum of the corresponding variable values in the current LP solution divided by the number of edges in the chain. The linking chain together with the interaction match forms an odd cycle if there is an even number of edges in the chain. For example in Figure 6.20 the set $\{e_4, e_5, e_6, e_7, e_8, e_9\}$ is a linking chain for the interaction matches $\{i_{1,6}, i'_{1,6}, e_1, e_{11}\}$ and $\{i_{1,6}, i'_{2,5}, e_3, e_{10}\}$. We compute a chain

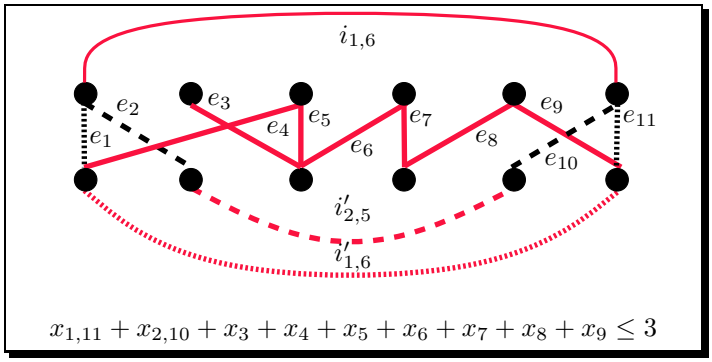

Figure 6.20: A SEAG containing an odd cycle of length 3.

of maximum value by dynamic programming. The following three arrays are used.

- $suc(e_i)$ stores the variable number of the edge right of e_i in the maximal linking chain for e_l and e_r.
- $num(e_i)$ stores the number of variables right of e_i in the maximal linking chain

for e_l and e_r.

- $val(e_i)$ stores the sum of the variable values of the edges right of e_i in the maximal linking chain for e_l and e_r.

For all edges f that are in conflict with e_r and not with e_l we initialize the arrays with $suc(f) = undef$, $num(f) = 1$ and $val(f) = x_f$.

Then we compute two lists of edges, the x-list and the y-list. In the x-list are edges e with $start(e)$ smaller or equal than $start(e_l)$ and $end(e)$ greater than or equal $end(e_l)$ that are not in conflict with e_r. In the y-list are edges e with $start(e)$ greater than or equal $start(e_l)$ and $end(e)$ smaller than or equal $end(e_l)$ that are not in conflict with e_r.

Consider for example interaction match $\{i_{1,6}, i'_{1,6}, e_1, e_{11}\}$ in Figure 6.20 on the preceding page. Then the x-list contains the edge e_2 while the y-list contains e_4. In order to illustrate our heuristic we assume that we are given the following fractional solution ($x_1 = 0.5, x_2 = 0, x_3 = 0, x_4 = 0.5$,$x_5 = 0.5, x_6 = 0.5, x_7 = 0.5, x_8 = 0.5$, $x_9 = 0.5, x_{10} = 0, x_{11} = 0.5$, $x_{1,11} = 0.5, x_{2,10} = 0$). This solution fulfills the ILP inequalities and violates the odd cycle inequality $x_{1,10} + x_{2,11} + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 \leq 3$.

The edges in these two lists are possible starting points of a maximal linking chain. We now recursively compute the maximal value linking chain starting with an edge in the x-list or y-list.

For an edge f in the x-list ($\{e_2\}$ in the example) we search the conflicting edges g with $end(e_l) < end(g) \leq end(f)$, *i.e.*, g is in conflict with f and not with e_l (in our example there is no such edge). Then we recursively compute the maximal linking chain for g and e_l starting in the y-list.

For an edge f in the y-list ($\{e_4\}$ in our example) we proceed analogously. We search the edges g such that $start(e_l) < start(g) \leq start(f)$, *i.e.*, g is in conflict with f and not with e_l (in our example e_3 and e_5 fulfill that condition). Then we recursively compute the maximal linking chain for g and e_r starting in the x-list.

Having recursively computed the maximal linking chain starting with each of the above edges ($\{e_3, e_6, e_7, e_8, e_9\}$ and $\{e_5, e_6, e_7, e_8, e_9\}$ in the example), we can compute the maximal linking chain starting with f as follows. We pick from all the edges g for which the quotient $val(g)/num(g)$ is maximal (in our example the quotient for the first chain is 2/5 and for the second chain 2.5/5, hence we choose the second). We then set $val(f) = val(g) + x_f$, $num(f) = num(g) + 1$ and $suc(f) = g$ (in our example $val(e_4) = val(e_5) + x_f = 2.5 + 0.5$, $num(e_4) = num(e_5) + 1 = 6$ and $suc(e_4) = e_5$) so that we can retrieve the maximal linking chain by following the pointers.

By choosing the edge f in the x-list or y-list that has the maximal quotient we can check whether we have found a violated odd cycle inequality by testing whether $x_{l,r}+val(f) > num(f)/2$ (in our example we find that $x_{1,11} + val(e_4) = 3.5 > 3 = num(e_4)/2$).

We now could add the violated, valid inequality ($x_{1,10} + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 \leq 3$ in the example) as solution of the heuristic separation procedure to the LP and solve it again. However, this is not an odd cycle inequality for the SEAG in Figure 6.20 on page 112. It is easy to see that setting x_5 to 0 and x_3 to 0.5 results again in a fractional solution where the above inequality is not violated.

As a remedy, we try to tighten the violated inequality as follows. For each edge e in the linking chain we compute the extended clique with the maximal number of objects (note that an extended clique can contain alignment edges and interaction matches) that contains e and with the property that all objects in the clique are in conflict with the two neighboring edges in the linking chain.

To do this we compute the maximal extended cliques containing e (for example $\{e_3, e_5, e_6\}$ is a maximal clique containing e_5). Each maximal extended clique c is intersected with the set of all objects that is in conflict with the two neighboring edges in the linking chain (for example $\{e_3, e_5, e_6\}$ is intersected with $\{e_1, e_2, e_3, e_5\}$ and $\{e_3, e_5, e_7\}$ yielding the set $\{e_3, e_5\}$). We choose the maximal cardinality intersection $c(e)$ and regard it as a preliminary extended clique in an odd cycle. We do this for each edge in the linking chain and for the interaction match $m_{l,r}$.

Clearly the sets $c(e)$ not necessarily fulfill the conditions of the odd cycle definition. Therefore we cycle through the maximal linking chain and the interaction match and enforce locally the odd cycle definition, which means that we delete from the current set $c(e)$ all objects that are not in conflict with all objects in the neighboring extended cliques. We iterate this procedure in a round robin fashion until none of the extended cliques changes anymore for one round.

Still, this is not sufficient, because an object in $c(e)$ may not only be in conflict with all objects in the neighboring extended cliques, but also with objects in other extended cliques. In order to ensure that $c(e)$ contains only edges that are exactly in conflict with edges in the neighboring extended cliques we subtract the intersection between $c(e)$ and the conflict set of every object in the maximal value linking chain that is not a neighbor of e.

In our example this procedure finds indeed the odd cycle in Figure 6.20 on page 112 and instead of adding the weak inequality $x_{1,10} + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 \leq 3$ to the LP we can add the odd cycle inequality $x_{1,10} + x_{2,11} + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 \leq 3$ to the LP.

Separation and Branching Strategy

By using the pairgraph we can exactly separate all extended cliques that contain no interaction match. Since the interaction inequalities are part of the initial LP this implies that an integer solution is feasible.

If we cannot find a violated extended clique inequality and our solution is still fractional we apply the heuristic separation routine for finding violated odd cycle inequalities. If this also fails we branch. In the branching phase we choose the fractional interaction match variable that is closest to 0.5 and has the highest objective function coefficient.

6.3.4 Computational Results for the SMT Problem

In this section we report on the results of our computational experiments computed. We implemented the branch-and-cut algorithm in C++ using the library of efficient data types and algorithms LEDA (Mehlhorn and Näher 1995) and the branch-and-cut framework ABACUS (Jünger and Thienel 1997).

Our test sequences consist of 23S ribosomal RNA sequences from the Antwerpen rRNA database (de Rijk, de Peer, Chapelle, and de Wachter 1994). We used sequences from the small ribosomal subunit which are approximately 1500 bases long. According to the notation used in this database the following symbols are used to denote nucleotides:

- Completely identified nucleotides.
  `U,C,A,G` standing for (U)racil, (C)ytosine, (A)denine and (G)uanine.
- Partially identified nucleotides.
  - `Y` : stands for `U` or `C`.
  - `R` : stands for `A` or `G`.
  - `M` : stands for `A` or `C`.
  - `K` : stands for `U` or `G`.
  - `W` : stands for `U` or `A`.
  - `S` : stands for `G` or `C`.
  - `B` : stands for `U`, `C` or `G`.
  - `D` : stands for `U`, `A` or `G`.
  - `H` : stands for `U`, `C` or `A`.
  - `V` : stands for `C`, `A` or `G`.
- We did not consider sequences with unidentified nucleotides.

Apart from the sequence information, the Antwerpen database contains the secondary structure of the sequences in form of special characters that are intertwined with the sequence characters and indicate the beginning or end of complementary strands of a helix (also called stem, see Section 3.2). In addition there is a "helix numbering" string naming the strands of a helix. Basically this corresponds to the linear representation introduced in Section 3.2 (see Figure 3.8 on page 40) where the edges joining the complementary base pairs are implicitely given by the special symbols which are:

- `[` and `]`: symbolizes the beginning respectively the end of one strand of a helix.
- `^` : symbolizes in one character the simultaneous end of one strand and beginning of another, *i.e.*, `^=][`.
- `{` and `}`: symbolizes beginning and end of an internal loop or bulge loop interrupting a helix strand.
- `(` and `)` : encloses a base forming a non-standard base-pair (any pair other than `G-C,A-U or G-U`).

We adapted that notation to output a pairwise structural alignment as follows (see Figure 6.21 on the next page). The output starts with some statistical information for each of the two sequences as there is:

- the name and length of the sequence,
- the number of interactions in its annotation (resp. secondary structure),
- the number of helices in the secondary structure,
- the alignment score (in brackets the sequence similarity score and the interaction match score is given).

Then there follow four strings with a line break after a certain number of characters. The first and second string belong to the first sequence, while the third and the fourth belong to the second sequence.

The first string contains the helix numbering for the first sequence (and the fourth string for the second). For each number x that symbolizes one strand of a helix there is also a number x' that symbolizes the complementary strand of that helix. The helix itself is therefore denoted by $x : x'$.

The second string contains the first sequence itself (the third string the second sequence) intertwined with the above mentioned special symbols `(,),[,],{,},^`. A number in the helix numbering string starts always at the same position at which the symbol `[` or

```
Alignment of sequences
  0.Desulfurococcus mobilis (length = 1495)
  1.Halobacterium halobium (length = 1473)
Sequence 0 has secondary structure
# interacts. : 458
   # helices : 51
Sequence 1 has secondary structure
# interacts. : 429
   # helices : 51
Alignment has value 11562(4698,6864)
- - -1- - - - - - - -2- - - -1' - - - - -3- - - - - - - - - - -4-
A C U[C C G G U]U G A[U C C U^G C C G G]U[C C C G A C C G C U]A[U
A U U[C C G G U]U G A[U C C U^G C C G G]A[G G C C A U U G C U]A[U
- - -1- - - - - - - -2- - - -1' - - - - -3- - - - - - - - - - -4-

 - - - - - - - - - - - - - -5- - - - - - - -6- - - - - - - - -
 C G G G G U G G]G G C U A A[G C C{A}U G G]G A[G U C{G C}A C G C
 C G G A G U C C]G A U U U A[G C C{A}U G C]U A[G U U{G U}G C G -
 - - - - - - - - - - - - - -5- - - - - - - -6- - - - - - - - -

- - - - - - - -6' - - - - - - - - - - - - - - - -7- - - - - -
U C C G C]C G C U[G C G G G G C G U G G C]G G A C G G[C U G]A G U
- - - - G]- G U U[U{A G A C C}C G C A G C]G G A A A G[C U C]A G U
- - - - - - - -6' - - - - - - - - - - - - - - - -7- - - - - -
```

Figure 6.21: Part of a structural alignment output.

^ is written in the second string. Consider for example the first row of the alignment in Figure 6.21. One strand of helix $1 : 1'$ starts after the third non blank character of the second string (there is a [right behind that character and the [begins at the same position as the 1 in the helix numbering string). The complementary strand $1'$ starts after the fifteenth non blank character of the second string (there is a ^ right behind that character). Note that ^ symbolizes the end of one strand (here 2) and the beginning of another strand (here $1'$). The characters in complementary strands form complementary base pairs, the first character of x with the last of x', the second of x with the second last of x' and so forth. That means that in the above alignment the first sequence contains in its secondary structure, among others, the interactions $(3, 19), (4, 18), (5, 17), (6, 16), (7, 15)$. In helix $6 : 6'$ there is an internal loop symbolized by { and }. The characters within these brackets do not form an interaction with

another character.

Generating the Input

Recall that the input for the SMT problem is a structural extended alignment graph $G = (V, E, H, I)$. In our experiments we restricted ourselves to pairwise structural traces. For every pair of sequences we need to describe how we construct the set I of interaction edges and the set E of alignment edges. Once we have specified the SEAG (and therefore the set M of interaction matches) we only need to give a structural trace score function to run our algorithm.

Generating the interaction edges. In our experiments we tested the variant of structural sequence alignment where the secondary structure of the first sequence is known while the secondary structure of the second sequence is unknown. That means that the annotation of the first sequence is the secondary structure as given by the Antwerpen RNA database while the annotation of the second sequence contains all possible standard interactions (`A-U,C-G` and `G-U`).

