DIPLOMARBEIT

RNA STRUCTURAL ALIGNMENT BY MEANS OF LAGRANGIAN RELAXATION

ausgeführt
zum Zwecke der Erlangung des
akademischen Grades eines
Diplom - Ingenieurs
am
Institut für Computergraphik und Algorithmen 186/1
der
Technischen Universität Wien
unter der Leitung von
Univ. - Prof. Dr. Petra Mutzel
und
Dr. Gunnar W. Klau
durch

Markus Bauer
Matr. - Nr. 9925289
Westbahnstraße 21/1/21, 1070 Wien

Wien, am 23. Juni 2004 .....

RNA Structural Alignment by Means of Lagrangian Relaxation

This thesis deals with an important topic in computational molecular biology: structurally correct alignments of RNA sequences. Compared to DNA sequences where sequence information is normally sufficient for adequate alignments, the structural aspects of RNA have to be taken into account when dealing with RNA sequences: structure of RNA sequences tends to remain conserved throughout evolution.

By adapting an algorithm for the similar task of aligning proteins, a new algorithm based on Lagrangian relaxation is described that aligns two RNA sequences without restricting the secondary structure. Previous dynamic programming approaches are able to deal only with pseudoknot-free structures.

The comparison of the new method with previous ones shows that the new algorithm yields – in general – better structural alignment scores and requires less resources in terms of memory and CPU usage.

Strukturelle RNA-Alignments mittels Lagrange Relaxierung

Thema dieser Diplomarbeit ist ein wichtiges Gebiet der *computational molecular biology*: das Berechnen strukturell richtiger Alignments von RNA-Sequenzen. Im Gegensatz zu DNA ist bei RNA die sekundäre Struktur von enormer Bedeutung, da diese im Lauf der Evolution konserviert wird. Sie muss daher bei Alignments ebenfalls beachtet werden.

Ein Algorithmus für das verwandte Problem des Protein-Alignments wird vorgestellt und in weiterer Folge für das strukturelle Alinieren von zwei RNA-Sequenzen modifiziert. Im Gegensatz zu den meisten vorhergehenden Algorithmen schränkt die hier präsentierte Methode die Sekundärstruktur von RNA nicht ein, d.h. es werden auch Pseudoknoten berücksichtigt.

Ein Vergleich mit anderen Ansätzen zeigt, dass der Algorithmus zu den Schnellsten mit den geringsten Anforderungen hinsichtlich CPU und Hauptspeicher zählt, wobei die Ergebnisse aber im Allgemeinen besser als die der anderen Algorithmen sind.

Danksagung

An dieser Stelle möchte ich mich bei all jenen bedanken, die in direkter oder indirekter Weise zum Gelingen dieser Arbeit beigetragen haben:

Gunnar Klau danke ich für die umfassende, ausgezeichnete Betreuung während der letzten Monate, Prof. Knut Reinert von der FU Berlin für die zahlreichen Hinweise und Vorschläge und René Weiskircher für das Aushelfen bei Problemen mit LEDA. Ebenfalls möchte ich Prof. Petra Mutzel für die Möglichkeit danken, die Diplomarbeit an ihrer Abteilung zu schreiben, und Prof. Günther Raidl für die Bereitschaft, Prof. Mutzel zu vertreten.

Meinen Studienkollegen und Freunden bin ich für die moralische Unterstützung und den Spass der letzten Jahre zu Dank verpflichtet: Danke Schero, Steff, Bernhard, Anton, Roel, Guido, Esther und vielen anderen.

Zuletzt möchte ich mich bei meinen Eltern und Geschwistern für ein sorgloses Studium bedanken und bei Martina für die schöne gemeinsame Zeit.

Contents

List of Figures

List of Tables

I'll need some information first
Just the basic facts

PINK FLOYD

Chapter 1

Introduction

1.1 Biological Background

In the 1860's Gregor Mendel performed experiments on heredity and discovered that heredity is not just a random process. By observing some generations of begonias he was able to conceive the *Mendel's law* and pointed out the existence of what we now call *genes*.

After that, it took some decades to fully understand the mechanisms behind the fundamental processes like heredity and reproduction in nature. Nowadays we know that the entire information is stored in *genes*, sequences of *desoxyribonucleic acid* (*DNA* in short) or '*ribonucleic acid* (*RNA* in short). These *genes* encode the *proteins*, the main functional units of an organism.

Proteins basically control every part of a human organism, they set, *e.g.*, the color of our hair, the shape of our body or the length of our arms. The interplay between DNA, RNA and proteins can schematically be drawn as

$$\textbf{DNA} \rightarrow \textbf{RNA} \rightarrow \textbf{protein}$$

Basically, DNA stores all the information. RNA is – roughly stated – a helper molecule that is necessary to convert the genetic information into functional units, the *proteins*.

1.1.1 Desoxyribonucleic Acid

DNA is one of the two main biopolymers in nature. *DNA* is built of four different *nucleotides*, namely *adenine*, *guanine*, *cytosine*, and *thymine*. Each nucleotide is commonly designated with its first letter: A, G, C and T. The difference between the single nucleotides is the different nitrogenous base.

DNA builds a "double helix", a linear, double-stranded structure, discovered in 1953 by James Watson and Francis Crick. The structure is held together by hydrogen bonds between *Watson-Crick-pairs*, namely **A-T** and **G-C** (see Figure 1.1).

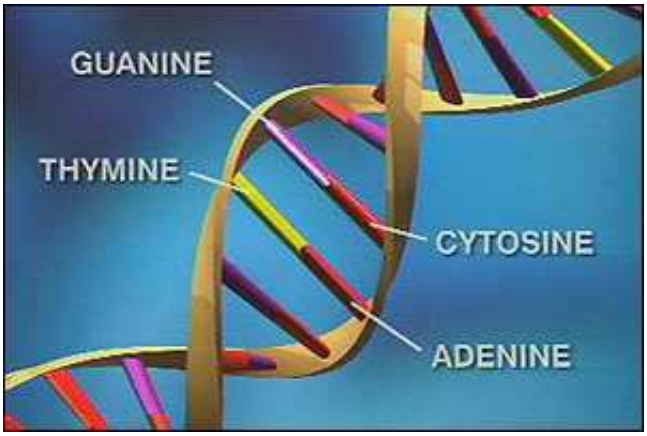

Figure 1.1: Structure of DNA, taken from Lindsay (2004)

Triplets of nucleotides form a *codon* (*e.g.*, CCG codes proline, AUU for isoleucine). Each codon denotes one out of the twenty *amino acids*.

1.1.2 Ribonucleic Acid

The second important biopolymer is *ribonucleic acid*, or *RNA* in short. Compared to DNA, there are some differences:

Firstly, there are also four nucleotides forming a RNA strand, but *uracil* is replaced by *thymine*. Secondly, RNA does not have a double-helix as its structure, but it is a single-stranded molecule that has the property of folding back onto itself. By folding back, it forms pairs of **GC**, **AU** and less stable **GU**. These pairs form the *secondary structure* of an RNA sequence (details will be presented in the following sections, see Figure 1.2 for a preview).

RNA is biologically important in several ways:

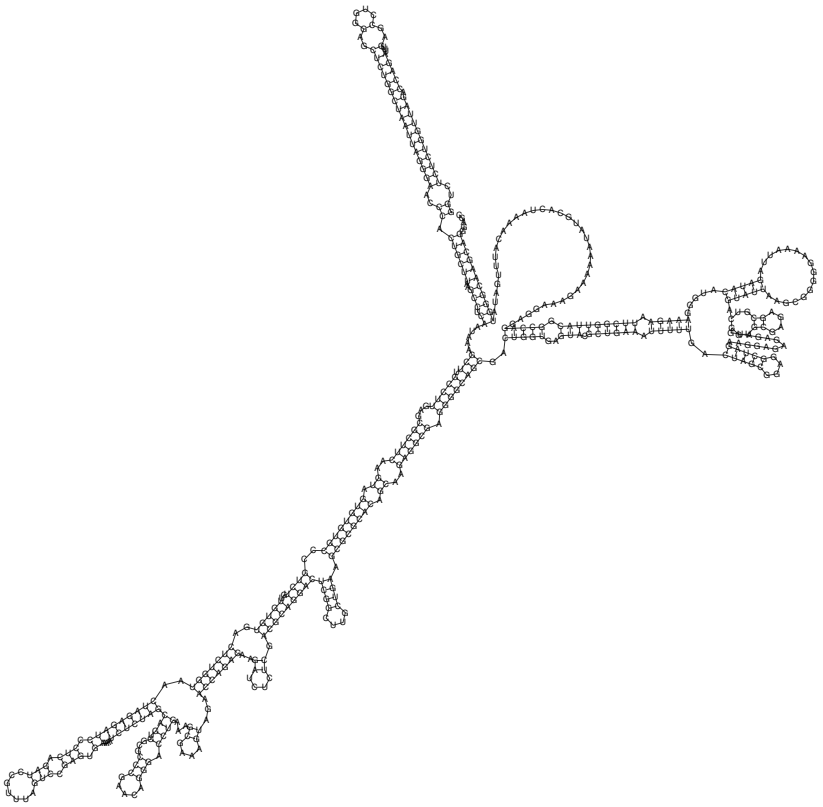

Figure 1.2: Structure of RNA, generated by alidot, Hofacker et al. (1998)

- The list of all genetic information for an organism is called its *genome*. Normally, genomes are made of DNA, but in some viruses the genome is made of RNA.
- RNA plays an important role as a "copy molecule" (*messengerRNA*, or *mRNA* in short) The plan for building a new protein is first copied to mRNA and afterwards taken to the *ribosome*, the "working machinery" in a living cell. The ribosome builds a new protein by reading the information provided by the mRNA.
- RNA acts as a carrier for other molecules. There are twenty of these *transfer RNA*, each for one of the twenty amino acids. *tRNA* brings the amino acids that are needed for the protein production (called the *protein synthesis*) to the ribosome.
- RNA can drive reactions between molecules by itself (this property is called *catalytic property*). Catalysts reduce the amount of energy that

a reaction needs to take place and make sure that the desired reaction is specific and accurate. Until the early 1980's the general opinion was that only molecules built on DNA (so called *enzymes*) have this property, but the research group around Tom Cech showed 1982 that RNA has catalytic properties as well. In 1989, together with Sidney Altman, he received the Nobel prize in chemistry for the discovery of the *ribozymes*.

The large variety of important cell functions that RNA offers depend on special structural properties and therefore it is worth taking a closer look at the RNA secondary structure.

As mentioned above, RNA is a single-stranded molecule that forms a structure by folding back onto itself. There are three hydrogen bonds between **GC** and **AU**, but only two between **GU**. All interactions together form the *secondary structure*. There are three common ways to draw secondary structures:

1. "(", respectively ")" denote paired base pairs, "." denote unpaired nucleotides (this is the *string representation*, see Figure 1.3).

```
GUCUAAAUUGACCGUACAUGAGGGCACGGCCAGCAUG
(((......)))....((((..(((...)))..))))
```

Figure 1.3: Secondary structure in string representation

2. Nucleotides denote vertices, interactions denote edges in a graph (the *linked diagram representation*, see Figure 1.4).

Figure 1.4: Secondary structure in linked diagram representation

3. A more illustrative way to show a secondary structure is given in Figure 1.5).

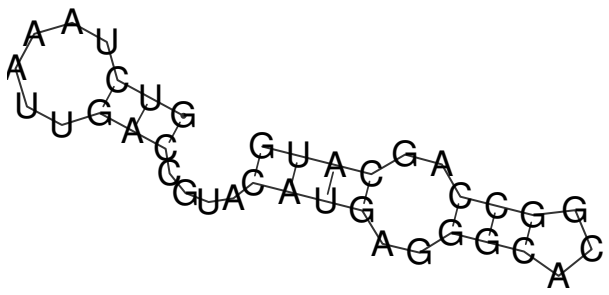

Figure 1.5: Secondary structure in two-dimensional representation

More formally, a RNA secondary structure is defined as follows (see Waterman, 1978):

A *secondary structure* S is a vertex labeled graph on n vertices with an adjacency matrix A satisfying the following properties:

1. $a_{i,i+1} = 1$ for $a \leq i < n$.
2. For each i there is at most a single $k \neq i-1, i+1$ such that $a_{i,k} = 1$.
3. If $a_{i,j} = a_{k,l} = 1$ and $i < k < j$ then $i < l < j$.

An edge (i, j) with $|i - j| \neq 1$ is called a *bond* or *base pair*. A vertex, not connected to any k except $k = i - 1$ and $k = i + 1$, is called *unpaired*.

If property 3 is not satisfied, the secondary structure is said to contain *pseudoknots*. Pseudoknots were not taken into account until recently. On the one hand there are not that many known pseudoknots in RNA structures, on the other hand pseudoknots make computations on secondary structures much more difficult. However, recently research was done on RNA pseudoknots (*e.g.*, Haslinger (2001)) showing that adding certain types of pseudoknots leads to better structure prediction of RNA sequences.

One of the most prominent questions related to RNA secondary structure is the question of predicting the accurate structure in case only the se-

quence itself (the *primary structure*) is given. If one knows the structure, one may draw conclusions about functional motifs within the structure. Since determining the structure experimentally is very expensive, structure prediction tools are used to obtain a set of reasonable structures.

RNA Secondary Structure Prediction

A very simple approach to predict the secondary structure due to Nussinov, Pieczenik, Griss, and Kleitman (1978) and Waterman (1978) is to maximize the number of paired base pairs without pseudoknots. This can efficiently be done by means of *dynamic programming*, but it is not very accurate. Folds of RNA sequences minimize the energy that is inherently contained in the structure and the energy does not correlate with the number of base pairs alone.

A more sophisticated model that works much better depends on energy rules (see for example Zuker, 1989; Zuker and Sankoff, 1984). The simplest way to do that is the following: assign a certain amount of energy, say $e(i, j)$, to every base pair. The energy assigned to pairs of nucleotides depends on several parameters, as the number of hydrogen bonds between the base pairs or the location of the base pairs. The energy of the whole structure is then given by

$$E(S) = \sum_{i,j \in S} e(i, j)$$

One is searching for the structure that forms the minimum energy of all possible structures (this structure is called the *minimum free energy* structure or *MFE* in short). The most prominent program that approaches the problem of predicting secondary structure by means of *minimum free energy* is Michael Zuker's program `mfold` (Zuker, 2004).

Although the energy model is more elaborate than simply counting paired base pairs, there are some disadvantages:

- The real secondary structure of an RNA sequence does not have to be the one with minimum free energy. It is possible that suboptimal foldings describe the real structure.
- There might be more structures with a minimum of free energy and more suboptimal structures that are just, say, 5-10% above the minimum.

- The energy potentials assigned to each base pair are empirically measured and describe only an approximation to the real values.

By computing MFE-structures for RNA sequences, one can also retrieve the so called *partition function* of the sequence. If $\mathbb{S}$ denotes the set of all possible structures, the *partition function* gives the probability for any structure S to be one that actually folds the sequence. Starting from the *partition function*, it is subsequently possible to compute the probability for *every* pair (i, j) that (i, j) forms a base pair in the secondary structure. These probabilities are called the *base pair probabilities* and are initially described in McCaskill (1990).

Another way to get the structure that is more likely to be the correct one is to compare structures that are based on different, but related sequences, such as virus genomes of the same family. Then, one searches for common structural elements, the *functional motifs*, that are contained in every sequence.

RNA Structure Comparison

This section deals with structure comparison and structure alignment rather intuitively. A formal description will be given in Section 1.2.

As written in Section 1.1.2, the variety of functions of RNA depend on its structural properties. Since it is not certain that programs like `mfold` produce correct secondary structures, comparing *MFE*-structures or suboptimal structures of evolutionary related sequences helps to find a common structure, that is, structural motifs that can be found in every (or a majority) of the sequences.

Secondary structure is highly conserved during evolution and therefore of great importance in interpreting the function of RNA sequences. Simulations (see Fontana, Konings, Stadler, and Schuster (1993); Schuster, Fontana, Stadler, and Hofacker (1994) for details) show that mutations of 10% of all nucleotides in an RNA sequence lead to unrelated structure which means that the sequence itself does not change that much, but the secondary structures are completely different.

Since the number of mutations in RNA sequences is very high, the sequence itself changes very often. If the structure remains the same throughout the entire process of mutation, this is called a *compensatory mutation*. Compensatory mutations are a strong indication of highly conserved units within a RNA sequence, since nature tends to reuse successful

spots in a given sequence. If someone detects a part of highly conserved structure, it is very likely that this part of the sequence offers a specific function.

Figure 1.6 shows such compensatory mutations, indicated by a circle around the nucleotides. The nucleotides in the sequences are different, that is in sequence 10, for example, position 20 is a **G**, whereas in sequence 15 position 20 is a **C**. The structure, however, remains the same.

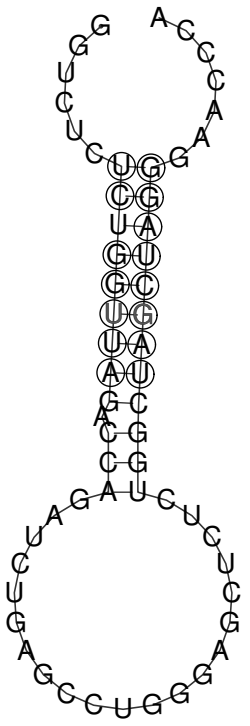

Figure 1.6: Compensatory mutations in 15 HIV sequences, generated by alidot, Hofacker et al. (1998)

This leads directly to the notion of an *alignment*. An alignment provides the possibility to measure how different or how similar two (or more) sequences are. A formal description of alignment and secondary structure will be given in Section 1.2.

1.2 Alignments

Changes in biopolymers like DNA or RNA are a natural process: errors in replication lead to insertions and deletions within a sequence and it is of

great importance to discover these evolutionary relationships afterwards.

Alignments are a way to quantify the relation between different sequences, respectively to assign a *score* to the comparison of sequences. This score, in the best case, should reflect the evolutionary correlation between the sequences: the higher the score, the higher the probability that the two sequences have a common ancestor sequence.

One of the first successful approaches of sequence alignment to the field of molecular biology is due to Doolittle, Hunkapiller, Hood, Devare, Robbins, Aaronson, and Antoniades (1983). The authors showed that genes in the *simian sarcoma virus*, leading to a cancer infection within the organism, correlate significantly with a gene that is responsible for cell growth. Knowing the correlation between the "cancer" and the "growth" gene, it was concluded that cancer may be caused by a normal growth gene switched on at the wrong time.

An intuitive definition of alignments is the following: we are given

- a finite set of characters
- a finite set Ω of operations (like insertions, deletions)

The score function assigns a value $f(o)$ to each of the given operations $o \in \Omega$. The overall score is then defined as

$$\min \parallel \max \sum_{o \in \Omega} f(o)$$

The following sections will give a formal definition of traditional sequence alignments, and afterwards move on to structural alignments. The treatment of this subject, especially the treatment of alignment algorithms, is not complete by any means. For further reading the reader is referred to the classical textbooks (like Waterman, 2000; Pevzner, 2000; Gusfield, 1997).

