Multiple Sequence Alignment and Phylogenetic Reconstruction: Theory and Methods in Biological Data Analysis

Gianluca Della Vedova

Advisors: Prof. Tao Jiang (UC Riverside)
Prof. Giancarlo Mauri (Univ. Milano-Bicocca)
Prof. Paola Bonizzoni (Univ. Milano-Bicocca)

Dottorato di Ricerca in Informatica - XII Ciclo
Università degli Studi di Milano

Contents

Abstract

Bioinformatics (or Computational Molecular Biology) is an emerging interdisciplinary field between Computer Science and Molecular Biology. This new field encompasses the study of a number of computational problems arising from the huge amount of biological data that is publicly available nowadays. The final goal is to design efficient algorithmic solutions to such problems or to identify which problems cannot be solved efficiently. This thesis explores two fundamental computational problems arising from the need of studying biological data: sequence comparison and phylogeny analysis.

One of the most widely adopted approaches to sequence comparison that have been introduced in literature is sequence alignment. The problem of Multiple Sequence Alignment is studied deeply in this thesis, in particular the SP-score version, where the cost of a global alignment is the sum of the costs of all pairwise alignments of two sequences. Initially the $\mathcal{NP}$-hardness of the problem is proved, even in the restricted case of metric scoring function and binary alphabet [18], then it is proved the $\mathcal{APX}$-hardness of the case where at most a constant number of spaces can be inserted in each sequence [61]. The restrictions studied are of particular relevance in Biology.

Notwithstanding such results, which state that some of the variants of Multiple Sequence Alignment cannot be efficiently solved, we are able to obtain some algorithmic improvements on a different version. We study the restriction of the problem where the ratio between the minimum and maximum possible costs of pairwise alignments is upper bounded by a constant, and we devise a polynomial-time approximation scheme for such restriction [61], based on the smooth polynomial programming technique.

A different version of multiple sequence alignment, called Trace Alignment, is studied: more precisely we prove the $\mathcal{APX}$-hardness of such problem. This new formulation has particular relevance in practice, since it aims at pointing out the highly conserved subregions that are present in the sequences; such regions are of great interest among biologists, since they are more likely to encode proteins whose existence is fundamental for the species studied.

Another approach to sequence comparison is based on the notions of subsequence and supersequence. More precisely, two different measures of sequence similarity (notably the Longest Common Subsequence and the Shortest Common Supersequence) are studied. An approximation algorithm is described for each of such problems, and the performance of the algorithm proposed is studied experimentally [17, 22]. In the experimental analysis both the length of the sequences computed and a measure expressing how much the solution computed is similar to the optimum one are considered: the analysis is focused on instances which simulates the evolution of species according to the well-known Jukes-Cantor model.

The second part of this thesis is devoted to two problems arising when the analysis of biological data is focused on the discovery and representation of evolutionary events: phylogeny comparison and phylogeny reconstruction. Among the possible formulations of phylogeny comparison, in this thesis we investigate the approximation complexity of computing the Maximum Isomorphic Agreement Subtree of a set of evolutionary trees, proving that there cannot exist a polynomial-time constant-ratio approximation algorithm even in the case of instances containing exactly three trees. Such negative results is successively strengthened [21].

Some of the most common methods for inferring evolutionary trees rely on the notion of quartet. Algorithms which are based on such technique take into account the information associated to each subset of four species (a quartet) and exploit such information to construct the whole tree. Unfortunately the information contained in all quartets is not necessarily consistent, hence the need for identifying and correcting discrepancies among quartets. The first framework for correcting such errors introduced in literature is called *quartet cleaning*. We describe two new algorithms that improve the previously known results, with respect to the number of errors recovered and the time complexity [31].

Computer Science is all about automated problem solving. Clearly solving a probelm requires analyzing closely such problem, identifying a suitable and precise formulation for it, so that designing an efficient algorith is easier. Recent advances in technology have given an impressive boost to research in Molecular Biology: one well-known project is that field is to fully understand the function and the inner working of each portion of DNA in human beings. Since the human genome consists of approximately 3 billions bases, such analysis could not be carried on without the fundamental support of computers.

In order to give efficient solutions to the problems arising in those projects, some knowledge of both Molecular Biology and Computer Science is necessary, such need has given birth to Bioinformatics (also called Computational

Molecular Biology), which is a new and emerging interdisciplinary field between Computer Science and Biology. As stated previously, one of the tasks that must be accomplished, in order to give efficient solutions, is finding a suitable formulation, as combinatorial problem, of real-world problem we are studying.

The recent announcement of the completion of the sequencing of the human genome is all but the final answer to the questions in this field. Just to point out that some of most important biological problems are quite far from getting a definitive solution, biologists still do not have a clear idea of how different genes interact with each other, that is they are not able to determine the so called gene regulatory network built upon the inhibit/activate relations. Another novel problem is that of comparing sequences in order to find homologies, that is regions that are highly conserved among the genome of different species. Since analogies between biological sequences are commonly believed to lead to functional similarities such homologies are of primary interest among researchers in the field.

This thesis contains a brief introduction to the field of Computational Molecular Biology and defines some of the most intriguing problems in the field, such as multiple sequence alignment and reconstruction of evolutionary trees. We will study some formulations of the Multiple Sequence Alignment, the Longest Common Subsequence and the Shortest Common Supersequence problmes, which model the comparison among biological sequences. More precisely we will prove that some of such problems cannot be solved efficiently, while we will describe efficient approximation algorithms for some of those problems.

We will analyze a formulation of the comparison of evolutionary trees called Maximum Isomorphic Agreement Subtree, proving that such problem is hard to approximate.

Finally we will study the problem of inferring phylogenies, in particular within the quartet-based paradigm, designing two efficient algorithms for quartet cleaning, which is a technique to improve the quality of the trees computed according to such paradigm.

Chapter 1

Introduction

Technological advances have lead to tremendous improvements in Molecular Biology, so nowadays researchers in the field have access to an amount of data that was not possible to think of two decades ago. The following sentence is an excerpt from [48]:

> In a short time it will be hard to realize how we managed without the sequence data. Biology will never be the same again. [99]

Molecular biologists have access to a number of data bases over the Internet (among others SwissProt and EMBL), moreover new devices have been developed to help researchers in obtaining biological sequences (i.e. DNA, RNA or proteins). Recently the completion of the sequencing of the whole human genome (which is made of about 3 billion bases) has been announced. This huge amount of data makes the need for efficient algorithms to analyze biological data even more stringent than before.

In order to give good solutions to the problems that arise in this field, it is necessary to have a working knowledge of the fundaments of Molecular Biology as well as Computer Science. This necessity has lead to the birth of Bioinformatics. In this thesis two of the most important problems in the field are investigated: *sequence comparison* and *phylogeny analysis*.

The comparison of sequences is a well-known problem in Computer Science, in fact a number of definitions of distance between two sequences have appeared in literature, such as the Hamming distance and the edit distance [73, 85], and various algorithms for computing such distances have been presented.

Moreover the notion of distance among strings finds a number of applications outside of Bioinformatics, for example the *diff* Unix command relies on a specific notion of distance, based on the definition of longest common

subsequence of two strings. Sequence comparison is at the core of Bioinformatics, as in biomolecular sequences (DNA, RNA, or amino acid sequences) high sequence similarity usually implies significant functional or structural similarity.

In fact "similarity" is a central phenomenon in biology. But sequence similarity is not a definitive tool for biologists: even though hemoglobin is the same protein in flies as in human beings, it comes as no surprise that flies and humans do differ. This means that human genome have some conserved regions when compared to the genome of flies, while some other regions are completely different. Such conserved regions are usually referred to as *homologies* which means inferred common ancestry, although it is commonly misused to mean similarity [52]. This abuse of language is due to the fact that similarity between sequences is an observation and can be quantified, while the fact that there is a common ancestor cannot be measured. Moreover it is possible, as a first approximation, to state that sequence similarity implies functional similarity which, in turn, is likely to point out a common ancestor.

Even though the objects to compare are not necessarily restricted to be sequences, it is usually easier to acquire and examine sequences, than to analyze the phenotypes or to investigate biochemical properties. Some problems on sequences have been studied even before the advent of Bioinformatics, hence studying biological data at the sequence level allows to use some of the results already presented in literature.

Frequently biologists need to look for a given sequence in a data base. Since the amount of data involved in such a data base search is so huge(due to the length of the sequences and the number of sequences stored in the data bases) having efficient methods to compare sequences is of fundamental relevance. In order to get more precise information about how much a sequence is homologous to some sequence stored in a data base we must be able to compare quickly a set of sequences, that is to compute a multiple sequence comparison.

Evolutionary history is often represented by an evolutionary tree (or *phylogeny*) where known sequences of extant species are represented at the leaves of the tree, and their unknown ancestors are represented at internal nodes of the tree. When the tree is known (from previous data and deduction) the problem is to deduce the sequences for the internal nodes optimizing an objective function depending on the tree.

This is a particular version of a more general problem in Biology: to determine the evolutionary history of a set of extant species. One of the most intriguing aspects of such problems is to determine the topology of the tree representing such history. The method that is used more frequently in practice for inferring phylogenies is that of reconstruction from quartets [39].

This method is based on the observation that it is feasible to determine experimentally the evolutionary history of small subsets of the extant species (usually subsets of 4 species, called quartets), but this leaves open the problem of computing an evolutionary tree that is compatible with the results of all experiments.

Advances in Molecular Biology have led to an explosion in the amount of data available for analysis, and in the size of these data sets. Nowadays it is no longer possible to assemble data and build trees by hand, hence there is a growing need for sophisticated techniques to analyze and understand data, balanced by the need of recognizing which problems are simply too difficult for our current resources.

Since such amount of available data allows to compute inexpensively different trees for the same species set (for instance taking into account different DNA sites), the need for computing a common evolutionary history inferred from a set of phylogenies naturally arises: such new biological problem can have a number of different, but sound, combinatorial formulations.

In Chapter 3 we study some formulations of the Multiple Sequence Alignment problems; more precisely we focus on some restriction of great biological relevance, proving that all such formulations cannot be solved exactly efficiently, for some of such formulations we also prove that they cannot be approximated arbitrarily efficiently.

In Chapter 4 we cope partially with such negative results providing an approximation scheme for an interesting restriction of Multiple Sequence Alignment when the number of spaces that can be inserted is bounded.

The results stated in these chapters have been described in previous papers or manuscripts coauthored with P. Bonizzoni, T. Jiang, W. Just and G. Mauri [18, 61, 19].

In Chapter 5 we propose two different approximation algorithms for two related problems: the Longest Common Subsequence and the Shortest Common Supersequence problems, studying their experimental behaviour. Such studies have been presented previously in [17, 22, 11], coauthored with P. Bonizzoni and G. Mauri.

In Chapter 6 the approximation complexity of computing the Maximum Isomorphic Agreement Subtree is investigated, showing that such problem is hard to approximate. Such results appeared also in [20, 21], coauthored with P. Bonizzoni and G. Mauri.

Finally in Chapter 7 two new algorithms for computing the Local Vertex Cleaning are described, showing that one of the algorithm has optimal time complexity, while the second one is able to recover an optimal number of errors. These algorithms have not been published previously and are a joint work with H. T. Wareham [31].

Chapter 2

Basic Definitions

2.1 Computational Complexity

This thesis focuses on designing efficient solutions for some problems, or proving that such problems are hard (that is they cannot be solved efficiently). The mathematical model that we will assume for describing a computation is the RAM (see [79]). It seems natural to begin with the formal definition of problem.

Definition 2.1.1 (Problem). An optimization problem, or simply *problem*, consists of 3 distinct components:

- a set I of *instances*, that is the set of objects on which the problem is applied;
- a formulation of feasible *solutions* of a given instance. Such formulation must be checkable by a RAM;
- a *goal* that is the *objective function* (usually denoted by *cost* or *value*) from the set of feasible solutions to $\mathbb{R}$, and whether we want to minimize or maximize such objective function

A feasible solution that minimize (maximize) the objective function is called *optimal* solution. In Appendix A we have listed the problems we deal with in the thesis. Sometimes it is interesting even to determine if a given instance has at least one feasible solution: we will call such problems *decision* problems.

The formal description of how to solve a problem as a sequence of operations is called *algorithm*. It is possible to see an algorithm for a given problem P as a function from the instances of P to the set of the feasible

solutions (in the case of decision problems such function returns either yes or no).

As customary we will use the term *time complexity* of an algorithm, to identify the worst-case running time of an algorithm as function of the size of the instance.