Generating alignment edges. We have to determine a set of reasonable alignment edges. In principle, we use for this purpose alignment edges realized by some suboptimal alignment, *i.e.*, an alignment with a score close to optimal. In contrast to Lenhof, Reinert, and Vingron (1998), we do not take all the edges realized by any suboptimal alignment scoring better than a fixed threshold s below the optimal. Rather we employ a windowing technique to make the alignment graph denser in certain regions and thinner in others. The reason for applying the windowing technique described below is

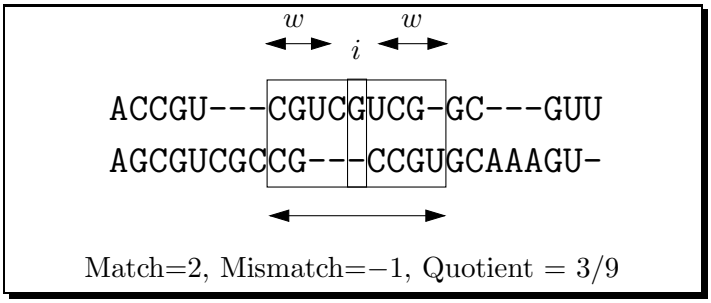

Figure 6.22: Calculation of quotient at position i with window of width 4

that conventional suboptimal alignments have frequently shown insufficient deviation from the optimal alignment to cover the alignment edges necessary to build the struc-

turally correct alignment. On the other hand, upon inclusion of a sufficient number of suboptimal alignments the number of edges to consider became too large.

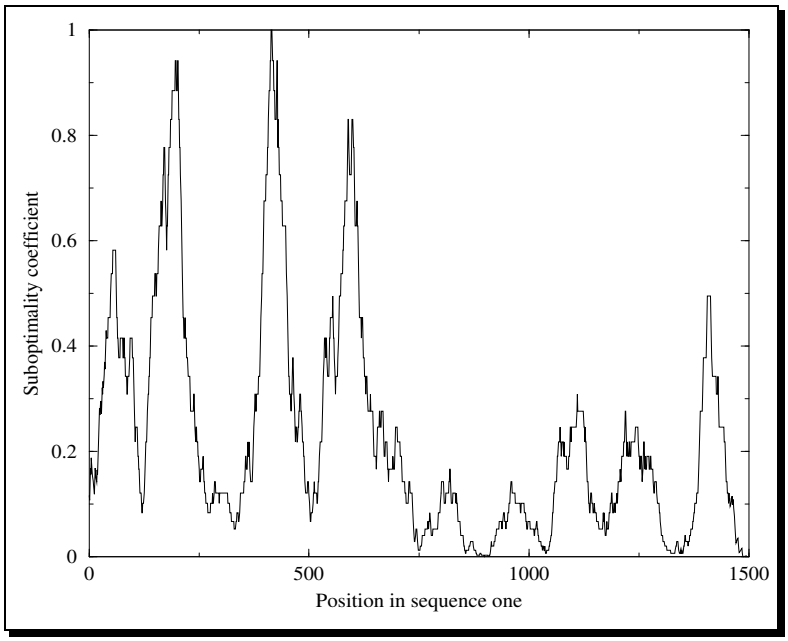

Figure 6.23: Plot of $c(i)$ for an optimal conventional alignment.

As a remedy we designed a windowing technique that adjusts the suboptimality cutoff according to the local quality of an alignment. Where the alignment appears to be very good no suboptimal alternatives are considered. In alignment regions showing little sequence conservation more suboptimal alternatives are taken into account.

We proceed as follows: For a given optimal conventional alignment we compute for each position i in the first sequence, say, an coefficient $q(i)$. Let $a(i)$ be the position of the i-th character of the first sequence in the alignment and n_1 be the length of the first sequence. Then, for a given window size w, we sum the mutation score matrix values of the aligned characters from alignment position $\max\{0, a(i) - w\}$ to alignment position $\min\{a(l), a(i) + w\}$ and divide it by the length of the window. This *quality coefficient* $q(i)$ is a measure for the local quality of the optimal alignment at sequence position i. See Figure 6.22 on the preceding page for an example with window width 4.

Now we compute a *suboptimality coefficient* $c(i)$ as follows: First we normalize $q(i)$ to a value $p(i)$ between 0 and 1 and then define $c(i) = (1-p(i))^2$. The coefficient $c(i)$ is near 0 in regions where the alignment is reliable and near 1 in regions where it is not reliable. Finally, we insert in the SEAG all alignment edges at position i that are realized by some alignment that is at most $c(i) \cdot s$ worse than the optimal conventional alignment, where

the parameter s is the (maximal) suboptimality. Figure 6.23 on the preceding page

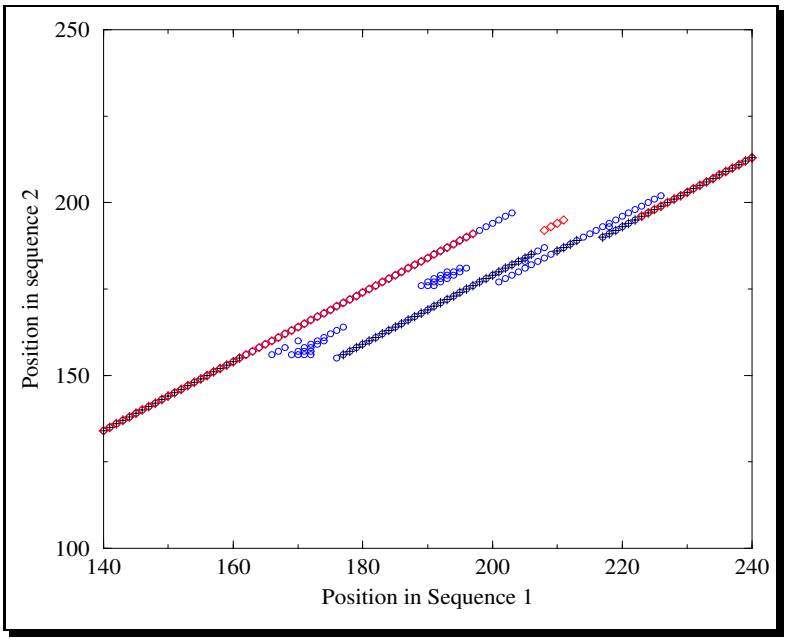

Figure 6.24: Dot plot to show effectivness of $c(i)$ computation.

shows a plot of $c(i)$ for the optimal conventional alignment between Desulfurococcus Mobilis (1495 nucleotides) and Halobacterium Halobium (1473 nucleotides). One can see three prominent peaks where indeed our experiments show that the conventional alignment is wrong and hence we need to insert more alignment edges into the SEAG in the hope that the correct ones are contained.

Figure 6.24 shows the effectiveness of this approach. We use a dot plot representation to overlay the correct alignment with the optimal conventional alignment and the all possible matches in the SEAG. A mark at position (i, j) in the plot indicates, that the i-th character of the first sequence is aligned with the j-th character of the second sequence. The more (mis)matches of an alignment coincide with the (mis)matches from the correct alignment, the better the alignment.

Figure 6.24 shows the region between characters 140 and 240 of the first sequence (Desulfurococcus Mobilis). The black plus sign indicate (mis)matches occurring in an optimal conventional alignment, the red diamonds indicate (mis)matches occurring in the correct alignment. A blue circle at position i indicates a (mis)match occurring in some suboptimal alignment that is at most $c(i) \cdot s$ worse than the optimal conventional alignment.

In the displayed region the suboptimality coefficient $c(i)$ has its first peak (see Fig-

ure 6.23 on page 119) indicating a possibly unreliable conventional alignment. Indeed, the optimal conventional alignment (black plus sign) deviates considerably from the correct alignment (red diamonds). However, since $c(i)$ is near 1.0 in this region, the alignment edges contained in the SEAG build a "cloud" covering all but a few correct (mis)matches. Note that this is essential for the branch-and-cut algorithm to identify the correct structural alignment. Hence, our windowing strategy is an effective way to limit the number of alignment edges in the SEAG and therefore the number of variables in the ILP formulation, while inserting enough alignment edges to guarantee good results.

Assessing the Quality of the Results

The given secondary structure should direct the optimal alignment towards the detection of conserved structural patterns. For reasons of convenience we call the structural alignment that realizes the optimal structural trace the *structural alignment* and the optimal conventional alignment simply the *conventional alignment*. Both alignments realize a certain number of interaction matches, which in turn determines their secondary structure. We used three ways of assessing the quality of a structural alignment compared to the quality of a conventional alignment.

- The first is the number of realized interaction matches. We compare it to the number of interaction matches of the correct alignment given in the database. Since the second annotation in the input SEAG contains only standard interactions, we can compute only interaction matches, with a standard interaction in the second annotation. Hence, we assign a positive score only to such interaction matches. It should be noted, that in the correct alignment occasionally occur other interaction matches. Generally, the more interaction matches we realize, the better the alignment.

- The second is the comparison of the sequence alignment from the database (which we assume to be the "correct" alignment) with the structural and conventional alignment. We use two ways to display the quality of the alignment in selected regions. The alignment itself as displayed in Figure 6.21 on page 117 and a dot plot representation as described above to overlay the correct alignment with either the conventional alignment or the structural alignment. A mark at position (i, j) in the plot indicates, that the i-th character of S_1 is aligned with the j-th character of S_2. The more (mis)matches of an alignment coincidence with the (mis)matches from the correct alignment, the better the alignment.

- The third way is the alignment score. For each alignment (correct, conventional, and structural) we compute their score. Since our structural trace score function

is a combination of a score for the sequence similarity and for the structural similarity, we give both parts of the score. The hope is that a better overall score can be achieved by realizing more interaction matches while sacrificing some sequence similarity.

Results

We initially tested the algorithm on small problem instances like tRNA alignments and alignments of 5S RNA sequences. For the cases studied the algorithm reproduced the correct alignments. Here we want to present more challenging examples of 23S ribosomal RNA sequences from the Antwerpen rRNA database (de Rijk, de Peer, Chapelle, and de Wachter 1994). We computed a structural optimal trace for all pairs of the

# Seq.	Family	Sequence	Length	# inter.
1	Crenarchaeota	Acidianus Brierley	1492	438
2	Crenarchaeota	Acidianus Sp	1493	441
3	Crenarchaeota	Desulfurococcus Mobilis	1495	458
4	Crenarchaeota	Pyrodictium Occultum	1497	446
5	Crenarchaeota	Sulfolobus Acidocaldarius	1493	446
6	Crenarchaeota	Thermofilum Pendens	1509	444
7	Euryarchaeota	Archaeoglobus Fulgidus	1492	433
8	Euryarchaeota	Halobacterium Halobium	1473	429
9	Euryarchaeota	Halococcus Morrhuae	1475	426
10	Euryarchaeota	Haloferax Denitrificans	1469	418
11	Euryarchaeota	Methanobacterium Formicium	1476	430
12	Euryarchaeota	Natronobacterium Magadii	1466	412

Figure 6.25: Set of test sequences

sequences shown in Figure 6.25. Note that the alignments are not symmetric, because we use the known secondary structure of the first sequence while we allow any standard interaction in the annotation of the second sequence. We used 6 sequences from each of two different families (Crenarchaeota and Euryarchaeota) of archaebacteria. For these twelve sequences we build 132 test sets which we number as an ordered pair. The test set (i, j) corresponds to an SEAG for the i-th and j-th sequence that is constructed as described in the previous section.

For each test set we construct a SEAG as described above, *e.g.*, for test set $(3, 11)$ the SEAG constructed for the sequences Desulfurococcus Mobilis and Halobacterium Halobium, where for the first sequence the secondary structure is given, while the

-	0														
A	0	4													
U	0	1	4												
C	0	1	1	4											
G	0	1	1	1	4										
Y	0	1	2	2	1	4									
R	0	2	1	1	2	1	4								
M	0	2	1	2	1	2	2	4							
K	0	1	2	1	2	2	2	1	4						
W	0	2	2	1	1	2	2	2	2	4					
S	0	1	1	2	2	2	2	2	2	1	4				
B	0	1	2	2	2	2	1	1	2	1	2	4			
D	0	2	2	1	2	1	2	1	2	2	1	2	4		
H	0	2	2	2	1	2	1	2	1	2	1	2	2	4	
V	0	2	1	2	2	1	2	2	1	1	2	2	2	2	4
	-	A	U	C	G	Y	R	M	K	W	S	B	D	H	V

Figure 6.26: Mutation score matrix used for experiments.

annotation of the second contains all standard base pairs. The conventional alignment is computed with a dynamic programming algorithm with affine gap costs. The gap initiation penalty was 6 and the gap prolongation penalty 2. The mutation score matrix we used is given in Figure 6.26 (values above the main diagonal are symmetrical).

We computed for each pair a number of structural traces with increasing suboptimality. Our results are summarized tables that depict data for different structural alignments (*e.g.*, the table in Figure 6.27 on the next page).

For each alignment we display its total score followed by the structural similarity score and the sequence similarity score (columns `tsc`, `isc` and `asc`). The next column contains the number of scored interactions followed by the number of alignment edge variables, the number of interaction match variables, and the time in seconds that the branch-and-cut algorithm ran. Those three columns are of course only filled for structural alignments computed by our algorithm.