1.2.1 Sequence Alignments

Definition 1.2.1. *Let Σ be a finite alphabet without the blank character '-' and let $\hat{\Sigma} = \Sigma \cup '-'$. If S_1, S_2 are two strings over the alphabet Σ with lengths n_1 and n_2 then a* pairwise sequence alignment *A of S_1 and S_2 are two strings $\hat{S}_1, \hat{S}_2 \in \hat{\Sigma}$ such that – displayed in a $2 \times n$-dimensional matrix (the* alignment matrix*) – the following properties hold:*

- $a_{i,j} \in \widehat{\Sigma}, \forall i = 1, 2$ and $1 \leq j \leq n$
- Sequence $\widehat{\Sigma}_i$ without blanks is equal to sequence S_i
- No column contains only blanks. This implies $\max\{n_1, n_2\} \leq n \leq n_1 + n_2$

Two characters a and b are said to be *aligned*, if they are placed in the same column of the alignment matrix. We call $a \neq b$ with $a, b \neq' -'$ a *mismatch*. $a = b$ denotes a *match* and if one of the two characters is a '-', it is called an *indel*[1]. Figure 1.7 shows an example for these operations.

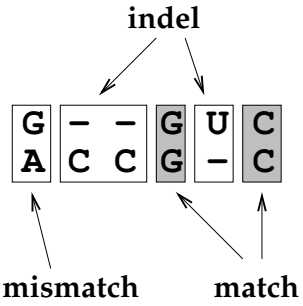

Figure 1.7: Possible occurrences in a sequence alignment

Scoring Sequence Alignments

Scores assigned to sequence alignments should be an indication of the correlation of the alignment:

Sequences that are very close to each other should have a high score, whereas sequences that do not have anything in common, for example two randomly chosen sequences from GenBank, a major source for sequence data (see for Biotechnology Information, 2004), should have a low score. The task of alignment algorithms is to find the alignment of the highest score.

A pairwise alignment of two sequences S_1, S_2 is written as $A(S_1, S_2)$. The set of all alignments between S_1 and S_2 is denoted by $A^2(S_1, S_2)$ and the set of all possible alignments between any two sequences by A^2. A pairwise sequence alignment score function is defined as follows

[1] *indel* is a composition of *insertion* and *deletion*. Since these two operations cannot be distinguished in an alignment, they are called *indels*.

Definition 1.2.2. *A function* $sc : A^2 \to \mathbb{R}$ *is called* alignment score function. *If* A *is an alignment of two sequences, then* $sc(A)$ *is called the score of the alignment* A*. The optimal alignment score of* S_1, S_2 *is defined as* $sc^{opt}(S_1, S_2) := opt_{A \in A^2(S_1,S_2)} sc(A)$*, where opt either denotes the maximum or the minimum.*

The common scoring schemes are based on comparing letters in $\hat{\Sigma}$. Every possible pair of letters (a, b) with $a, b \in \hat{\Sigma}$ is assigned a certain value that represents the score of aligning these two letters. There are two different ways of scoring pairs of letters:

- **Distance alignment** assigns a score $dist : \widehat{\Sigma} \times \widehat{\Sigma} \to \mathbb{R}$ that represents the distance between the two letters. Chemically similar letters get a low score, whereas chemically different letters get high scores.
- **Similarity alignment** assigns a score $sim : \widehat{\Sigma} \times \widehat{\Sigma} \to \mathbb{R}$ that represents how similar the two letters are. Chemically dissimilar letters get low scores, whereas chemically similar letters get high scores.

The alignment functions *dist* and *sim* are based on *mutation score matrices*. The simplest one assigns a match a value of 1, a mismatch counts $-\mu$ and an indel operation $-\delta$. The overall score is given by

$$sc(A^2(S_1, S_2)) := \#matches - \mu \#mismatches - \delta \#indels$$

A more elaborate example of *mutation score matrices* are given by the *PAM matrices* (*point access mutation*) due to Dayhoff, Schwartz, and Orcutt (1978). These matrices reflect the frequency with which a replaces b in evolutionary related sequences.

Given such score matrices, the distance score function is defined as

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

and a similarity score function defined as

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

There has been some discussion whether either similarity scores or distance scores are biologically more accurate. The discussion, however, is only of academic use, because similarity score functions are equivalent to distance score functions (see Waterman, 2000).

A simple dynamic programming approach to compute a similarity alignment of two sequences with a given *mutation score matrix* is based on a paper by Needleman and Wunsch (1970). Since this algorithm is an integral part of the algorithms described later on, we will present the main ideas.

Two sequences $S_1 = (a_1, \ldots, a_m)$ and $S_2 = (b_1, \ldots, b_n)$ are given. The idea is to compute for every pair (x, y), with $1 \leq x \leq m$ and $1 \leq y \leq n$, the maximum score D_{xy} that can be achieved including the positions x in S_1 and y in S_2, that is the substrings $[a_1 \ldots a_x]$ and $[a_1 \ldots a_y]$ of S_1 and S_2, respectively. We have two different choices to compute $D_{x,y}$:

1. **MATCH** a_x matches b_y, the overall score is given by $D_{x-1,y-1} + sim(a_x, b_y)$

2. **GAP** Either a_x is matched and a gap is inserted into the second sequence or vice versa. Therefore there are two choices for $D_{x,y} = D_{x-1,y} + sim(a_x, -)$ or $D_{x,y} = D_{x,y-1} + sim(-, b_y)$.

A recurrence relation follows right away:

$$D_{x,y} = \max \begin{cases} D_{x-1,y-1} & + \text{sim}(a_x, b_y) \\ D_{x-1,y} & + \text{sim}(a_x, -) \\ D_{x,y-1} & + \text{sim}(-, b_y) \end{cases}$$

Backtracking the solution is pretty simple: for every entry $D_{m,n}$ the "direction" from where the value was achieved (that is either a match or a gap in the first or second sequence) is stored. Then, backtracking is done by starting at position $D_{x,y}$ and going back until the beginning is reached. Figure 1.8 shows an example of computing the similarity score between the sequence $S_1 = \mathsf{AUCUGAU}$ and $S_2 = \mathsf{UGCAUA}$. The scoring function *sim* is simple: same letters have a score of one, different letters a score of 0. As one can easily verify is the running time of the algorithm in $\mathcal{O}(nm)$.

It is worth mentioning that there are interesting connections between the algorithm above and the problems of computing the *edit distance* between two strings or the computation of the *longest common subsequence*. In graph-theoretical terms, the algorithm computes a *non-crossing matching of maximum weight in a bipartite graph*. Again, the reader is referred to the literature, especially to the book by Gusfield (1997).

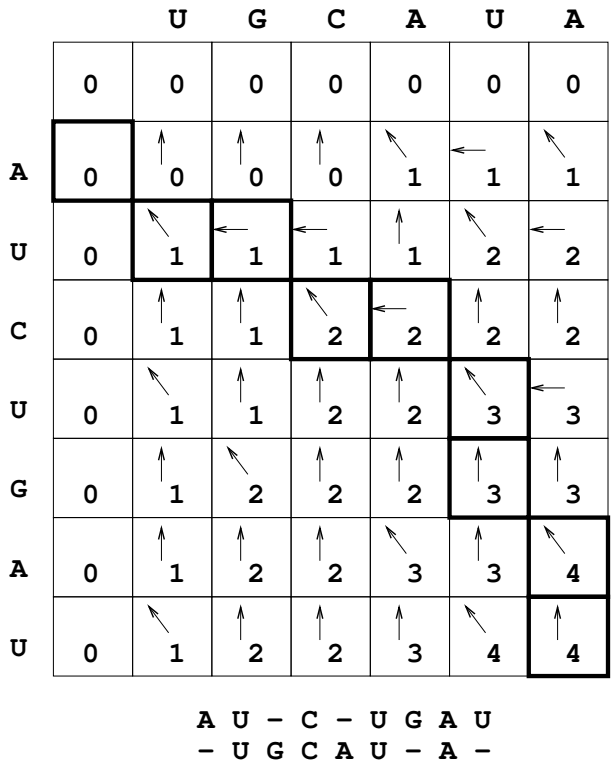

Figure 1.8: Example for Needleman-Wunsch

Scoring Gaps

Using point mutation matrices alone do not reflect the biological reality very precisely. Mutations like insertions or deletions normally do not affect a single point but rather longer parts of it. In a point mutation-based scoring scheme all these operations would be treated independently, whereas it is desirable to consider all these mutations as one single event. These considerations lead to the notion of a *gap*.

Definition 1.2.3. *A gap of length l is a maximal continuous sequence of blank characters ('-') in one of the two sequences.*

Basically, we distinguish between two main scoring schemes for gaps:

- linear gap costs
- affine gap costs

If inserting a gap has a score of g, in a linear scoring scheme the overall costs for a gap of length l would be gl. It is reasonable to argue that the penalty for a gap of length l should be less than the sum of the penalties for l independent gap, that is

$$g(l_1 + l_2 + l_3 + \cdots + l_n) \leq g(l_1) + g(l_2) + g(l_3) + \cdots + g(l_n)$$

In a scoring scheme with affine gap costs there are two parameters: the gap open penalty o and gap prolongation costs p, leading to

$$g(l) = o + pl \quad .$$

Nowadays, affine gap costs are the commonly used model and there are efficient algorithms that compute pairwise sequence alignments with affine gap costs (see Pevzner, 2000). Until very recently, the problem of computing an alignment of more than two sequences with affine gap costs (a so called *multiple alignment*) was not attacked because of exponential space and time requirements. Newer results due to Althaus, Caprara, Lenhof, and Reinert (2002) demonstrate an algorithm for computing alignments of more than two sequences with arbitrary gap costs. Although still exponential in nature, the algorithm can compute optimal multiple alignments of 20 sequences with up to 100 nucleotides.

1.2.2 Structural Alignments

As described in Section 1.1.2 a large collection of RNA specific functions depends on its secondary structure. Sequence conservation might not be that high (for example because of *compensatory mutations*, see Section 1.1.2) leading to a bad alignment score, whereas the secondary structure itself is highly conserved. Therefore, the problem of *RNA secondary structure alignment* (or *RSA* in short) is of peak importance in computational biology.

The first approach in considering sequence and structure simultaneously is due to Sankoff (1985) where the author proposes a dynamic programming approach to align and fold a set of given RNA sequences at once. Due to its high computational demands–$\mathcal{O}(n^6)$ and $\mathcal{O}(n^4)$ in CPU time and memory, respectively. Later on, genetic algorithms (see Notredame, O'Brien, and Higgins, 1997), Branch-and-Cut approaches (see Reinert, 1999) or more elaborate dynamic programming approaches due to Hofacker, Bernhart, and Stadler (2004) were used to approach structural alignments.

Before giving a formal definition for *RSA*, we need the following definition (taken from Reinert (1999)).

Definition 1.2.4. *Let S be a sequence over $\hat{\Sigma}$. A paired base pair (i, j) is called an* interaction *if $s_i \neq' -'$ and $s_j \neq' -'$. The set P of interactions is called the* annotation *of sequence S. Two interactions (i, j) and (k, l) are said to be in conflict, if they share one of the base pairs, that is either $i = k$ or $i = l$ or $j = k$ or $j = l$. In a secondary structure there are no two interactions in conflict. A pair (S, P) is called an* annotated sequence.

Definition 1.2.5. *Let Σ be a finite alphabet without the blank character '-' and let $\hat{\Sigma} = \Sigma \cup \{'-'\}$. If (S_1, P_1) and (S_2, P_2) are two annotated sequences $\in \hat{\Sigma}$ with lengths n_1 and n_2, a pairwise structural alignment A of (S_1, P_1) and (S_2, P_2) is a $2 \times n$-dimensional matrix consisting of two structured sequences $(\hat{S}_1, \hat{P}_1)$ and $(\hat{S}_2, \hat{P}_2)$ with the following properties:*

- $a_{i,j} \in \hat{\Sigma}, \forall 1 \leq j \leq n, i = 1, 2$
- *Sequence $\hat{\Sigma}_i = S_i$ without blanks*
- *There is no column consisting of blanks only, implying that max$(n_1, n_2) \leq n \leq n_1 + n_2$*
- *$\forall (l, m) \in \hat{P}_i$ holds $(l - gaps(i, l), m - gaps(i, j)) \in P_i$*

A pair of interactions (i, j) and (k, l) are said to be realized, if the corresponding base pairs are aligned, that is, $i = k$ and $j = l$. Figure 1.9 gives an example of realized interactions.

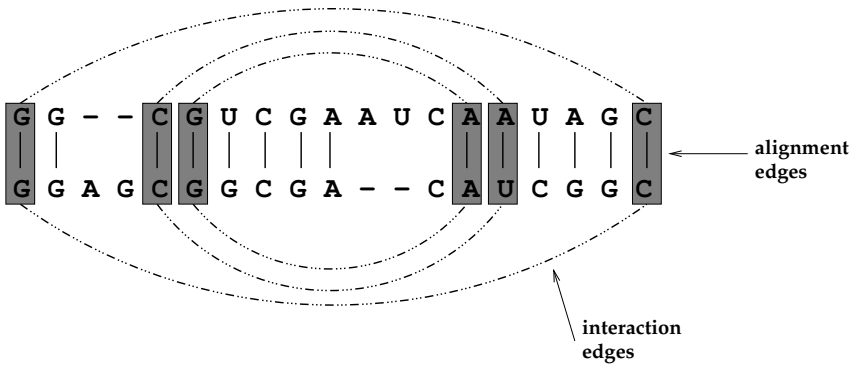

Figure 1.9: Structural alignment

In graph-theoretical terms (see Section 2.1) a structural alignment can be seen as a graph with a set of *alignment edges* between the sequence itself and a set of *interaction edges* between the nodes of each sequence. Then, a *structural alignment* is a set of *non-crossing* alignment edges with the set of *interaction edges*, such that the start- and endnodes of the interaction edges are realized by interaction edges (see Figure 1.10).

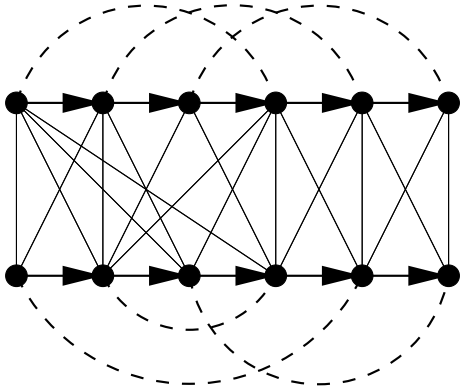

Figure 1.10: Graph-theoretic interpretation of structural alignments

Scoring Structural Alignments

As described earlier, structural alignments do not only consider sequence evolution and sequence correlation, but also the corresponding structure. Therefore, it is necessary to adapt the scoring schemes developed in Section 1.2.1 in order to reflect structural correlation as well.

Formally, a structural scoring scheme of two annotated sequences (S_1, P_1) and (S_2, P_2) is denoted by $A_s^2((S_1, P_1), (S_1, P_2))$. The concept of a structural alignment and a a structural alignment scoring function, respectively, can easily be extended to k annotated sequences (see Reinert, 1999). Since the algorithms developed later on are capable of computing the structural alignment of only two sequences, the reader is referred to the literature, *e.g.*, Reinert (1999), for details.

Definition 1.2.6. *A function* $sc : A_s^2 \to \mathbb{R}$ *is called* structural alignment score function. *If* A_s *is a structural alignment of two annotated sequences, then* $sc(A_s)$ *is called* pairwise structural alignment score *of* A_s. *The optimal* structural alignment score *of two annotated sequences* (S_1, P_1) *and* (S_2, P_2) *is given by* $sc_{opt}((S_1, P_1), (S_2, P_2)) := max_{A_s \in A^2((S_1,P_1),(S_2,P_2))} sc(A_s)$.

Normally, there are only *similarity scores* in use for defining structural correlation between two annotations. Therefore, there is no distinction between *similarity scores* and *distance scores* – like in case of *sequence alignments* – and no distinction between *min* and *max*.

In the special case of *RNA*, a scoring function for pairwise structural alignment is defined as

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

The function *isim* is defined as:

Definition 1.2.7. *A function* $isim : \Sigma^4 \rightarrow \mathbb{R}$ *assigns a score to two pairs of aligned characters that are defined by two matching interactions.*

Simply stated, two interactions that are realized by their aligned nucleotides contribute a certain score to the overall score. Whereas previous work (see Reinert, 1999) did not take any structure prediction information into account and every possible interaction has the same value, other authors (Hofacker et al., 2004) argue in favor for using *base pair probabilities* as weights (see Section 1.1.2).

Though, the task remains the same for all possible variations of $isim$:

Align the two given sequences in such a way that the sum of sequence and structural information is a maximum.

1.3 Guide to the Thesis

In Chapter 2 we give the basic definitions and notations, including the mathematical background to some key concept used in this thesis, *e.g.*, graph theory, linear programming or the theory behind Lagrangian relaxation.

Chapter 3 summarizes previous work done in the field of structural alignments. Some of these results are of great importance to the rest of the thesis, because the key ideas for the algorithms developed later on are based on the algorithms described in this chapter.

In Chapter 4 some previous results described in Chapter 3 will be put together to formulate a new algorithm for RNA structure alignments. Basically, this will be accomplished by putting together the mathematical for-

mulations presented in Section 3.1 and Section 3.2 and afterwards modifying the algorithm for aligning folded protein structures.

Chapter 5 presents the results of tests on RNA sequences. Finally, in Chapter 6 we draw some conclusions from the work presented in this thesis and sketch further work that could be done based on the ideas described in this thesis.

I was just guessing at numbers and figures
Pulling the puzzles apart

COLDPLAY

Chapter 2

Preliminaries

2.1 Graph Theory

The following definitions are taken from Diestel (1997) and Reinert (1999):

Definition 2.1.1. *A* graph *is a pair $G = (V, E)$ of sets satisfying $E \subseteq [V]^2$. The elements of* V *are called* nodes *(or* vertices*), elements of* E *are called* edges*. The set of nodes in G are denoted by $V = V(G)$, the set of edges by $E = E(G)$.*

Elements of $E(G)$ are denoted by the set of their *endnodes* (or *endvertices*), i.e., we write $e = (v, w)$ for $e \in E(G)$. A vertex v is *incident* with an edge e if $v \in e$; then e is an edge at v. Two vertices $x, y \in V(G)$ are *adjacent*, or *neighbors*, if $(x, y) \in E(G)$. Two edges $e \neq f$ are *adjacent* if they have an end in common. Pairwise non-adjacent vertices or edges are called *independent*. A set of vertices or of edges is said to be *independent*, if no two of its elements are adjacent.

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 $E(V') := \{(v, w) \in E | v, w \in V'\}$ and call the subgraph $G' = (V', E')$ *node-induced* by V'. Similarly if $E' \subseteq E$ we define $V(E') := \{v, w \in V | (u, v) \in E'\}$ and call the subgraph $G' = (V', E')$ *edge-induced* by E'.

The *degree* $d_G(v) = d(v)$ of a node v is the number $|E(v)|$ of adjacent nodes in $V(G)$.

Definition 2.1.2. *A graph $G = (V, E)$ is said to be* k-partite*, with $k \geq 2$, if V admits a partition into r classes such that every edge has its endnodes in different classes: nodes in the same class must not be adjacent.*

Instead of *2-partite*, one usually says *bipartite*. Figure 2.1 shows a 3-partite and a bipartite graph.