A *computational class* is a set of problem sharing some common properties, such as having similar worst-case time or space complexity. An example of computational class is $\mathcal{P}$, that is the class of all problems for which an optimal solution (that is an exact solution) can be computed in polynomial time, i.e. $O(n^k)$ time for some constant k. Analogously $\mathcal{NP}$ is the class of the problems where both checking if a solution is feasible and the objective function can be computed in polynomial time. In the classical book by Garey and Johnson [44] the relevance of the computational classes $\mathcal{P}$ and $\mathcal{NP}$ is amazingly pointed out.

A fundamental notion that allows to compare the hardness of solving different problems is that of *reduction* (see Fig. 2.1):

Definition 2.1.2 (Reduction). Let P_1 and P_2 be two optimization problems, then a *reduction* from P_1 to P_2 is a pair $\langle \mathsf{f}, \mathsf{g} \rangle$ of algorithms, where f receives as input an instance x of P_1 and outputs an instance $\mathsf{f}(x)$ of P_2, and g receives as input a solution y of P_2 and outputs a solution $\mathsf{g}(y)$ such that if y is an optimal solution of $\mathsf{f}(x)$ then $\mathsf{g}(y)$ is an optimal solution of x.

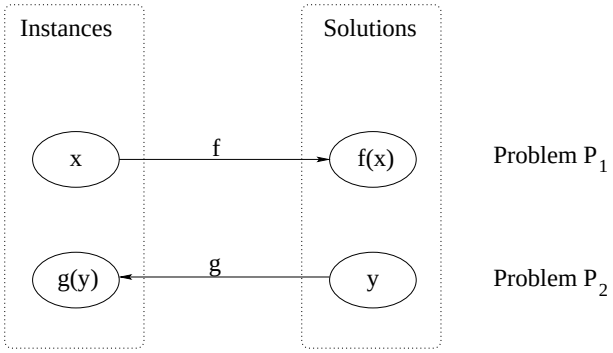

Figure 2.1: Representation of reduction

Usually the sum of the time complexities of f and g is called the time complexity of the reduction. It is immediate to note that, if P_1 is reduced to P_2 and A is the best time complexity known algorithm for P_2, then the time complexity of P_1 is at most the sum of the time complexities of A and that of the reduction.

A commonly considered conjecture is that $\mathcal{NP} \supset \mathcal{P}$, the notion of reduction allows to point out some of the evidences suggesting that such conjecture is true, more precisely it allows to identify a set of problems that are not believed to be in $\mathcal{P}$: the $\mathcal{NP}$-hard problems.

Definition 2.1.3 ($\mathcal{NP}$-hardness). Let P be a problem in $\mathcal{NP}$. Then P is $\mathcal{NP}$-*hard* if all problems in $\mathcal{NP}$ can be reduced to P in polynomial-time.

An immediate consequence of Def. 2.1.3 is that no $\mathcal{NP}$-hard problem can be solved in polynomial time, unless $\mathcal{P} = \mathcal{NP}$. In practice this means that computing an exact solution of a $\mathcal{NP}$-hard problem is considered intractable.

While it is not be tractable to solve exactly $\mathcal{NP}$-hard problems, it still may be possible to determine a near-optimal solution. We now need to introduce a formal definition of near-optimal solution.

Definition 2.1.4 (Approximation ratio). Let P be an optimization problem, let A be an algorithm for such problem, let I be an instance of P and let $\mathrm{Opt}(I)$ be the optimum value of I. Then the *approximation ratio* achieved by A on I is:

$$\max\{\frac{\mathrm{cost}(\mathsf{A}(I))}{\mathrm{Opt}(I)}, \frac{\mathrm{Opt}(I)}{\mathrm{cost}(\mathsf{A}(I))}\}$$

Please note that such ratio is always at least 1, and the lower such ratio is, the better the solution computed by the algorithm is. The *guaranteed* approximation ratio (or simply approximation ratio) of an algorithm is an upper bound on the approximation ratio over all instances. Consequently we are interested in efficient algorithms whose approximation ratio is as small as possible.

A target that is highly desirable is to describe polynomial-time algorithm with constant approximation ratio: the computational class containing exactly such problems is called $\mathcal{APX}$. Clearly not all problems in $\mathcal{NP}$are in $\mathcal{APX}$, nonetheless something that is even better than a constant approximation ratio algorithm sometimes can be described: an approximation scheme.

Definition 2.1.5 (Approximation Scheme). Given a problem P, an algorithm for P with guaranteed approximation ratio $1 + \epsilon$ and polynomial time complexity for each fixed constant $\epsilon > 0$ is called *polynomial-time approximation scheme*, or shortly *ptas*.

The typical time complexity of a ptas resembles $O(n^{1/\epsilon})$ or $O(f(\epsilon)n^k)$. The computational class that contains exactly the problems admitting a ptas is denoted by $\mathcal{PTAS}$. In Fig. 2.2 are represented the relations among the computational classes introduced so far where all inclusions are commonly

believed to be strict. By elaborating on the definition of reduction it is possible to generalize the notion of $\mathcal{NP}$-hard problem introducing the notion of "hard" problems with respect to an inclusion between computational classes. In order to formalize such notion we exploit the definition of reduction.

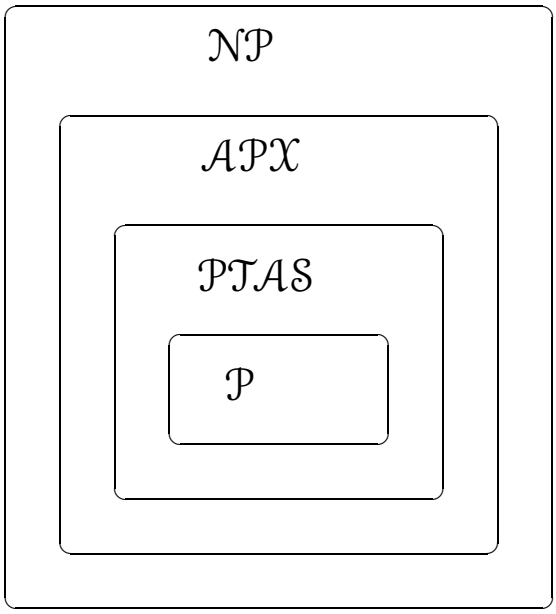

Figure 2.2: Inclusions between computational classes

Definition 2.1.6 (L-reduction). A polynomial-time reduction is an *L-reduction* if $\max\{\frac{\text{cost}(g(y))}{\text{Opt}(x)}, \frac{\text{Opt}(x)}{\text{cost}(g(y))}\} \leq \alpha \max\{\frac{\text{cost}(y)}{\text{Opt}(f(x))}, \frac{\text{Opt}(f(x))}{\text{cost}(y)}\}$, for a given constant $\alpha > 0$ and for each instance x of P_1,

Definition 2.1.7 ($\mathcal{APX}$-hardness). Let P be a problem in $\mathcal{APX}$. Then P is *$\mathcal{APX}$-hard* if all problems in $\mathcal{APX}$ can be L-reduced to P.

Just as in the case of the $\mathcal{NP} \supset \mathcal{P}$ conjecture, it is widely believed that $\mathcal{APX} \supset \mathcal{PTAS}$ (actually, thanks to a celebrated result by Arora *et.al.* [9] the latter inclusion is strict if and only if the former one is strict). Hence proving that a problem is $\mathcal{APX}$-hard is considered equivalent to proving that it does not admit a ptas.

In [8] a technique to devise polynomial-time approximation schemes has been introduced: such technique is called *smooth polynomial programming*. We will briefly recall the relevant material from that paper. A *c-smooth*

polynomial integer program (or PIP) is a problem of the form

$$\begin{aligned} &\text{minimize } p_0(x_1, \ldots, x_n) \\ &\text{subject to } l_j \leq p_i(x_1, \ldots, x_n) \leq u_j \\ &\qquad x_i \in \{0, 1\} \text{ for } i = \{1, \ldots, n\} \end{aligned} \tag{2.1}$$

where each p_j is an n-variate polynomial of maximum degree d, and each coefficient of each degree ℓ monomial (term) is at most $c \cdot n^{d-\ell}$.

The fundamental result is Theorem 1.10 of [8]:

Theorem 2.1.1. *There is a randomized polynomial-time algorithm that approximately solves smooth PIPs, in the following sense.*

- *Given a feasible c-smooth degree d PIP with n variables, objective function p_0 and K constraints, the algorithm finds a 0/1 solution z satisfying*
$$p(z_0, \ldots, z_n) \leq OPT + \delta n^d,$$
where OPT is the optimum of the PIP.

- *This solution z also satisfies each degree d' constraint to within an additive factor of $\delta n^{d'}$ for $d' > 1$, and satisfies each linear constraint to within an additive error of $O(\delta\sqrt{n \log n})$.*

- *The running time of the algorithm is $O((dKn^d)^t)$, where $t = 4c^2e^2d^2/\delta^2$ (hence $t = O(1/\delta^2)$).*

- *The algorithm can be derandomized (i.e., made deterministic), while increasing the running time by only a polynomial factor.*

2.2 Graph-Theoretic Notions

Definition 2.2.1 (Graph). Let V be a finite set and let $E \subseteq V \times V$. Then the pair $\langle V, E \rangle$ is called *graph* with vertex set V and edge set E.

Given a graph G we denote with $V(T)$ and $E(T)$ set of vertices and edges of G respectively. An edge of the form (v, v) is called *loop*, a loopless graph is called *simple* graph. Let $G = \langle V, E \rangle$ be a graph such that $(v_1, v_2) \in E$ if and only if $(v_2, v_1) \in E$ for every two vertices $v_1, v_2 \in V$, then G is an *undirected* graph, otherwise G is called *directed*. Unless stated otherwise the graphs in this thesis are assumed to be simple and undirected.

Definition 2.2.2 (Subgraph). Let $G = \langle V, E \rangle, G_1 = \langle V_1, E_1 \rangle$ be two graphs. Then G_1 is a subgraph of G if and only if $V_1 \subseteq V$ and $E_1 \subseteq E$.

Let $G = \langle V, E\rangle$ be a graph, and let $V_1 \subseteq V$, then the subgraph of G induced by V_1 is $\langle V_1, E \cap (V_1 \times V_1)\rangle$. A sequence $v_0, e_1, v_1, e_2, \ldots, e_l, v_l$ where each $v_i \in V$ and each $e_i = (v_{i-1}, v_i) \in E$, is called a *path* from v_o to v_l. The *length* of a path is the number of edges in it. A *cycle* is a path from a vertex to itself. A graph is *connected* if for every two vertices v_1, v_2 there is a path from v_1 to v_2. The *degree* of a vertex is the number of edges that are incident on such edge.

Let $G = \langle V, E\rangle$ be a graph and let $e = (v_i, w_j)$ be an edge of G, then e is *incident* on v_i and v_j, while v_i and v_j are the *endpoints* of e. Moreover we will say that v_i and v_j are adjacent. Let v be a vertex of G, then by G_{-v} we denote the subgraph of G obtained by removing from G the vertex v and all edges incident on v.

Definition 2.2.3 (Tree). A *tree* is a connected graph with no cycles.

The vertices of a tree with degree one are called *leaves*, while the other vertices are called *internal* nodes. A tree is *rooted* if there is a distinguished internal node called *root*. An unrooted tree is called *binary* if all its internal nodes have degree three.

Definition 2.2.4 (Isomorphism). Let $G_1 = \langle V_1, E_1\rangle$ and $G_2 = \langle V_2, E_2\rangle$ be two graphs. Then G_1 and G_2 are *isomorphic* if there exists a bijection $f : V_1 \to V_2$ such that $(v_i, v_j) \in E_1$ if and only if $(f(v_i), f(v_j)) \in E_2$.

Definition 2.2.5 (Homeomorphic contraction). Let $G = \langle V, E\rangle$ be a graph, let v be a vertex of G with degree 2 and let w_1, w_2 be the two vertices of G adjacent to v. Then the result of the *homeomorphic contraction* of v in G is the graph $G_1 = \langle V_1, E_1\rangle$ where $V_1 = V - \{v\}$ and $E_1 = E - \{(v, w_1), (v, w_2)\} \cup \{(w_1, w_2)\}$.

Two graphs G_1, G_2 are *homeomorphic* if a sequence of contractions of G_1 and G_2 gives the same graph.

2.3 Sequences and alignment

Let Σ be a finite set called *alphabet*. The elements of Σ are called *symbols* of *characters*.

Definition 2.3.1 (Sequence). A juxtaposition $s = s_1 s_2 \cdots s_m$ of symbols of Σ is called *sequence* over Σ. Moreover m is the *length* of s and is denoted by $|s|$.