The tables starts with the correct structural alignment from the Antwerpen database, followed by the conventional alignment which was computed using affine gap costs. Then we give the data for the structural alignments opt_x where x is the level of suboptimality the alignment was computed with.

set (1,0) Acidianus Sp(455) Acidianus Brierleyi(451)							
alignment	tsc	isc	asc	#sci	#nav	#niv	time
corr.	12458	7056	5402	441	-	-	-
conv.	12419	7008	5411	438	-	-	-
opt_0	12419	7008	5411	438	1499	445	17
opt_1	12419	7008	5411	438	1499	445	17
opt_2	12419	7008	5411	438	1499	445	18
opt_3	12419	7008	5411	438	1499	445	18
opt_4	12419	7008	5411	438	1499	445	20
opt_5	12419	7008	5411	438	1499	445	22
opt_6	12419	7008	5411	438	1503	445	36
opt_7	12419	7008	5411	438	1508	447	43
opt_8	12448	7040	5408	440	1519	455	20
opt_9	12448	7040	5408	440	1519	455	21
opt_{10}	12448	7040	5408	440	1530	465	25

Figure 6.27: Test run of two sequences stemming from the same family.

set (1,11) Acidianus Sp(455) Natronobacterium Magadii(439)							
alignment	tsc	isc	asc	#sci	#nav	#niv	time
corr.	11195	6688	4507	418	-	-	-
conv.	10790	6208	4582	388	-	-	-
opt_0	10870	6288	4582	393	1486	405	17
opt_1	10870	6288	4582	393	1486	405	17
opt_2	10957	6384	4573	399	1525	428	20
opt_3	11116	6528	4588	408	1570	449	22
opt_4	11136	6560	4576	410	1620	469	31
opt_5	11136	6560	4576	410	1662	502	78
opt_6	11136	6560	4576	410	1710	541	146
opt_7	11184	6608	4576	413	1808	628	317
opt_8	11213	6640	4573	415	1917	738	322
opt_9	11277	6704	4573	419	2021	900	512
opt_{10}	11277	6704	4573	419	2109	997	717

Figure 6.28: Test run of two sequences stemming from different families.

After testing different strategies of assigning different scores to standard or non-

set (0,7) Acidianus Brierleyi(451) Halobact. Halobium(447)							
alignment	tsc	isc	asc	#sci	#nav	#niv	time
corr.	11272	6736	4536	421	-	-	-
conv.	10888	6288	4600	393	-	-	-
opt_0	10936	6336	4600	396	1490	402	24
opt_1	10936	6336	4600	396	1490	402	40
opt_2	10987	6384	4603	399	1503	407	41
opt_3	11108	6496	4612	406	1537	420	34
opt_4	11111	6496	4615	406	1558	427	45
opt_5	11140	6528	4612	408	1594	448	54
opt_6	11195	6608	4587	413	1699	499	40
opt_7	11197	6608	4589	413	1804	578	82
opt_8	11297	6704	4593	419	1894	662	190
opt_9	11314	6720	4594	420	1949	723	237
opt_{10}	11342	6736	4606	421	2111	977	4809

Figure 6.29: Test run of two sequences stemming from different families.

standard interaction matches, we found out that making such a distinction seemed to have little effect on the quality of the structural alignments. Hence each interaction match has the same score of 16.

Note, however, that for the computation of a optimal structural alignment we did not use a score function with gaps. Therefore we applied this gapless score function to the optimal conventional alignment in order to derive a lower bound for our algorithm. That implies that the score of the conventional alignment may be less than the score of the structural alignment, simply because the conventional alignment is optimal with respect to another score function.

Figure 6.27 on the facing page shows a table for a pair of sequences stemming from the same family. It can be seen that the improvement in the score and in the number of realized interaction matches is not very significant. This behavior can be observed for most sequence pairs stemming from the same family. The reason for this is simply that within a family sequence conservation is high. Hence, conventional algorithms perform well and we cannot improve the alignment by taking into account the structural similarity.

This changes completely if the pair of sequences is chosen from different families as shown in the three Figures 6.28, 6.29, and 6.30. Here, the structural elements are conserved although in some helices the sequence conservation is very poor.

The figures show that conventional alignments realize significantly less interaction matches than correct alignments, whereas optimal alignments with suboptimality 0 already realize more interaction matches than conventional alignments. With increasing suboptimality this number rises. Note that while the total score of the optimal alignments increases this is mainly due to the structural similarity score. The sequence similarity score even decreases in most cases. This is exactly the effect we expected. Sequence similarity is sacrificed in order to give a better structural alignment.

While the running times in Figure 6.27 on page 124 basically stay the same, the other examples show that they become (sometimes significantly) worse with increasing suboptimality. However, our experiments show that in many cases the structural alignment computed in a couple of minutes is already significantly better than the conventional alignment and only seldomly one would like to increase the suboptimality even further.

set (2,9) Desulf. Mobilis(458) Haloferax Denitrificans(440)							
alignment	tsc	isc	asc	#sci	#nav	#niv	time
corr.	11423	6768	4655	423	-	-	-
conv.	11116	6400	4716	400	-	-	-
opt_0	11321	6608	4713	413	1516	427	19
opt_1	11321	6608	4713	413	1516	427	20
opt_2	11321	6608	4713	413	1518	427	22
opt_3	11362	6640	4722	415	1537	436	24
opt_4	11372	6656	4716	416	1552	441	25
opt_5	11408	6688	4720	418	1571	453	26
opt_6	11456	6736	4720	421	1607	470	40
opt_7	11530	6816	4714	426	1638	496	32
opt_8	11578	6864	4714	429	1700	532	48
opt_9	11602	6896	4706	431	1765	592	86
opt_{10}	11606	6912	4694	432	1824	643	176

Figure 6.30: Test run of two sequences stemming from different families.

Another set of figures shows the total score, the alignment score, and the interaction score of all 132 test sets for suboptimality 0 (Figures 6.31,6.33, and 6.32) and for suboptimality 8 (Figures 6.34,6.36, and 6.35). For this values all but 6 computations needed less than 30 minutes to find an optimal solution, most of them only a couple of minutes (see Figure 6.38 on page 131 for the timings with suboptimality 8 and Figure 6.37 on page 131 for timings with suboptimality 0).

All of the above figures are sorted. In the figures that display the scores, the test sets

are sorted in increasing order according to the correct score. The figures displaying the timings are sorted in increasing order according to the time. Inspecting the results of all test sets validates the statements we made by looking at the selected test sets in the tables.

Finally we show for the pair $(0, 8)$ a part of the actual alignment. Figures 6.39 on page 132, 6.40 on page 133, and 6.41 on page 134 show the beginning of the correct, the conventional and the structural alignment. Looking, for example, at the two helices $6 : 6'$ and $10 : 10'$ reveals that the structural alignment indeed almost perfectly reconstructed the correct alignment while the conventional alignment succeeded only partially for helix $6 : 6'$ and not at all for helix $10 : 10'$. A closer look at the alignment part that contains helix $10 : 10'$ reveals that the sequence similarity is indeed rather small.

Figure 6.42 on page 135 shows a dot plot of the correct, conventional and structural alignment for the region in which helix $10 : 10'$ lies. In this example the structural alignment does not completely coincide with the correct alignment. This may be due to the lacking (mis)matches in the input SEAG (see Figure 6.43 on page 135). Nevertheless, as can be seen in Figure 6.41 on page 134, the structural alignment does realize almost all interactions of helix $10 : 10'$, thereby giving an alternative structural alignment compared to that of the database.

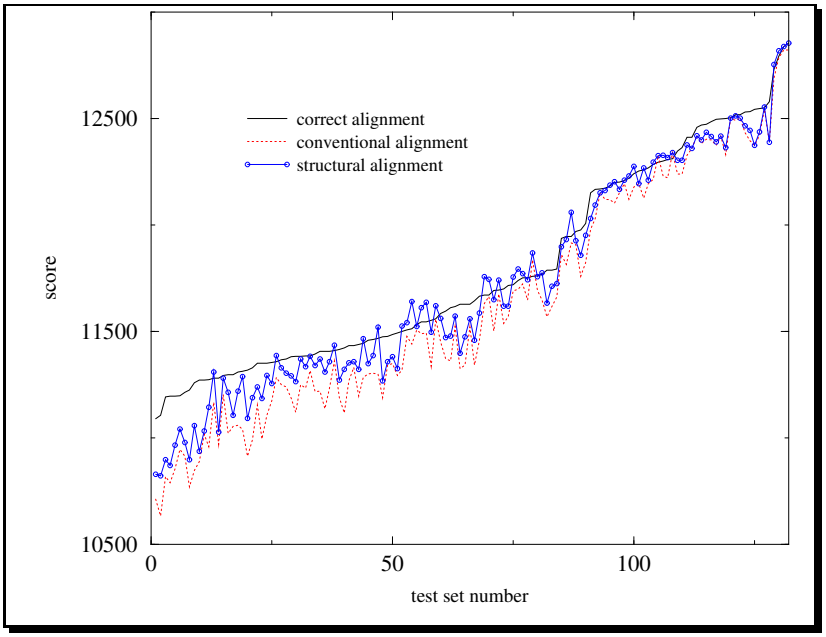

Figure 6.31: Plot of the total structural alignment scores for all test sets, computed with suboptimality 0.

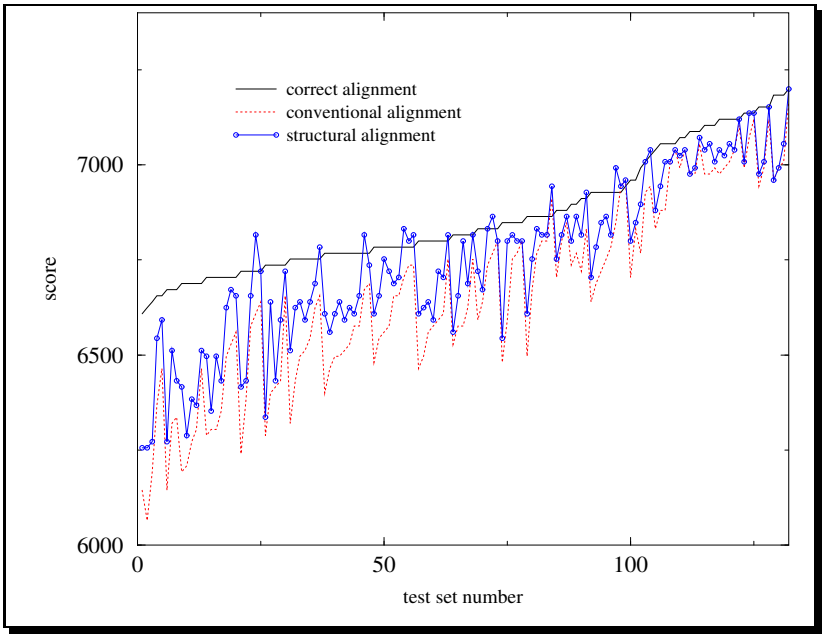

Figure 6.32: Plot of the interaction match scores for all test sets, computed with suboptimality 0.

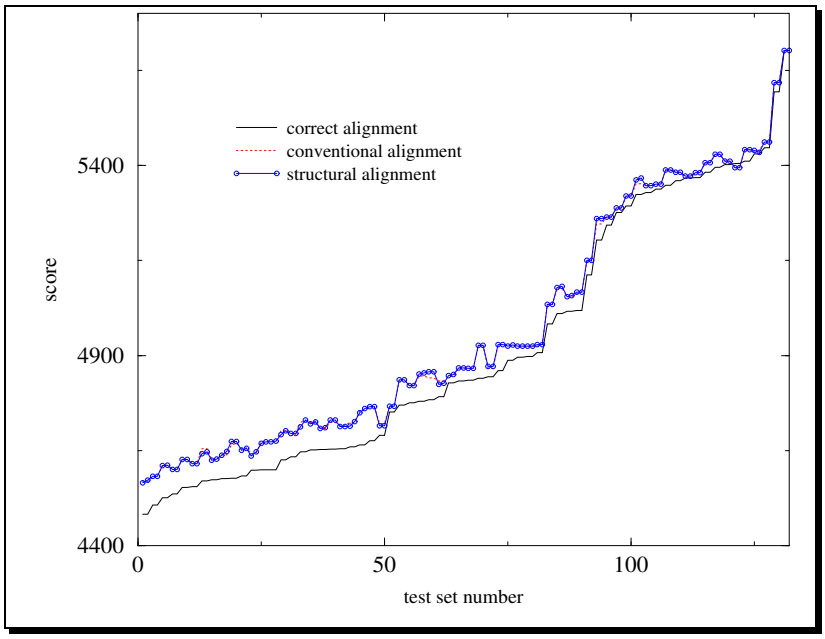

Figure 6.33: Plot of the mutation matrix scores for all test sets, computed with suboptimality 0.

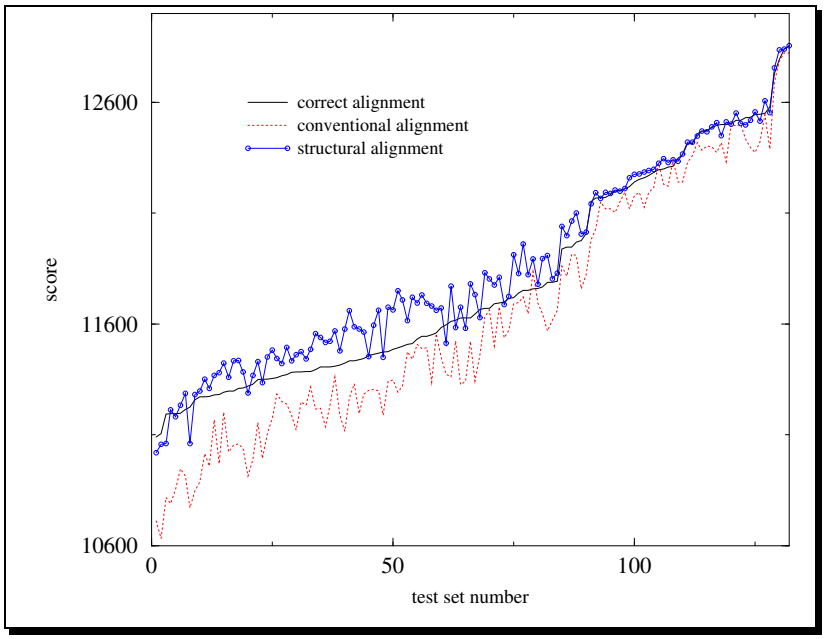

Figure 6.34: Plot of the total structural alignment scores for all test sets, computed with suboptimality 8.