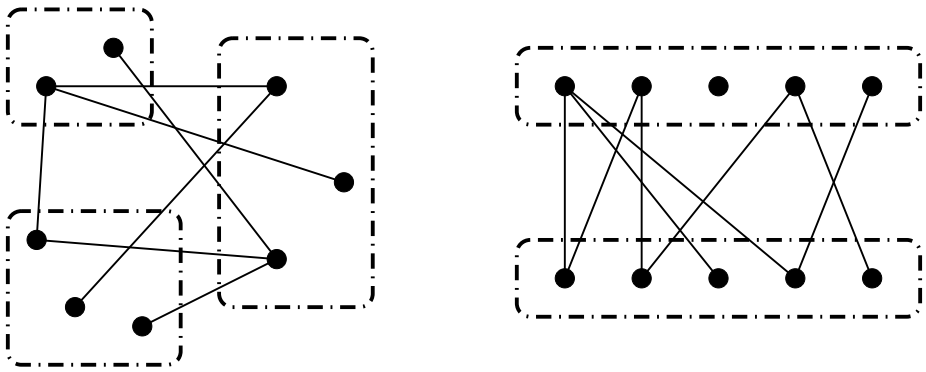

Figure 2.1: 3-partite and bipartite graph

For a *directed graph*, *digraph* in short, $D = (V, A)$ we denote the node set by $V = V(D)$ and the *arc* set by $A = A(D)$. An arc $a \in V \times V$ is an ordered pair of elements of V. If $a = (v, w) \in A$ then a is said to be *incident from* v and *incident to* w or v is the *source* and w is the *target*. Two nodes $u, v \in V$ are called *adjacent* in a digraph $D = (V, A)$ if $(u, v) \in A$ or $(v, u) \in A$. The definitions of a *node-induced, arc-induced subgraph* follow analogously from the ones of graphs.

We call a graph $G = (V, E)$ a *weighted graph* if we assign *weights* to each edge $e = (v, w) \in E$.

Definition 2.1.3. *A set M of independent edges in a graph G is called a* matching. *M is a matching of $U \subseteq V$ if every vertex in U is incident with an edge in M. The vertices in U are then called* matched *(by M),* unmatched *otherwise.*

We distinguish between two types of matchings: (a) a *maximum cardinality matching* maximizes the cardinality of M. If $|M| = \frac{|V|}{2}$, that is every node is incident to an edge in a matching M, the matching is called *perfect*. Figure 2.2 shows such a perfect matching.

(b) a *matching of maximal weight* tries to maximize the sum of the edge weights in M. The *maximum cardinality matching* can be seen as a special case of the *maximum weight matching* where all edges have the same weight.

A matching in a bipartite graph $G = (V, E)$ with the node classes $S_1 = (a_1, \ldots, a_n)$ and $S_2 = (b_1, \ldots, b_m)$ is a special case of matchings in general graphs. If an order $\prec$ is defined on the nodes of each class such that

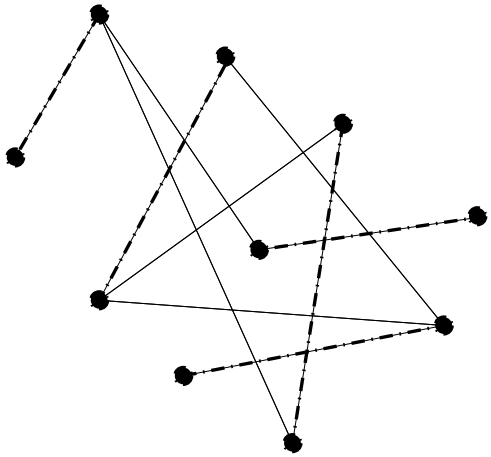

Figure 2.2: Perfect matching in a general graph

$x_a \prec x_b \Leftrightarrow a < b$, a *non-crossing matching* is defined as a matching where two edges – graphically spoken – do not cross, that is $\forall e_1 = (a_v, b_w), e_2 = (a_x, b_y) \in E(G)$ either $v < x \Rightarrow w < y$ or $x < v \Rightarrow y < w$. Figure 2.3 shows such a non-crossing matching in a bipartite graph.

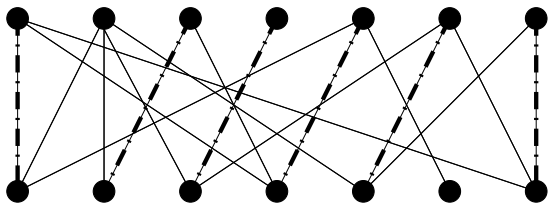

Figure 2.3: Non-crossing matching in a bipartite graph

In a weighted bipartite graph the *non-crossing matching of maximum weight* is the matching M whose edges do not cross and whose sum of edge weights in M is maximal. Note that the problem of computing such a matching can be solved by means of labeling algorithms (see Malucelli, Ottmann, and Pretolani, 1993) in $\mathcal{O}(n\log n)$ or by using the approach due to Needleman and Wunsch, as described in Section 1.2.1 (in terms of computational biology, a *maximum weight non-crossing matching* denotes a traditional sequence alignment).

2.2 Linear Algebra

In this section some mathematical tools that will be used throughout the rest of the thesis will be introduced.

The set of real (rational, integer, natural) numbers is denoted by $\mathbb{R}$ ($\mathbb{Q}$, $\mathbb{N}$, $\mathbb{Z}$). For an ordered set $E = \{e_1, \ldots, e_n\}$ and a field $\Re$ we denote by $\Re^E$ the set of vectors in which the components of each vector are indexed by the members of E. In particular, $\mathbb{R}^E$ denotes the $|E|$-dimensional vector space over the field $\mathbb{R}$.

Definition 2.2.1. *For any field $\Re$ a vector $x \in \Re^E$ is called a* linear combination *of $x_1, \ldots, x_n$, if there exist $a_i \in \mathbb{R}$ such that $x = \sum_{i=1}^{n} a_i x_i$. If additionally $\sum_{i=1}^{n} a_i = 1$ and $a_i \geq 0, i = 1, \ldots, n$, x is a convex combination of $x_1, \ldots, x_n$. If $S \subseteq \Re^E$, $CH(S)$ denotes the* convex hull *of S, that is the set of all possible convex combinations of elements in S.*

A special class of functions that are of great importance in linear optimization are described in the following definition:

Definition 2.2.2. *A function $f : \Re^n \to \Re$ is called* convex *if for every $x, y \in \Re^n$, and every $\lambda \in [0, 1]$, we have*

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda) f(y)$$

A function $f : \Re^n \to \Re$ is called concave *if for every $x, y \in \Re^n$, and every $\lambda \in [0, 1]$, we have*

$$f(\lambda x + (1 - \lambda)y) \geq \lambda f(x) + (1 - \lambda) f(y)$$

2.3 Linear Programming

Cook, Cunningham, Pulleyblank, and Schrijver (1998) define the *Linear Programming Problem* as finding a vector x that maximizes (or minimizes) the *objective function* $c^T x$, where x ranges over all vectors satisfying a set of given linear inequalities $Ax \leq b$. A vector x, such that $Ax \leq b$ holds, is called a *feasible* solution.

More formally, a *linear programming problem* or *LP*-problem in short, consists of a matrix $A \in \Re^{m \times n}$, a vector $b \in \Re^m$ and a vector $c^T \in \Re^n$ with

$$\begin{aligned} &\min c^T x \\ &Ax \leq b \end{aligned}$$

or

$$\min\{c^T x | Ax \leq b\}$$

A feasible solution x is called an optimal solution if $c^T x \geq c^T \hat{x}$ for all feasible solutions $\hat{x}$. If a LP has no feasible solutions, it is said to be *infeasible*.

Definition 2.3.1. *Given a LP*

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

the linear program

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

is called the dual problem.

Problem (P) is called the *primal problem*. Note, that a fundamental result of *duality theory* states that the optimal value of the primal problem and the dual problem are equal. Duality theory is of great importance in linear programming. It allows conclusions between different linear programs that basically describe the same problem to be drawn.

A class of linear programs are the *Integer Linear Programs*, or *ILP* in short. An *ILP* is defined as

$$\begin{aligned} \min\ & c^T x \\ & Ax \leq b \\ & x \text{ integral} \end{aligned}$$

It has been shown that in general solving ILPs is *NP-complete*. However, there exist efficient algorithms to tackle ILPs, *e.g.*,

- *Branch-and-Cut*-algorithms (see Section 3.1).
- *Lagrangian Relaxation* will be described in detail in Section 2.4.

2.4 Lagrangian Relaxation

2.4.1 General Description

Informally spoken, *Lagrangian relaxation* is well suited for problems where the constraints can be divided into two sets: the problem, constrained only

by a set of "good" constraints, is solvable very easily, whereas adding the set of "bad" constraints makes it very hard to solve. The main idea is to relax the problem by the "bad" constraints – called the *Lagrangian relaxed problem* – and put them into the objective function, assigned with weights (the *Lagrangian multiplier*). Each weight represents a penalty which is added to a solution that does not satisfy the particular constraint.

The *Lagrangian relaxed problem* yields a bound on the original instance. The core question is to find optimal *Lagrangian multipliers* that give the best bound to the original problem.

In the following, *Lagrangian relaxation* will be described formally. The description follows the one from Bertsimas and Tsitsiklis (1997). For an easy introduction to *Lagrangian relaxation* see Fisher (1981) and especially Fisher (1985), for a thorough mathematical treatment of the subject see Lemaréchal (2001).

Assume the following *integer linear problem*:

$$\min c^T x \tag{2.1}$$
$$Ax \geq b \tag{2.2}$$
$$Dx \geq d \tag{2.3}$$
$$x \text{ integer} \tag{2.4}$$

with A, D, b, c, d having integer entries. Let Z_{IP} be the optimal value to the ILP above and let

$$X = \{x \text{ integral } \mid Dx \geq d\}$$

We assume that optimizing over the set X can be done very easily, whereas adding the "bad" constraints $Ax \geq b$ makes the problem infeasible to solve. Therefore, we introduce a dual variable for every constraint of $Ax \geq b$. The vector $\lambda \geq 0$ is the vector of dual variables (the *Lagrangian multipliers*) that has the same dimension as vector b. For a fixed λ, the problem

$$\begin{aligned} \min\ & c^T x + \lambda^T (b - Ax) \\ & Dx \geq d \end{aligned}$$

is introduced and its optimal value is denoted by $Z(\lambda)$. By assumption, the optimal value for the relaxed problem with a fixed vector $\lambda \geq 0$ can be efficiently computed and it is easy to see that $Z(\lambda)$ provides a lower bound on Z_{IP}.

Lemma 2.4.1. *If 2.1 has an optimal solution and if $\lambda \geq 0$, then $Z(\lambda) \leq Z_{IP}$.*

Of particular interest is the tightest of all bounds, that is

$$\begin{aligned} &\max Z(\lambda) \\ &\lambda \geq 0 \end{aligned}$$

The problem above is called the *Lagrangian dual*. Let

$$Z_D = \max_{\lambda \geq 0} Z(\lambda) \tag{2.5}$$

If X is a finite set, say $X = \{x^1, \ldots, x^m\}$, then $Z(\lambda)$ can also be written as

$$Z(\lambda) = \min_{i=1,\ldots,m} (c^T x^i + \lambda^T (b - Ax^i))$$

The function $Z(\lambda)$ is the minimum of a finite set of linear functions of λ and therefore it is concave and piecewise linear, as the definition of of concave functions implies (see Definition 2.2.2). Furthermore, the problem of computing Z_D can be written as a linear program (with a very large number of constraints, though).

It is important to note, however, that – unlike in linear programming – integer linear programming does not have strong duality theory. This implies that the optimal value of the Lagrangian dual does not have to be the same as the optimal value of the original (primal) problem. Instead of

$$Z_D = Z_{IP}$$

the following holds:

$$Z_D \leq Z_{IP}$$

An example for the concepts described so far is given below.

Example 2.4.1. ***General assignment problem****. The following ILP is given:*

$$Z_{IP} = min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{2.6}$$

$$\sum_{i=1}^{m} x_{ij} = 1, \qquad j = 1, \ldots, n \tag{2.7}$$

$$\sum_{j=1}^{n} a_{ij} x_{ij} \leq b_i, \qquad i = 1, \ldots, m \tag{2.8}$$

$$x_{ij} \in \{0, 1\}, \forall i, j \tag{2.9}$$

The ILP is relaxed by (2.7) *and these constraints are taken into the objective function, assigned with* Lagrangian multiplier. *This yields*

$$Z(\lambda) = min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} + \sum_{j=1}^{n} \lambda (\sum_{i=1}^{m} x_{ij} - 1) \tag{2.10}$$

$$\sum_{j=1}^{n} a_{ij} x_{ij} \leq b_i, \qquad i = 1, \ldots, m \tag{2.11}$$

$$x_{ij} \in \{0, 1\}, \forall i, j \tag{2.12}$$

The problem is reduced to m 0-1-knapsack problems that can be solved by means of dynamic programming in (pseudo-)polynomial time.

2.4.2 Solving the Lagrangian Dual

In Section 2.4.1 we defined *Lagrangian multipliers* and *Lagrangian dual*. As outlined in Lemma 2.4.1, the optimal value of the *Lagrangian dual* is always smaller or equal to the optimal value of the original problem.

The core question is: how should the $\lambda \geq 0$ be set, such that the gap between $Z(\lambda)$ and Z_{IP} is as small as possible (or 0 in the best case)?

For sake of simplicity, we assume that X is finite and can be written as $X = \{x^1, \ldots, x^m\}$. Then – as described above – $Z(\lambda)$ can be written as

$$Z(\lambda) = \min_{i=1,\ldots,m} (c^T x^i + \lambda^T (b - A x^i))$$

With $f_i = b - Ax^i$ and $h_i = c^T x^i$ this can be rewritten as

$$Z(\lambda) = \min_{i=1,\ldots,m} (h_i + \lambda f_i^T)$$

a piecewise linear and concave function.

For the moment, assume that $Z(\lambda)$ was also differentiable. Then, the classical approach of maximizing the function would be the *steepest ascent method*, that is computing a sequence of iterations with

$$\lambda^{t+1} = \lambda^t + \gamma_t \nabla Z(\lambda^t)$$

We are following the gradient at the current position – with a specified stepsize γ – to find points with a higher function value.

Unfortunately, this procedure is no longer valid for our function, since it is piecewise linear and concave, but as one can easily see, it is not differentiable everywhere.

Therefore, we extend the notion of a *gradient* in concave functions that are not differentiable everywhere.

Lemma 2.4.2. *A function $f : \Re^n \to \Re$ is concave iff for any $x^* \in \Re^n$, there exists a vector $s \in \Re^n$ such that*

$$f(x) \leq f(x^*) + s^T(x - x^*)$$

holds $\forall x \in \Re^n$.

In the two-dimensional case a subgradient is a tangent in a differential point, as Figure 2.4 illustrates. In a non-differential point there exists more than one subgradients, as one can easily see.

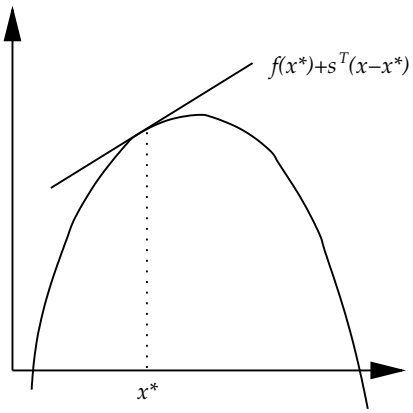

Figure 2.4: Concave function with a subgradient

Using the definition of concave functions, the notion of a *gradient* can be extended as follows:

Definition 2.4.1. *Let f be a concave function. A vector s such that*

$$f(x) \leq f(x^* + s(x - x^*))$$

for all $x \in \Re^n$, is called a subgradient *of function f at point x^*. The set of all* subgradients *of f at x^* is denoted by $\partial f(x^*)$ and is called the* subdifferential *of f at x^*.*

As soon as $0 \in \partial f(x^*)$, we are done, as the following lemma says:

Lemma 2.4.3. *Let $f : \Re^n \to \Re$ be a concave function. A vector x^* maximizes f over $\Re^n$ iff $0 \in \partial f(x^*)$.*

Lemma 2.4.3 follows directly from Definition 2.4.2. If $(s = 0) \in \partial f(x^*)$, it implies – according to Definition 2.4.2 – that

$$f(x) \leq f(x^*)$$

which describes the maximum of function f.

To get the counterpart to the *steepest ascent method* for piecewise linear functions, the notion of the *subdifferential* at a point x^* has to be extended.

Lemma 2.4.4. *Let*

$$Z(\lambda) = min_{i=1,\ldots,m}(h_i + \lambda f_i^T)$$

and

$$E(\lambda) = \{i \mid Z(\lambda) = h_i + \lambda f_i^T\}$$

Remember that $h_i = c^T x^i$ and $f_i = b - Ax^i$. Then, the following is true:

- *For every $i \in E(\lambda^*)$, f_i is a subgradient of the function Z at point λ^**
- *$\partial Z(\lambda^*) = CH(\{f_i, i \in E(\lambda^*)\})$, that is a vector s is a subgradient of the function Z at λ^* iff $Z(\lambda^*)$ is a convex combination of the vectors $f_i, i \in E(\lambda^*)$.*

Remember that the function under consideration is piecewise linear and concave. Then, $E(\lambda)$ is nothing else than the set of indices of the linear functions that form a minimum at that specific point (it has to be the minimum, otherwise the sequence would not converge). In the two-dimensional case this set contains the indices of functions that cross at that specific point (see Figure 2.5 for an illustration).

The definitions above offer the tools to generalize the *steepest ascent method* to non-differentiable, piecewise linear functions. The algorithm can be written as

Definition 2.4.2. ***Subgradient optimization method.***

1. *Choose a starting point λ^0; $t = 0$.*
2. *Choose a subgradient s^t of the function Z at λ^t. If $s^t = 0 \rightarrow$ **STOP**, because the optimal value has been reached.*
3. *Compute $\lambda^{t+1} = \lambda^t + \gamma_t s^t$, where γ_t denotes the stepsize.*
4. *Increment t and go to 2.*

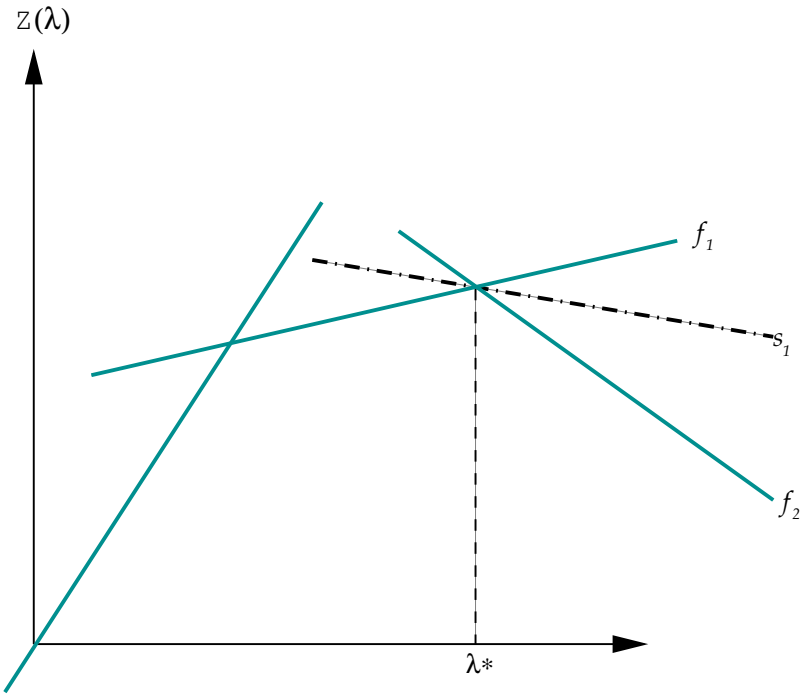

Figure 2.5: Subdifferential of Z(λ) at λ^*

The definition of stepsize γ is of crucial importance, since the speed of convergence depends heavily on the stepsize. Poljak (1967) proves that $Z(\lambda)$ converges to the optimal value of Z – assuming it is finite – for any stepsize γ_t such that

$$\sum_{t=1}^{\infty} \gamma_t = \infty$$

and

$$\lim_{t \to \infty} \gamma_t = 0$$

Such a sequence would be, *e.g.*, $\gamma_t = \frac{1}{t}$. A more sophisticated formula is the following:

$$\gamma_t = \mu \frac{\hat{Z}_D - Z(\lambda^t)}{\| s^t \|^2} \tag{2.13}$$

(2.13) goes back to the 1970's, *e.g.*, when Held and Karp applied the concept of *Lagrangian relaxation* to the problem of the *Travelling salesman* for the very first time (see Held, Wolfe, and Crowder, 1974; Held and Karp, 1971).