We will use $s[i]$ to denote the i-th symbol of the sequence s. A distinguished symbol of the alphabet Σ is the space symbol and is denoted by Δ. In this thesis all sequences are assumed to be over an alphabet Σ, with $\Delta \in \Sigma$.

Definition 2.3.2 (Subsequence). Let $a = a_1, \ldots, a_n$ and $b = b_1, \ldots, b_m$ be two sequences over the alphabet Σ. Then a is a *subsequence* of b if it is possible to obtain a from b by removing some (eventually zero) symbols.

For instance `ACCTGTG` is a subsequence of `ACTCCTGCTAG`. If a is a subsequence of b then, conversely, we will say that b is a *supersequence* of a.

Let A be a set of sequences and let b be a sequence. Then b is a *common* subsequence of A if b is a subsequence of each sequence in A. Analogously we can introduce the notion of common supersequence.

Now we shift our attention to the notion of alignment of sequences, as such notion allows to compare more precisely a set of sequences, in order to find homologies, that is patterns conserved during evolution.

Definition 2.3.3 (Multiple sequence alignment). Let $S = \langle s_1, \ldots, s_n \rangle$ be an ordered set of n sequences over the alphabet Σ. Then a *multiple sequence alignment*, or shortly *msa* of S is an ordered set $\langle as_1, \ldots, as_n \rangle$ of equal length sequences, where each as_i can be obtained from s_i by inserting some spaces.

The sequences $\langle as_1, \ldots, as_n \rangle$ are called *aligned* sequences. Sometimes given a set $\langle s_1, \ldots, s_n \rangle$ of sequences we will describe an alignment of such set as a matrix A of n rows, where in each cell there is an element of Σ and in the i-th row there is the sequence as_i.

Given two sequences s_1 and s_2 in the alignment, then each symbol $as_1[i]$ is *opposite* to $as_2[i]$. By abuse of language, we will write that the $s_1[i]$ is opposite to $s_2[j]$ under an alignment A, actually meaning that the corresponding symbols in at_1, at_2 are opposite. A *match* occurs where two identical symbols are opposite in the two sequences as_1 and as_2, otherwise two non-identical opposing symbols give a *mismatch* which can be thought of as a replacement. The insertion of a space in a sequence opposing a symbol σ of a second sequence, is viewed as the deletion in the first sequence of the symbol σ or an insertion of σ into the second one.

Definition 2.3.4 (Score). A *score* is a function $d : (|\Sigma \cup \{\Delta\}|) \times (|\Sigma \cup \{\Delta\}|) \to \mathbb{N}$ that assigns a cost to each pair of symbols.

Following the definition, a score can be intuitively described by a matrix. The following mathematical properties characterize some interesting subsets of score matrices [27]:

(i) $d(a,a) = 0$, for every $a \in \Sigma \cup \{\Delta\}$,

(ii) $d(a,b) = 0$ implies that $a = b$, for every $a, b \in \Sigma \cup \{\Delta\}$,

(iii) $d(a,b) = d(b,a)$, for every $a, b \in \Sigma \cup \{\Delta\}$,

(iv) $d(a,c) \leq d(a,b) + d(b,c)$, for every $a, b, c \in \Sigma \cup \{\Delta\}$,

(v) $d(a,c) \leq \max\{d(a,b), d(b,c)\}$, for every $a, b, c \in \Sigma \cup \{\Delta\}$.

A score scheme that satisfies properties (i) – (iii) is a semi-metric, the score is a metric if property (iv) is also satisfied and is an ultrametric if all above specified properties hold. By means of a score scheme a value is assigned to a multiple alignment. A very popular score scheme, called SP-score, is defined as follows:

Definition 2.3.5 (SP-score). Let A be an alignment with m rows and k columns, and let s_i, s_j be respectively the ith and jth rows of A. Then the cost of the pairwise alignment in A of s_i and s_j, denoted as $d_{\mathsf{A}}(s_i, s_j)$ is $\sum_{1 \leq l \leq k} d(s_i[l], s_j[l])$, where $s_i[l]$ ($s_j[l]$) is the l-th symbols of s_i (s_j respectively). The *SP-score* (or cost) of A is defined as the following summation

$$\sum_{1 \leq i < j \leq m} d_{\mathsf{A}}(s_i, s_j) = \sum_{1 \leq i < j \leq m} d(as_i, as_j)$$

Alternatively we may think that the cost of a multiple alignment is the sum of the scores of all columns, where the score of each column is the sum of the scores of all distinct unordered pairs of symbols in the column. Then, the value of the alignment of a column x of height l is $\sum_{1 \leq i < j \leq l} d(x[i], x[j])$, where $x[i]$ is the symbol in i-th row of column x and $d(x[i], x[j])$ is the score between the two symbols $x[i]$ and $x[j]$. We assume that an alignment cannot contain a column of only Δ's.

Sometimes it is useful, from a biological point of view, to penalize alignments that present a large number of *gaps*, that is consecutive runs of space symbols. Let g be a constant called *gap opening penalty*, then we modify the definition of cost of an alignment as follows:

Definition 2.3.6. Let $as_1 = \langle as_1[1], \ldots, as_1[m] \rangle, as_2 = \langle as_2[1], \ldots, as_2[m] \rangle$ be the alignment, of length m, of two sequences s_1, s_2, then the cost of the alignment is $d(s_1, s_2) = g(G_1 + G_2) + \sum_{i=1}^{m} d_M(as_1[i], as_2[i])$, where G_j is the number of gaps in s_j.

A *space-L alignment* A of a set $\langle s_1, \ldots, s_k \rangle$ of sequences is an alignment $\langle as_1, \ldots, as_k \rangle$ of $\langle s_1, \ldots, s_k \rangle$ such that $|as_i| \leq |s_i| + L$ for each sequence s_i.

Note that space-L-alignments exist only if the length of the shortest of these sequences is at least $n - L$, where n is the length of the longest among the sequences $s_1, \ldots, s_k$. Please also note that there are no restriction about where the space symbols can be inserted.

Let B be a subset of a set S of sequences and A an alignment of S. Then, by A_B we denote the array consisting of all rows of A containing sequences in B (in this case in A_B there may be some columns containing only Δ's).

Definition 2.3.7 (Alignment graph). An *alignment graph* $G = \langle V, E, F \rangle$ for a set S of sequences is a graph whose vertices V correspond to the characters of the sequences in S, F is a set of directed edges (called *special* edges) such that $(v, w) \in F$ if and only if v immediately precedes w in a sequence of S. Moreover E is a set of weighted undirected edges (called *alignment* edges) such that two vertices incident on an alignment edge cannot belong to the same sequence in S.

It is possible to describe the notion of alignment graph purely with graph-theoretic notions, without referring to sequences. In this case let V be a finite set and let $\prec$ be a partial order over V. By $\prec^*$ we denote the reflexive transitive closure of $\prec$. Then a graph $G = \langle V, E, F \rangle$ is an alignment graph for $V, \prec$ if the following conditions hold for each pair of vertices v, w:

- the sets $\{z \in V : z \prec^* v\}$ and $\{z \in V : z \prec^* w\}$ are disjoint or one of them is contained into the other one;
- $(v, w) \in F$ if and only if $v \prec w$;
- $(v, w) \in E$ implies that $v \not\prec^* w$.

Moreover a weight function $w : E \to \mathbb{Q}^+$ is given as part of the alignment graph. Whenever all edges have weight one, we will not explicitly state the weight function.

For consistency with the terminology used in [33] we will call *special* the cycles intersecting at least an oriented edge.

Definition 2.3.8. A multiple sequence *trace alignment* of an alignment graph $G = \langle V, E, F \rangle$ is a graph $G_1 = \langle V, E_1, F \rangle$ such that $E_1 \subseteq E$ and there is no cycle including any special edge.

2.4 Phylogenies

An interesting field of Computational Biology is phylogenetics, whose aim is to determine the evolutionary history of a set of species. Such history

is usually represented by means of a tree. In this section we will introduce some conventions and notions that will be used throughout this thesis: we will denote a set of species by S, and the cardinality of S by n.

Definition 2.4.1. Let S be a set of species, then an *evolutionary tree* or *phylogeny* over S is a tree T whose leaves are exactly the members of S.

Given a tree T with leaf set S, a *quartet* from S, or equivalently a quartet of T, is any subset of S of 4 elements. A *quartet topology* is a partition of a quartet into two subsets of two elements each, written in the form $ab|cd$. The quartet topology or quartet resolution *induced* by a quartet $\{a, b, c, d\}$ in T is $ab|cd$ if and only if in T a is closer to b than to c or d. , the set of quartets of T is the set In the following we will denote by Q_T the set of quartet topologies induced in T together with their resolutions, it is not hard to see that it is possible to reconstruct T given the set Q_T of all the quartet topologies induced in T.

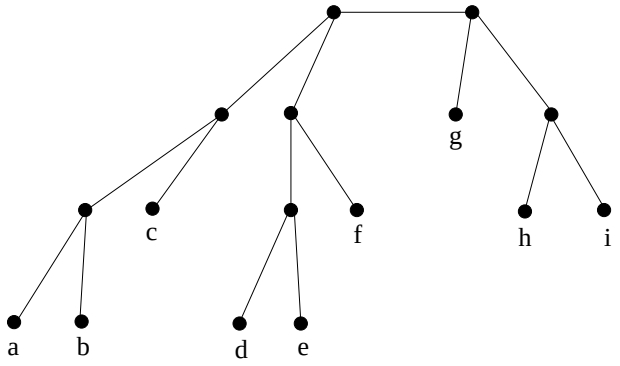

Figure 2.3: Example of evolutionary tree

Let T be an evolutionary tree over S, then each internal node v induces a *tripartition* (A_v, B_v, C_v) of S where T_{-v} consists of three trees whose set of leaves are A_v, B_v, C_v. By $Q(A_v, B_v, C_v)$ (or $Q_T(v)$) we denote the set of quartets $\{a, b, c, d\}$ such that $a \in A_v, b \in B_v, c \in C_v$: each such quartet is called *across the vertex* v. Similarly given an edge e of an evolutionary tree T we will say that e induces the bipartition (A_e, B_e) where removing the edge e from T gives two trees with leaves A_e and B_e respectively.

Definition 2.4.2 (Induced partitions). Let T be a phylogeny. Then the set of all *bipartitions induced* in T is $\cup_{e \in E(T)}(A_e, B_e)$, while the set of all *tripartitions induced* in T is $\cup_{v \in V(T)}(A_v, B_v, C_v)$.

A set of bipartitions (or tripartition) is called *compatible* if there exists a phylogeny inducing such set of partitions (please note that the phylogeny may induce some partition not in the set). In [25] it has been shown how to compute efficiently the tree inducing a given set of compatible bipartitions. Such result can be immediately extended to the case where a set of tripartitions is given as input, instead of bipartitions.

A quartet $\{a, b, c, d\}$ is across the edge e (or across $\langle A_e, B_e \rangle$) if and only if $|A_e \cap \{a, b, c, d\}| = 2$. In Fig. 2.4 two quartets regarding an evolutionary tree are represented.

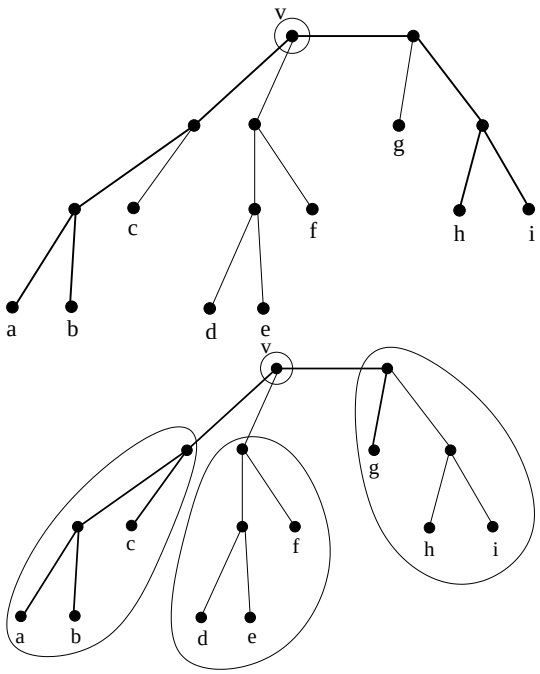

Figure 2.4: Quartet (a, b, h, i) is across v and (a, b, c, g) is not across v

Let Q be a complete set of quartet topologies over S (that is the resolutions for all possible quartets over S) and let T be a phylogeny over S. A quartet $\{a, b, c, d\}$ over S is a *quartet error* for T if its resolution in Q is different from that in Q_T. The number of quartets across a vertex v is $\frac{1}{2}(|S| - 3)|A_v||B_v||C_v|$, clearly this is also an upper bounds for the number of quartet errors across a vertex.