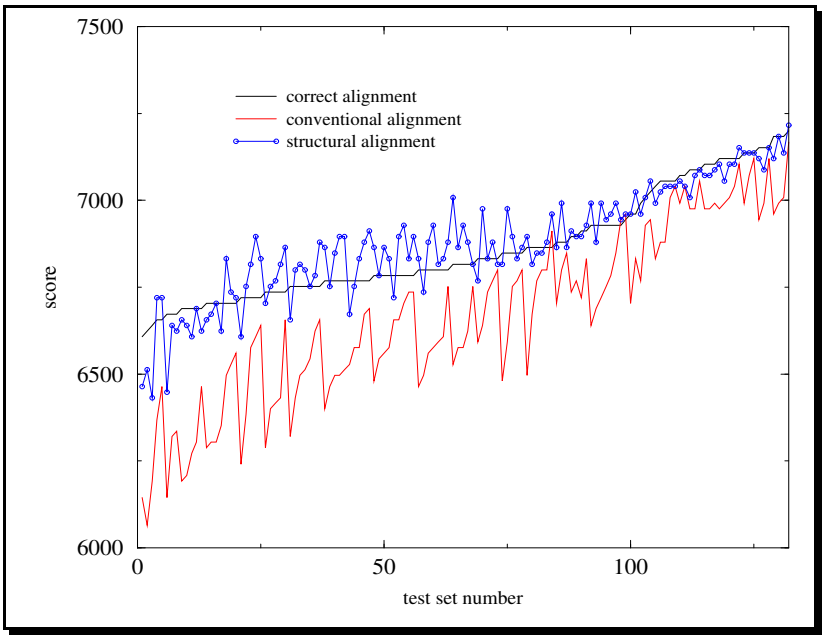

Figure 6.35: Plot of the interaction scores for all test sets, computed with suboptimality 8.

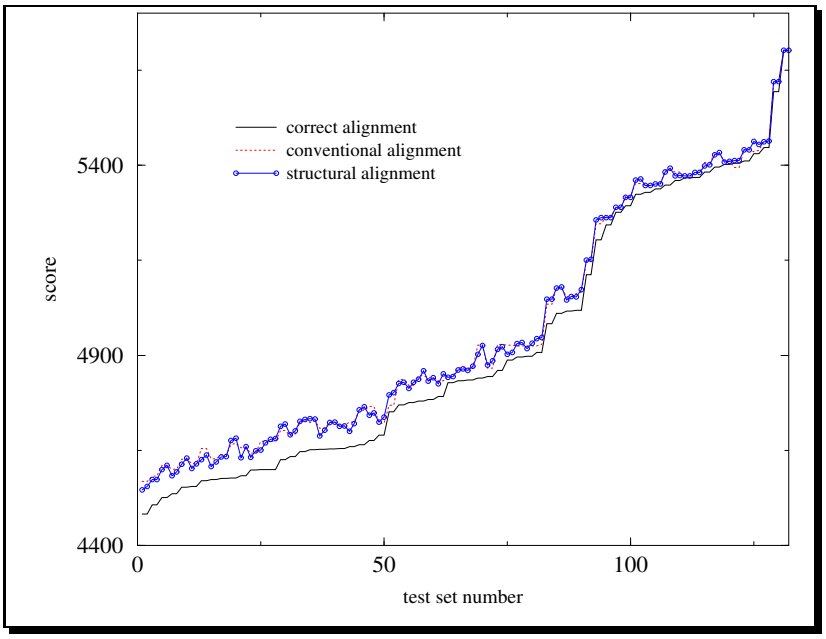

Figure 6.36: Plot of the mutation matrix score scores for all test sets, computed with suboptimality 8.

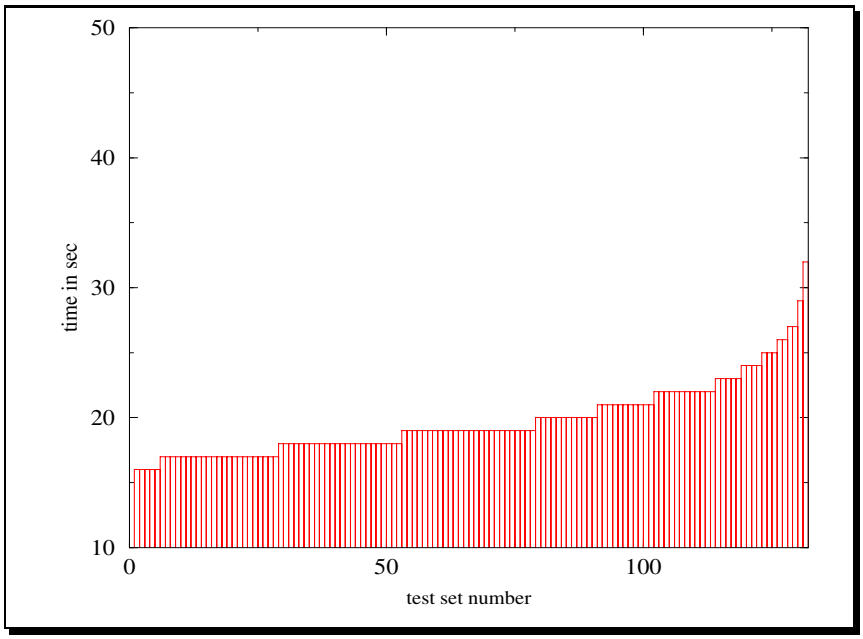

Figure 6.37: Plot of running time for all test sets, computed with suboptimality 0.

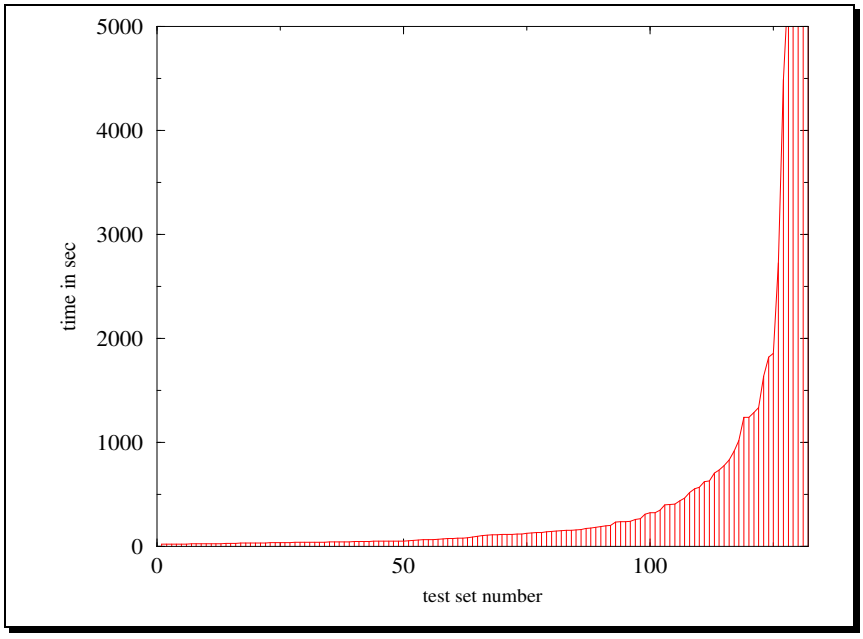

Figure 6.38: Plot of running time for all test sets, computed with suboptimality 8.

```
Alignment of sequences
  0.Acidianus brierleyi (length = 1492)
  1.Halococcus morrhuae (length = 1475)
Sequence 0 has secondary structure
# interacts. : 451
   # helices : 51
Sequence 1 has secondary structure
# interacts. : 445
   # helices : 51
Alignment has value 11258(4570,6688)
- - -1- - - - - - - - 2- - - -1' - - - -1' - - - - -3- - - - - -
A U U[C C G G U]U G A [U C C U^G C C G U^G C C G G]A[C C A G A U
A U U[C C G G U]U G A [U C C U^G C C G U^G C C G G]A[G G C U A U
- - -1- - - - - - - - 2- - - -1' - - - -1' - - - - -3- - - - - -

- - - - -4- - - - - - - - - - - - - - - -5- - - - - - - - -6- - - -
C G C U]A[U G G G G A U A G]G G C U A A[G C C{A}U G G]G A[G U C{G
U G C U]A[U C G G G G U C C]G A U U C A[G C C{A}U G C]U A[G U U{G
- - - - -4- - - - - - - - - - - - - - - -5- - - - - - - - -6- - - -

 - - - - - - - - - - - - - - - - 6'- - - - - - - - - - - - - - - -7-
 U}A C G C U C U C]G G U A A[G A G G G C G U G G C]G G A C G G[C
 U}A C G G G U]- - - U C A G - -[A C C C G U A G C]A A A U A G[C
 - - - - - - - - - - - - - - - - -6' - - - - - - - - - - - - - -7-

- - - - - - - - -8- - - - - - - - - - - - - - - - - - - - - -9- -
U G]A G U A A C A[C G U G G U C{A A}C C{U A A}C C U C G G G]A[C U
U C]C G U A A C A[C G U G G U C{A A}A C{U A C}C C U C U G G]A[C C
- - - - - - - - -8- - - - - - - - - - - - - - - - - - - - - -9- -

 - - - - - - - - - - - - - - - - - -9' - - - - - - - - - - - - - -
 U G{G A U A C}C U C C G G]G A A A[C U G G A G{C U A A U C}C A A
 G G{G A U A U}C C U C G G]G A A A[C U G A G G{U C A A U C}C C A
 - - - - - - - - - - - - - - - - - -9' - - - - - - - - - - - - - -

- - - - -10 - - - - - - - - - - - - - - - -10'- - - - - - - - - -
G]A U A G[G C A A A G G A{A}U C]U G G A A C[G A U C C U U U G C]U
G]A U A C U[G C U U U C{A}U G U]U G G A A U[A C A G A A A G U]C G
- - - - - -10 - - - - - - - - - - - - - - -10'- - - - - - - - - -
```

Figure 6.39: Part of the correct alignment of test set $(0, 8)$.

```
Alignment of sequences
  0.Acidianus brierleyi (length = 1492)
  1.Halococcus morrhuae (length = 1475)
Sequence 0 has secondary structure
# interacts. : 451
   # helices : 51
Sequence 1 has secondary structure
# interacts. : 387
   # helices : 51
Alignment has value 10847(4655,6192)
- - -1- - - - - - - -2- - - -1' - - - - -3- - - - - - - - - - -4-
A U U[C C G G U]U G A[U C C U^G C C G G]A[C C A G A U C G C U]A[U
A U U[C C G G U]U G A[U C C U^G C C G G]A[G G C U A U U G C U]A[U
- - -1- - - - - - - -2- - - -1' - - - - -3- - - - - - - - - - -4-

 - - - - - - - - - - - - - - -5- - - - - - - - -6- - - - - - - - -
 G G G G A U A G]G G C U A A[G C C{A}U G G]G A[G U C{G U}A C G C
 C G G G G U C C]G A U U C A[G C C{A}U G C]U A[G U U{G U}A C G]-
 - - - - - - - - - - - - - - -5- - - - - - - - -6- - - - - - - - -

- - - - - - - - -6' - - - - - - - - - - - - - - - -7- - - - - - -
U C U C]G G U A A[G A G G G C G U G G C]G G A C G G[C U G]A G U A
- - - G G U U C A G A - C C[C G U A G C]A A A U A G[C U C]C G U A
- - - - - - - - - - - - - -6' - - - - - - - - - - -7- - - - - - -

 - - -8- - - - - - - - - - - - - - - - - - - - - - -9- - - - - - -
 A C A[C G U G G U C{A A}C C{U A A}C C U C G G G]A[C U U G{G A U
 A C A[C G U G G U C{A A}A C{U A C}C C U C U G G]A[C{C}G G{G A U
 - - -8- - - - - - - - - - - - - - - - - - - - - - -9- - - - - - -

- - - - - - - - - - - - -9' - - - - - - - - - - - - - - - - - - -10
A C}C U C C G G]G A A A[C U G G A G{C U A A U C}C A A G]A U A G[G
A U}C C U C G G]G A A A[C U G A G G{U C A A U C}C C{A}G]A U A - -
- - - - - - - - - - - -9' - - - - - - - - - - - - - - - - - - - -

 - - - - - - - - - - - - - - - - -10'- - - - - - - - - - - - - - -1
 C A A A G G A{A}U C]U G G A A C[G A U C C U U U G C]U U A A A G[
 - - - - - - - - - C U G - - - C U U U C - - A U G U U G G A A U
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Figure 6.40: Part of the conventional alignment of test set $(0, 8)$.