Normally, the parameter μ is initialized with 1 and halved, if the value of the *Lagrangian dual* does not increase within a fixed number of iterations.

Note, that *subgradient optimization* is easy to develop and implement. But there are other methods, like *cutting plane*-approaches or *bundle*-methods, to compute a series of *Lagrangian multipliers* that promise even faster speed of convergence. For details, the reader is referred to Lemaréchal (2001).

2.4.3 Solving Hard Problems

Although *Lagrangian relaxation* performs quite well in *NP-complete* real-world applications, like the *General Assignment Problem*, *Travelling Salesman Problem* or *Scheduling Problems*, it is not guaranteed that the sequence of iterations converges to the optimal value.

Therefore, *Lagrangian relaxation* can be used in a *Branch-and-Bound*-framework. Just like a *linear programming relaxation* on *integer linear programs*, *Lagrangian relaxation* provides upper and lower bounds on the problem to solve and it helps to decide, which branches are most likely to lead to the optimal solution and should be explored.

Well it's a strange old game
You learn it slow

DIRE STRAITS

Chapter 3

Previous Work

This chapter describes some of the results achieved so far on the field structural alignments. After a description of a *Branch-and-Cut* algorithm for *RNA structural alignment*, newer results in structural alignments of proteins are presented. The chapter is concluded with some references to other papers that deal with structural alignments.

3.1 Structural Alignment via Branch-and-Cut

Lenhof, Reinert, and Vingron (1998) and subsequently Reinert (1999) introduce methods of polyhedral combinatorics to problems of computational biology. The main idea is to phrase the problem of computing a structural alignment of maximum score as an ILP and solve it afterwards by means of *Branch-and-Cut*.

3.1.1 General Branch-and-Cut Framework

This section describes the *Branch-and-Cut* paradigm only very briefly. For an extensive treatment of the subject, see, *e.g.*, Jünger, Reinelt, and Thienel (1995).

Branch-and-Cut algorithms are a well-studied tool for solving integer linear programs that are known to be NP-complete. Basically, *Branch-and-Cut* combines two approaches: (a) traditional *Branch-and-Bound* and (b) *cutting plane techniques*.

The general framework works the following way:

We are given an integer linear program (see Section 2.3) that cannot be solved exactly in polynomial time because of its NP-completeness. Therefore, we drop the constraint that $x \in \{0, 1\}$ and add the inequalities $x \geq 0$ and $x \leq 1$ instead. Unfortunately, the number of inequalities can be exponential which makes a polynomial computation of the linear program infeasible. Now, *Branch-and-Cut* comes into play:

We start with a small (say polynomial) number of inequalities and solve the associated linear program. If the result satisfies *all* inequalities (not only the ones that were considered in solving the linear program), the solution is also a solution for the original problem. Otherwise, we add the violated inequalities to the linear program and solve it again (that is we "cut" off the invalid solution).

If no violated constraints can be found, we have to consider two cases:

1. The solution vector x is integral.

   Then we have an optimal solution to the original problem, because none of the ILP constraints is violated.

2. The solution vector x is fractional.

   In this case, we branch the problem, that is we introduce new subproblems by setting a fractional variable to 0 or 1.

So, the main task in *Branch-and-Cut* algorithms lies in finding violated inequalities. These algorithms, called *separation algorithms*, are highly problem-specific and, depending on the hardness of the separation problem, exact or heuristic methods are used.

3.1.2 ILP Formulation

Recall the definition of *secondary structure* in Section 1.1.2 and the graph-theoretic description of a structural alignment at the end of Section 1.2.2.

A structural alignment of two annotated sequences (S_1, P_1) and (S_2, P_2) is a set of non-crossing alignment edges and the corresponding interaction edges. Reinert (1999) gives another definition of a valid alignment.

Two annotated sequences (S_1, P_1) and (S_2, P_2) with a set A of alignment edges between the sequences and a set I of interaction edges between nodes of the same sequence are given. We then add a set H of directed *horizontal* arcs between the nodes of the sequence, that is

$$H = \{(s_{i,j}, s_{i,j+1}) | i \in \{1, 2\}, 1 \leq j \leq n_i\}$$

and call the resulting graph the *structural enhanced alignment graph*, or *SEAG* in short, of the two sequences (see Figure 3.1 for an example: solid lines describe alignment edges, dashed lines are interaction edges and arrows represent arcs between nodes).

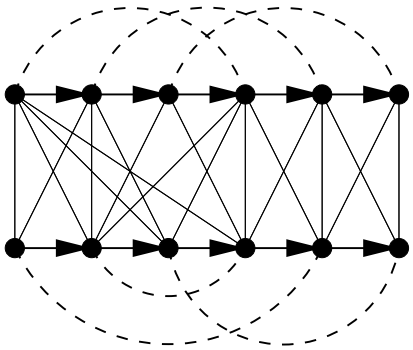

Figure 3.1: *SEAG* of two sequences

The set M denotes the set (i_p, i_q, e_l, e_k) with $i_{p,q} \in I, e_{l,k} \in A$ representing all possible interaction matches in a secondary structure. Recall that an interaction match needs two interaction edges whose endnodes are realized by two alignment edges (see Figure 3.2 for an example with two realized interaction matches).

For the ILP formulation another definition is needed:

Definition 3.1.1. *A mixed cycle R in a* SEAG *is called* critical *if for $i \in \{1, 2\}$ all vertices in $R \cap S_i$ occur consecutively in R.*

Informally stated, that means that a cycle enters and leaves a sequence exactly once. Based on this definition, we define a structural alignment as

Definition 3.1.2. *Let Σ be a finite alphabet, (S_1, P_1) and (S_2, P_2) be two annotated sequences over Σ. Let $G = (V, E, A, I)$ be the* SEAG *of the two sequences. A* structural alignment *is a pair (T, B) with $T \subseteq E, B \subseteq I$ with the property that the subgraph induced by $T \cup H$ does not contain a critical cycle and no two interaction edges are in conflict.*

Figure 3.2 shows a valid structural alignment with two realized interaction matches.

Using the definitions above, Reinert (1999) gives the following ILP definition for the structural alignment problem:

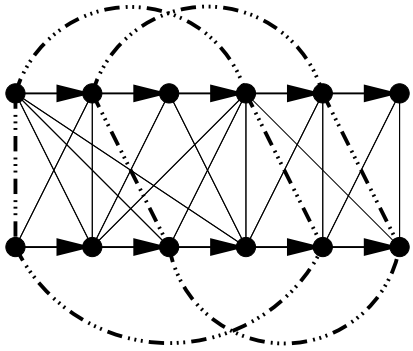

Figure 3.2: Structural alignment based on *SEAG*

$$
\begin{aligned}
\max & \sum_{e_i \in A} w_i x_i + \sum_{m_{i,j} \in M} w_{i,j} x_{i,j} & \\
& \sum_{e \in C} x_e \leq |C \cap E| - 1 & \forall \text{critical mixed cycles C} \\
& \sum_{j} x_{i,j} \leq x_i & \forall x_i, x_{i,j} \\
& \sum_{i} x_{i,j} \leq x_j & \forall x_j, x_{i,j} \\
& x_i, x_{i,j} \in \{0, 1\} & \forall x_i, x_{i,j}
\end{aligned}
$$

Variable x_i is set to 1, if alignment edge i is realized. $x_{i,j}$ is set to 1, if interaction match $m_{i,j}$ is realized.

As one can easily check, all requirements for a structural alignment are met: the first constraints guarantee that the structural alignment contains only valid alignment edges (putting it in different words: there are no *critical mixed cycles* in the solution and therefore the alignment is valid). Secondly, the constraints $\sum_j x_{i,j} \leq x_i$ and $\sum_i x_{i,j} \leq x_j, \forall x_{i,j}$ serve two purposes:

1. An interaction is realized if and only if the two corresponding alignment edges are realized.

2. Every alignment is used by at most one interaction edge (that is there is no pair of realized interactions that share a start- or endnode).

Solving the ILP above by means of *Branch-and-Cut*, the author was able to optimally align two annotated sequences that had a length of about 1400-1500 nucleotides. For details, especially about the separation techniques for the mixed cycle inequalities that were used within in the Branch-and-Cut algorithm, the reader is referred to Reinert (1999).

3.2 Protein Alignment via Lagrangian Relaxation

A problem, similar to the one of computing a structural alignment between two annotated sequences, is the one of aligning two proteins. There are several important applications to the comparison of different proteins which are, again, similar to the ones of RNA:

- **Function determination** Just like with RNA molecules, the structure of a protein determines the function and the interaction with other proteins. Therefore, the function of an unknown protein can often be derived by comparing its structure to the structure of known proteins.
- **Structure prediction** If the structure of a protein was experimentally evaluated, for example by means of *X-ray crystallography*, it is considered to be the correct one. If a new protein is found, its structure can be predicted and afterwards be compared to known structures.
- **Protein clustering** The alignment of protein structures allows groups of proteins to be formed, based on their structure similarity.

One way to compare different proteins is to compare their *contact maps*. The notion of a *contact map* is based on the local vicinity between residues of a protein.

Formally stated, a *contact map* of a folded protein of n residues is a $n \times n$ matrix C with entries of 0 or 1, depending on the distance between the two residues: $C_{ij} = 1$ if and only if the distance of two heavy atoms – the first one from the i-th and the second one from the j-th residue – is within a given threshold, for example 3Å or 5Å. Hence, the basic idea is that in case of similar contact maps the corresponding structures will be similar as well.

Given a matrix C, a protein can be seen as a graph G whose nodes correspond to the residues. An edge between the i-th and the j-th residue is

inserted, if $C_{ij} = 1$. The *Contact Map Overlap* problem, or *CMO* in short, calls for an alignment of residues in the first protein with residues in the second protein. The *overlap* of the two proteins is the number of contacts in the first protein whose endpoints are aligned with residues that are also in contact in the second protein.

Then, the problem of aligning two proteins is reduced to the problem of finding the maximum overlap between two proteins. Figure 3.3 shows an alignment of value 5.

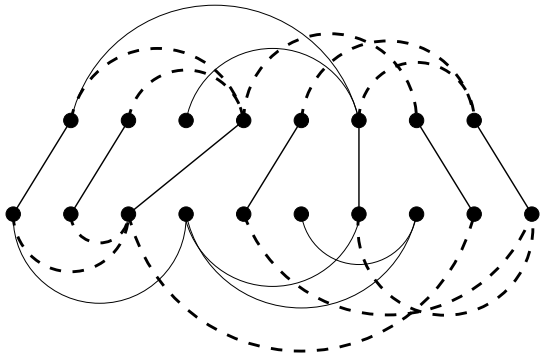

Figure 3.3: Contact map overlap of value 5

At *RECOMB'01* Lancia, Carr, Walenz, and Istrail (2001) presented an algorithm based on *Branch-and-Cut*. They were able to align proteins with proven optimality for the very first time. A year later, at *RECOMB'02* (see Caprara and Lancia, 2002), the approach was modified and the maximum overlap was computed by means of *Lagrangian relaxation*. New results by the same authors (see Caprara, Lancia, Carr, Walenz, and Istrail, 2004) combine both approaches yielding the best algorithm for aligning protein structures known so far.

The following description of computing the *maximum overlap* of two folded proteins follows the one from Caprara and Lancia (2002), since the *RECOMB'02* algorithm forms the basis for the method presented later in this thesis.

3.2.1 Mathematical Formulation

We are given two undirected graphs G_1 and G_2, with $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Vertices denote residues of a protein. These vertices are connected by an edge (a *interaction edge*), if and only if they are close enough.

Let $L = V_1 \times V_2$ (in this context, a set of non-crossing edges of L denotes a traditional sequence alignment $\mathcal{A}$), and let $\mathcal{I}$ be the collection of all (maximal) sets of pairwise incompatible lines I.

Two lines $l = (i_1, i_2)$ and $m = (j_1, j_2)$ are said to be incompatible, if either $i_1 > j_1$ and $i_2 < j_2$, or $i_1 < j_1$ and $i_2 > j_2$ (in other words, incompatible lines cross or touch each other). For each $l = (i, j) \in L$ a binary variable x_l is introduced with $x_l = 1$, if residue i of the first protein is mapped to residue j in the second one. For each $l, m \in L, l < m$ with $l = (i_1, i_2)$ and $m = (j_1, j_2)$ the variable $a_{lm} = 1$, if $(i_1, j_1) \in E_1$ and $(i_2, j_2) \in E_2$ (note that there is an order $<$ defined on all lines to avoid double variable definitions and constraints, respectively). In terms of *RNA structural alignments*, a_{lm} is set to 1, if two alignment edges realize the corresponding interaction edges.

Now, the main idea of Caprara and Lancia (2002) is not to maximize the single a_{lm} directly, but rather split the weight a_{lm} into two separate profits b_{lm} and b_{ml}. Afterwards the profits of $b_{lm}, l, m \in L$ are maximized. Although this may sound odd at the first time, but the separation into two *independent* profits is the core idea behind the whole approach. However, one has to ensure that $b_{lm} + b_{ml} = a_{lm}$ for all $l, m \in L$. As an initial (arbitrary) choice a_{lm} is divided by 2, that is $b_{lm} = b_{ml} = \frac{a_{lm}}{2}$.

The problem of computing the *maximum overlap* between two proteins can then be written as

$$\max \sum_{l \in L} \sum_{m \in L} b_{lm} x_l x_m$$

with

$$\sum_{l \in I} x_l \leq 1, \forall I \in \mathcal{I}$$
$$x \geq 0, \text{integer}$$

The integer program above ensures all conditions that a solution of the *CMO* problem has to meet: $\sum_{l \in I} x_l$ guarantees that the solution does not contain crossing lines, because in this case the sum of x_l would be greater than 1. The objective function itself aims for the maximum profit that can be achieved by realizing lines between the first and second protein.

After linearizing the integer program above (Caprara and Lancia, 2002, Section 2.1), the authors get the following ILP:

$$\max \sum_{l \in L} \sum_{m \in L} b_{lm} y_{lm} \tag{3.1}$$

that is constrained by

$$\sum_{l \in I} x_l \leq 1 \qquad \forall I \in \mathcal{I} \tag{3.2}$$

$$\sum_{l \in I} y_{lm} \leq x_m \qquad \forall I \in \mathcal{I}, m \in L \tag{3.3}$$

$$y_{lm} = y_{ml} \qquad \forall l, m \in L, l < m \tag{3.4}$$

$$x, y \geq 0 \qquad \text{integer} \tag{3.5}$$

Variable y_{ml} replaces the product $x_m x_l$ and $l < m$ in equation (3.4) is introduced as an (arbitrary) order on all lines to avoid unnecessary constraints (otherwise $y_{ml} = y_{lm}$ *and* $y_{lm} = y_{ml}$ would be part of the ILP).

The constraints (3.2) guarantee, like in the first formulation, that

- Only valid alignments, that is alignments where no lines are crossing (or touching), form a solution. Inequalities (3.3) take care that the corresponding lines are realized, if an interaction edge is realized.
- Interaction edges, say $l = (e_1, e_2)$ and $m = (f_1, f_2)$, are realized if and only if $(e_1, f_1) \in E_1$ and $(e_2, f_2) \in E_2$

Note, however, that in the *CMO* problem, we do not have to deal with restricted degrees of the nodes. As long as the alignment edges are realized, an arbitrary number of interaction edges adjacent to a single node can be realized.

3.2.2 Relaxing the ILP

It is not possible to solve the ILP described above in polynomial time, therefore we relax the problem.

If the ILP is relaxed by omitting constraints (3.4), interaction edges might be realized "half way", that is y_{ml} might be 1, whereas $y_{lm} = 0$. Nevertheless, the relaxed ILP is solvable in $\mathcal{O}(|G_1(E)||G_2(E)|)$. The next paragraphs will show how this can be accomplished.

After the removal of $y_{ml} = y_{lm}$, each variable y_{lm} appears only in the constraints (3.3) associated with x_m. For each $m \in L$, this implies that, if variable x_m takes the value 0 all variables y_{lm} take the same value, whereas, if variable x_m takes the value 1, the optimal choice of y_{lm} with $l \in L$ is given

by the solution of

$$\max \sum_{l \in L} b_{lm} y_{lm} \tag{3.6}$$

$$\sum_{l \in I} y_{lm} \leq 1, \forall I \in \mathcal{I}, m \notin I \tag{3.7}$$

$$\sum_{l \in I} y_{lm} \leq 0, \forall I \in \mathcal{I}, m \in I \tag{3.8}$$

$$y \geq 0, \text{integer} \tag{3.9}$$

This simply means the following:

For every line $m \in L$ we compute a maximum profit value p_m. The higher the value, the higher the profit that might be achieved, if line m is part of the solution. The distinction between $m \in I$ and $m \notin I$ guarantees that the profit is computed correctly in the sense that two lines must not cross (therefore $\sum_{l \in I} y_{lm} \leq 0$ with $m \in I$ has to be fulfilled: as soon as $\sum_{l \in I} y_{lm}$ was greater than 0, two lines of the **same** set I would be part of the solution, which simply means that two lines would be crossing).

After calculating the profits for all lines the original problem is solved by solving the alignment

$$\max \sum_{m \in L} p_m x_m \tag{3.10}$$

$$\sum_{l \in I} x_m \leq 1, \forall I \in \mathcal{I} \tag{3.11}$$

$$x \geq 0, \text{integer} \tag{3.12}$$

The above formulation means the following:

For every single set I of pairwise incompatible lines it is desirable to take the one with the maximum possible profit. Inequalities (3.11) guarantee that the calculated solution is valid, since only one line of every set I is part of the solution.