Let T be a tree and let a, b be two nodes of T, then we will denote by $d_T(a, b)$ the distance between a and b in T, that is the number of edges in the unique simple path from a to b in T. Let T be a rooted tree, and let t be a node of T, then the depth of t in T is the distance of t from the root of

T. The depth of a tree T, denoted by $\text{depth}(T)$, is the maximum among the depths of its nodes. Given two leaves a, b of T we define the *least common ancestor*, of a and b in T, denoted by $\text{lca}_T(a, b)$, as the maximum depth node of T which is ancestor of both a and b.

Chapter 3

Comparing Sequences: Hardness of Aligning

3.1 Introduction

Multiple sequence alignment is one of the most popular and important problems in Computational Biology [68]. It finds different applications in Molecular Biology, mainly in two related areas: finding information about the secondary structure of the molecules, such as residue conservation along sequences, and estimate the evolutionary distance between species from their associated sequences.

The similarity of sequences in the alignment is measured by using different scores or *distances* between elements of the matrix. A popular (and mathematically sound) assumption in Molecular Evolution is to measure evolutionary distance by means of a molecular clock, that is the number of mutations is roughly proportional to the time: this justifies to consider only metric scores, that is scores where the distance between identical letters is zero and it satisfies the triangle inequality. Among different score schemes, the *sum of all pairs* score, in short the SP-score (Def. 2.3.5), is the one that has received more attention. By means of the SP-score a value is assigned to a multiple alignment; an *optimal alignment* is the one that minimizes the value over all possible alignments.

Several methods have been developed for multiple sequence alignment [27, 26], but no efficient methods are known to find the optimal alignment. Recently, a polynomial time approximation algorithm for the problem has been proposed by Gusfield [47] who achieved a $2 - 2/k$ approximation factor by assembling an alignment of k sequences from optimal alignment of pairs of sequences. The approximation ratio has been improved to a $2 - l/k$ factor,

for any fixed l, by Bafna, Lawler and Pevzner [10]. But, besides these results it was an open question whether the problem is $\mathcal{NP}$-complete. The computational complexity of multiple sequence alignment has been investigated in [97] where is given a simple proof of $\mathcal{NP}$-completeness of the alignment with score scheme over a fixed alphabet of four letters that satisfies the triangle inequality, and assigns a non zero distance between identical letters. But, this result leaves open the problem of analyzing the complexity of computing optimal SP-score multiple sequence alignments for instances of this problem which are of practical biological relevance. Mainly, the result in [97] does not consider an important requirement for score schemes ([41, 98]) which is the property of *metricity*: this one implies a zero distance between identical letters.

Here, we prove the intractability of multiple sequence alignment in the very restricted case in which sequences are over a binary alphabet and the score is a metric. The significance of the intractability in this case is that it establishes the $\mathcal{NP}$-completeness for all cases encountered in practice, as well as for general instances of the alignment problem in which $|\Sigma| > 2$ and the distance verifies specific properties. Then we strengthen such results by showing that the problem is $\mathcal{APX}$-hard if the cost of a gap (that is a substring of spaces inserted in an alignment) is fixed, and the number of spaces that can be inserted is bounded by a constant.

3.2 Preliminaries

Various notions of cost of an alignment have been proposed, in this chapter we will focus mainly on the SP (sum of pairs) formulation. This means that the cost of an alignment is the sum of the costs of all pairwise alignments. By cost(A) we will denote the cost of an alignment A of a set of sequences, and by $\mathsf{A}[i]$, we denote the column of A of index i. Let B and C be two disjoint subsets of sequences of S, and let $B(i)$ and $C(i)$ be the i-th sequence in B and C, respectively, then by cost($\mathsf{A}_{B,C}$) we denote $\sum_{i,j} d_{\mathsf{A}}(B(i), C(j))$.

We prove that multiple sequence alignment is $\mathcal{NP}$-complete over a fixed score scheme that is a metric, by using a reduction from the Vertex Cover problem (VC, Problem 10) which is $\mathcal{NP}$-complete [44].

The decision versions of the problems Vertex Cover and Multiple Sequence Alignment are defined in the following, the reader can find the optimization versions in Appendix A.

Problem 1 (Vertex Cover).
Instance: an unoriented graph $G = \langle V, E \rangle$ and an integer k.
Solution: a cover of G of k vertices, that is a subset $C \subseteq V$ such that C

contains at least one of the endpoints of each edge $e \in E$ and $|C| = k$.
Goal: to minimize the cardinality of the cover.

Problem 2 (Multiple Sequence Alignment).
Instance: A set $\mathcal{S} = \{s_1, \cdots, s_n\}$ of finite sequences over a fixed alphabet Σ, a SP-score d and an integer c.
Solution: a multiple alignment of the sequences in $\mathcal{S}$ of cost c or less.
Goal: to minimize the cost of the alignment.

A variant of Multiple Sequence Alignment is when the cost of a gap (that is a maximal substring containing spaces) is fixed to a constant: such problem is called Fixed-Gap Multiple Sequence Alignment. We will also investigate the computational complexity of the Fixed-Gap Multiple Sequence Alignment problem when the number of spaces that can be inserted in a sequence is bounded, showing that the problem is $\mathcal{APX}$-hard. The last section of this chapter deals with a different formulation of MSA called Maximum Weight Trace Alignment: we prove here that the problem is $\mathcal{APX}$-hard.

3.3 Multiple Sequence Alignment Over Alphabet of Size 6

We first describe a reduction from the Vertex Cover problem on graphs [44] to Multiple sequence alignment over an alphabet of size 6. Then, we generalize the idea of this reduction to the case of a SP-score over a binary alphabet.

Now we are able to prove some technical results that will be used in our $\mathcal{NP}$-hardness proofs.

Lemma 3.3.1. *Let s_1, s_2 be two sequences over Σ such that $l_1 = |s_1|$, $l_2 = |s_2|$, $l_2 \geq l_1$ and there are m symbols of s_1 that are not in s_2. Then any alignment of the set $\{s_1, s_2\}$ has at least $m + l_2 - l_1$ mismatches.*

The following two properties hold for every alignment over a score which is a metric and has non null values greater or equal to 1.

Corollary 3.3.2. *Let s_1, s_2 be two sequences over Σ, such that $l_1 = |s_1|$, $l_2 = |s_2|$, $l_2 \geq l_1$ and there are m symbols of s_1 that are not in s_2. Then for any alignment of the set $\{s_1, s_2\}$, $\text{cost}(\mathsf{A}_{\{s_1,s_2\}}) \geq m + l_2 - l_1$.*

Proof. It follows from Lemma 3.3.1. □

Lemma 3.3.3. *Let U be a subset of a set S of sequences over Σ such that U contains only identical sequences, and let A be an optimal alignment of S. Then* $\text{cost}(\mathsf{A}_U) = 0$.

Proof. Assume to the contrary that A is an optimal alignment of S and that $\text{cost}(\mathsf{A}_U) > 0$. Let $u \in U$ be the sequence that minimizes the value $\text{cost}(\mathsf{A}_{\{u\},S-U})$. Then, let A_1 be the alignment obtained from A by assuming that all sequences in $S - U$ are aligned as in A (i.e. $\mathsf{A}_{S-U} = \mathsf{A}_{1S-U}$), while all sequences in U are aligned identically to the alignment of u in A. Since $\text{cost}(\mathsf{A}_{S-U}) = \text{cost}(\mathsf{A}_{1S-U})$ and $\text{cost}(\mathsf{A}_{1U}) < \text{cost}(\mathsf{A}_U)$, it follows that $\text{cost}(\mathsf{A}_1) < \text{cost}(\mathsf{A})$, which is a contradiction. □

The SP-score for multiple alignment over alphabet $\Sigma = \{a, b, c, d, e, f\}$ is the one described in the following Table 3.1.

	a	b	e	f	c	d	Δ
a	0	1	1	1	1	2	2
b	1	1	2	0	1	1	2
c	1	2	0	2	2	2	1
d	2	1	2	1	2	0	2
e	1	2	2	1	0	2	2
f	1	0	2	1	2	1	2
Δ	2	2	1	2	2	2	0

Table 3.1: SP-score for alphabet of size 6

The transformation from VC to MSA consists of constructing a set S of sequences encoding the graph G and a value c, depending on k and on the number m of edges of G, such that c is an upper bound for the value of an optimal alignment of S if and only if k is the size of a vertex cover for G.

Let $G = \langle V, E\rangle$ be a graph with $V = \{v_1, \ldots, v_n\}$ and $E = \{e_1, \ldots, e_m\}$. Now we construct an encoding for the edges of the graph that gives the set of sequences which is instance of the alignment problem.

Given an edge $e = (v_i, v_j)$, where we assume that $i < j$, we construct an encoding of such an edge with a sequence, called *edge sequence* constructed as follows:

$$s(i, j) = a^{3i} b\, a^{3(j-i)-2} b\, a^{3(n-j)+3}.$$

Note that for each edge (v_i, v_j) the edge sequence $s(i, j)$ has length $3n+3$. Moreover, we construct a *template sequence* t of length $3n + 4$:

$$t = c(eef)^n eec$$

We also construct the *test sequence* $x(k)$ of the form:

$$x(k) = cd^k c$$

Note that the test sequence depends on k. The set of sequences that is instance of the alignment problem associated to the instance (G, k) of the VC problem, is the set $S = Y \cup T \cup X$, where $Y = \{s(i,j) : (v_i, v_j) \in E\}$, T contains K_2 sequences t and X contains K_1 sequences $x(k)$. In Fig. 3.1 is represented an alignment of the encoding of a graph G.

Definition 3.3.1 (Standard alignment). Let A be an alignment of $\mathcal{S}$. Then A is a *standard alignment* if it satisfies the following properties:

(i) there are no Δs in A_T;

(ii) all Δ's in A_S are aligned with cs of A_T;

(iii) all d's of A_X are aligned with fs of A_T;

(iv) all c's of A_X are aligned with cs of A_T;

(v) no column of A_X contains both Δs and ds.

The main idea on which the encoding of S is based, is that an optimal alignment A of S is obtained when A is a standard alignment.

In the following we will prove a fundamental property of standard alignments, that is their values are bounded by a given value c only when G has a vertex cover of a given size k. This fact is obtained by forcing ds of the test sequences to be opposite to bs of the edge-sequences. By construction, only one b of each edge sequence can be opposite to a d, and the number of such bs determines the value of the alignment. If the total number of bs opposite to ds is equal to the number of edges, which is possible only if there are k vertices which cover one end of each edge sequence, then $\text{cost}(\mathsf{A}) < c$, otherwise $\text{cost}(\mathsf{A}) > c$.

It follows easily that each standard alignment has exactly $r = 3n + 4$ columns. Note that in Fig. 3.1 is represented a standard alignment of S where, for simplicity, all Δs are not shown.

In the following we give some useful properties of standard alignments:

Proposition 3.3.4. *Let A be a standard alignment of S. Then for each edge sequence encoding the edge (v_i, v_j), the b encoding one end vertex v_h, for $h \in \{i, j\}$, is opposite to the h-th f of each template sequence, while the other b is opposite to es of the template sequences.*

c	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	c	Template
c	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	c	sequences
c	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	c	
c	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	c	
c			d												d			d			c	Test
c			d												d			d			c	sequences
a	a	a	b_1	a	b_1	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a		(v_1, v_2)
a	a	a	b_2	a	a	a	a	b_1	a	a	a	a	a	a	a	a	a	a	a	a		(v_1, v_3)
a	a	a	b_3	a	a	a	a	a	a	a	b_1	a	a	a	a	a	a	a	a	a		(v_1, v_4)
	a	a	a	a	a	a	b_2	a	a	a	a	a	a	a	b_1	a	a	a	a	a	a	(v_2, v_5)
	a	a	a	a	a	a	b_3	a	a	a	a	a	a	a	a	a	a	b_1	a	a	a	(v_2, v_6)
	a	a	a	a	a	a	a	a	a	b_2	a	a	a	a	b_2	a	a	a	a	a	a	(v_3, v_5)
	a	a	a	a	a	a	a	a	a	b_3	a	a	a	a	a	a	a	b_2	a	a	a	(v_3, v_6)
	a	a	a	a	a	a	a	a	a	a	a	a	b_2	a	b_3	a	a	a	a	a	a	(v_4, v_5)
	a	a	a	a	a	a	a	a	a	a	a	a	b_3	a	a	a	a	b_3	a	a	a	(v_4, v_6)

Figure 3.1: An example of the encoding of a graph

Lemma 3.3.5. *Let* A *be an optimal alignment of* S *and let* A_1 *be a standard alignment of* S*. Then* $\text{cost}(\mathsf{A}_{X,T})$ *is minimum over all possible alignments of* S *if and only if* $\text{cost}(\mathsf{A}_{X,T}) = \text{cost}(\mathsf{A}_{1X,T})$*, otherwise* $\text{cost}(\mathsf{A}_{X,T}) \geq \text{cost}(\mathsf{A}_{1X,T}) + K_2$*.*

Proof. Assume that $\text{cost}(\mathsf{A}_{X,T})$ is minimum over all possible alignments of S. Then, $\mathsf{A}_{\{x,t\}}$ contains exactly $r-2$ mismatches of value 1, for every $x \in X$ and $t \in T$. In fact, since the mismatches (d, f) have a value 1, while the mismatches (d, e), (d, Δ), (d, c) all have value 2 it is advantageous to align all ds of x with fs of t. By the SP-score $d(c, c) = 0$, $d(c, \Delta) = 1$, $d(c, e) = d(c, f) = d(\Delta, e) = d(\Delta, f) = 2$, it follows that it is advantageous to align the cs of x with the cs of t. Note that any other alignment of $\{x, t\}$ cannot be optimal. It is immediate to verify that $\text{cost}(\mathsf{A}_{X,T}) \geq \text{cost}(\mathsf{A}_{1X,T})$.