```
Alignment of sequences
  0.Acidianus brierleyi (length = 1492)
  1.Halococcus morrhuae (length = 1475)
Sequence 0 has secondary structure
# interacts. : 451
   # helices : 51
Sequence 1 has secondary structure
# interacts. : 416
   # helices : 51
Alignment has value 11281(4625,6656)
- - -1- - - - - - - -2- - - -1' - - - - -3- - - - - - - - - - -4-
A U U[C C G G U]U G A[U C C U^G C C G G]A[C C A G A U C G C U]A[U
A U U[C C G G U]U G A[U C C U^G C C G G]A[G G C U A U U G C U]A[U
- - -1- - - - - - - -2- - - -1' - - - - -3- - - - - - - - - - -4-

 - - - - - - - - - - - - - - -5- - - - - - - - -6- - - - - - - - -
 G G G G A U A G]G G C U A A[G C C{A}U G G]G A[G U C{G U}A C G C
 C G G G G U C C]G A U U C A[G C C{A}U G C]U A[G U U{G U}A C G G
 - - - - - - - - - - - - - - -5- - - - - - - - -6- - - - - - - - -

- - - - - - - - -6' - - - - - - - - - - - - - - - -7- - - - - - -
U C U C]G G U A A[G A G G G C G U G G C]G G A C G G[C U G]A G U A
G{U}U C]- - - - A[G A - C C C G U A G C]A A A U A G[C U C]C G U A
- - - - - - - - -6' - - - - - - - - - - - - - - - -7- - - - - - -

 - - -8- - - - - - - - - - - - - - - - - - - - - -9- - - - - - -
 A C A[C G U G G U C{A A}C C{U A A}C C U C G G G]A[C U U G{G A U
 A C A[C G U G G U C{A A}A C{U A C}C C U C U G G]A[C{C}G G{G A U
 - - -8- - - - - - - - - - - - - - - - - - - - - -9- - - - - - -

- - - - - - - - - - - -9' - - - - - - - - - - - - - - - - - - -10
A C}C U C C G G]G A A A[C U G G A G{C U A A U C}C A A G]A U A G[G
A U}C C U C G G]G A A A[C U G A G G{U C A A U C}C C{A}G]A U A C[U
- - - - - - - - - - - -9' - - - - - - - - - - - - - - - - - - -10