Then, the overall solution is given by the following computation:

1. Compute the profit p_m for all $m \in L$, consisting of a set of $\hat{y}_{lm} = 1, l \in L$.

2. Compute a solution $\bar{x}$ to the alignment using the values of $p_m, \forall m \in L$,

3. The solutions of step 1 and step 2 are brought together by calculating:

$$\bar{y}_{lm} = \hat{y}_{lm}\bar{x}_m \tag{3.13}$$

This simply means that the maximum profit lines (computed in step 1) of a line $m \in L$ are realized in the overall solution, if the line m is part of the solution of the non-crossing matching calculated in step 2.

Taking a look at the computational complexity of the calculation above, we can easily check that $\mathcal{O}(|N(E_1)||N(E_2)|)$ holds: In the first step ((3.6)-(3.9), for every line $l = (i, j) \in L$ the maximum profit is computed. This can be done by means of dynamic programming: for every line $l = (i, j) \in L$ we are looking at all interaction edges that are either incident to i in $G_1(E)$ or to j in $G_2(E)$ (like in the original paper, we call them $N_1(i)$ and $N_2(j)$). Now, we have to compute an alignment among the "left" and "right" neighbors of line l such that no two lines cross. This can be done by dynamic programming (for example by the algorithm of Needleman and Wunsch) in $\mathcal{O}(|N_1(i)||N_2(j)|)$. Figure 3.4 gives an example of $N(i)$ and $N(j)$ illustrating the fact that in the first step of the algorithm, an alignment is being computed.

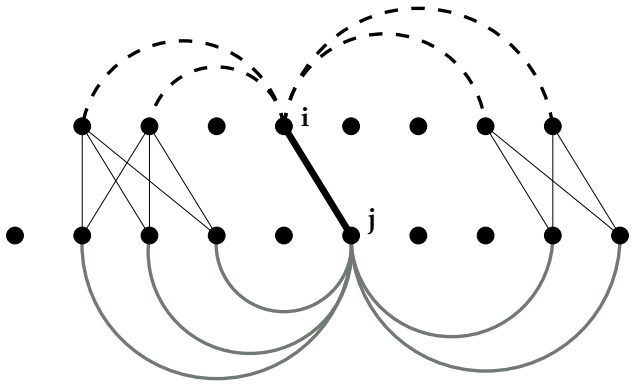

Figure 3.4: First step in solving the relaxed problem

The endnodes of the dashed edges represent $N(i)$, the endnodes of the grey edges $N(j)$. Among the lines that connect those endnodes one has to find the *non-crossing matching of maximum cardinality*, since every edge has the same weight.

Solving the second step, that is problem (3.10)-(3.12), takes $\mathcal{O}(|L|)$, as one can easily verify. Every line $l \in L$ has a certain profit p_l and among all these

profits we have to compute an alignment with the single p_l as weights. Again, this can be done by dynamic programming using the algorithm of Needleman and Wunsch.

Taking these considerations into account, the overall complexity is given by

$$\mathcal{O}(|L| + \sum_{i \in G_1(V)} \sum_{j \in G_2(V)} |N_1(i)||N_2(j)|) = \mathcal{O}(|G_1(E)||G_2(E)|) \ .$$

One will have noticed that the algorithm described so far computes the solution to a relaxed ILP, but it is no *Lagrangian relaxation* as described in Section 2.4. The approach written above can, however, easily be adapted:

Instead of just relaxing the ILP by the constraints (3.4), we move the constraints to the objective function assigned with *Lagrangian multipliers*. Then, (3.6) has to be replaced by

$$\max \sum_{l \in L} \sum_{m \in L} b_{lm} y_{lm} + \sum_{l \in L} \sum_{m \in L : l < m} \lambda_{lm}(y_{lm} - y_{ml})$$

Defining $\lambda_{ml} = -\lambda_{lm}$ for $l < m$ and $\lambda_{mm} = 0$ for all $m \in L$, (3.6) can be written as

$$\max \sum_{l \in L} \sum_{m \in L} (b_{lm} + \lambda_{lm}) y_{lm}$$

What remains is the computation of the *Lagrangian multipliers* λ_{lm}. As described in Section 2.4, we need to compute a series of such multipliers to find the global maximum of the objective function. Adopting the general framework given in Section 2.4, the vector of subgradients is given by

$$s_{lm} = \bar{y}_{lm} - \bar{y}_{ml}, \forall l, m \in L, l < m$$

By using these subgradients, the series of λ_{lm}^k with $k = 0, \ldots, n$ is computed by

$$\lambda_{lm}^{k+1} = \begin{cases} \lambda_{lm}^k & \text{if } s_{lm} = 0 \\ \max(\lambda_{lm}^k - \gamma, -b_{ml}) & \text{if } s_{lm} = 1 \\ \min(\lambda_{lm}^k + \gamma, b_{ml}) & \text{if } s_{lm} = -1 \end{cases} \tag{3.14}$$

with

$$\gamma = \mu \frac{\mathrm{UB} - \mathrm{LB}}{\sum_{m,l} s_{ml}^2}$$

and

$$\lambda_{lm}^0 = 0$$

for all $m, l \in L$ as described in Section 2.4.

Parameter μ has to be initialized with a fixed value. Variable *UB* denotes the best upper bound on the problem (the value of the solution of (3.10)-(3.12)), whereas *LB* is a feasible solution that was found by an appropriate heuristic (in Caprara and Lancia (2002) the authors simply take the solution to (3.10)-(3.12) and compute the corresponding overlap).

Two interesting facts should be written down explicitly:

1. The equation $\lambda_{ml} + \lambda_{lm} = 1$ holds for all lines m and l. Hence, one can easily see the purpose of the *Lagrangian multipliers*: $b_{ml} = b_{lm}$ in the beginning, but as soon as the multiplier are computed, the profit between b_{ml} and b_{lm} is adjusted according to the single s_{lm}:

   If $s_{lm} = 0$ holds, it basically says that the chosen edges match or that they do not match at all. On the other hand, $s_{lm} = 1$ and $s_{lm} = -1$, respectively, expresses that from one side (either from the left or the right) the edge forming the optimal profit was realized, from the other side, however, other lines were chosen to form the maximum profit. Now, the goal of using the multiplier consists of adapting the profits in a way that – in the optimal case – the same lines forming the optimal value are chosen (seen from the left *and* the right side).

2. As written in Section 2.4, *Lagrangian relaxation* itself is a heuristic procedure and does not guarantee that the optimal value was found. In fact, the algorithm described above works well for protein structures that are pretty similar, whereas aligning different protein structures is completely out of reach, as the authors write in Caprara and Lancia (2002). The main problem with different structures lies in the selection of lines that form the maximum profit value:

   If the two proteins do not provide a common structure, different lines forming the maximal profit will be chosen for each line. Therefore, almost all s_{ml} will be 0, not because of the fact that the same profit lines were chosen, but because of the fact that completely different lines forming the profit were chosen, as one of the authors confirmed in Lancia (2004) (remember that either $s_{ml} = 1$ or $s_{ml} = -1$ has to hold in order to adapt the Lagrangian multiplier).

3.3 Related Work

Besides the algorithms described in Section 3.1 and Section 3.2 other approaches have been developed for the problem of structurally aligning two structures. What they all have in common is that no pseudoknots are allowed within the structure.

1. A *secondary structure* without pseudoknots can be transformed into a tree structure. The first algorithms for comparing tree structures (and therefore RNA *secondary structures* without pseudoknots) are due to Zhang and Sasha (1989) and have a complexity of $\mathcal{O}(n^2)$. A widely used software package, the *Vienna RNA Package* (see Hofacker, 2004), contains a program that allows the comparison of RNA *secondary structures* based on their tree distance.

2. Bafna, Muthukrishnan, and Ravi (1995) present exact and heuristic algorithms that allow the comparison of RNA sequences. The authors state that "our algorithms for RNA string matching extend to structures that allow crossing edges", but it is written nowhere, how this could be accomplished and if the complexity of the algorithms – $\mathcal{O}(n^2m^2)$ with n and m being the lengths or the two RNA sequences – would change.

3. Newer results due to Hofacker et al. (2004) approach the comparison of RNA sequences by aligning the corresponding *base pair probability matrices*: the entries of the matrices are the base pair probabilities for each sequence.

   Formally stated, we are given two matrices P^A and P^B with entries $\phi_{i,j}^{A|B}$ that denote the base pair probability between residue i and j in sequence A and B, respectively. Then, one basically tries to maximize the score of the realized interaction matches between the interaction edge (i,j) in the first and (k,l) in the second sequence, that is

   $$\max \sum_{(i,j),(k,l)} (\phi_{(i,j)}^{A} + \phi_{(k,l)}^{B}) \ .$$

   Besides the probabilities the algorithm takes traditional sequence information into account as well, that is, it scores (mis-)matches, indels and gaps.

   The approach is a variation of Sankoff's dynamic programming algorithm (see Sankoff, 1985) for simultaneous folding and alignment

of two RNA sequences. The high computational demands – memory and CPU requirements are in $\mathcal{O}(n^4)$ and $\mathcal{O}(n^6)$ – limit the length of the input sequences to approximately 150 nucleotides per sequence though.

Han var interessert i så mangt en ting
Ja alt frå Helvete til Tensing

KAIZERS ORCHESTRA

Chapter 4

RNA Structural Alignment Using Lagrangian Relaxation

Aligning proteins and RNA *secondary structure* turns out to be a remarkably similar task. There are, however, some differences that will be addressed in Section 4.1.

Section 4.2 deals with the changes that have to be applied to the ILP presented in Section 3.2.1 to reflect the differences between proteins and RNA sequences.

Finally, Section 4.3 presents a modified version of the algorithm described in Section 3.2.2. The modified algorithm will be able to compute structural alignments of two annotated RNA sequences.

4.1 Differences Between Proteins and RNA

The problems of aligning protein structures and RNA secondary structures share a set of common properties, *e.g.*,

- Interaction edges represent correlations between residues.
- There are alignment edges between the two structures.
- The goal is the maximization of a certain score.

There are, however, some differences that have to be taken into account, specifically during the adaption of the algorithm presented in Section 3.2.2 for the problem of aligning secondary structures.

A quick glance at Figure 4.1 will make some of the differences apparent: the upper figure shows the alignment of two protein structures, whereas the lower one shows the structural alignment of two RNA sequences.

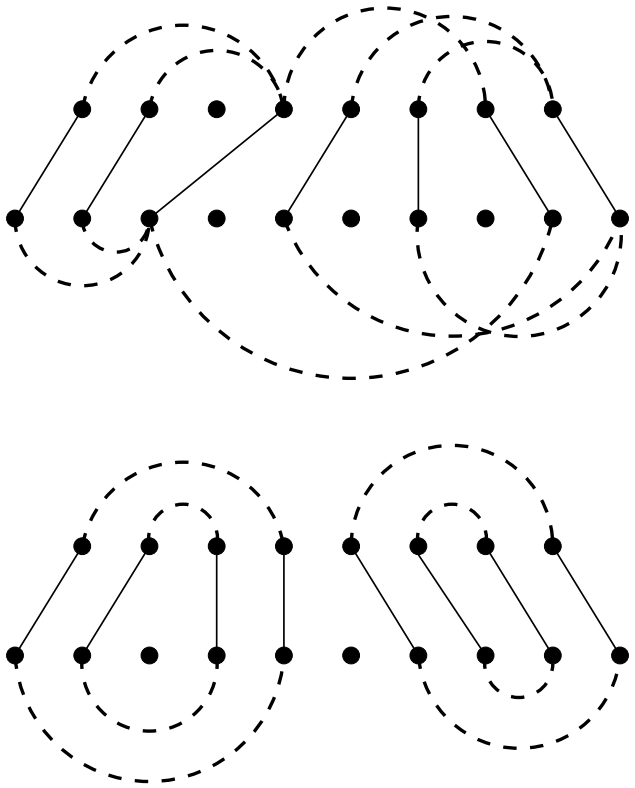

Figure 4.1: Differences between protein alignment and RNA structural alignment

Table 4.1 summarizes the differences between protein structural alignment and RNA structural alignment.

Note that the first and second difference, the weights of alignment and interaction edges, would let the ILP for protein alignment unchanged, a_{ml}, the weight of an interaction edge in the algorithm for the CMO problem, has to be changed from 1 to the weight of the single interaction and the alignment edge, respectively.

The third difference, namely the realization of alignment edges, leads to no changes as well, since the realization of an alignment edge that is not incident to any of the interaction edges is contained in the ILP formulation for aligning proteins. The thing is, however, that these alignment edges are of no use for the overall goal, the maximization of the structural alignment

	Protein structure alignment	**RNA secondary structure alignment**
Weights of interaction edges	Edges between two nodes simply denote that the two residues are within a given threshold.	Interaction edges do contain information, for example the *base pair probability* and the number of the hydrogen bonds between the two residues.
Weights of alignment edges	Edges between the two structures represent a mapping of nodes from the first to nodes of the second protein structure. There is no weight assigned with these edges.	Alignment edges get the *mutation score matrices* value, indexed by the two residues, as their weight. Hence, alignment edges contain additional information.
Realization of alignment edges	Alignment edges are realized, if interaction edges incident to the endnodes of the alignment edge are realized. An alignment edge that is not incident to any interaction edge is irrelevant in aligning protein structures.	Alignment edges contribute additional information (namely the information about the tradition sequence alignment) and there do not have to be any interaction edges that are incident to the endnodes of an realized alignment edge.
Degree of nodes	A node can be linked to several other nodes (an interaction edge denotes spatial vicinity between residues) and therefore no restrictions on the degree of a node exist.	Due to the special structure of RNA a node can be incident to at most one other residue. That is, the degree of nodes is restricted to one in an RNA structural alignment.

Table 4.1: Differences between protein and RNA structural alignments

score: alignment edges do not contribute to the score.

Finally, the fourth difference causes a significant change within the ILP formulation and the way the Lagrangian method works:

We have to take care that the degree of every node is one at most. Hence, the solution to the first step of the algorithm (computing the maximal

"profit" that a line l can possibly achieve) does not select several other lines, but has to select the *best* line with which the maximal profit is achieved (otherwise the degree of the node would be greater than 1).

The restriction of the node degrees, however, makes the computation of the single profits much easier: instead of solving an alignment for every single alignment edge, only the best line has to be taken. By means of *priority queues*, for example, computing the best edge can be done in constant time. Details will be presented in Section 5.1.1.

4.2 ILP Formulation for RSA

Taking a look at the differences between the *Contact Map Overlap* problem and the problem of aligning RNA secondary structures, one immediately recognizes that the original ILP formulation describing the *CMO* problem, that is,

$$\max \sum_{l \in L} \sum_{m \in L} b_{lm} y_{lm} \tag{4.1}$$

constrained by

$$\sum_{l \in I} x_l \leq 1 \qquad \forall I \in \mathcal{I} \tag{4.2}$$

$$\sum_{l \in I} y_{lm} \leq x_m \qquad \forall I \in \mathcal{I}, m \in L \tag{4.3}$$

$$y_{lm} = y_{ml} \qquad \forall l, m \in L, l < m \tag{4.4}$$

$$x, y \geq 0 \qquad \text{integer} \tag{4.5}$$

has to be extended with inequalities constraining the degree of each node. Taking a look at the ILP formulation from Reinert (1999),

$$\max \sum_{e_i \in A} w_i x_i + \sum_{m_{i,j} \in I} w_{i,j} x_{i,j} \tag{4.6}$$

$$\sum_{e \in C} x_e \leq |C \cap E| - 1 \qquad \forall \text{critical mixed cycles C} \tag{4.7}$$

$$\sum_{j} x_{i,j} \leq x_i \qquad \forall x_{i,j}, x_i \tag{4.8}$$

$$\sum_{i} x_{i,j} \leq x_j \qquad \forall x_{i,j}, x_j \tag{4.9}$$

$$x_i, x_{i,j} \in \{0, 1\} \tag{4.10}$$

there are two obvious changes to (4.1)-(4.5):

- constraints (4.8) and (4.9) have to be added to the *CMO*-ILP.
- objective function (4.1) has to be replaced by objective function (4.6).

Then, a first ILP formulation for aligning RNA secondary structure is

$$\max \sum_{m \in L} \sum_{l \in L} b_{lm} y_{lm} + \sum_{m \in L} w_m x_m \tag{4.11}$$

$$\sum_{l \in I} x_l \leq 1 \qquad \forall I \in \mathcal{I} \tag{4.12}$$

$$\sum_{l \in I} y_{lm} \leq x_m \qquad \forall I \in \mathcal{I}, m \in L \tag{4.13}$$

$$y_{lm} = y_{ml} \qquad \forall l, m \in L, l < m \tag{4.14}$$

$$\sum_{l \in L} y_{lm} \leq x_m \qquad \forall m \in L \tag{4.15}$$

$$x, y \geq 0 \qquad \text{integer} \tag{4.16}$$

(4.11)-(4.16) takes the differences listed in Table 4.1 into account:

- (4.11) maximizes the weight of interaction *and* realized alignment edges.
- (4.15) and (4.14) take care that the degree of each node is one at most. To be more precise, the number of interaction edges realized at a specific node is 2 at most (one outgoing and one incoming interaction edge, that is $y_{lm} = 1$ and $y_{lm} = 1$). Since in the original formulation y_{lm} and y_{ml} belong to the same interaction edge, we count the incoming and outgoing edge as one.

However, the ILP can be simplified:

As described above, the degree of each node is constrained by (4.15) and (4.14). Therefore, every line $m \in L$ can be incident to at most one other line $l \in L$. Consequently, constraints (4.13) can be dropped, since there is at most one line m, such that $y_{lm} = 1$, for all other $y_{lm} = 0$ holds. Then, the only thing that has to be checked is that line l and m do not cross, but this is already done by constraints (4.12).

$\sum_{l \in I} x_l \geq 2$ implies that $y_{ml} = 1$ for two lines $l, m \in L$. The lines l and m are in the same set I of incompatible lines and therefore the two lines would cross (which is not a valid structural alignment).

Dropping constraints (4.13), the final ILP is

$$\max \sum_{m \in L} \sum_{l \in L} b_{lm} y_{lm} + \sum_{m \in L} w_m x_m \tag{4.17}$$

$$\sum_{l \in I} x_l \leq 1 \qquad \forall I \in \mathcal{I} \tag{4.18}$$

$$y_{lm} = y_{ml} \qquad \forall l, m \in L, l < m \tag{4.19}$$

$$\sum_{l \in L} y_{lm} \leq x_m \qquad \forall m \in L \tag{4.20}$$

$$x, y \geq 0 \qquad \text{integer} \tag{4.21}$$

Section 4.3 explains how the ILP above can be solved on the analogy of solving the ILP for *CMO*.

4.3 Solving the ILP

Solving the ILP (4.17)-(4.21) can be done as described in Section 3.2. First, a basic procedure will be explained, yielding solutions of poor quality on the original problem. Consequently, the algorithm based on *Lagrangian relaxation* will be presented afterwards leading to significantly better results.

4.3.1 Relaxing the ILP

Like in Section 3.2.2 the ILP written above cannot be tackled directly. Hence, the problem is relaxed by the equality constraints (4.19). Hence, the relaxed ILP is solvable in $\mathcal{O}(|G_1(E)||G_2(E)|)$.