Now, assume that $\text{cost}(\mathsf{A}_{X,T}) \neq \text{cost}(\mathsf{A}_{1X,T})$. Since for every sequence $x \in X$ and $t \in T$, $\mathsf{A}_{1\{x,t\}}$ contains exactly $r-2$ mismatches of value 1, then there is a sequence x_1 such that $\mathsf{A}_{\{x_1,t\}}$ must contain at least $r-1$ mismatches. Since, by Lemma 3.3.3, $\text{cost}(\mathsf{A}_X) = 0$, $\text{cost}(\mathsf{A}_T) = 0$, and $|T| = K_2$, it follows that $\text{cost}(\mathsf{A}_{X,T}) = K_2|X|\,\text{cost}(\mathsf{A}_{\{x_1,t\}})$. Consequently, since $\text{cost}(\mathsf{A}_{1X,T}) = K_2|X|\,\text{cost}(\mathsf{A}_{1\{x_1,t\}})$, it follows that $\text{cost}(\mathsf{A}_{X,T}) \geq \text{cost}(\mathsf{A}_{1X,T}) + K_2$, as required. □

Lemma 3.3.6. *Let* A *be an optimal alignment of* S *and let* A_1 *be a standard alignment of* S*. Then* $D(\mathsf{A}_{S,T})$ *is minimum over all possible alignments of* $\mathbb{S}$ *if and only if* $\text{cost}(\mathsf{A}_{S,T}) = \text{cost}(\mathsf{A}_{1S,T})$*, otherwise* $\text{cost}(\mathsf{A}_{S,T}) \geq \text{cost}(\mathsf{A}_{1S,T}) + K_2$*.*

Proof. Assume that $\text{cost}(\mathsf{A}_{S,T})$ is minimum over all possible alignments of S. Then, $\mathsf{A}_{\{s,t\}}$ contains exactly r mismatches of value 1, for every $s \in S$ and $t \in T$. In fact, by Corollary 3.3.2, since there is no symbol common to both sequences s and t, and $|t| = r$, $|s| = r - 1$ it follows that every alignment A' for S is such that $\text{cost}(\mathsf{A}'_{\{s,t\}}) \geq r$. By construction of standard alignment and by the SP-score, it follows easily that $\text{cost}(\mathsf{A}_{1\{s,t\}}) = r$.

Now, assume that $\text{cost}(\mathsf{A}_{S,T}) \neq \text{cost}(\mathsf{A}_{1S,T})$. Then, there is a sequence $s_1 \in S$ such that $\mathsf{A}_{\{s_1,t\}}$ must contain either at least $r+1$ mismatches or r mismatches one of which is of value 2. Since, by Lemma 3.3.3, $\text{cost}(\mathsf{A}_T) = 0$, and $|T| = K_2$, it follows that $\text{cost}(\mathsf{A}_{S,T}) = K_2\,\text{cost}(\mathsf{A}_{\{s_1,t\}}) + \text{cost}(\mathsf{A}_{S-\{s_1\},T})$. But, $\text{cost}(\mathsf{A}_{1S,T}) = K_2\,\text{cost}(\mathsf{A}_{1\{s_1,t\}}) + \text{cost}(\mathsf{A}_{1S-\{s_1\},T})$. It follows that $\text{cost}(\mathsf{A}_{S,T}) \geq \text{cost}(\mathsf{A}_{1S,T}) + K_2$, as required. □

Lemma 3.3.7. *Let* A *be a standard alignment of* S*. Then all values* $\text{cost}(\mathsf{A}_X)$, $\text{cost}(\mathsf{A}_T)$, $\text{cost}(\mathsf{A}_{X,T})$ *and* $\text{cost}(\mathcal{A}_{S,T})$ *are fixed and minimum over all possible alignments.*

Proof. By definition of standard alignment and by Lemmata 3.3.5, 3.3.6, the proof is immediate. □

In the following by cost_{SD} we denote the sum $\text{cost}(\mathsf{A}_X) + \text{cost}(\mathsf{A}_T) + \text{cost}(\mathsf{A}_{X,T}) + \text{cost}(\mathsf{A}_{S,T})$ over a standard alignment A of S.

Lemma 3.3.8. *Let* A *be a standard alignment of* S*. Then* $\text{cost}(\mathsf{A}_S) < 8l^2r$ *and* $\text{cost}(\mathsf{A}_{S,X}) < 4K_1lr$.

Proof. It follows easily from the SP-score and the fact that a standard alignment consists of r columns. □

We will denote such upper bounds for $\text{cost}(\mathsf{A}_S)$ and $\text{cost}(\mathsf{A}_{S,X})$ with U_Y and $U_{Y,X}$ respectively. By Lemma 3.3.7 and Lemma 3.3.8 it follows easily that each standard alignment (hence each optimal alignment) has value not greater than $\text{cost}_{SD} + U_Y + U_{Y,X}$. We will assume that $K_1 > U_Y$ and $K_2 > U_Y + U_{Y,X}$.

In the following we will prove that an optimal alignment must be a standard one.

Lemma 3.3.9. *Let* A *be an optimal alignment of* S*. Then* A *must be a standard alignment.*

Proof. Let A_1 be a standard alignment of S. If A does not satisfy one of the properties (i) – (iv) of standard alignment, it is immediate to verify that $\text{cost}(\mathsf{A}_{X,T}) \neq \text{cost}(\mathsf{A}_{1X,T})$ or $\text{cost}(\mathsf{A}_{S,T}) \neq \text{cost}(\mathsf{A}_{1S,T})$. By Lemmata 3.3.5,

3.3.6, it follows that $\text{cost}(\mathsf{A}_{X,T})+\text{cost}(\mathsf{A}_{S,T}) \geq \text{cost}(\mathsf{A}_{1X,T})+\text{cost}(\mathsf{A}_{1S,T})+K_2$. Since $K_2 > U_Y + U_{Y,X}$, by Lemma 3.3.7 if follows that A is not optimal.

Assume now that A does not satisfy property (v) of standard alignment. Then by Lemma 3.3.3, A cannot be optimal. Consequently, A must be a standard alignment. □

We are now able to prove that the multiple alignment problem is $\mathcal{NP}$-complete with a fixed SP-score that is a metric and with an alphabet of six symbols. In the following, if A is an alignment of S, by $n_\sigma(i)$ we denote the number of σ's occurring in the column of index i of A.

Theorem 3.3.10. *Let (G, k) be an instance of the* VC *problem and let S be the encoding of such instance. Then:*

(i) *if G has a vertex cover of size k, then there exists a standard alignment A of S such that $\text{cost}(\mathsf{A}_{S,X}) \leq K_1(l + 2l(r-2))$;*

(ii) *if G has a minimum vertex cover of size $k_1 > k$, then for each standard alignment A of S it holds that $\text{cost}(\mathsf{A}_{S,X}) > U_Y + K_1(l + 2l(r-2))$.*

Proof. Let A be a standard alignment of S, and let I be the set of indices of the columns of A containing some ds. By the definition of standard alignment the value $\text{cost}(\mathcal{A}_{S,X})$ can be computed as the sum of the value of the column of index 1 and r and the value of all other columns of $\mathsf{A}_{S\cup X}$. Then, $\text{cost}(\mathsf{A}_{S,X}) = K_1(ld(c,\Delta)+ld(c,a))+K_1(\sum_{i\in I}(d(d,b)n_b(i)+d(d,a)(l-n_b(i)))$ $+\sum_{i\notin I\cup\{1,r\}}(d(\Delta,b)n_b(i)+d(\Delta,a)(l-n_b(i)))) = K_1(2l+2l(r-2)-\sum_{i\in I} n_b(i))$.

Let us assume that G has a vertex cover C of size k, then we will construct a standard alignment A such that $\text{cost}(\mathsf{A}_{S,X}) \leq K_1(l + 2l(r-2))$. From the vertex cover C we construct the set K_1 consisting of the indices of the columns in A_T that contain the fs encoding the vertices in C. Since C is a vertex cover of G each edge (v_i, v_j) has at least an end vertex v_h in C, for $h \in \{i, j\}$, moreover it is possible to align, in each edge sequence, the b encoding the vertex v_h with the h-th 1 of each template sequence. The alignment of the test sequences in $\mathsf{A}_{S\cup X}$ is obtained by aligning the ds exactly in the columns whose index is in K_1. By Proposition 3.3.4, since only a b for each edge sequence can be aligned with a f of each template sequence. It follows that $\sum_{i\in I} n_b(i) = l$. Substituting this value in the above relation for $\text{cost}(\mathsf{A}_{S,X})$, then (i) easily follows.

Let us assume that G has a vertex cover of minimum size $k_1 > k$. By Proposition 3.3.4 for each edge sequence encoding the edge (v_i, v_j), one b of each edge sequence, encoding the end vertex v_h, for $h \in \{i, j\}$, is aligned with the h-th f of each template sequence, hence there must be at least k_1 columns of A that contain some f's of the template sequences and at least a b of the

	a	b	Δ
a	0	1	2
b	1	0	1
Δ	2	1	0

Table 3.2: SP-score for binary alphabet

edge sequences. By properties (i), (iii) and (v) of standard alignment, in all test sequence each d is aligned with distinct f's of the template sequences: it follows that there is at least one edge sequence such that both b's are in columns of $\mathsf{A}_{S\cup X}$ that do not contain any d's. Consequently, given I the set of indices of the columns of A that contain some d's of the test sequences, $\sum_{i\in I} n_b(i) \leq l-1$, hence $\text{cost}(\mathsf{A}_{S,X}) \geq K_1(l+2l(r-2))$. Since $K_1 > U_Y$ we obtain that $\text{cost}(\mathsf{A}_{S,X}) > U_Y + K_1(l+2l(r-2)+1)$, which proves (ii). □

Corollary 3.3.11. *The graph G has a vertex cover of size k if and only if the set S has an optimal alignment A of value* $\text{cost}(\mathsf{A}) < \text{cost}_{SD} + U_Y + K_1(l+2l(r-2))$.

Proof. Let A be an optimal alignment of S. By Lemma 3.3.9, A is a standard alignment. Then $\text{cost}(\mathsf{A}) = \text{cost}_{SD} + \text{cost}(\mathsf{A}_S) + \text{cost}(\mathsf{A}_{S,X})$. By Theorem 3.3.10, if G has a vertex cover of size k, then $\text{cost}(\mathsf{A}_{S,X}) \leq K_1(l+2l(r-2))$.