 - - - - - - - - - - - - - - - -10'- - - - - - - - - - - - - - - -
 C A A A G G A{A}U C]U G G A A C[G A U C C U U U - G C]U U A A A
 G{C}U U U C A{U}G]- - - - - - - -[U U G G A A{U A}C A]- - - - -
 - - - - - - - - - - - - - - - - -10'- - - - - - - - - - - - - -
```

Figure 6.41: Part of the optimal alignment of test set $(0, 8)$.

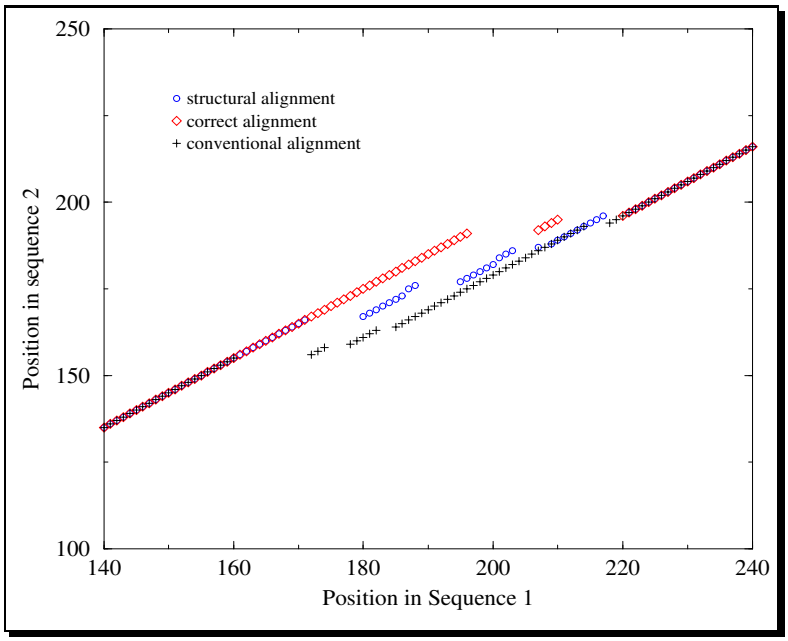

Figure 6.42: Dot plot displaying the correct, conventional, and optimal alignment of test set $(0, 8)$.

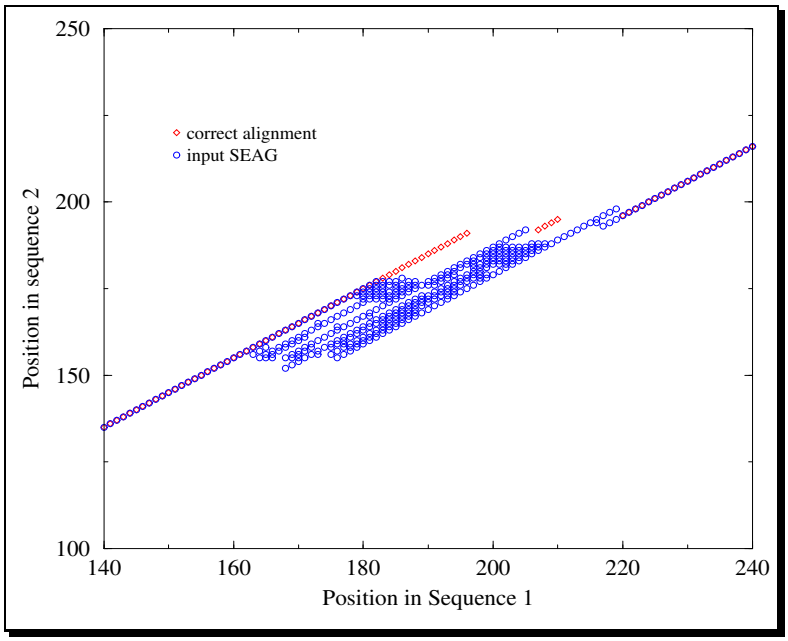

Figure 6.43: Dot plot of (mis)matches contained in the input SEAG and the correct alignment for test set $(0, 8)$.

Chapter 7

Discussion

In this thesis we present for the first time the application of methods from polyhedral combinatorics to the field of sequence alignment. We define two general sequence alignment problems, the GMT problem and the SMT problem and formulate both problems in terms of integer linear programs. The investigation of the associated problem polytopes leads to the characterization of several classes of facet-defining inequalities for the polytopes, for most of which we give exact or heuristic separation routines. We implemented these separation routines with the help of the branch-and-cut framework ABACUS and show that even this first implementation is able to solve to optimality problem instances to big for dynamic programming based approaches.

Our approach may have some disadvantages, such as the dependence on a lot of additional software or the unpredictable running time of the branch-and-cut algorithm. However, we think that the strengths of our approach outweigh the disadvantages by far:

- Various alignment problems can be expressed within the GMT and SMT problem by simply choosing an appropriate (structural) alignment score.
- The polyhedral approach is new and leaves a lot of room for improvement, while the dynamic programming based approaches are thoroughly studied and hard too improve.
- Once a scoring function is chosen, the SEAG and EAG can easily encode restrictions on the input. An optimal multiple trace always corresponds to an alignment that agrees as much as possible with the information encoded in the extended alignment graph. We give a few examples of how to restrict the set of alignment edges in the input EAG or SEAG:
  - Choose from (sub)optimal pairwise alignments. For example, take all edges that are realized by alignments that are optimal under a pairwise alignment score function with gaps (Kececioglu 1993; Reinert *et al.* 1997). One could also take edges that are realized by alignments whose score is at most a certain constant off the optimal alignment score. An edge could be weighted using a (shifted) mutation score matrix.
  - Context dependencies could be encoded. For example, we could consider a match only significant if it occurs between two additional matches and insert only edges into the alignment graph that meet this condition. This would reduce the number of edges in the EAG and make the problem computationally easier (Wilbur and Lipman 1984).
  - Consider a set of diagonals, *i.e.*, a consecutive run of aligned characters. Insert alignment edges between the aligned pairs of characters of a diagonal

and regard them as a block. The weight of a block can be determined by some probabilistic considerations (Morgenstern *et al.* 1998). As a subset of all possible blocks one could consider blocks that can be found in (sub)optimal pairwise alignments (Lenhof, Morgenstern, and Reinert 1999).

  - Consider short local alignments as blocks. Each (mis)match creates an edge in the EAG. In contrast to choosing diagonals, a block now symbolizes a possibly gapped local alignment. The weight of a block could again be determined by probabilistic considerations.
  - When structural traces are considered one can also encode restrictions for the alignment edges similar to the ones mentioned above. But also for the interaction edges it is straightforward to encode restrictions. For example, for structured RNA sequences one could require a minimum distance m between two interacting nucleotides, *i.e.*, $j - i \geq m$ for an interaction (i, j).

- The solution of each relaxation provides good upper bounds for a problem. These bounds can be used to evaluate the quality of heuristic methods.

- We extended the GMT formulation in order to handle arbitrary gap costs.

Obviously there are many open questions and new problems that we could not solve or attack in this thesis. Partially they are of theoretical nature, but there are also algorithmic problems that could directly be used to improve the performance of the branch-and-cut algorithm. Below we give a list of interesting open problems.

- In Section 4.3 on page 56 we gave a graph-theoretic formulation for gapped traces. This formulation can be used to describe multiple sequence alignment with arbitrary gap costs. Below we formally define the *Gapped Trace Problem* and give the corresponding ILP. The question is, whether the study of the gapped trace polytope also leads to practical algorithms for multiple alignment with arbitrary gap costs.

  Gapped Trace Problem:
  Given an GEAG $G = (V, E, H, A)$ with weights w_e $(\forall e \in E)$ and w_g $(\forall g \in A)$.
  Find the gapped trace (T, C), $T \subseteq E$, $C \subseteq A$ with maximum weight.

  Define for each gap edge $g \in A$ a binary variable x_g and for each alignment edge $e \in E$ a binary variable x_e that indicate whether the respective edges are contained in a gapped trace. Then the following ILP describes the gapped trace problem. The objective function is

$$\text{maximize } \sum_{e \in E} w_e \cdot x_e - \sum_{g \in A} w_g \cdot x_g.$$

According to the definition of gapped trace (see Definition 4.3.1 on page 57) we have to ensure three conditions for a pair (T, C). First the alignment edges in T must not induce a mixed cycle in G. This is ensured by

$$\sum_{e \in P} x_e \leq |P \cap E| - 1, \ \forall \text{ critical mixed cycles } P \text{ in } G.$$

The second condition for a gapped trace is that for any pair of sequences a node must either be incident to an alignment edge or must be enclosed by exactly one gap edge. This is ensured by inequalities that are defined for any node $s_{j,q}$, $1 \leq q \leq n_j$, and any sequence S_i, $i \neq j$. We denote the set of all alignment edges that are incident to $s_{j,q}$ and a node in S_i by $E_{j,q,i}$. Then the inequality is as follows:

$$\sum_{e \in E_{j,q,i}} x_e + \sum_{\substack{g_{i,j,l_1,l_2} \in A \\ \text{with } l_1 \leq q \leq l_2}} x_{i,j,l_1,l_2} = 1.$$

Finally we require that a consecutive run of gap characters is regarded as one gap rather than the concatenation of two shorter gaps. This is ensured by the following inequality for $1 \leq i \neq j \leq k$, $1 \leq q < n_j$:

$$\sum_{1 \leq l_1 \leq q} x_{i,j,l_1,q} + \sum_{q+1 \leq l_2 \leq n_j} x_{i,j,q+1,l_2,} \leq 1.$$

- Neither for the MT nor for the GMT problem a constant factor approximation is known. It is also not clear whether it exists at all.
- For the SMT problem, we could not find an efficient way to exactly separate all extended clique inequalities. Is the general separation problem for this class hard, or can an efficient algorithm be found?

We hope that our work provides some insight in the beauty and power of polyhedral combinatorics. Our novel approach may be a first step to establish this optimization technique in the field of sequence alignment and that our methods and results can provide a basis for more practical algorithms for (multiple) sequence alignment.

Chapter 8

Deutsche Zusammenfassung

Die Erforschung funktionaler Zusammenhänge zwischen verschiedenen biologischen Makromolekülen basiert zu großen Teilen auf Techniken aus dem Gebiet des Sequenzvergleiches. Zu den wichtigsten Methoden in diesem Gebiet gehören Algorithmen, welche zwei oder mehr Sequenzen so alignieren, daß ihre Gemeinsamkeiten und Unterschiede zu Tage treten. Ein Alignment von k Sequenzen kann als eine Matrix mit k Zeilen dargestellt werden. Jede Zeile enthält eine der Sequenzen vermischt mit einem sogenannten *Lücke*-Symbol, wobei keine Spalte nur aus Lücke-Symbolen bestehen darf.

Seit Needleman und Wunschs Veröffentlichung (1970) über paarweises Sequenz-Alignment hat die Anzahl und Vielfalt der verschiedenen Alignment-Methoden beträchtlich zugenommen. Die Anwendungen sind vielfältig und zum Teil recht unterschiedlich. Sie reichen von schnellen Datenbanksuchen (Altschul *et al.* (1990)) über Anwendungen in der Sequenzassemblierung (Kececioglu und Myers (1995)) bis hin zur genauen Analyse verschiedener Proteinfamilien (McClure *et al.* (1994)).

Trotz der großen Anzahl verschiedener Problemformulierungen beruhen die meisten Algorithmen zum Sequenz-Alignment auf dem Prinzip der dynamischen Programmierung. Obwohl diese Technik ein oft genutztes und weit verbreitetes Optimierungsverfahren ist, hat sie im Rahmen des Sequenz-Alignments den Nachteil, daß sie im Allgemeinen einen Algorithmus liefert, dessen Zeit- und Platzkomplexität exponentiell in der Anzahl der betrachteten Sequenzen ist. Sogar komplizierte Implementierungen, welche ausgefeilte Techniken benutzen (Lermen und Reinert (1997), Gupta, Kececioglu und Schaeffer (1995)) erreichen relativ schnell ihre Grenzen.

In dieser Arbeit untersuchen wir einen neuen Ansatz, um Sequenz-Alignment-Probleme zu lösen, welcher auf einem Gebiet der kombinatorischen Optimierung basiert, das als *polyhedrische Kombinatorik* bekannt ist (Schrijver (1986), Nemhauser *et al.* (1989)). Wir zeigen auf, wie man diesen Ansatz benutzen kann, um für gewisse Alignment-Probleme Algorithmen zu entwickeln, die nicht auf dem Prinzip der dynamischen Programmierung beruhen. Diese Algorithmen nennt man *Branch-and-Cut*-Algorithmen (Jünger, Reinelt und Thienel (1995b)). Sie vereinen *Branch-and-Bound*-Techniken mit linearer Programmierung und gehören momentan zu den erfolgreichsten Algorithmen, um schwierige kombinatorische Probleme zu lösen. Sie wurden bei bekannt schwierigen Problemen wie dem *Linear-Ordering-Problem* (Gr̈otschel *et al.* (1984)) und dem *Traveling-Salesman-Problem* (Padberg und Rinaldi (1987)) zum ersten Mal erfolgreich angewandt und gehören inzwischen zu den anerkannten Optimierungstechniken in vielen naturwissenschaftlichen und technischen Gebieten (z. B. Christoph *et al.* (1997), Jünger und Reinelt (1995a), Applegate *et al.* (1995); für eine hervorragende Übersicht siehe auch Kapitel 4 in Dell'Amico *et al.* (1997)). In dieser Arbeit werden Branch-and-Cut-Algorithmen zum ersten Mal für Probleme des Sequenz-Alignments entwickelt.

Eine wesentliche Voraussetzung beim Design eines Branch-and-Cut-Algorithmus besteht darin, das betrachtete Problem als ganzzahliges lineares Programm (GLP) zu formulieren. Nehmen wir an, daß wir eine solche Formulierung gefunden haben. Das heißt, daß jede Lösung des Problems durch einen hochdimensionalen (0/1)-Vektor repräsentiert werden kann, welchen wir als *Inzidenz*-Vektor bezeichnen. Die konvexe Hülle aller zulässigen Inzidenz-Vektoren bildet das sogenannte Problempolytop P. Das Problempolytop ist nicht mit dem Polytop zu verwechseln, welches durch den Schnitt der Hyperebenen gebildet wird, die als Nebenbedingungen im GLP erscheinen, obwohl auch dieses Polytop keinen unzulässigen ganzahligen Punkt enthält.

Da das Lösen ganzzahliger, linearer Programme ein NP-schweres Problem ist, relaxieren wir das GLP, indem wir z. B. die Ganzzahligkeitsbedingung fallen lassen. Das dadurch entstehende lineare Programm läßt sich effizient entweder mit dem Simplex-Algorithmus oder *interior-Point*-Methoden lösen. Wenn die Lösung $\bar{x}$ des linearen Programms ganzzahlig ist, entspricht dies einem Inzidenz-Vektor, der eine optimale Lösung repräsentiert. Anderenfalls suchen wir eine gültige Hyperebene f mit $fx \leq f_0$. Gültig bedeutet hier, daß die Hyperebene f die unzulässige Lösung $\bar{x}$ vom Polytop "abschneidet", ohne eine zulässige Lösung abzuschneiden, d.h. $fy \leq f_0$ für alle $y \in P$ und $f\bar{x} > f_0$. Die Menge $\{x \mid fx = f_0\}$ wird auch *Schnittebene (cutting plane)* genannt. Eine Schnittebene wird gefunden, indem man das *Separierungsproblem* für alle Klassen von gültigen Ungleichungen löst. Jegliche Schnittebenen, die gefunden werden, werden zu den Nebenbedingungen des linearen Programms hinzugefügt, worauf das so erweiterte lineare Programm wieder gelöst wird. Die Generierung von Schnittebenen wird wiederholt, bis man entweder eine ganzzahlige (und somit optimale) Lösung erhält oder bis man keine Schnittebene mehr findet. Falls man keine Schnittebene mehr findet, erfolgt ein Verzweigungsschritt (branch step). Wir generieren zwei Unterprobleme, indem wir eine gebrochene Variable des linearen Programms im ersten Unterproblem auf eins und im zweiten Unterproblem auf null setzen und dann diese Unterprobleme rekursiv lösen. Durch diese Vorgehensweise erhält man einen Enumerationsbaum von Unterproblemen. In jedem Knoten dieses Enumerationsbaumes löst der Branch-and-Cut Algorithmus eine gewisse Anzahl von relaxierten, linearen Programmen, wobei zu jedem solchen Programm eine gewisse Anzahl neuer Nebenbedingungen durch das Lösen des Separierungsproblems hinzukommt. Es zeigt sich, daß problemspezifische Schnittebenen wesentlich effektiver sind. Insbesondere Schnittebenen, die eine Facette des Problempolytops bilden, scheinen sehr geeignet, da sie in einer irredundanten Beschreibung des Problempolytops nicht durch andere Ungleichungen dominiert werden. Dies bedeutet, daß eine möglichst genaue Beschreibung des Poblempolytops unerläßlich für die Effizienz eines Branch-and-Cut-Algorithmus ist.

Graphen, Traces und multiples Alignment

Wir beschreiben nun die untersuchten Probleme im Detail. Es handelt sich

zum einen um das *Generalized-Maximum-Trace*-Problem (GMT), zum anderen um das *Structural-Maximum-Trace*-Problem (SMT). Dazu betrachten wir zunächst eine graphentheoretische Formulierung von multiplem Sequenz-Alignment, eingeführt von Kececioglu (1991), und zeigen dann, wie wir diese Formulierung so erweitern können, daß sie die beiden oben genannten Probleme beinhaltet.

Sei $S = \{S_1, S_2, \ldots, S_k\}$ eine Menge von k Strings über einem Alphabet Σ und sei $\hat{\Sigma} = \Sigma \cup \{-\}$, wobei "$-$" (minus) ein Symbol ist, welches "Lücken" in Strings repräsentiert. Ein *Alignment* von S ist eine Menge $\hat{S} = \{\hat{S}_1, \hat{S}_2, \cdots, \hat{S}_k\}$ von Strings über dem Alphabet $\hat{\Sigma}$, mit den folgenden beiden Eigenschaften: (1) die Strings in $\hat{S}$ haben alle die gleiche Länge l, und (2) entfernt man die Minuszeichen, dann ist der String $\hat{S}_i$ identisch mit dem String S_i. Somit kann man ein Alignment der Länge l als eine Matrix mit k Zeilen und l Spalten interpretieren, wobei die i-te Zeile dem String $\hat{S}_i$ entspricht. Wir bezeichnen zwei Buchstaben in verschiedenen Strings als *aligniert* durch $\hat{S}$, wenn sie sich in der gleichen Spalte der Alignment-Matrix befinden.