By dropping (4.19), the ILP is changed to

$$\max \sum_{m \in L} \sum_{l \in L} b_{lm} y_{lm} + \sum_{m \in L} w_m x_m \tag{4.22}$$

$$\sum_{l \in I} x_l \leq 1 \qquad \forall I \in \mathcal{I} \tag{4.23}$$

$$\sum_{l \in L} y_{lm} \leq x_m \qquad m \in L \tag{4.24}$$

$$x, y \geq 0 \qquad \text{integer} \tag{4.25}$$

Taking a close look at the ILP, one recognizes that it can be solved by the same procedure as described in Section 3.2.2. The only difference is that there is one more classe of inequalities, namely (4.24).

The following considerations are analog to the ones in Section 3.2.2:

$x_m = 0, m \in L$ implies that all y_{lm} in (4.24) have the same value, that is, all values equal 0. However, for $x_m = 1$ the optimal choice of y_{lm} is given by

$$\begin{aligned} \max \sum_{m \in L} \sum_{l \in L} b_{lm} y_{lm} + w_m & \\ \sum_{l \in I} x_l \leq 1 & \qquad \forall I \in \mathcal{I} \\ \sum_{l \in L} y_{lm} \leq 1 & \qquad m \in L \\ y \geq 0, \text{integer} & \end{aligned}$$

That means that the maximal "profit" is not given by a set of non-crossing lines that are incident to the current line – like it was the case with aligning proteins. The restriction on the degree of every node leads to the selection of exactly *one* line that promises the highest profit, if line m was realized. Consequently, the main difference to the algorithm of aligning protein structures is the following:

Not a set of other, non-crossing lines is chosen to form the maximal profit, but just *one* single line. Otherwise the degree of every node could be greater than one, that is,

$$\sum_{l \in L} y_{lm} \geq 2$$

for $m \in L$.

As soon as the profits p_m are computed for every $m \in L$, the algorithm is straightforward:

Given a single $p_m, m \in L$, choose the lines that maximize the overall profit, that is compute

$$\begin{aligned} \max \sum_{m \in L} p_m x_m & \\ \sum_{l \in I} x_m \leq 1, \forall I \in \mathcal{I} & \\ x \geq 0, \text{integer} & \end{aligned}$$

The value of the alignment gives an upper bound on the original problem. Intuitively, this is clear: Every line m chooses locally the best line that maximizes the overall profit. In an optimal solution this means that

line m chooses l as its maximum profit line and vice versa. This yields the maximum value that both lines can possibly contribute to the overall profit.

Since the constraints $y_{ml} = y_{lm}$ was dropped, this does not have to be the case anymore. Consider the following case (see also Figure 4.2): Line m realizes its maximum profit with line l, whereas line l achieves the best value choosing k. Since $y_{ml} = y_{lm}$ this situation is valid in the relaxed ILP.

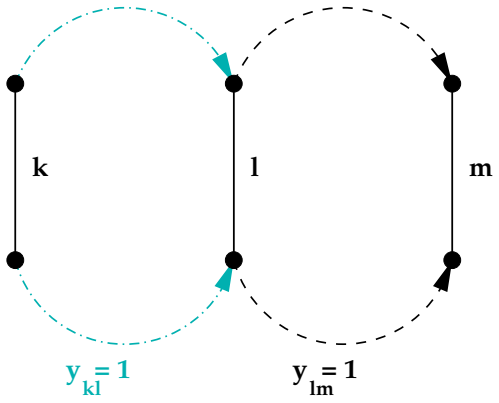

Figure 4.2: Valid situation in the relaxed ILP

The remaining algorithm remains the same, that is the optimal solution to the relaxed ILP is computed in a *two-phase selection algorithm*:

1. **First selection** Compute the maximal profit p_m that every line $m \in L$ can possibly achieve with another line $l \in L$.

2. **Second selection** Maximize the sum of profits such that no two lines cross, that is compute an alignment with the single $p_m, m \in L$ as weights.

A difficulty that comes along with constraining the degree of every node is the calculation of a valid solution for a structural alignment, given a set of non-crossing lines (see Section 4.3.2).

4.3.2 Computing Feasible Solutions

In Section 3.2.2 the calculation of the Lagrangian multiplier is described. One of the key parameters is the stepsize γ that is computed the following

way:

$$\gamma = \mu \frac{\mathrm{UB} - \mathrm{LB}}{\sum_{m,l} s_{ml}^2}$$

The values are

- **UB** An upper bound on the problem: this value is given by the result of the non-crossing matching of maximum weight, that is the result of the first alignment phase.
- **LB** A lower bound on the problem: a (heuristic) feasible solution for the problem.

In case of the *CMO*-problem, computing a lower bound on a problem instance, given a set of alignment edges that form the non-crossing matching, is easy, since the degree of the nodes is not constrained. Therefore, *every* interaction edge that is realized contributes to the solution. Dealing with RNA secondary structures, the situation is different.

Here, the degree of the nodes is constrained, leading to the following question: Given a set of fixed alignment edges, which interaction edges should be chosen, such that the overall score is maximal?

The upper part of Figure 4.3 gives an example for the starting point of the computation of a lower bound *LB*.

The problem can be formulated the following way:

Given a set of alignment edges and a set of interaction edges associated with a certain value (in Figure 4.3 the values are represented by w_i and w_j). Which interaction edges have to be chosen, such that the profit is maximal under the constraint that every node is incident to at most one interaction edge?

It turns out that this problem can be formulated as a *general matching of maximum weight* (see Section 2.1).

Consider the alignment edges as nodes and every pair of interaction edges (i_1, i_2) whose endpoints are adjacent to a pair of alignment edges are the edges of the graph. We call the resulting graph the *interaction matching graph* $\mathbb{G}$. The sum of the weights of i_1 and i_2 is the weight of an edge in $\mathbb{G}$. The lower part of Figure 4.3 shows the matching graph of the graph shown in the upper part of Figure 4.3.

Lemma 4.3.1. *A matching of maximum weight in the interaction matching graph corresponds to the structural alignment of maximum weight in the original graph.*

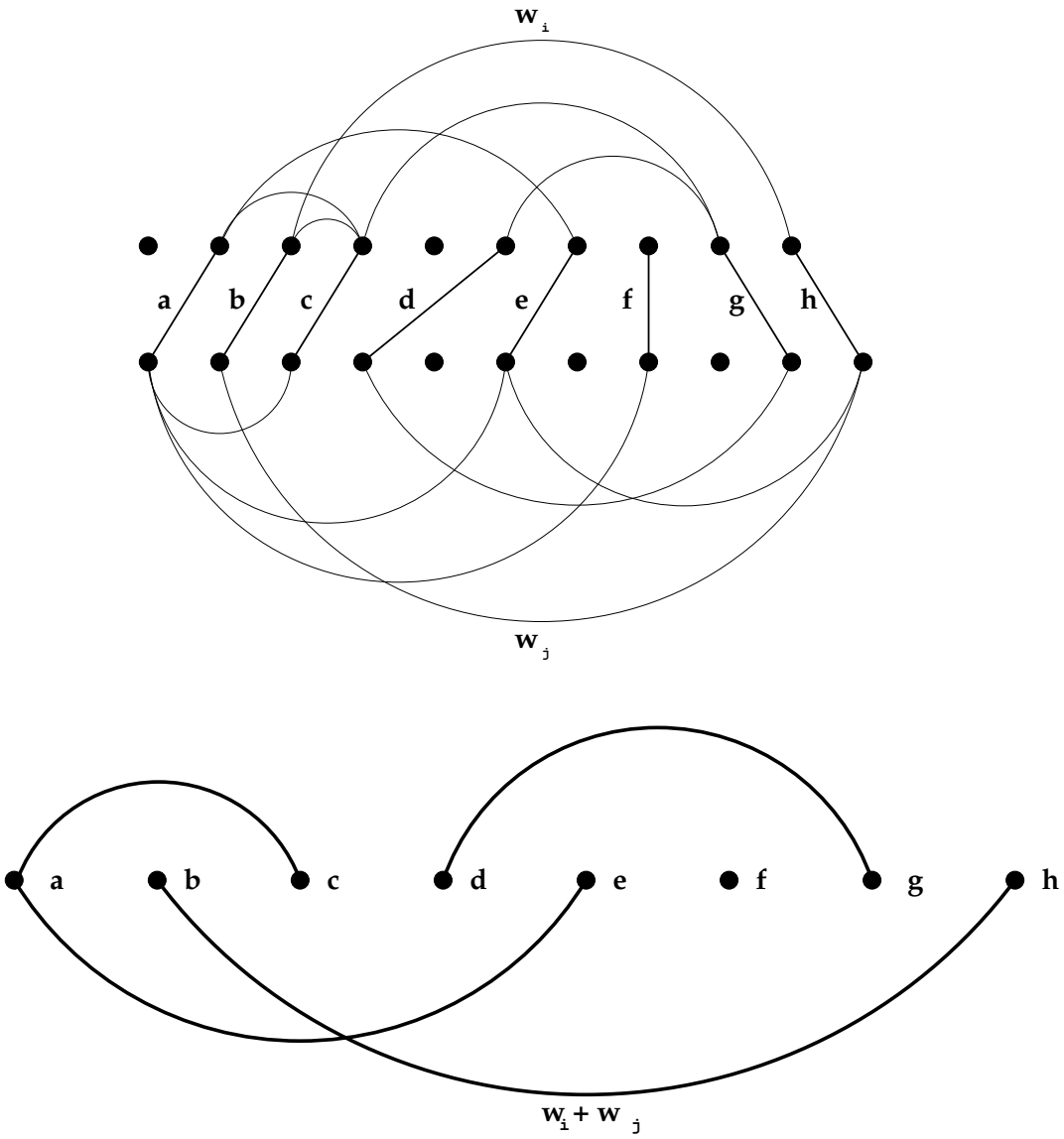

Figure 4.3: Matching graph for computing a lower bound

Proof. Recall the definition of a matching of maximum weight from Section 2.1. In a matching every node is incident to one edge at most, the sum of the weights of the matching edges is maximal. It is not hard to see that the two properties for a structural alignment are satisfied:

1. Every node in the original graph is incident to (at most) one interaction edge (remember that nodes in the interaction matching graph

are a bijective mapping of alignment edges in the original graph).

2. The sum of weights of the interaction edges is maximal.

□

A "backtracking" of the solution, that is, which interaction edges are part of the matching in the matching graph, is simple in the way that there is a one-to-one mapping between the edges in the matching graph and pairs of interaction edges in the original graph. Every edge in the matching graph corresponds to a pair of interaction edges in the original graph.

4.3.3 Solving the Lagrangian Relaxed Problem

The problem described in Section 4.3.1 relaxes the ILP by just dropping some constraints. A Lagrangian relaxation can be obtained by the same approach as the one described in Section 4.3.1.

Instead of just dropping constraints (4.19), they are taken to the objective function, assigned with a multiplier:

$$\max \sum_{m\in L}\sum_{l\in L} b_{lm}y_{lm} + \sum_{m\in L} w_m x_m + \sum_{l\in L}\sum_{m\in L, l<m} \lambda_{lm}(y_{lm} - y_{ml})$$

$$\sum_{l\in I} x_l \leq 1 \qquad \forall I \in \mathcal{I}$$

$$y_{lm} = y_{ml} \qquad \forall l, m \in L, l < m$$

$$\sum_{l\in L} y_{lm} \leq x_m \qquad \forall m \in L$$

$$x, y \geq 0 \qquad \text{integer}$$

Again, with $\lambda_{ml} = -\lambda_{lm}$ and $\lambda_{ll} = 0$ the objective function can be written as

$$\max \sum_{m\in L}\sum_{l\in L} (\lambda_{lm} + b_{lm})y_{lm} + \sum_{m\in L} w_m x_m \tag{4.26}$$

The multipliers are, like in Section 3.2.2, computed the following way.

First, compute the subgradients

$$s_{lm} = \bar{y}_{lm} - \bar{y}_{ml}, \forall l, m \in L, l < m$$

Then, by using these subgradients, the series of λ_{lm}^k with $i = 0, \ldots, n$ is given by

$$\lambda_{lm}^{k+1} = \begin{cases} \lambda_{lm}^k & \text{if } s_{lm} = 0 \\ \max(\lambda_{lm}^k - \gamma, -b_{lm}) & \text{if } s_{lm} = 1 \\ \min(\lambda_{lm}^k + \gamma, b_{lm}) & \text{if } s_{lm} = -1 \end{cases} \tag{4.27}$$

with

$$\gamma = \mu \frac{\mathrm{UB} - \mathrm{LB}}{\sum_{m,l} s_{ml}^2}$$

and

$$\lambda_{lm}^0 = 0, \forall m, l \in L$$

4.4 An Algorithm Based on Lagrangian Relaxation

The considerations from the previous sections lead to the following algorithm that computes a structural alignment for two annotated sequences (S_1, P_1) and (S_2, P_2):

Algorithm 1: Computation of pairwise RNA structural alignment

1: Compute a traditional sequence alignment of S_1 and S_2 to get a set of alignment edges.
2: Insert additional alignment edges to the ones computed by the traditional sequence alignment
3: Compute interaction edges using the annotations P_1 and P_2
4: Determine weights w_{ml} and $b_{ml} = b_{lm} = \frac{w_{ml}}{2}$ for every pair m and l, with $m, l \in L$
5: **while** stop condition not met **do**
6: Compute maximal profit of each alignment edge $m \in L$
7: Compute an alignment with the single profits as weights.
8: Compute a feasible solution, given the alignment edges.
9: Compute subgradients according to non-crossing matching and the chosen profit alignment edges
10: Adapt Lagrangian multiplier according to the subgradients
11: **end while**
12: Output structural alignment

The stopping condition of the *while loop* is very vague, because there are different strategies to stop the computation, *e.g.*

- The authors of Caprara and Lancia (2002) limit the number of iterations to $\max\{1000, 10 \max\{|G_1(E)|, |G_2(E)|\}\}$, since they did not experimentally observe major improvements afterwards.
- Fisher (1981) proposes a fixed number of iterations.
- If UB $=$ LB, the computations can be stopped, since the optimal solution has been reached (or to put it in other words: $\sum_{m,l} s_{ml}^2 = 0$).

In Chapter 5 we will discuss our implementation in detail and show the computational results computing

- Structural alignments from similar sequences (*e.g.*, sequences from the same family) that provide already high sequence conservation.
- Structural alignments from sequences that do not have high sequence conservation (*e.g.*, sequences stemming from different families).

And what have you got at the end of the day?
What have you got to take away?

DIRE STRAITS

Chapter 5

Computational Results

This section describes the results that were obtained while performing computations with real-world data.

First, a short summary of the implementation of the algorithm described in Section 4.3 is given. Second, the generation of the alignment and interaction edges is illustrated, since the running of the implementation depends heavily on the number of edges.

Finally, the results of the implementation are discussed.

5.1 Implementational Issues

5.1.1 Implementation

The program for computing structural alignments of two RNA sequences was implemented in C++ using LEDA, the *Library of Efficient Data Types and Algorithms* (for details see LEDA, 2004).

Two points are of crucial importance for keeping the running time low:

1. Finding the maximum profit edge, that is the edge with which the maximal possible profit is realized, should take constant time.
2. Computing a feasible solution of maximum weight given a set of alignment edges, as described in Section 4.3.2. This leads to the computation of a matching during every iteration of the algorithm.

Finding the maximum profit edge in constant time can be done as follows: A set of possible partner lines (the dashed alignment edges in Figure 5.1)

for an alignment edge (i, j) is given. All possible partner lines are stored in a *priority queue* with the possible profit as the priority. Extracting the element with the highest priority means nothing else than selecting the edge with which the highest profit can possibly be achieved.

During every iteration each alignment edge has to change one profit at most, because only one single profit edge is selected which might cause an adaption of the corresponding Lagrangian multiplier.Therefore, updating a single profit (or priority in terms of priority queues) can be done in $\mathcal{O}(\log n)$.

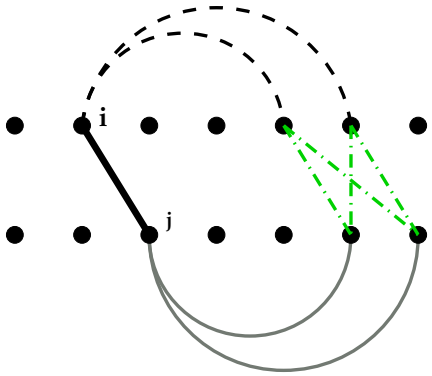

Figure 5.1: Handling possible partner edges

The computation of the lower bound during each iteration is done as described in Section 4.3.2. Given a set of select alignment edges, we search for the matching of maximum weight in the corresponding *interaction matching graph*. This can be done by using the built-in LEDA matching algorithms.

5.1.2 Generating the Alignment Edges

The number of alignment edges influences the speed of the computation significantly: the more alignment edges, the more possible structural alignments have to be considered and the more Lagrangian multipliers have to be adapted. Therefore, we generate the alignment edges in an intelligent way, improving the chance to generate the correct alignment edges that form the structural alignment of maximum weight.

The process of generating the alignment edges is basically the same as the one described in Reinert (1999):

We start from computing a traditional sequence alignment with affine gap costs. Then, we do not only consider the set of alignment edges forming the best sequence alignment with respect to the score function, but also all other alignment edges that are part of a sequence alignment seq_s scoring better than a fixed value s below the optimal. The term seq_0 denotes alignment edges that are part of an optimal sequence alignment (remember that, in general, an optimal sequence alignment of two sequences is not uniquely defined) are taken into account. seq_{50} denotes the set of alignment edges that are part of a sequence alignment whose score is at most 50 below the optimal score.

This can efficiently done using the algorithm of Vingron and Argos (see for example Vingron (1996)).

5.1.3 Generating the Interaction Edges

There are two ways to generate the interaction edges of an RNA sequence:

- **Using the base pair probabilities as weights for the interaction edges**

  The program **RNAfold** that is part of the Vienna RNA package allows the computation of the base pair probabilities of a RNA sequence. The output is a *dotplot* of the sequence (see Figure 5.2). The upper part of the matrix shows the probability with which the two nucleotides, specified through the matrix position, fold onto each other: the bigger the rectangle, the higher the probability.

  The lower part shows one of the possible minimum free energy structures of the sequence (this part, however, is of no further interest for the following computations).

- **Using fixed secondary structures**

  As a second method to generate interaction edges, secondary structures from the *European ribosomal RNA database* (Wuyts, de Peer, Winkelmans, and Wachter, 2002) were used. The database offers a large variety of annotated RNA sequences: data files containing both the sequence and the biologically verified secondary structure.