Assume now that G has a minimum vertex cover of size $k_1 > k$: by Theorem 3.3.10, $\text{cost}(\mathcal{A}_{S,X}) > U_Y + K_1(l+2l(r-2))$. Consequently, $\text{cost}(\mathsf{A}) > \text{cost}_{SD} + U_Y + K_1(l+2l(r-2))$, which proves what required. □

3.4 Multiple Alignment Over Binary Alphabet

In this section, we show that the MULTIPLE SEQUENCE ALIGNMENT problem is $\mathcal{NP}$-complete even if the sequences are over a binary alphabet and the score scheme, which is a metric, is given in Table 3.2:

The proof consists of showing a reduction from VERTEX COVER to MULTIPLE SEQUENCE ALIGNMENT. The technique used to build the reduction is similar to the one illustrated in the case of $|\Sigma| = 6$. Given (G,k) the VC instance, where G is the graph $\langle V, E\rangle$, with $V = \{v_1, \cdots, v_n\}$ and $E = \{e_1, \cdots, e_m\}$, while $1 \leq k \leq n$, we construct the following sequences over alphabet $\Sigma = \{a, b\}$:

the *edge sequence* $s(i,j)$ of length $3(n+1)$, for each edge (v_i, v_j),

$$s(i,j) = a^{3i}ba^{3(j-i)-2}ba^{3(n+1-j)}$$

the *template sequence* t of length $3(n+1)+1$,

$$t = b((a^2)b)^n(a^2)b$$

the *fixed sequence* q of length $3(n+1)+1$,

$$q = ba^{3n+2}b,$$

the *test sequence* $x(k)$, given k the integer of the VC instance,

$$x(k) = a^{3n+2-k}.$$

Then, let Y be the set $\{s(i,j) : (v_i, v_j) \in E\}$ of all possible edge sequences, T the set of K_1 template sequences t, Q the set of K_2 fixed sequences q and X the set of K_3 test sequences $x(k)$. The constants K_1, K_2 and K_3 are related to the number of edges, and will be fixed later in the section.

Finally, the sequences in $Y \cup T \cup Q \cup X$ give the set S that is instance of the alignment problem.

b	a	a	b	a	a	b	a	a	b	a	a	b	a	a	b	a	a	b	a	a	b	a	a	b
b	a	a	b	a	a	b	a	a	b	a	a	b	a	a	b	a	a	b	a	a	b	a	a	b
b	a	a	b	a	a	b	a	a	b	a	a	b	a	a	b	a	a	b	a	a	b	a	a	b
b	a	a	b	a	a	b	a	a	b	a	a	b	a	a	b	a	a	b	a	a	b	a	a	b
b	a	a	b	a	a	b	a	a	b	a	a	b	a	a	b	a	a	b	a	a	b	a	a	b
b	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	b
b	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	b
	a	a		a	a		a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	
	a	a		a	a		a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	
a	a	a	b	a	a	a	a	a	a	a	b	a	a	a	a	a	a	a	a	a	a	a	a	
a	a	a	b	a	a	a	a	b	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	
a	a	a	b	a	a	a	a	a	a	a	a	a	a	b	a	a	a	a	a	a	a	a	a	
	a	a	a	b	a	b	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
a	a	a	a	a	a	b	a	b	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	
a	a	a	a	a	a	b	a	a	a	a	a	a	a	a	a	a	b	a	a	a	a	a	a	
a	a	a	a	a	a	b	a	a	a	a	a	a	a	a	a	a	a	a	b	a	a	a		
a	a	a	b	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	b	a	a	a		

Template sequences

Fixed sequences

Test sequences

(v_1, v_4)
(v_1, v_3)
(v_1, v_5)
(v_1, v_2)
(v_2, v_3)
(v_2, v_6)
(v_2, v_7)
(v_1, v_7)

Figure 3.2: Alignment A for $\mathcal{S}$ in the case of binary alphabet

In the following, we give some properties that allow us to show that a vertex cover for G is of size k if and only if the cost of an alignment of S can be bounded by a value C, depending on k and on the graph G, as stated in Theorem 3.4.12. By this result, the proof that the construction of the instance $(\mathcal{S}, C)$ for the alignment problem is a reduction is immediate.

Definition 3.4.1. An alignment A of the set S of sequences is a *standard alignment* if it satisfies properties (i), (ii), (iii) and (iv):

(i) all columns of $\mathsf{A}_{T\cup Q}$ containing some Δs are such that they contain only Δs in $\mathsf{A}_{T\cup Q\cup X}$ and contain no as in A_Y;

(ii) all Δs in A_Y are opposite only to Δs or to bs in A_Q;

(iii) in A_X there is no column with both as and Δs, and the first and last column of A_X consist of Δs;

(iv) the Δs of A_X are contained only in columns that do not contain any as of A_T.

We will show that an optimal alignment must be a standard alignment; the properties of definition 3.4.1 will allow us to relate the value of the alignment to the size of the vertex cover of the graph.

Let A be a standard alignment. Then, by the previous definition, it is immediate to verify that conditions (i) and (ii) imply that in A, all Δ's in internal columns of A_Y have a mismatch only with bs of sequences in S. Moreover, Δs in the first and last column of A_S mismatch only with bs of A_Q, otherwise by condition (i), (ii) and (iv), there is a column in A of only Δs, which is not possible. By this fact and definition 3.4.1, the Proposition 3.4.1 easily follows.

Proposition 3.4.1. *Let A be an alignment of S that satisfies properties (i) and (ii) of a standard alignment. For each edge sequence $s(i,j)$ in Y, one of the two bs encoding one end vertex v_h of the edge (v_i, v_j) is aligned in A with the $(h+1)$-th b of each template sequence in T, while the other b of $s(i,j)$ has a mismatch with each symbol of the template sequences to which it is opposite.*

Lemma 3.4.2. *Let A be a standard alignment of S. Then* $\mathrm{cost}(\mathsf{A}_Y) < 6l^2$, $\mathrm{cost}(\mathsf{A}_{Y,X}) < 8lkK_3$ *and* $\mathrm{cost}(\mathsf{A}_{Y,T}) < 4l^2K_1$.

Proof. Let us first prove the upper bound for $\mathrm{cost}(\mathsf{A}_Y)$. By condition (i) and (ii) of definition 3.4.1, Δs in internal columns of A_Y have a mismatch only with bs. Then, mismatches occur only in columns with bs. Since there are exactly $2l$ bs, it follows that the cost of all internal columns is bounded by $2l^2$, while the cost of the first and last column of A_Y is bounded by $4l^2$, as in such columns Δs have mismatch with as. Consequently, $\mathrm{cost}(\mathsf{A}_Y) < 6l^2$.

Now, let us prove that $\mathrm{cost}(\mathsf{A}_{Y,X}) < 8lkK_3$. Let y and x be two arbitrary sequences in Y and X, respectively. By condition (i), (ii) and (iii), Δs in $\mathcal{A}_Y$

are opposite only to Δs of A_X. Since $|y| = 3(n+1)$, while $|x| = 3n + 2 - k$ and y contains two bs which are not in x, it follows that $\mathsf{A}_{\{y,x\}}$ contains at most $k+3$ mismatches, from which we prove that $\text{cost}(\mathsf{A}_{Y,X}) < 2(k+3)lK_3$. Thus the required bound follows.

By Proposition 3.4.1 and just as in the above proof, it easily follows that $\text{cost}(\mathsf{A}_{Y,T}) < 4l^2K_1$. □

In the rest of this section, the upper bounds given in Lemma 3.4.2, we will be denoted respectively as U_Y, $U_{Y,X}$ and $U_{Y,T}$. pose that $K_1 > l$, $K_2 > U_Y + U_{Y,X} + U_{Y,T}$ and $K_3 > U_Y$.

Lemma 3.4.3. *Let y, q be two sequences with $y \in Y$, $q \in Q$ and let A be an alignment of Y. Then $\mathsf{A}_{\{y,q\}}$ contains at least four mismatches (and $\text{cost}(\mathsf{A}_{\{y,q\}}) \geq 4$). Moreover if A is a standard alignment, then $\mathsf{A}_{\{y,q\}}$ contains exactly four mismatches and $\text{cost}(\mathsf{A}_{\{y,q\}}) = 4$.*

Proof. By construction of sequences y, q, and by definition of standard alignment, it is immediate to note that, in a standard alignment A_1, both bs in y have a mismatch with some as or Δs of $\mathsf{A}_{1\{q\}}$, and both bs in q have a mismatch with some as or Δs of $\mathsf{A}_{1\{y\}}$. By the SP-score $\text{cost}(\mathcal{A}_{\{y,q\}}) = 4$. Along the same line it is immediate to note that if $\mathsf{A}_{\{y,q\}}$ is an arbitrary alignment where all bs have a mismatch, then $\text{cost}(\mathcal{A}_{\{y,q\}}) \geq 4$.

Assume now that in a non standard alignment a b of y and a b of q are aligned in the same column; we will prove that $\text{cost}(\mathcal{A}_{\{y,q\}}) > 4$. Clearly, the smallest number of mismatches is given by assuming that the first b of y is aligned with the first b of q, or the second b of y is aligned with the last b of q. Then, by construction of y and q the first three or last three as of y have a mismatch with some Δs, hence $\text{cost}(\mathcal{A}_{\{y,q\}}) \geq 6$. □

The following lemmata are direct consequences of Definition 3.4.1 and Lemma 3.4.3.

Lemma 3.4.4. *Let A be a standard alignment of S. Then $\text{cost}(\mathsf{A}_X)$, $\text{cost}(\mathsf{A}_T)$, $\text{cost}(\mathsf{A}_{T,Q})$, $\text{cost}(\mathsf{A}_Q)$, $\text{cost}(\mathsf{A}_{X,T\cup Q})$ and $\text{cost}(\mathsf{A}_{Y,Q})$ are fixed and minimum over all possible alignments.*

Lemma 3.4.5. *Let A be a standard alignment of S. Then in $\mathsf{A}_{Y\cup X}$ there are $K_3l(k+1)$ mismatches of the form (σ, Δ), where $\sigma \in \mathsf{A}_Y$ and $\Delta \in \mathsf{A}_X$.*

Proof. Let x and y be respectively a test sequence and an edge sequence, and let us consider the alignment $\mathsf{A}_{\{y,x\}}$. By properties (i), (ii) and (iii) of standard alignment each Δs in $\mathsf{A}_{\{y\}}$ is opposite only to Δs of $\mathsf{A}_{\{x\}}$ in $\mathsf{A}_{\{y,x\}}$. Since, by construction, each edge sequence contains $k+1$ symbols more than

each test sequence, it follows that in each test sequence x there are exactly $k+1$ Δs that have a mismatch with a symbol of y in $\mathsf{A}_{\{y,x\}}$. The claim follows immediately. □

Lemma 3.4.6. *Let* A *be a standard alignment of* S*. Then* $\text{cost}(\mathsf{A}_{X\cup T\cup Q}) + \text{cost}(\mathsf{A}_{Y,T\cup Q})$ *is fixed over all possible standard alignments of* S*.*

Proof. By Lemma 3.4.4, $\text{cost}(\mathsf{A}_{X\cup T\cup Q})$ and $\text{cost}(\mathsf{A}_{Y,Q})$ are fixed. Let y, t be respectively an edge sequence and a template sequence, and let A be a standard alignment of S. Since in t there is one symbol more than in y it follows that in $\mathsf{A}_{\{y\}}$ there is one Δ more than in $\mathcal{A}_{\{t\}}$. By Proposition 3.4.1, all bs of t, except for one, have a mismatch with the symbol of y to which they are opposite. Moreover, by Proposition 3.4.1, there is an a or a Δ inserted in t that has a mismatch with a b of y. By definition of standard alignment, there cannot be any other mismatch in $\mathsf{A}_{\{y,t\}}$. □

By previous Lemma 3.4.6, the sum $\text{cost}(\mathsf{A}_{X\cup T\cup Q}) + \text{cost}(\mathsf{A}_{Y,T\cup Q})$ is fixed for every standard alignment A; in the following we will denote such sum cost_{SD}. Moreover, by Lemma 3.4.2 it is immediate that every standard alignment A and hence every optimal alignment has a value $\text{cost}(\mathsf{A}) \leq \text{cost}_{SD} + U_Y + U_{Y,X}$.

Lemma 3.4.7. *Let* A *be an optimal alignment of* S*. Then* A *must satisfy property* (i) *of a standard alignment.*

Proof. Let A_1 be an arbitrary standard alignment and let A be an optimal alignment of S that does not satisfy property (i) of standard alignment. Then the following cases must be considered.

1. There is a column in A_Q containing some Δs and some σs, for $\sigma \in \{a, b\}$. By Lemma 3.3.3, A cannot be optimal.

2. Let us assume that there is a column of $\mathsf{A}_{T\cup Q}$ that contains at least a symbol σ and Δ. Clearly, if Δ is in A_T, then a symbol Δ must be in A_Q. By case 1, since each column of A_Q contains either as or bs or Δs, it follows that there is a mismatch (Δ, σ) in $\mathcal{A}_{T\cup Q}$, consisting of a Δ in A_Q and a σ in A_T, that occurs in the i-th column of A.