Wir können die Positionen der Buchstaben der k Eingabe-Strings auch als die Knotenmenge V eines k-partiten Graphen $G = (V, E)$ betrachten, den wir den *Eingabe-Alignment-Graph* nennen. Dabei repräsentieren die Kanten in E Paare von Buchstaben, welche in einem Alignment möglicherweise aligniert werden können. Wir nennen eine Kante in E *Alignment-Kante* und bezeichnen in einem Alignment eine Alignment-Kante als *realisiert*, wenn sich die Endpunkte der Kante in der gleichen Spalte der Alignment-Matrix befinden. Darüber hinaus bezeichnen wir die Teilmenge von E, die insgesamt von einem Alignment $\hat{S}$ realisiert ist, als den *Trace* von $\hat{S}$. Der Begriff des Traces von zwei Strings ist ein grundlegendes Konzept im Bereich des Sequenzvergleiches (siehe z. B. Sankoff und Kruskal (1983) S. 10–18), der durch Kececioglu (1991) auf multiples Sequenz-Alignment erweitert wurde. Kececioglu führte auch als erster den Begriff des Alignment-Graphen ein. Die Beziehung zwischen multiplem Alignment und multipartiten Graphen wurde auch von Vingron und Pevzner (1995) betrachtet.

Das Generalized-Maximum-Trace-Problem

Im sogenannten *Maximum-Trace-Problem* (MT), welches ursprünglich bei der Sequenzassemblierung dem Erstellen eines Konsensus-Alignments diente, hat jede Kante im Alignment-Graph ein positives Gewicht, welches den Nutzen der Realisierung dieser Kante repäsentiert. Das Ziel besteht darin, ein Alignment zu finden, dessen Trace von insgesamt maximalem Gewicht ist. Kececioglu (1991) bewies, daß das MT-Problem NP-vollständig ist und entwickelte einen Branch-and-Bound-Algorithmus, der auf dem Prinzip der dynamischen Programmierung beruht und eine worst-case Zeitkomplexität von $O(k^2 2^k N)$ und Platzbedarf von $O(N)$ hat, wobei $N = \prod_i |S_i|$. Dieser Algorithmus kann relativ kleine Probleminstanzen zur Optimalität lösen.

Wir generalisieren das Maximum-Trace-Problem so, daß man die verschiedensten Be-

wertungsfunktionen verwenden kann. Im *Generalized-Maximum-Trace-Problem* erlauben wir Mehrfachkanten zwischen zwei Knoten des Alignment-Graphen G, und partitionieren die Kantenmenge E in eine Menge D von sogenannten *Blöcken*. Ein Block ist ein Trace, in dem jede Kante inzident zu Knoten im gleichen Sequenzpaar ist. Wir nennen einen Block *realisiert*, wenn alle Kanten in ihm realisiert sind.

Jeder Block $d \in D$ hat ein Gewicht w_d, welches den Nutzen der Realisierung dieses Blocks repräsentiert. Das Gewicht eines Alignments ist die Summe der Gewichte der Blöcke, die es realisiert. Man beachte, daß diese Formulierung die Konstruktion eines multiplen Alignments aus paarweisen, lokalen Alignments beinhaltet.

Die meisten Bewertungsschemata für Alignments basieren auf der Ähnlichkeit einzelner Paare von Buchstaben (welche für Amino- oder Nukelinsäuren stehen; siehe auch Dayhoff *et al.* (1979) oder Henikoff und Henikoff (1992)). Solche Schemata können im Alignment-Graph modelliert werden, indem wir die Alignment-Kanten in einelementige Mengen partitionieren, was dem ursprünglichen Maximum-Trace-Problem entspricht. Im Gegensatz zum MT-Problem kann das GMT-Problem auch allgemeinere Bewertungsschemata modellieren, die auf dem Vergleich ganzer Segmentpaare basieren (siehe auch Altschul und Erickson (1986), Morgenstern *et al.* (1998) und Wilbur und Lipmann (1984)).

In der Formulierung des Problems als ganzzahliges, lineares Programm definieren wir für jeden Block d in D eine binäre Variable x_d, welche anzeigt, ob der Block realisiert ist ($x_d = 1$) oder nicht ($x_d = 0$). Eine ganzzahlige Lösung ist zulässig, wenn die Alignment-Kanten der realisierten Blöcke einen Trace darstellen. Das Ziel ist es, eine zulässige Lösung zu finden, die eine Menge von Blöcken mit insgesamt maximalem Gewicht realisiert.

Wie weiter oben bemerkt, ist es nun notwendig, das Problempolytop näher zu untersuchen, um einen effizienten Branch-and-Cut-Algorithmus zu erhalten. Im Falle des GMT-Problems konnten wir verschiedene Klassen von facetten-definierenden Ungleichungen identifizieren und für viele dieser Klassen effiziente Separierungsalgorithmen entwickeln. Im paarweisen Fall zeigen wir, daß die *clique*-Ungleichungen zusammen mit den trivialen Ungleichungen eine vollständige Beschreibung des Problempolytops bilden. Zusammen mit einem Polynomzeitseparierungsalgorithmus für die clique-Ungleichungen impliziert dies die Existenz eines Polynomialzeitalgorithmus für Sequenz-Alignment, der nicht auf dem Prinzip der dynamischen Programmierung beruht. Im Falle mehrerer Sequenzen beschreiben wir zwei weitere Klassen von gültigen Ungleichungen (die *mixed cycle*-Ungleichungen und die *ladder*-Ungleichungen) und zeigen auf, unter welchen Bedingungen sie facetten-definierend sind. Wir beschreiben eine Datenstruktur, welche wir *pairgraph* nennen, die eine exponentielle Anzahl von clique-Ungleichungen in nur polynomiell viel Platz repräsentiert und wie man

diese Datenstruktur dazu benutzen kann, einen effizienten, exakten Separierungsalgorithmus für clique-Ungleichungen zu entwickeln. Zur Separierung der *mixed-cycle*-Ungleichungen benutzen wir die graphentheoretische Formulierung selbst, um eine effiziente Separierungsroutine zu entwerfen. Unsere Implementierung des Branch-and-Cut-Algorithmus für das GMT-Problem zeigt, daß der Einsatz von Methoden der kombinatorischen Optimierung beim Lösen harter Sequenz-Alignment-Probleme zu Algorithmen führt, die vergleichbar oder besser sind als existierende Algorithmen, die auf dynamischer Programmierung beruhen. So konnten wir z. B. 18 Sequenzen einer Länge von ≈ 200 optimal alignieren; eine Problemgröße, die nicht durch Algorithmen gelöst werden kann, die auf dynamischer Programmierung beruhen.

Das Structural-Maximum-Trace-Problem

Das zweite Alignment-Problem, das wir betrachten, ist das *Structural-Maximum-Trace-Problem* (SMT). Hier ist das Ziel, ein Alignment zu berechnen, welches simultan Sequenz- und Strukturübereinstimmung maximiert. Präziser ausgedrückt, benutzen wir als Bewertungsfunktion eine gewichtete Summe von Bewertungen der Sequenzähnlichkeit und Strukturähnlichkeit beider Sequenzen. Strukturähnlichkeit bedeutet in diesem Kontext die Ähnlichkeit der beiden Sekundärstrukturen der Sequenzen, in unseren Beispielen RNA-Sequenzen.

Bafna *et al.* (1995) schlagen einen Algorithmus mit Laufzeit $O(n^4)$ vor, wobei n die Länge der Sequenzen ist. Diese Laufzeit erlaubt es nicht, reale Beispiele zu betrachten, in denen man Sequenzen einer Länge von ≈ 1400 untersuchen möchte.

Die Eingabe für das SMT-Problem kann man ebenfalls als Alignment-Graph betrachten, wobei wir für jede Sequenz zusätzlich eine Liste von möglichen *Interaktionen* oder *Basenpaaren* zwischen zwei Buchstaben der Sequenz haben. Eine solche Liste kann z. B. durch ein Sekundärstruktur-Vorhersage-Programm geliefert werden, oder sie kann alle möglichen Watson-Crick Basenpaare (`A-U` or `C-G`) enthalten.

Ein strukturelles Alignment kann im Unterschied zu einem konventionellen Alignment nicht nur eine Alignment-Kante realisieren, d.h. zwei Buchstaben der Sequenz alignieren, sondern es kann auch zwei Interaktionen alignieren, oder anders ausgedrückt, einen *Interaktions-Match* realisieren. Dabei bezeichnen wir zwei Interaktionen als aligniert, wenn die in der Interaktion beteiligten Buchstaben in beiden Sequenzen miteinander aligniert sind.

In der Formulierung des SMT-Problems als ganzzahliges lineares Programm assoziieren wir mit jeder Alignment-Kante e in E eine binäre Variable x_e, welche angibt, ob die Kante realisiert ist ($x_e = 1$) oder nicht ($x_e = 0$). Zum gleichen Zweck ordnen wir jedem Interaktions-Match m eine binäre Variable x_m zu. Eine ganzzahlige Lösung ist zulässig, wenn die realisierten Alignment-Kanten einen Trace bilden und wenn jeder Buchstabe

in höchstens einem realisierten Interaktions-Match vorkommt. Jede Alignment-Kante und jeder Interaktions-Match hat ein Gewicht, welches den Nutzen der Realisierung dieser Kante beziehungsweise dieses Interaktions-Matches repräsentiert. Das Ziel ist es, eine zulässige Lösung maximalen Gewichts zu finden.

Bei der näheren Untersuchung des SMT-Polytops zeigt sich, daß die *trivialen* und *Mixed-Cycle*-Ungleichungen im Prinzip die gleichen sind wie beim GMT-Polytop. Wir konnten drei neue Klassen von gültigen Ungleichungen finden und zeigen, unter welchen Bedingungen sie facetten-definierend sind: die *Extended-Clique*-Ungleichungen, die *Interaktions*-Ungleichungen und die *Odd-Cycle*-Ungleichungen.

Das Studium des SMT-Polytops führte zu einem Branch-and-Cut-Algorithmus zum strukturellen Alignment zweier RNA-Sequenzen, mit welchem wir Sequenzen einer Länge von ≈ 1400 nachweislich besser alignieren können als konventionelle Alignment-Algorithmen. Uns ist kein anderes Verfahren bekannt, welches Sequenzen dieser Länge optimal strukturell alignieren kann.

Fazit

Als ein Hauptbeitrag unserer Arbeit betrachten wir die Einführung von Methoden der polyedrischen Kombinierung in das Gebiet des Sequenz-Alignments. Wir definieren zwei allgemein gehaltene Alignment-Probleme, die leicht adaptiert werden können, so daß sie zahlreiche Sequenz-Alignment-Probleme beinhalten. Die Formulierung des SMT- und GMT-Problems als graphentheoretisches Problem erlaubt eine einfache Kodierung zusätzlicher Restriktionen. So könnte der Alignment-Graph z. B. nur Kanten enthalten, die von der Alignierung zweier Buchstaben in (sub)optimalen Alignments stammen (Kececioglu (1993), Reinert *et al.*(1997)), oder aber, er könnte nur Kanten enthalten, die kontextabhängig sind (Wilbur und Lipmann (1984)).

Basierend auf einer bestehenden Formulierung als ganzahliges, lineares Program können neue Probleme oft durch das Hinzufügen von Nebenbedingungen oder Variablen leicht modelliert werden. Bei dynamischer Programmierung hingegen erfordert die Anpassung einer bestehenden Formulierung, wie z. B. die Erweiterung von konventionellem Alignment zu strukturellem Alignment (Bafna *et al.* (1995)), zumindest eine beträchtliche Restrukturierung der zugrundeliegenden Rekursionsformel. Beim polyedrischen Ansatz können große Teile des Programm-Codes, der für das ursprüngliche Problem formuliert wurde, wiederverwendet werden. So basieren z. B. das GMT- und auch das SMT-Problem beide auf dem gleichen ganzzahligen, linearen Programm, und Separationsroutinen für Ungleichungen in dieser Formulierung werden in beiden Programmen gleichermaßen benutzt. Unser Ansatz hat weiterhin den Vorteil, daß andere Forscher auf den erreichten theoretischen und praktischen Ergebnissen leicht aufbauen können. Unsere ersten Implementierungen der Branch-and-Cut-Algorithmen für das SMT- und GMT-Problem zeigen, daß eine weitere Untersuchung unseres Ansatzes für Sequenz-

Alignment vielversprechend erscheint. Wir meinen, daß die vorgestellte Methode noch reichlich Raum für Verbesserungen bietet, wohingegen traditionelle Methoden, welche auf dem Prinzip der dynamischen Programmierung beruhen, schon seit langem untersucht werden und deshalb schwierig zu verbessern sind.

Übersicht
In Kapitel 2 definieren wir grundlegende mathematische Begriffe und Notationen. Wir wiederholen unter anderem fundamentale Sätze aus der polyedrischen Kombinatorik, der linearen Programmierung und der Theorie der Polyeder. Dieses Kapitel dient als Referenz für mathematische Notationen, die in der Arbeit ohne weitere Definiton vorkommen.

In Kapitel 3 führen wir zunächst den Begriff des paarweisen und multiplen Alignments ein. Wir definieren in einem allgemeinen Rahmen gebräuchliche Bewertungsfunktionen und erweitern diesen Rahmen so, daß er auch auf strukturelle Alignments angewandt werden kann. Trotzdem wir uns in unseren Experimenten ausschließlich mit RNA-Sequenzen beschäftigen, sind unsere Formulierungen so ausgelegt, daß wir jegliche Information verarbeiten können, die auf der Interaktion zweier Buchstaben in der Sequenz beruht.

In Kapitel 4 definieren wir den Begriff des paarweisen und multiplen Traces. Traces sind eine elegante Repräsentierung von Sequenzähnlichkeit, obwohl sie gewisse Unterschiede zu Alignments aufweisen. Wir stellen die Unterschiede und Gemeinsamkeiten von Traces und Alignments dar und zeigen, wie man viele verschiedene Alignment-Probleme mit Hilfe gewisser gewichteter Eingabe-Graphen dargestellen kann. Ein Alignment entspricht dann einem Untergraphen des Eingabegraphens, der gewisse Bedingungen erfüllt. Somit reduziert sich das Problem, ein optimales Alignment zu berechnen, auf das Problem, einen Untergraphen maximalen Gewichtes zu finden, der diese Bedingungen erfüllt.

In Kapitel 5 definieren wir formal das GMT- und SMT-Problem. Im Anschluß daran stellen wir in kurzer Form die momentan besten Algorithmen für beide Probleme vor, die auf dynamischer Programmierung beruhen.

In Kapitel 6 präsentieren wir den von uns gewählten Ansatz. Wir beginnen damit, in Abschnitt 6.1 die grundlegende Funktionsweise eines Branch-and-Cut-Algorithmus detailiert zu beschreiben. Danach beschreiben wir die problemspezifischen Details zuerst für das GMT-Problem in Abschnitt 6.2 und dann für das SMT-Problem in Abschnitt 6.3. In jedem der beiden Abschnitte gehen wir wie folgt vor: Zuerst formulieren wir das Problem als ganzzahliges, lineares Programm in Abschnitt 6.2.1 beziehungsweise 6.3.1. Das ganzzahlige, lineare Programm bildet die Grundlage des von uns gewählten Ansatzes. Es kann in natürlicher Weise mit einem hochdimensionalen Polytop assozi-

iert werden (die konvexe Hülle der Inzidenz-Vektoren aller zulässigen Lösungen). In der Praxis zeigt sich, daß die facetten-definierenden Ungleichungen des Problempolytops gute Schnittebenen darstellen. Aus diesem Grund untersuchen wir die Struktur des Problempolytops näher in Abschnitt 6.2.2 respektive 6.3.2. Das so gewonnene theoretische Wissen über das Problempolytop wird in Abschnitt 6.2.3 beziehungsweise 6.3.3 in effiziente Separierungsroutinen umgewandelt. Diese benutzen wir dann in einer Implementierung unseres Algorithmus, deren Resultate wir in den Abschnitten 6.2.4 und 6.3.4 beschreiben. Dabei zeigen wir auf, daß der Einsatz von Methoden der kombinatorischen Optimierung für Sequenz-Alignment Probleme zu Algorithmen führt, die vergleichbar oder sogar besser als vorhandene Algorithmen sind, welche auf dynamischer Programmierung beruhen.

In Kapitel 7 diskutieren wir unsere Ergebnisse und zeigen interessante offene Probleme auf, welche wir im Zuge der Arbeit identifiziert haben.

Chapter 9

Glossary

3'-end
: The 3'-end of a single DNA strand is named after the orientation of the 3' and 5' carbon atoms in the sugar ring at this end of the strand. By convention a DNA string is read from the 5'-end to the 3'-end.

5'-end
: The 5'-end of a single DNA strand is named after the orientation of the 3' and 5' carbon atoms in the sugar ring at this end of the strand. By convention a DNA string is read from the 5'-end to the 3'-end.

adenine
: A purine base that pairs with thymine.

anticodon
: A nucleotide triplet in a tRNA molecule that aligns with a particular codon in mRNA under the influence of the ribosome, so that the amino acid carried by the tRNA is inserted in a growing protein chain.

codon
: A section of DNA (three nucleotide pairs) that codes for a single amino acid.

complementary base pair
: In double helix DNA ansd intramoleculary folded RNA bases build hydrogen-bonded pairs. The most favorable ones are adenine-thymine (or uracil) and cytosine-guanine.

cytosine
: A pyrimidine base that pairs with guanine.

gene