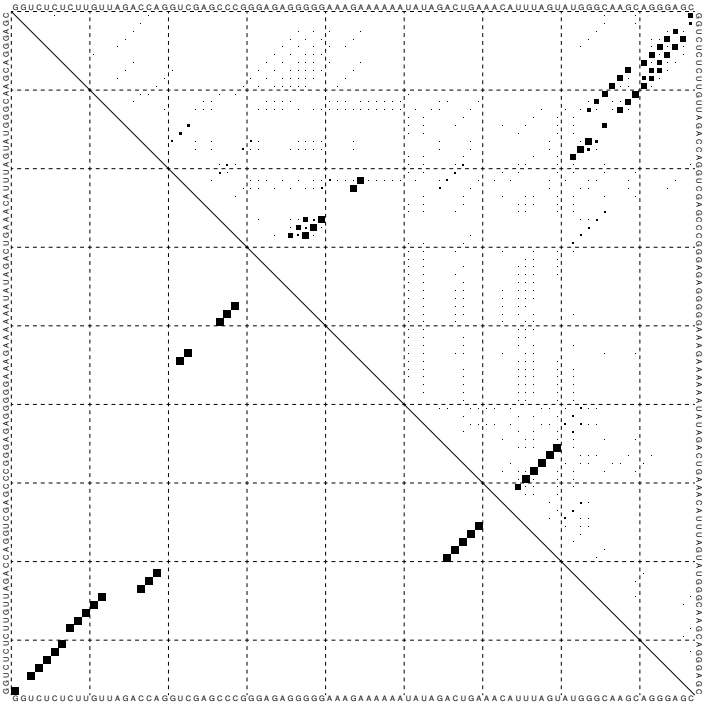

Figure 5.2: Dotplot of a HIV sequence

5.2 Results

5.2.1 Testing Strategies

Basically, two different testing strategies were pursued:

1. **Aligning two RNA sequences with the base pair probabilities as weights**

   In Section 1.1.2 we outlined that RNA alignments based on sequence information alone are generally not sufficient to detect structural elements. Therefore, the base pair probabilities for each sequence are taken into account to compute structurally accurate alignments.

2. **One given secondary structure**

   As the second testing strategy the (known) structure of one RNA sequence was taken to form the interaction edges, whereas all possible interaction edges were allowed for the second RNA sequence.

The idea is that the fixed structure should help to identify conserved structures within the structural alignment.

5.2.2 Chosen Parameters

The following parameters were used throughout the computation process:

- Figure 5.3 shows the mutation score matrix that was used to score pairs of nucleotides:

	–	A	U	C	G
–	0	0	0	0	0
A		4	1	1	1
U			4	1	1
C				4	1
G					4

Figure 5.3: Mutation score matrix used for experiments

- For the initial traditional sequence alignment with affine gap costs we used 6 as gap open and 2 as gap extension penalty, respectively. The level of suboptimality was set to 50, *i.e.* every alignment edge is generated that is part of a sequence alignment whose score is not more than 50 below the optimal score.

- Like Reinert (1999) the weight of an interaction edge was set to 8, leading to a score of 16, if a pair of interaction edges (one interaction edge in the first and the other one in the second sequence) is realized.

- Computing the optimal Lagrangian multiplier using subgradient optimization is an iterative computations (as outlined in Section 2.4.2): the number of iterations was 1500 iterations at most, since it turned out that afterwards no substantial improvements occurred anymore. If the upper and lower bound coincide earlier, the computation is stopped since the optimal value of the Lagrangian dual has been reached.

Compared to the tests conducted by Caprara and Lancia (2002) this is a bit different: they set the number of iterations to 1000 at most.

5.2.3 General Observations

As described in Section 4.4 the optimal value is reached as soon as the upper bound UB and the lower bound LB are the same. Two main observations regarding the upper and lower bound were made during the experiments:

1. If an optimal value is found, the gap between upper and lower bound becomes very small within just a few hundred iterations. The rest of the iterations is used to adapt the Lagrangian multiplier in such a way that – in the end – the gap becomes 0. Figure 5.4 shows an example of the development of lower and upper bound where the gap finally becomes 0.

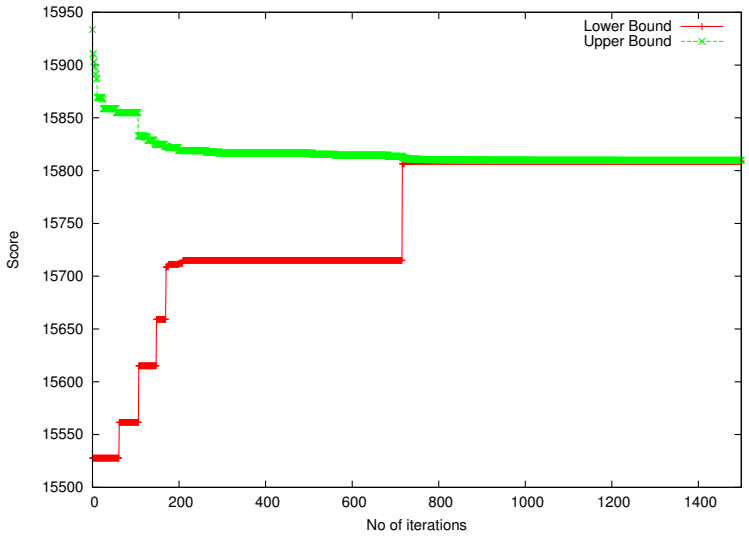

Figure 5.4: Development of lower and upper bound yielding a gap of 0

2. A low traditional sequence score, however, is a strong indication that the Lagrange method will fail to find an optimal value, leading to a gap between lower and upper bound. It was tried to increase the number of iterations, but it turned out that the gap remained the same even after a few thousand iterations. Figure 5.5 shows an example where the gap between lower and upper bound remains the same.

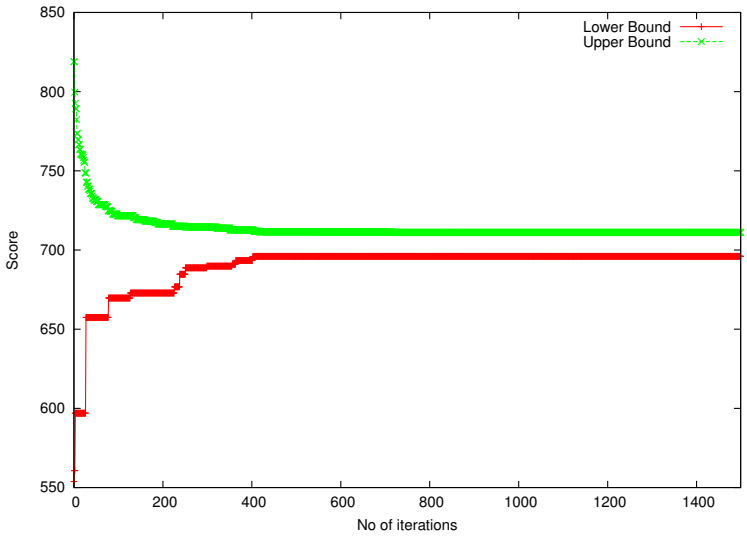

Figure 5.5: Development of lower and upper bound yielding a non-decreasing gap

This observation corresponds to the paragraph in Caprara and Lancia (2002) where the authors state that

> (...) it has to be remarked that the optimal solution of instances associated with substantially different proteins seems completely out of reach not only for our algorithm (...).

The score of the traditional alignment measures how similar the two sequences under consideration are: the lower the score, the dissimilar the two sequences are. It will be shown in the next sections that whenever the traditional sequence score is low, the gap between lower and upper bound is big, yielding solutions that are not guaranteed to be optimal or near-optimal.

5.2.4 Tests with 5S RNA Sequences

In the beginning, we conducted some experiments with the following 5S RNA sequences, all taken from the *5S Ribosomal RNA Database* (Szymanski, Barciszewska, Barciszewski, and Erdmann, 2000). The twelve 5S sequences, all belonging to the family of the *Cytophagales*, are listed in Table 5.1.

♯	Group	Name	Length
1	A	Bacteroides Thetaiotaomicron	111
2	A	Bacteroides Veroralis	112
3	A	Cytophaga Aquatilis	111
4	A	Anaerorhabdus Furcosus	114
5	A	Bacteroides Capillosus	115
6	A	Bacteroides Fragilis	110
7	B	Cytophaga Heparina	114
8	B	Cytophaga Johnsonae	116
9	B	Flavobacterium Breve	121
10	B	Flexibacter Sp	117
11	B	Porphyromonas Gingivalis	111
12	B	Saprospira Grandis	122

Table 5.1: Twelve 5S RNA sequences

For all sequences the base pair probabilities were computed using *RNAfold* from the *Vienna RNA Package*. Subsequently, for all possible pairs between the two groups a structural alignment was computed using the base pair probabilities, multiplied by 8, as the weight of the interaction edges. The multiplication by 8 is necessary to adapt the weight to the mutation score matrix presented in Section 5.2.2 (otherwise the maximal weight of an interaction weight is one, whereas even different letters have a score of one).

Figure 5.6 shows the results of the structural alignments computed from all possible pairs of sequences listed in Table 5.1.

Lower Bound and *Upper Bound* denote the best lower and upper bound found during the computations. *Conventional Alignment* is the value of the *structural alignment* induced by a traditional sequence alignment and is computed the following way:

The alignment edges computed by a traditional sequence alignment with affine gap costs are taken and form the input edges for the *interaction matching graph* (see Section 4.3.2). Then, the structural alignment score is computed that is defined by the traditional alignment edges.

It can be seen that the Lagrange method performs very well on test instances that have a fairly high conventional alignment score, whereas sequences that provide a low conventional alignment score fail to be structurally aligned to optimality (and hence lead to a gap between the lower and the upper bound).

It is also worth noting that in one case – the structural alignment between

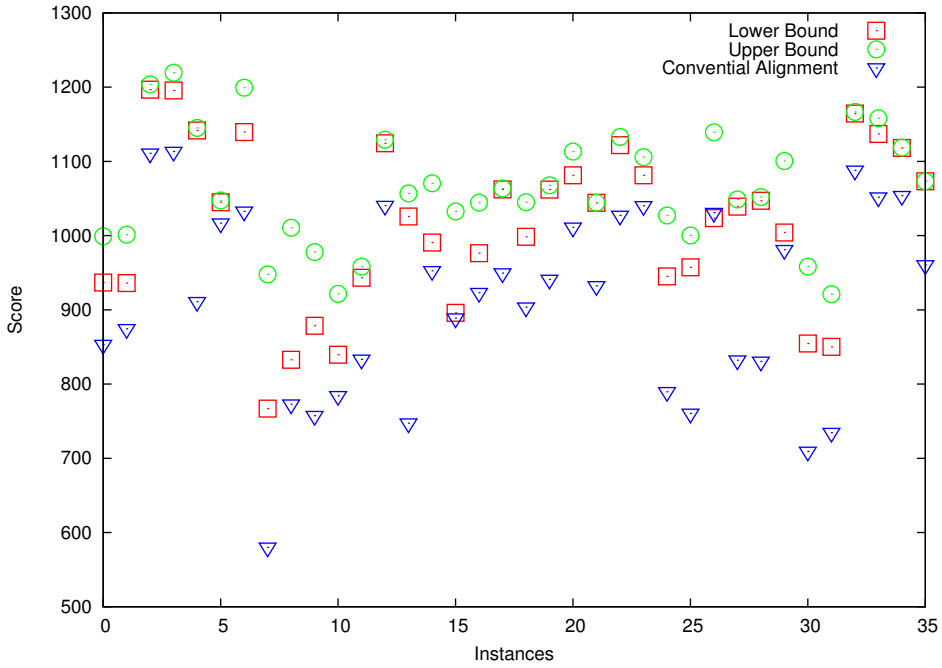

Figure 5.6: Structural alignments of 5S RNA sequences

Saprospira Grandis and *Bacteroides Veroralis* – the Lagrange method even failed to produce a better score than the one provided by the conventional sequence alignment. This might indicate that the two sequences are very closely related to each other, yielding high sequence conversation that is already optimal or near-optimal.

In the following, the score of some test instances that was computed using the Lagrange method was compared to alignment produced by `pmmulti`, the program by Hofacker et al. (2004) to align base pair probability matrices using dynamic programming. Table 5.2 shows the results.

The weights of the interaction edges were computed the same way as described in (Hofacker et al., 2004) for the sake of greater comparability of the results. This means that not the pure base pair probabilities were taken, but the probability $\bar{\phi}_{i,j}$ between residue i and j was computed by

$$\bar{\phi}_{i,j} = \log \frac{\phi_{i,j}}{\phi_{\min}}$$

with $\phi_{\min}$ being the lowest probability in the matrix.

Surprisingly, the gap between the value of `pmmulti` and the Lagrange method turned out to be big. The primary reason for the different val-

Sequence 1	Sequence 2	pmmulti	Lagrange
Cytophaga Heparina	Bacteroides Veroralis	743.67	829.214
Cytophaga Heparina	Bacteroides Thetaiotaomicron	748.351	819.883
Cytophaga Johnsonae	Cytophaga Aquatilis	640.335	673.783
Flectobacillus Major	Euplotes Woodruffi	679.295	732.62
Flexibacter Sp	Bacteroides Thetaiotaomicron	675.284	731.189
Flexibacter Sp	Anaerorhabdus Furcosus	748.559	792.912
Flexibacter Sp	Bacteroides Capillosus	497.828	702.525
Flexibacter Sp	Cytophaga Aquatilis	531.358	696.05

Table 5.2: Comparison between Lagrange and pmmulti

ues lies in the secondary structure that is allowed during the computation: `pmmulti` follows a dynamic programming approach that does not allow pseudoknots, whereas the Lagrange method does not restrict the secondary structure (and indeed, a closer look at the structural alignment produced by the Lagrange method reveals that the realized interaction edges form around 15 pseudoknots).

5.2.5 Tests with 23S RNA Sequences

General Tests

The fourteen ribosomal 23S RNA sequences listed in Table 5.3, taken from the *European ribosomal RNA database*, formed the input data for the following tests. The idea is that taking one known structure directs the structural alignment to conserved structural elements within the two sequences.

The column *Acc.-Num.* lists the *GenBank* identifier for the specific sequence, whereas column *R* has a star, if the sequence was also used during the tests by Reinert (1999). This will be important later on, because the results computed by Reinert will be compared to the ones obtained by the Lagrangian method.

In the first test round structural alignments for all possible pairs of sequences within each of two families were computed. The base pair probabilities – again computed by means of `RNAfold` – were used as weights of the interaction edges for both sequences. Figure 5.7 and Figure 5.8 show the results for the family of Crenarchaeota and Euryarchaeota, respectively.

Again, it turns out that, in general, the better the conventional alignment

♯	Family	Name	Acc.-Num.	Length	R
1	Crenarchaeota	Pyrodictium Occultum	M21087	1497	∗
2	Crenarchaeota	Sulfolobus yangmingensis	AB010957	1493	
3	Crenarchaeota	Pyrobaculum oguniense	AB029339	1472	
4	Crenarchaeota	Acidianus tengchongensis	AF226987	1496	
5	Crenarchaeota	Sulfolobus shibatae	M32504	1495	
6	Crenarchaeota	Thermoproteus tenax	M35966	1504	
7	Crenarchaeota	Cenarchaeum symbiosum	U51469	1474	
8	Crenarchaeota	Thermofilum Pendens	X14835	1509	∗
9	Euryarchaeota	Natronobacterium tibetense	AB005656	1474	
10	Euryarchaeota	Halobacterium halobium	AJ002949	1463	
11	Euryarchaeota	Haloferax denitrificans	D14128	1469	∗
12	Euryarchaeota	Methanobacterium formicium	M36508	1476	∗
13	Euryarchaeota	Halococcus morrhua	X00662	1475	∗
14	Euryarchaeota	Archaeoglobus fulgidus	X05567	1492	∗

Table 5.3: Fourteen ribosomal 23S RNA sequences

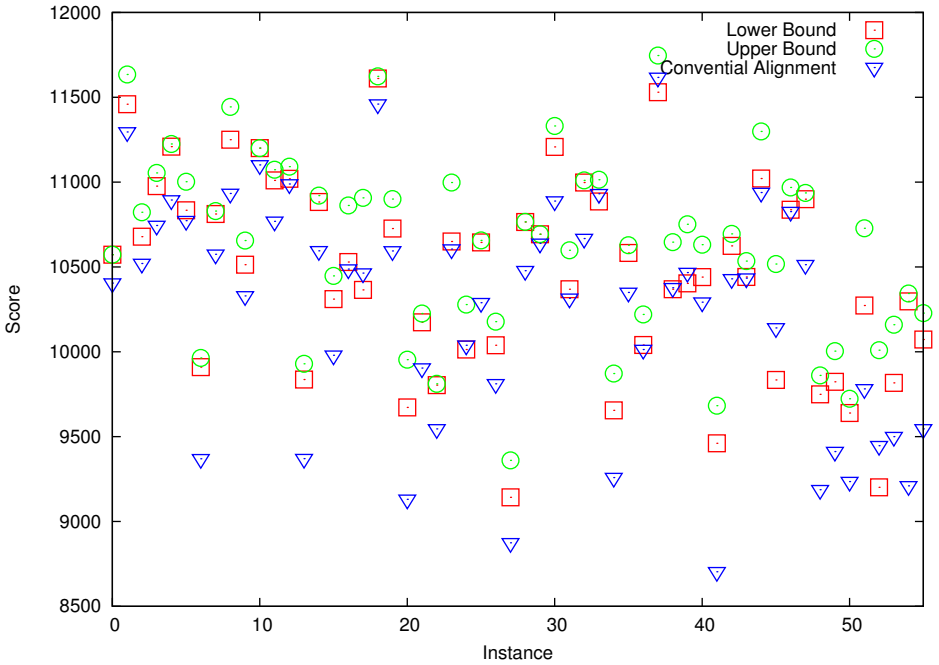

Figure 5.7: Results computing structural alignments of Crenarchaeota sequences

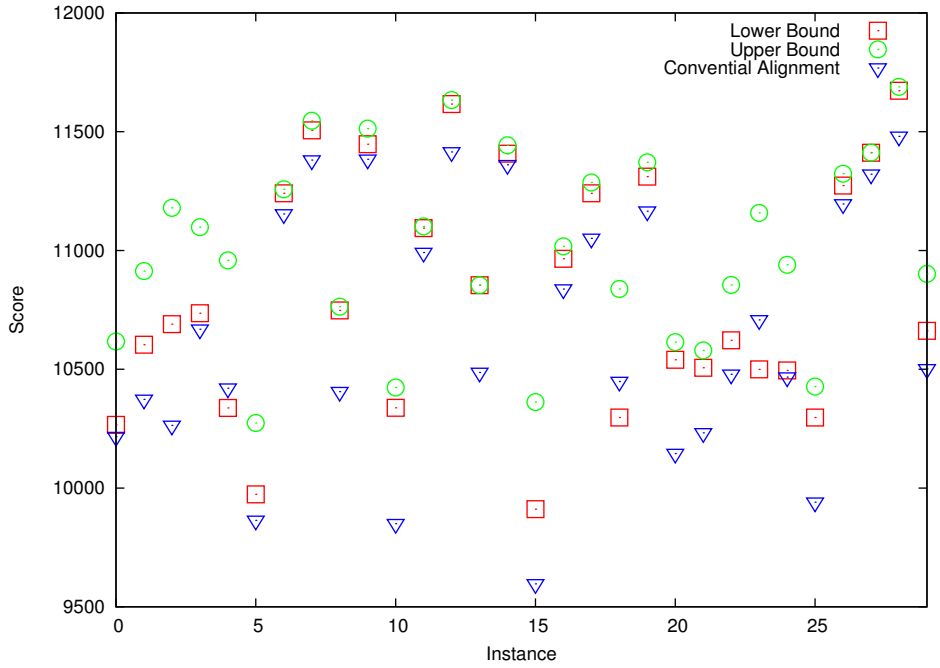

Figure 5.8: Results computing structural alignments of Euryarchaeota sequences

score, the smaller the gap between the lower and the upper bound computed by the Lagrange method. Furthermore, the lower the conventional alignment is, the higher the number of alignment edges (see Figure 5.9). This is clear, since we compute the set of alignment edges by taking all alignment edges whose score is a fixed threshold t below the optimal score. Then, a high conventional alignment score indicates a better quality of the sequence alignment, yielding a smaller number of alignment edges, simply because there are not that many suboptimal possibilities to align the two sequences.