   Then, we can show that $\text{cost}(\mathsf{A}_{T,Q}) \geq \text{cost}(\mathsf{A}_{1T,Q}) + K_2$. In fact, let t be the sequence of T that contains the symbol σ in the i-th column and let q be an arbitrary sequence in Q. Then, $\mathsf{A}_{\{t,q\}}$ contains at least $n+1$ mismatches, as $|t| = |q|$ and t contains $n+2$ bs, while $\mathsf{A}_{\{q\}}$ contains 2 bs and a Δ not in $\mathsf{A}_{\{t\}}$. Clearly, $\mathsf{A}_{1\{t,q\}}$ contains exactly n mismatches, all of value 1. Since $|Q| = K_2$, it follows that $\text{cost}(\mathsf{A}_{T,Q}) \geq \text{cost}(\mathsf{A}_{1T,Q}) + K_2$.

3. Assume now that every column containing Δs in $\mathsf{A}_{T\cup Q}$ has only Δs in $\mathsf{A}_{T\cup Q}$, but at least one a in A_X. Let y be the sequence in X that has at least one a in such column and let q be a sequence in Q. Since $\mathsf{A}_{\{q\}}$ contains at least a Δ opposite to an a in $\mathsf{A}_{\{y,q\}}$, while y has at least $k+2$ Δs, it follows that $\mathsf{A}_{\{y,q\}}$ contains at least $k+3$ mismatches, of which two are of value 1, while the other ones are of value 2. By condition (i) of standard alignment, for any arbitrary sequence $x \in X$, $\mathsf{A}_{1\{x,q\}}$ contains exactly $k+2$ mismatches. It follows that $\text{cost}(\mathsf{A}_{X,Q}) \geq \text{cost}(\mathsf{A}_{1X,Q})+K_2$.

4. Each column containing Δs in $A_{T\cup Q}$ consists of only Δs in $\mathsf{A}_{T\cup Q\cup X}$ and has at least an a in A_Y. We are now able to show that $\text{cost}(\mathsf{A}_{S,Q}) \geq \text{cost}(\mathsf{A}_{1S,Q})+K_2$. By Lemma 3.4.3, for each sequence $y \in Y$ and $q \in Q$, $\text{cost}(\mathsf{A}_{1\{y,q\}}) = 4$.

   Let y_1 be a sequence in Y that contains an a in a column where there are only Δs in $\mathsf{A}_{T\cup Q\cup X}$. Then $\mathsf{A}_{\{y_1,q\}}$ must contain at least a mismatch (a, Δ) besides four mismatches of value 1. Hence $\text{cost}(\mathsf{A}_{\{y_1,q\}}) \geq 5$. It follows that $\text{cost}(\mathsf{A}_{Y,Q}) \geq \text{cost}(\mathsf{A}_{1Y,Q}) + K_2$.

By previous cases, Lemma 3.4.4 and Lemma 3.4.2, since $K_2 > U_{Y,T} + U_{Y,X} + U_Y$ the Lemma follows. □

Lemma 3.4.8. *Let* A *be an optimal alignment of* S*. Then* A *must satisfy property* (ii) *of a standard alignment.*

Proof. By Lemma 3.4.7, A must satisfy property (i). Assume that A satisfies property (i) and assume to the contrary that A does not have property (ii). Let A_1 be a standard alignment. As in the proof of Lemma 3.4.7, case 4, it is easy to show that $\text{cost}(\mathsf{A}_{Y,Q}) \geq \text{cost}(\mathsf{A}_{1Y,Q}) + K_2$. Since $K_2 > U_{Y,T} + U_Y + U_{Y,X}$, it follows that $\text{cost}(\mathsf{A}_{Y,T\cup Q}) > \text{cost}(\mathsf{A}_{1Y,T\cup Q}) + U_Y + U_{Y,X}$. By applying Lemma 3.4.4 and Lemma 3.4.2, we obtain that $\text{cost}(\mathsf{A}) > \text{cost}(\mathsf{A}_1)$, which is a contradiction. □

Lemma 3.4.9. *Let* A *be an optimal alignment of* S*. Then* A *must satisfy property* (iii) *of a standard alignment.*

Proof. By Lemma 3.3.3, there is no column of A_X containing Δs and as. Moreover, by the SP-score, in an optimal alignment it is more advantageous that Δs of A_X are opposite to bs of $\mathsf{A}_{T\cup Q}$. Consequently, in the first and last column of A_X there are only Δs. □

Lemma 3.4.10. *Let* A *be an optimal alignment of* S*. Then* A *must satisfy property* (iv) *of a standard alignment.*

Proof. By Lemmata 3.4.7, 3.4.8 and 3.4.9, A must satisfy properties (i), (ii) and (iii). Consequently, if (iv) does not hold, it means that there is a column of index l_1 in A containing only Δs of X and only as of $T \cup Q$, and eventually bs of S. Moreover, since for each sequence x, $x \in X$, $|x| = 3n + 2 - k$, while for each sequence $t \in T$, t contains $n + 2$ bs, it follows that there is a column of index l_2 in A containing as of each test sequence in X and bs of the template sequences. Then let A_1 be the alignment obtained from A as follows: in A_X substitute the Δs in the column of index l_1 with the as in the column with index l_2 and vice versa.

By construction of A_1, $\text{cost}(\mathsf{A}) - \text{cost}(\mathsf{A}_1)$ is equal to the sum $\text{cost}(\mathsf{A}[l_1]) - \text{cost}(\mathsf{A}_1[l_1]) + \text{cost}(\mathsf{A}[l_2]) - \text{cost}(\mathsf{A}_1[l_2])$. We will prove that $\text{cost}(\mathsf{A}) - \text{cost}(\mathsf{A}_1) > 0$, thus obtaining a contradiction with the assumption that A is an optimal alignment of S. In the following, by $n_\sigma(l_i)$ we will denote the number of σ symbols in the column of index l_i of A_Y. By construction of the sequences, and by the SP-score it is easy to note that:

$$\begin{aligned}
\text{cost}(\mathsf{A}[l_1]) =& (K_1 + K_2 + n_a(l_1))n_b(l_1)d(a,b) + (K_1 + K_2 + n_a(l_1))K_3 d(a,\Delta) \\
&+ n_b(l_1)K_3 d(\Delta, b) \\
\text{cost}(\mathsf{A}[l_2]) =& (K_1 + n_b(l_2))(K_2 + K_3 + n_a(l_2))d(a,b) \\
\text{cost}(\mathsf{A}_1[l_1]) =& (K_1 + K_2 + K_3 + n_a(l_1))n_b(l_1)d(a,b) \\
\text{cost}(\mathsf{A}_1[l_2]) =& (K_1 + n_b(l_2))(K_2 + n_a(l_2))d(a,b) + (K_2 + n_a(l_2))K_3 d(a,\Delta) + \\
&(K_1 + n_b(l_2))K_3 d(\Delta, b)
\end{aligned}$$

Consequently $\text{cost}(\mathsf{A}) - \text{cost}(\mathsf{A}_1) = 2K_3(K_1 - (n_a(l_2) - n_a(l_1)))$. Since $n_a(l_2) - n_a(l_1) \leq l$, it follows $\text{cost}(\mathsf{A}) - \text{cost}(\mathsf{A}_1) \geq 2K_3(K_1 - l)$. By posing $K_1 > l$, we obtain that $\text{cost}(\mathcal{A}_1) < \text{cost}(\mathsf{A})$, which contradicts the fact that A is optimal.

Thus, A must satisfies property (iv). □

By Lemmata 3.4.7, 3.4.8, 3.4.9 and 3.4.10 it follows directly that:

Corollary 3.4.11. *An optimal alignment of S is a standard alignment.*

The result of Theorem 3.4.12, relates the value of an alignment of S to the size of a vertex cover.

Theorem 3.4.12. *Let G be a graph and S the encoding of G. Then:*

(1) *if G has a vertex cover of size k, then there is a standard alignment A of S such that $\text{cost}(\mathsf{A}_{Y,X}) \leq 2lK_3 + 2lkK_3$,*

(2) *if G has a minimum vertex cover of size $k_1 > k$, then for every standard alignment A of S it holds that $\text{cost}(\mathsf{A}_{Y,X}) > U_Y + 2lK_3 + 2lkK_3$.*

Proof. Assume first that G has a vertex cover of size k. Let A be a standard alignment of S, where the sequences in $Y \cup X$ are aligned as follows: A_Y does not contain Δs in internal columns. For each test sequence the first and last Δs are respectively aligned in the first and last column of A_Y. Moreover, for each edge sequence $s(i,j)$ encoding the edge e, $e = (v_i, v_j)$, one of the two bs encoding one end vertex of e is aligned in a column of A containing Δs of each test sequence $x(k)$. Such an alignment is possible since each edge has one end in the vertex cover, and the number of Δs in each test sequence is equal to $k+2$, where k is the size of a vertex cover. In fact, if the set C of vertices, with $C = \{v_{i_1}, \cdots, v_{i_k}\}$ is a vertex cover for G, then A can be obtained by aligning for each $1 \leq h \leq k$, the $(h+1)$th Δs of each test sequence with the (i_h+1)th b of each template sequence, and with a b of an edge sequence. In fact, every edge sequence encodes an edge (v_i, v_j) such that either $v_i \in C$ or $v_j \in C$, (Fig. 3.2). It follows, by Proposition 3.4.1, that the total number of bs in A_Y opposing Δs in A_X is equal to the number l of edges of the graph.

Let us determine $\text{cost}(\mathsf{A}_{Y,X})$. Let I be the set of indices of the columns of A containing Δs of X and let $n_\sigma(i)$ be the number of σs in the column of $\mathsf{A}_{Y\cup X}$ of index i. Moreover, let r be the number of columns in A. Then $\text{cost}(\mathsf{A}_{Y,X}) = \sum_{i\in I-\{1,r\}} K_3(d(\Delta,b)n_b(i) + d(\Delta,a)(l-n_b(i)) + \sum_{i\notin I\cup\{1,r\}} K_3 d(a,b)n_b(i) + \text{cost}(\mathsf{A}_{S,X}[1]) + \text{cost}(\mathsf{A}_{S,X}[r])$. Consequently $\text{cost}(\mathsf{A}_{S,X}) = 2klK_3 + 2lK_3$, since $\sum_{i\in I} n_b(i) = l$.

Now, assume that G has a minimum vertex cover of size k_1, with $k_1 > k$. Let A be an arbitrary standard alignment of S. Let us compute $\text{cost}(\mathsf{A}_{Y,X})$: since G has a vertex cover of size $k_1 > k$ and, by Proposition 3.4.1, for each edge sequence $s(i,j)$ encoding the edge (v_i, v_j), exactly one b of $s(i,j)$ encoding an end vertex v_h is aligned with the $(h+1)$th b of each template sequence, there must be at least k_1 columns of A that contain bs of the template sequences opposing one b of at least an edge sequence. By properties (iii) and (iv) of standard alignment, in $\mathsf{A}_{X\cup T}$, each Δ of A_X is aligned with a b of A_T. Since A_X contains k Δs internal columns, it follows that there is at least one edge sequence such that no one of the two bs of these sequences is in a column of $\mathsf{A}_{Y\cup X}$ containing Δs of X. Consequently, given I_1 the set of indices of the columns of A that contain Δs of the test sequences, $\sum_{i\in I_1} n_b(i) \leq l-1$. Clearly, $D(\mathsf{A}_{Y,X}) = K_3(\sum_{i\in I_1}(d(\Delta,a)n_a(i) + d(\Delta,b)n_b(i)) + \sum_{i\notin I_1}(d(a,\Delta)n_\Delta(i) + d(a,b)n_b(i))$.

By Lemma 3.4.5 the number of mismatches (σ,Δ), where $\Delta \in \mathsf{A}_X$ is $K_3 l(k+1)$. Consequently, there are $K_3 l(k+1) - K_3 \sum_{i\in I_1} n_b(i)$ mismatches (a,Δ). It follows that $\text{cost}(\mathsf{A}_{Y,X}) = K_3(2l(k+1) - 2\sum_{i\in I_1} n_b(i) + \sum_{i\in I_1} n_b(i) + \sum_{i\notin I_1} n_b(i) + 2\sum_{i\notin I} n_\Delta(i)) \geq 2lkK_3 + 2lK_3 + 2K_3$, as by properties (i) and (ii)

of standard alignment $\sum_{i \notin I} n_\Delta(i) = 0$ and $\sum_i n_b(i) = 2l$. By Lemma 3.4.6, since $K_3 > U_Y$ it follows that $\text{cost}(\mathsf{A}_{Y,X}) > U_Y + 2lK_3 + 2lkK_3$. □

By Corollary 3.4.11 and Theorem 3.4.12, the following result is immediate.