: The fundamental physical and functional unit of heredity that carries information from one generation to the next; a segment of DNA, composed of a transcribed region and a regulatory sequence that make transcription possible.

guanine
: A purine base that pairs with cytosine.

hydrogen bond
: A weak bond involving the sharing of an electron with a hydrogen atom; hydrogen bonds are important in the specifity of base pairing in nucleic acids and the determination of protein shape.

inosine
: A rare base that is important at the wobble position of some tRNA anticodons.

mRNA (messenger RNA)
: An RNA molecule transcribed from the DNA of a gene, from which a protein is translated by the actions of ribosomes.

phylogenetic tree
: A rooted tree that depicts the anchestral relationship of species. The vertices correpond to species. The children of a vertex represent direct descendents of that vertex whereas that vertex is the direct ancestor of its children. Two leaves that have a common ancestor near the root are assumed to have diverged earlier in evolutionary history than two leaves that have a common ancestor far from the root. The edges of the tree may be weighted to measure the evolutionary distance.

promoter
: A regulator region at a short distance from the 5'-end of a gene that acts as the binding site for RNA polymerase.

pseudoknot
: RNA molecules fold intramolecularly and form hydrogen-bonded base pairs. This secondary structure occurs normally such that it can be depicted as a planar graph. Base pairs that destroy the planarity are called *pseudoknots*.

purine
: A type of nitrogen base; the purine bases in DNA are adenine and guanine.

pyrimidine
: A type of nitrogen base; the pyrimidine bases in DNA are cytosine and thymine; in RNA uracil instead of thymine.

RNA

Ribo **N**ucleic **A**cid. A generally single-stranded nucleic acid similar to DNA but having ribose sugar rather than deoxyribose sugar and uracil rather than thymine as one of the bases.

RNA polymerase

An enzyme that catalyzes the synthesis of an RNA strand from a DNA template. In eukaryotes, there are several classes of RNA polymerase. Structural genes for proteins are transcribed by RNA polymerase II.

ribosome

A complex organelle that catalyzes translation of messenger RNA into amino acid sequence. Composed of proteins and ribosomal RNA (rRNA).

template

A molecular "mold" that shapes the structure or sequence of another molecule; for example, the nucleotide sequence of DNA acts as a template to control the nucleotide sequence of RNA during transcription.

thymine

A pyrimidine base that pairs with adenine.

transcription

The synthesis of RNA using a DNA template.

translation

The ribosome-mediated production of a polypeptide whose amino acid sequence is derived from the codon sequence of an mRNA molecule.

tRNA

A class of small RNA molecules that bear specific amino acids to the ribosome during translation; the amino acid is inserted into the growing polypeptide chain when the anti-codon of the tRNA pairs with a codon on the mRNA being translated.

uracil

A pyrimidine that appears in RNA in place of thymine found in DNA.

Watson-Crick

J.D. Watson and Crick first showed the hydrogen bonded structure of DNA. The two most common base pairs `A-U` and `C-G` are therefore commonly referred to as Watson-Crick pairs.

Bibliography

Altschul, S. 1989. Gap costs for multiple sequence alignment. *Journal of Theoretical Biology* 138, 297–309.

Altschul, S., Carroll, R., and Lipman, D. 1989. Weights of data related by a tree. *Journal of Molecular Biology* 207, 647–653.

Altschul, S., and Erickson, B. 1986. Locally optimal subalignments using nonlinear similarity functions. *Bulletin Mathematical Biology* 48, 633–660.

Altschul, S., Gish, W., Miller, W., Myers, E., and Lipman, D. 1990. A basic local alignment search tool. *Journal of Molecular Biology* 215, 403–410.

Altschul, S., and Lipman, D. 1989. Tree, stars, and multiple biological sequence alignment. *SIAM Journal Applied Mathematics* 49(1), 179–209.

Applegate, D., Bixby, R., Chvátal, V., and Cook, B. 1995. Finding cuts in the TSP. DIMACS Technical Report 95-05, DIMACS.

Bafna, V., Lawler, E. L., and Pevzner, P. A. 1994. Approximation algorithms for multiple sequence alignment. In: *Proceedings Fifth Annual Symposium on Combinatorial Pattern Matching (CPM-94)*, 43–54.

Bafna, V., Muthukrishnan, S., and Ravi, R. 1995. Computing similarity between RNA strings. In: Z. Galil, and E. Ukkonen, eds., *Proceedings of the Sixth Annual Symposium on Combinatorial Pattern Matching (CPM-95)*, no. 937 in Lecture Notes in Computer Science, 1–16, Springer, Berlin.

Bairoch, A., and Apweiler, R. 1999. The swiss-prot protein sequence data bank and its supplement TrEMBL in 1999. *Nucleic Acids Research* 27, 49–54.

Benner, S., Cohen, M., and Gonnet, G. 1993. Empirical and structural models for insertions and deletions in the divergent evolution of proteins. *Journal of Molecular Biology* 220, 1065–1082.

Beyer, W., Stein, M., Smith, T., and Ulam, S. 1974. A molecular sequence metric and evolutionary trees. *Mathematical Biosciences* 19, 9–25.

Carrillo, H., and Lipman, D. J. 1988. The multiple sequence alignment problem in biology. *SIAM Journal on Applied Mathematics* 48(5), 1073–1082.

Christof, T., Jünger, M., Kececioglu, J., Mutzel, P., and Reinelt, G. 1997. A branch-and-cut approach to physical mapping with end-probes. In: *Proceedings of the First Annual International Conference on Computational Molecular Biology (RECOMB-97)*, 84–93, ACM Press, Santa Fe.

Chvátal, V. 1983. *Linear Programming*. A Series of Books in the Mathematical Sciences, Freeman, New York.

Cook, W., Cunningham, W., Pulleyblank, W., and Schrijver, A. 1998. *Combinatorial Optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons.

Corpet, F., and Michot, B. 1994. RNAlign program: alignment of RNA sequences using both primary and secondary structures. *CABIOS* 10(4), 389–399.

Dayhoff, M., Schwartz, R., and Orcut, B. 1979. A model of evolutionary change in proteins. In: M. Dayhoff, ed., *Atlas of Protein Sequence and Structure*, vol. 5, 345–352, National Biomedical Research Foundation, Washington, D.C.

de Rijk, P., de Peer, Y. V., Chapelle, S., and de Wachter, R. 1994. Database on the structure of the small ribosomal subunit RNA. *Nucleic Acids Research* 22, 3495–3501.

Dell'Amico, M., Maffioli, F., and Martello, S., eds. 1997. *Annotated bibliographies in combinatorial optimization*. Wiley-interscience series in discrete mathematics and optimization, John Wiley & Sons, Chichester.

Eddy, S., and Durbin, R. 1994. RNA sequence analysis using covariance models. *Nucleic Acids Research* 22(11), 2079–2088.

Fitch, W. 1966. An improved method of testing for evolutionary homology. *Journal of Molecular Biology* 16, 9–16.

Garey, M., and Johnson, D. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman.

Gomory, R. 1958. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society* 64, 275–278.

Gonnet, G., Cohen, M., and Benner, S. 1992. Exhaustive matching of the entire protein sequence database. *Science* 256, 1443–1445.

Gorodkin, J., Heyer, L., and Stormo, G. 1997. Finding the most significant common sequence and structure motifs in a set of RNA sequences. *Nucleic Acids Research* 25, 3724–3732.

Gotoh, O. 1982. An improved algorithm for matching biological sequences. *Journal of Molecular Biology* 162, 705–708.

Grantham, R. 1974. Amino acid difference formula to help explain protein evolution. *Science* 185, 862–864.

Grötschel, M., Jünger, M., and Reinelt, G. 1984. A cutting plane algorithm for the linear ordering problem. *Operations Research* 32, 1195–1220.

Grötschel, M., Lovász, L., and Schrijver, A. 1981. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica* 1, 169–197.

Grötschel, M., and Padberg, M. 1985. Polyhedral theory. In: E. Lawler, J. Lenstra, and A. R. Kan, eds., *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley Interscience, Chichester.

Gupta, S., Kececioglu, J., and Schaeffer, A. 1995. Improving the practical space and time efficiency of the shortest-paths approach to sum-of-pairs multiple sequence alignment. *Journal Computational Biology* 2, 459–472.

Hammer, P., Johnson, E., and Peled, U. 1975. Facets of regular 0-1-polytopes. *Mathematical Programming* 8, 179–206.

Henikoff, S., and Henikoff, J. 1992. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Science* 89, 10915–10919.

Hoffmann, A. 1974. A generalization of max flow-min cut. *Mathematical Programming* 6, 352–359.

Isaacson, W. 1999. The biotech century. *Time* 153(1), 42–43.

Jones, D., Taylor, W., and Thornton, M. 1992. The rapid generation of mutation data matrices from proteins sequences. *CABIOS* 8, 275–282.

Jünger, M., Reinelt, G., and Rinaldi, G. 1995a. The traveling salesman problem. In: M. Ball, T. Magnanti, C. Monma, and G. Nemhauser, eds., *Handbook on Operations Research and Management Science*, 225–330, Elsevier, North Holland.

Jünger, M., Reinelt, G., and Thienel, S. 1995b. Practical problem solving with cutting plane algorithms in combinatorial optimization. In: W. Cook, L. Lovász, and P. Seymour, eds., *Combinatorial Optimization: Papers from the DIMACS Special Year*, no. 20 in DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 111–152, AMS, Providence, RI.

Jünger, M., and Thienel, S. 1997. The design of the branch and cut system ABACUS. Tech. Rep. 97.260, Institut für Informatik, Universität zu Köln.

Karp, R., and Papadimitriou, C. 1980. On linear characterizations of combinatorial optimization problems. In: *Proceedings of the 21st Annual Symposium on the*

Foundations of Computer Science (FOCS 80), 1–9.

Kececioglu, J. 1991. *Exact and approximation algorithms for DNA sequence reconstruction*. Ph.D. thesis, University of Arizona.

Kececioglu, J. 1993. The maximum weight trace problem in multiple sequence alignment. In: *Proceedings of the 4th Symposium on Combinatorial Pattern Matching (CPM 93)*, no. 684 in Lecture Notes in Computer Science, 106–119, Springer.

Kececioglu, J. 1998. Personal communication.

Kececioglu, J. 1999. Personal communication.

Kececioglu, J., and Zang, W. 1998. Aligning alignments. In: *Proceedings of the 9th Annual Symposium on Combinatorial Mattern Matching (CPM-98)*, no. 1448 in Lecture Notes in Computer Science, 189–208.

Kececioglu, J. D., and Myers, E. W. 1995. Combinatorial algorithms for DNA sequence assembly. *Algorithmica* (13), 7–51.

Lenhof, H.-P., Morgenstern, B., and Reinert, K. 1999. An exact solution for the segment-to-segment multiple sequence alignment problem. *BIOINFORMATICS* 15(3), 203–210.

Lenhof, H.-P., Reinert, K., and Vingron, M. 1998. A polyhedral approach to RNA sequence structure alignment. In: *Proceedings of the Second Annual International Conference on Computational Molecular Biology (RECOMB-98)*, 153–162, ACM Press, New York.

Lermen, M., and Reinert, K. 1997. The practical use of the $\mathcal{A}^*$ algorithm for exact multiple sequence alignment. Research Report MPI-I-97-1-028, Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany.

Levitt, M. 1969. Detailed molecular model for transfer ribonucleic acid. *Nature* 224, 759–763.

McClure, M., Vasi, T. K., and Fitch, W. M. 1994. Comparative analysis of multiple protein-sequence alignment methods. *Mol. Biol. Evol.* 4(11), 571–592.

Mehlhorn, K., and Näher, S. 1995. LEDA, a platform for combinatorial and geometric computing. *Communications of the ACM* 38(1), 96–102.

Morgenstern, B., Atchley, W., Hahn, K., and Dress, A. 1998. Segment-based scores for pairwise and multiple sequence alignments. In: J. Glasgow, T. Littlejohn, F. Major, R. Lathrop, D. Sankoff, and C. Sensen, eds., *Proceedings of the Sixth International Conference on Intelligent Systems for Molecular Biology (ISMB-98), (in press)*, 115–121, AAAI Press.

Morgenstern, B., Dress, A., and Werner, T. 1996. Multiple DNA and protein sequence alignment based on segment-to-segment comparison. In: *Proceedings of the National Academy of Science*, no. 93, 12098–12103.

Needleman, S., and Wunsch, C. 1970. A general method applicable to the search for similarities in the amino-acid sequence of two proteins. *Journal Molecular Biology* 48, 443–453.

Nemhauser, G., Kan, A. R., and Todd, M., eds. 1989. *Optimization*, vol. 1 of *Handbooks in Operations Research and Management Science*. North-Holland, Amsterdam.

Nemhauser, G., and Trotter, L. 1973. Properties of vertex packing and independence system polyhedra. *Mathematical Programming* 6, 48–61.

Niefind, K., and Schomburg, D. 1991. Amino acids similarity coefficients for protein modeling and sequence alignment derived from main-chain folding angles. *Journal of Molecular Biology* 219, 481–497.

Notredame, C., O'Brien, E., and Higgins, D. 1997. RAGA: RNA sequence alignment by genetic algorithm. *Nucleic Acids Research* 25, 4570–4580.

Padberg, M., and Rao, M. 1981. The Russian method for linear inequalities III: Bounded integer programming. GBA Working Paper 81–39, New York University.

Padberg, M. W., and Rinaldi, G. 1987. Optimization of a 532 city symmetric traveling salesman problem by branch and cut. *Operations Research Letters* 6, 1–7.

Pevzner, P. A., and Waterman, M. S. 1993. Generalized sequence alignment and duality. *Advances in Applied Mathematics* 14, 139–171.

Pulleyblank, W. 1989. Polyhedral combinatorics. In: G. Nemhauser, A. R. Kan, and M. Todd, eds., *Handbooks in Operations Research and Management Science*, North Holland, Amsterdam.

Reinert, K., Lenhof, H.-P., Mutzel, P., Mehlhorn, K., and Kececioglu, J. 1997. A branch-and-cut algorithm for multiple sequence alignment. In: *Proceedings of the First Annual International Conference on Computational Molecular Biology (RECOMB-97)*, 241–249, ACM Press, Santa Fe, NM.

Risler, J., Delorme, M., Delacroix, H., and Henaut, A. 1988. Amino acid substitutions in structurally related proteins. A pattern recognition matrix. *Journal of Molecular Biology* 204, 1019–1029.

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

Sankoff, D., and Kruskal, J. 1983. *Time Warps, String Edits and Macromolecules: the Theory and Practice of Sequence Comparison*. Addison-Wesley.

Schrijver, A. 1986. *Theory of Linear and Integer Programming*. Wiley-interscience series in discrete mathematics and optimization, Wiley, Chichester.

Schwikowski, B. 1998. *A new algorithmic approach to the construction of multiple alignments and evolutionary trees*. Ph.D. thesis, Universität Bonn.

Stoesser, G., Moseley, M., Sleep, J., McGowran, M., Garcia-Pastor, M., and Stark, P. 1998. The EMBL nucleotide sequence database. *Nucleic Acids Research* 26(1), 8–15.

Tsukiyama, S., Ide, M., Ariyoshi, H., and Shirawaka, I. 1977. A new algorithm for generating all the maximal independent sets. *SIAM Journal on Computing* 6, 505–517.

Vingron, M., and Pevzner, P. 1995. Multiple sequence comparison and consistency on multipartite graphs. *Advances in Applied Mathematics* 16, 1–22.

Waterman, M. 1989. Consensus methods for folding single-stranded nucleic acids. In: *Mathematical Methods for DNA Sequences*, 185–224, CRC Press.

Wilbur, W., and Lipman, D. 1984. The context dependent comparison of biological sequences. *SIAM Journal on Applied Mathematics* 44(3), 557–567.

Index

U

V

W

Z