Consequently, a higher number of alignment edges is harder to align, since a greater search space has to be explored.

Lagrange vs. Branch-and-Cut

Finally, the set of sequences from Table 5.3 marked with a star in column R was further examined and the results were compared to those computed by Reinert (1999).

There are two main differences to the tests conducted before:

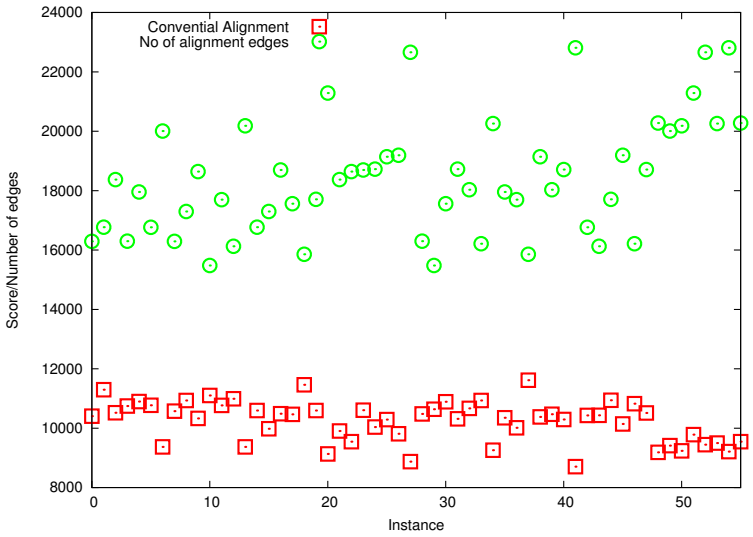

Figure 5.9: Correlation between conventional alignment and the number of alignment edges

1. The interaction edges for one of the two sequences is taken from the secondary structure taken from the *European ribosomal RNA database*.

2. The interaction edges for the second sequence were computed using the entries in the base pair probability matrices, disregarding the base pair probability, however (this will be explained in the following paragraphs).

Especially the second issue is of further interest:

Reinert (1999) performs his tests with holding one structure fixed, whereas he allows all possible interaction edges as the second secondary structure. During the the *worst case generation method* used by Reinert was tuned a bit:

Instead of using all possible base pairs, only the interaction edges with a base pair probability greater than 0 were taken. If a base pair has a probability of 0, it cannot be part of a valid secondary structure under any circumstances, simply because the RNA energy model would not allow such an interaction. Therefore, all entries of the base pair probability matrices were used, disregarding the probability as the weight of the interaction edge. Instead of the probability, a weight of 8 was assigned to each interaction edge to make the results comparable to those in Reinert (1999) (Reinert scores every interaction match with 16).

The assumption was that – despite the modified generation of the interaction edges – the structural scores should be the same. It turned out that this was not the case. Figure 5.10 shows the results for all pairs of sequences.

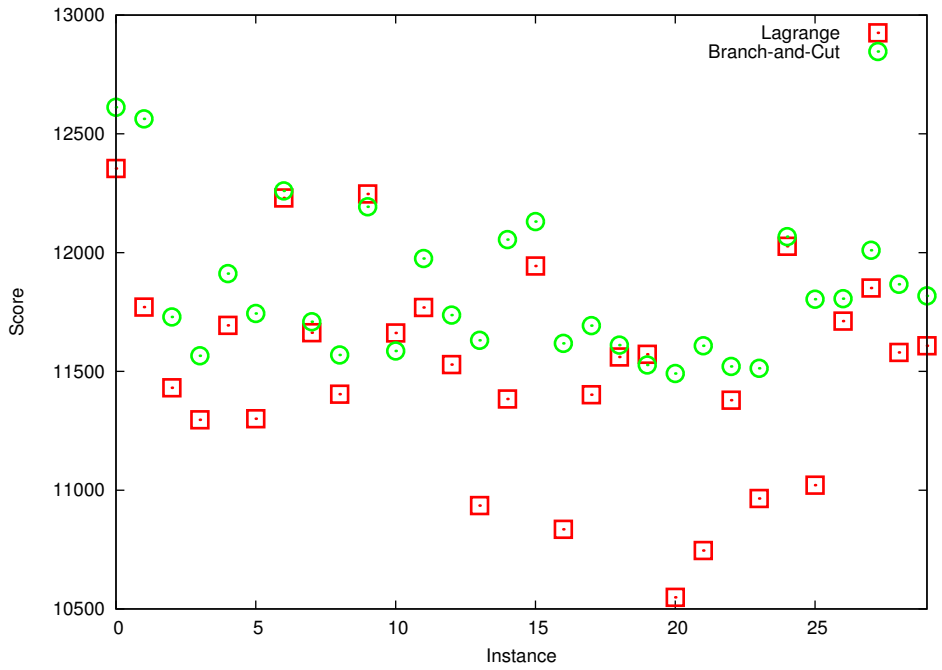

Figure 5.10: Comparison Lagrange vs. Branch-and-Cut without worst case generation

One can easily see that the score computed by the Lagrange method is – except for one instance – always worse than the score computed by the Branch-and-Cut algorithm. A comparison of the best upper bound of the Lagrange method and the Branch-and-Cut algorithm – shown in Figure 5.11 is even more illustrating.

Some instances yield even smaller *upper* bounds than the optimal results computed by the Branch-and-Cut method.

Switching from the enhanced worst-case generation of the interaction edges to the real worst-case generation changes the picture completely. Figure 5.12 shows the results of the computation. The scores computed are much higher then the ones shown in Figure 5.10, in 18 out of 30 cases the Lagrange method computed better results than the Branch-and-Cut algorithm. Furthermore, Reinert was only able to compute structural alignment with a suboptimality level of 10 at most with running times of 4000 and more seconds.

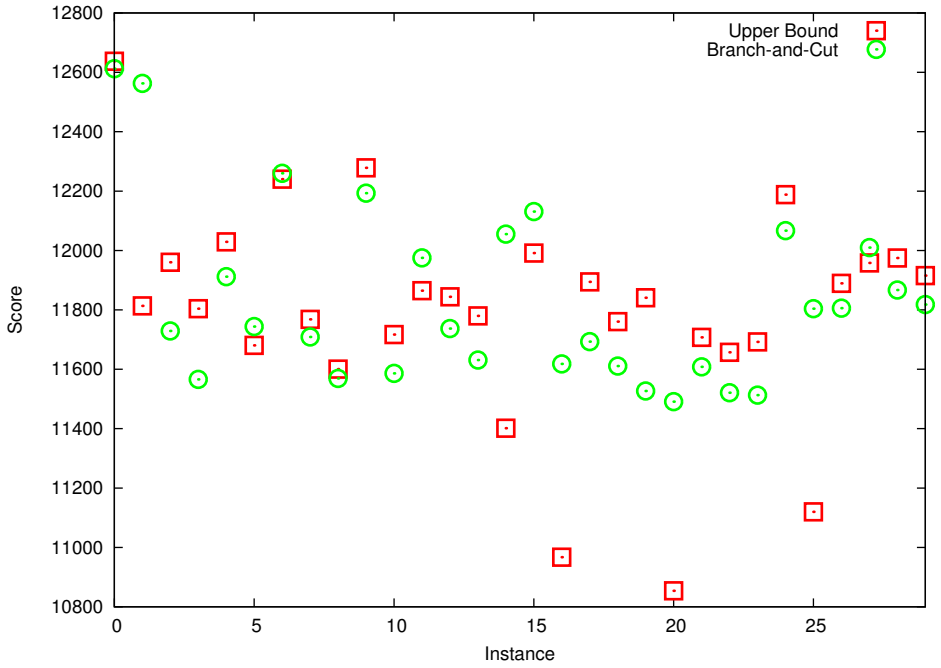

Figure 5.11: Comparison of the upper bound vs. Branch-and-Cut

The Lagrange method works with a suboptimality level of 50 with a running time of about $660 - 700$ seconds per instance.

Comparing Figure 5.10 and Figure t5.12 raises the question whether the structural alignment computed by Reinert are structurally correct. Since the score is much higher, interaction edges are used that would not have been generated by the enhanced worst-case generation of the interaction edges. Consequently, this implies that interaction edges are used that do not fit into the widely investigated RNA secondary structure energy model. In the end, this circumstance possibly yields RNA structural alignments that lack the foundation of the RNA energy model and that might not be observed in nature in that specific form.

Taking all things into account it has been shown that the Lagrangian method works well on sequences that are similar and therefore provide a (relatively) high traditional sequence alignment score.

It has to remarked, however, that there are still instances where the algorithm fails to find an optimal solution (*e.g.*, instances where the structural score of the traditional alignment score is higher than the one found by

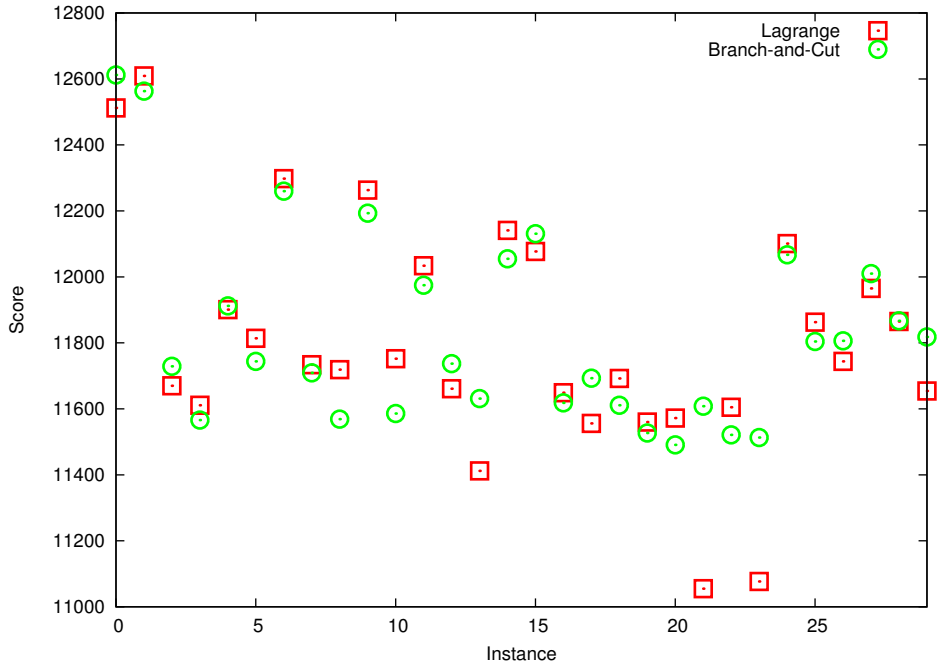

Figure 5.12: Comparison of the Lagrange method with Branch-and-Cut, part 2

the Lagrangian method). It definitely needs further investigation and research, how the algorithm can be modified to provide faster convergence to an optimal (or near-optimal) solution and to avoid big gaps between the lower and the upper bound (like it is the case with instances 21 and 23 in Figure 5.12).

Walk on, walk on
What you've got they can't deny it

U2

Chapter 6

Conclusions and Further Work

This thesis described a new algorithm for computing structural alignments of RNA sequences. After presenting a first ILP formulation for structure alignments the problem is relaxed in a Lagrangian way, that is some constraints are moved to the objective function, assigned with Lagrangian multiplier. The resulting ILP formulation can efficiently be solved and yields an upper bound on the original problem.

For the iterative adjustment of the Lagrangian multipliers it is necessary to compute an upper and lower bound during each iteration of the process. It has been shown how the computation of a lower bound can be reduced to a general matching of maximum weight of the structural enhanced alignment graph.

Since the new method does not follow the previous dynamic programming approaches, the secondary structure is not restricted, that is pseudoknots are allowed within the structure.

The new algorithm outperforms other approaches like Branch-and-Cut by some orders of magnitude by taking much more alignment edges of the structural enhanced alignment graph into account. This ultimately yields higher structural alignment scores within shorter time.

There are, however, several fields of research that could be approached in the future:

- Using other methods to adapt the Lagrangian multiplier that promise better converge than subgradient optimization.
- Embedding the Lagrangian method into a *Branch-and-Bound* approach to obtain provable optimal solutions.

- Extending the structural enhanced alignment graph in a way that linear, affine or even arbitrary gap costs are taken into account.

Bibliography

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

V. Bafna, S. Muthukrishnan, and R. Ravi. Computing similarity between RNA strings. In *Proc. Combinatorial Pattern Matching*, pages 1–14, 1995.

D. Bertsimas and J. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.

A. Caprara and G. Lancia. Structural alignment of large-size proteins via lagrangian relaxation. In *Proceedings of the Sixth Annual International Conference on Computational Biology*, pages 100–108. ACM Press, 2002.

A. Caprara, G. Lancia, B. Carr, B. Walenz, and S. Istrail. 1001 optimal PDB structure alignments: Integer programming methods for finding the maximum contact map overlap. *Journal of Computational Biology*, 11 (1):27–52, 2004.

W. Cook, W. Cunningham, W. Pulleyblank, and A. Schrijver. *Combinatorial optimization*. Wiley-Interscience series in discrete mathematics and optimization. John Wiley & Sons, 1998.

M. Dayhoff, R. Schwartz, and B. Orcutt. A model of evolutionary change in proteins. *Atlas of Protein Sequence and Structure*, pages 345–352, 1978.

R. Diestel. *Graph theory*. Graduate Texts in Mathematics. Springer-Verlag, 1997.

R. Doolittle, M. Hunkapiller, L. Hood, S. Devare, K. Robbins, S. Aaronson, and H. Antoniades. Simian sarcoma virus onc gene, v-sis, is derived from the gene (or genes) encoding a platelet-derived growth factor. *Science*, 221(4607):275–277, July 1983.

M. Fisher. The lagrangian relaxation method for solving interger programming problems. *Management Science*, 27(1):1–17, 1981.

M. Fisher. An applications oriented guide to lagrangian relaxation. *Interfaces*, 15(2):10–21, 1985.

W. Fontana, D. Konings, P. F. Stadler, and P. Schuster. Statistics of RNA Secondary Structures. *Biopolymers*, 33:1389–1404, 1993.

N. C. for Biotechnology Information. Genbank, April 2004. `http://www.ncbi.nlm.nih.gov/Genbank/index.html`.

D. Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology*. Cambridge University Press, 1997.

C. Haslinger. *Prediction Algorithms for Restricted RNA Pseudoknots*. PhD thesis, University of Vienna, March 2001. Institute for Theoretical Biochemistry.

H. Held, P. Wolfe, and H. Crowder. Validation of subgradient optimization. *Mathematical Programming*, 6(1):62–88, 1974.

M. Held and R. Karp. The traveling-salesman problem and minimum spanning trees: Part ii. *Mathematical Programming*, 1:6–25, 1971.

I. Hofacker. Vienna RNA package, March 2004. `http://www.tbi.univie.ac.at/~ivo/RNA`.

I. Hofacker, S. Bernhart, and P. Stadler. Alignment of RNA base pairing probability matrices. *Bioinformatics*, 2004. in press.

I. Hofacker, M. Fekete, C. Flamm, M. Huynen, S. Rauscher, P. Stolorz, and P. Stadler. Automatic detection of conserved RNA structure elements in complete RNA virus genomes. *Nucl.Acids Res.*, 26:3825–3836, 1998.

M. Jünger, G. Reinelt, and S. Thienel. Practical problem solving with cutting plane algorithms in combinatorial optimization. In *Combinatorial optimization. Papers from the DIMACS special year. Papers from workshops held at DIMACS at Rutgers University*, volume 20 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 111–152. 1995.

G. Lancia. personal communication, February 2004.

G. Lancia, R. Carr, B. Walenz, and S. Istrail. 101 optimal PDB structure alignments: a branch-and-cut algorithm for the maximum contact map overlap problem. In *Proceedings of the Fifth Annual International Conference on Computational Biology*, pages 193–202. ACM Press, 2001.

LEDA. *LEDA - Library of Efficient Data Types and Algorithms*. Algorithmic Solutions, May 2004. `http://www.algorithmic-solutions.com`.

C. Lemaréchal. Lagrangian Relaxation. In *Computational Combinatorial Optimization, Optimal or Provably Near-Optimal Solutions*, volume 2241 of *Lecture Notes in Computer Science*. Springer, 2001.

H.-P. Lenhof, K. Reinert, and M. Vingron. A Polyhedral Approach to RNA Sequence Structure Alignment. *Journal of Computational Biology*, 5(3): 517–530, 1998.

D. Lindsay, April 2004. `http://www.don-lindsay-archive.org/creation/glossary.html`.

F. Malucelli, T. Ottmann, and D. Pretolani. Efficient labelling algorithms for the maximum non crossing matching problem. *Discrete Applied Mathematics*, 47:175–179, 1993.

J. McCaskill. The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers*, 29:1105–1119, 1990.

S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, pages 443–453, 1970.

C. Notredame, E. O'Brien, and D. Higgins. RAGA: RNA sequence alignment by genetic algorithm. *Nucleic Acids Res.*, 25(22):4570–4580, November 1997.

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

P. Pevzner. *Computational Molecular Biology*. MIT Press, 2000.

B. Poljak. A general method of solving extremum problems. *Soviet Mathematics Doklady*, 8:593–597, 1967.

K. Reinert. *A Polyhedral Approach to Sequence Alignment Problems*. PhD thesis, Universtität des Saarlandes, 1999.

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

P. Schuster, W. Fontana, P. Stadler, and I. Hofacker. From sequences to shapes and back: A case study in RNA secondary structures. *Proceedings Royal Society London B*, 255:279–284, 1994.

M. Szymanski, M. Z. Barciszewska, J. Barciszewski, and V. A. Erdmann. 5S ribosomal RNA database Y2K. *Nucleic Acids Research*, 28(1):166–167, 2000.

M. Vingron. Near-optimal sequence alignment. *Current Opinion in Structural Biology*, 6(3):346–352, June 1996.

M. Waterman. Secondary structure of single-stranded nucleic acids. *Studies on foundations and combinatorics, Advances in mathematics supplementary studies*, 1978.

M. Waterman. *Introduction to Computational Biology*. CRC Press, 2000.

J. Wuyts, Y. V. de Peer, T. Winkelmans, and R. D. Wachter. The european database on small subunit ribosomal RNA. *Nucleic Acids Research*, 30: 183–185, 2002.

K. Zhang and D. Sasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6): 1245–1262, 1989.

M. Zuker. RNA folding. *Methods in Enzymology*, 180:262–288, 1989.

M. Zuker. mfold, 2004. `http://www.bioinfo.rpi.edu/applications/mfold/old/rna/`.

M. Zuker and D. Sankoff. RNA secondary structures and their prediction. *Bulletin of Mathematical Biology*, 46:591–621, 1984.