Corollary 3.4.13. *The graph G has a vertex cover of size k if and only if the set S has an optimal alignment A of value $\text{cost}(\mathsf{A})$, with $\text{cost}(\mathsf{A}) < \text{cost}_{SD} + U_Y + 2lK_3 + 2lkK_3$.*

By the previous result it follows that the construction of the set S of sequences from an instance G and k of VC is a reduction to sequence alignment.

Theorem 3.4.14. *Multiple alignment with SP-score that is a metric is $\mathcal{NP}$-complete even over a binary alphabet.*

3.5 Fixing the Cost of a Gap

Following the results presented in the previous section of this chapter, several such modifications of the Multiple Sequence Alignment problem were studied, most notably in [60] the Gap-0 Alignment problem, where spaces may be inserted at the beginning and at the end of sequences, but not between characters from Σ, and the Gap-0-1 Alignment problem, which is the restriction of Gap-0 Alignment where at most one space can be inserted in each sequence have been studied. Anyway most scoring schemes used in practice are *affine*, i.e., they specify a fixed *gap opening penalty* g (possibly 0) that is added to the score calculated according to d_M for each newly created gap in the alignment. In this context, the numbers $d_M(s, \Delta)$ for $s \in \Sigma$ are called *gap extension penalties.* Note that if all gap extension penalties are zero, then we have a scoring scheme with *fixed gap penalties.* If $d_M(s, \Delta) > 0$ for all $s \in \Sigma$, then we will say that the scoring scheme specifies *strictly positive gap extension penalties.* In [60] all the problems mentioned above have been proved to be $\mathcal{NP}$-hard for practically every affine scoring scheme with strictly positive gap extension penalties used by molecular biologists [60]. This left open the case of other ways of calculating the gap penalties that are sometimes used in Molecular Biology, in particular, the interesting case of fixed gap penalties, where all gaps are penalized equally, no matter where they occur and how long they are.

In this section, as well as in Chapter 4, we will study a formulation of Multiple Sequence Alignment that captures to some extent the pattern of space insertions observed in real biomolecular sequences and is different from the restrictions studied in [60]. A *space-L alignment* $\mathcal{A}$ of a set

	Δ	A	C	T
Δ	0	0	0	0
A	0	0	0	1
C	0	0	0	0
T	0	1	0	0

$< t_1, \ldots, t_k >$ of sequences is an alignment $< at_1, \ldots, at_k >$ of $< t_1, \ldots, t_k >$ such that $|at_i| \leq |t_i| + L$ for each sequence t_i. Note that space-L-alignments exist only if the length of the shortest of these sequences is at least $n - L$, where n is the length of the longest among the sequences $t_1, \ldots, t_k$. Please also note that there are no restrictions about where the space symbols can be inserted.

In this section we will prove that the Space-L Multiple Sequence Alignment problem is $\mathcal{APX}$-hard even if L is restricted to be a constant. The formal definition of Space-L Multiple Alignment is:

Problem 3 (Min Space-L Multiple Alignment).
Instance: a set of sequences $\langle t_1, \ldots, t_k \rangle$ and a scoring scheme (d_M, g).
Solution: a space-L multiple alignment $\mathcal{A}$ of $\langle t_1, \ldots, t_k \rangle$.
Goal: to minimize the SP-score of $\mathcal{A}$.

The main result in this section is the following:

Theorem 3.5.1. *There exists a scoring scheme (M, g) with fixed gap penalties such that:*
(a) *For the scoring scheme (M, g) and for every $L > 0$ the* Space-L Multiple Alignment *problem is $\mathcal{APX}$-hard.*
(b) *For the scoring scheme (M, g) the* Gap-0 Multiple Alignment *problem is $\mathcal{APX}$-hard.*
(c) *For the scoring scheme (M, g), the* Multiple Alignment *problem is $\mathcal{APX}$-hard.*

Here is the scoring scheme mentioned in the above theorem. The alphabet will be $\Sigma = \{A, C, T\}$, the gap opening penalty will be $g = 2$, and the scoring matrix M is representated in Fig. 3.5

Proof. We will prove Theorem 3.5.1 by reducing the Max Cut problem (Pb. 8 at page 99) on cubic graphs - such problem is denoted by 3-Max Cut - to the respective multiple alignment problems. Recall that an instance of size k of the 3-Max Cut problem is a simple graph $G = \langle V, E \rangle$ such that $|V| = k$ and each vertex of G has degree exactly 3. The problem is to find

$t[2(\lceil\alpha k\rceil m+1)]$ $t[2\lceil\alpha k\rceil\ell+2(h-1)k^3]$ $t[2\lceil\alpha k\rceil\ell+2(i-1)k^3]$

t_h: ... A C A C A ... A C A C A C ... C C C C C ...

t_i: ... T C T C T ... C C C C C C ... A C A C A ...

t_{k+h}: ... C C C C C ... T C T C T C ... C C C C C ...

t_{k+i}: ... C C C C C ... C C C C C C ... T C T C T ...

a partition of the set of vertices V into disjoint sets V_0 and V_1 such that the number of edges that connect a vertex in V_0 with a vertex in V_1, i.e., the size of the *cut determined by* $\langle V_0, V_1\rangle$, is as large as possible. It is well known that the 3-MAX CUT problem is $\mathcal{APX}$-hard, for our purposes, it is most important to note that the latter implies that there exists a real $\epsilon > 0$ such that no polynomial-time approximation algorithm can find a cut such that the number of edges that are NOT cut is within an additive constant of ϵk of minimum.

Given a cubic graph $G = \langle V, E\rangle$ with k vertices, we define a $2k$-tuple $\bar{t}^G = \langle t_1, \ldots, t_{2k}\rangle$ of sequences as follows: Enumerate $V = \{v_1, \ldots, v_k\}$, $E = \{e_0, \ldots, e_{\ell-1}\}$. Each sequence t_i will have length $2(k^4 + \lceil\alpha k\rceil\ell)$, where α is a constant that will be defined below. Intuitively speaking, for $1 \leq i \leq k$, the sequence t_i will encode the vertex v_i. Edge $e_m = \{v_i, v_h\}$ will be encoded by characters $t_h[j], t_i[j]$, where $j = 2(\lceil\alpha k\rceil m + 1), \ldots, 2\lceil\alpha k\rceil(m+1)$. More precisely, we define $t_i[j]$, the j-th character in t_i, as follows. For $1 \leq m \leq \ell$, $e_m = \{v_h, v_i\}$, $h < i$, $\lceil\alpha k\rceil m < j < \lceil\alpha k\rceil(m+1)$ we put: $t_h[2j] = A$, $t_i[2j] = T$, and $t_p[2j] = C$ for $p \notin \{i, h\}$.

The sequence t_{k+i} will act as a "mirror image" of t_i. The purpose of mirror images is to neutralize the effects of unbalanced cuts on the scores of alignments. For $1 \leq i \leq k$ and $\lceil\alpha k\rceil\ell + ik^3 < j < \lceil\alpha k\rceil\ell + (i+1)k^3$ we put: $t_i[2j] = A$, $t_{k+i}[2j] = T$, $t_p[2j] = C$ for $p \notin \{i, k+i\}$.

For all p, j, we let $t_p[2j-1] = C$. Let us illustrate this construction with a picture. In Fig. 3.5 we exhibit a situation where $e_m = \{v_h, v_i\}$.

Let us define a "benchmark alignment" of the above sequences. We will define this alignment by partitioning the sequences into two sets $\mathcal{L}$ and $\mathcal{R}$ and inserting one space to the left of each sequence in $\mathcal{L}$ and one space to the right of each sequence in $\mathcal{R}$. Let $\langle V_1, V_2\rangle$ be a cut of G. We will show how to associate a benchmark alignment to such cut. For each $1 \leq i \leq k$ we let $t_i \in \mathcal{L}$ if and only if $t_{k+i} \in \mathcal{R}$. Moreover for each $1 \leq i \leq k$ we let $t_i \in \mathcal{L}$ if and only if $v_i \in V_1$.

Note that the score for the benchmark alignment is $4k^2 + \alpha kU$, where U

is the number of edges that *are not* in the cut $\langle V_1, V_2\rangle$. Moreover, the benchmark alignment is a gap-0-1 alignment, and hence both a gap-0 alignment and a space-1 alignment.

We will show that there exists a fixed $\delta > 0$ such that if an alignment a of the above sequences is found that scores within a factor of $(1+\delta)$ of the benchmark alignment, then it will be possible to reconstruct, in polynomial time, from this alignment a partition of the vertex set that induces a cut whose size is within a additive constant of ϵk of maximum. Suppose we have *any* alignment a that scores within $1+\delta$ of our benchmark alignment, where δ is sufficiently small and will be determined later. Let us say that a sequence pair $\langle t_p, t_q\rangle$ is *static in* a if there is no space in the induced pairwise alignment $\langle bt_p, bt_q\rangle$. Being static in a is easily seen to be an equivalence relation. Let T_1 and T_2 denote the two largest equivalence classes of the "static" relation, and let T_3 denote the set of sequences that are neither in T_1 nor in T_2. Note that none of the sequence pairs $\langle t_i, t_{k+i}\rangle$ can be static in a, otherwise the cost of the alignment of $\rangle t_i, t_{k+i}\rangle$ is too large. Thus the size of T_1 and T_2 is at most k. Let $|T_1| = k - k_1$, $|T_2| = k - k_2$. Then $|T_3| = k_1 + k_2$. Since each pair of sequences from different equivalence classes adds at least 4 to the SP-score of a, we have

$$SP(\langle at_1, \ldots, at_{2k}\rangle) \geq$$

$$\geq 4((k-k_1)(k-k_2) + (k-k_1)(k_1+k_2) + (k-k_2)(k_1+k_2)) =$$

$$= 4(k^2 + k_1k_2 + k(k_1+k_2) - (k_1+k_2)^2) = 4(k^2 + k_1k_2 + (k - |T_3|)|T_3|).$$

Thus the numbers k_1 and k_2 must be such that $k_1k_2 + (k - |T_3|)|T_3| < \delta k^2 + \delta\alpha kU$, where U is the number of edges that are not cut by the partition used in the benchmark alignment. Note that $U \leq 3k$. We will choose $\delta < \frac{\epsilon\alpha}{100}$. It follows that as long as α is sufficiently small, we can assume that $|T_3| < k\frac{\epsilon}{6}$. Now let α, δ be as above, and let V_i be the set of all vertices such that $t_i \in T_i$ for $i \in \{1, 2\}$. Consider the partition $\langle V_1, V\backslash V_1\rangle$. Let W be the number of edges of G that are not cut by $\langle V_1, V\backslash V_1\rangle$. Note that this number differs from the number Z of edges $\{v_i, v_j\}$ such that $\langle t_i, t_j\rangle$ is static by at most $3|T_3|$, since every edge in the difference must have an endpoint in T_3 and the degree of the graph is 3. If the SP-score of the alignment is within a factor of $(1+\delta)$ of that of the benchmark alignment, then we have:

$$4k^2 + \alpha kW \leq 4k^2 + \alpha k(Z + k\frac{\epsilon}{2}) \leq (1+\delta)(4k^2 + \alpha kU) + \alpha\frac{\epsilon}{2}k^2.$$

By the choice of δ and since $U \leq 3k$, we get

$$\alpha kW - \alpha kU < 4\delta k^2 + \delta\alpha kU + \alpha\frac{\epsilon}{2}k^2.$$

Assuming, as we may, that $\alpha \leq 1$, and noting that $U \leq 3k$, our choice of δ gives:

$$W - U < 4\frac{\epsilon}{100}k + 3\frac{\epsilon}{100}\alpha k + \frac{\epsilon}{2}k < \epsilon k.$$

□

3.6 Maximum Trace Alignment is $\mathcal{APX}$-hard

The formulations of Multiple Sequence Alignment that have been presented previously have some drawbacks, namely the cost of two symbols is the same in the whole alignment, since it is defined as the weight assigned to each pair of symbols of the alphabet. Consequently it is not possible to have different costs when the same alphabet symbols are aligned with different occurrences of such symbols. Moreover it is not always true that the SP alignment of maximum cost is the one that is more interesting from the biological point of view. For example it is well known that in DNA sequences the encoding part (exons) are relatively short and sparse with respect to the non-encoding parts (introns), hence it seems natural to penalize differences between encoding regions more than ones in non-encoding regions, while the fact that exons relatively short and sparse with respect to introns makes the latter ones more relevant in determining the optimum alignment.

In this section we study a different formulation of Multiple Sequence Alignment that has been given by Kececioglu [68] as Maximum Trace Alignment, where the set of k sequences is represented as a k-partite graph (see [96] for another application of k-partite graphs to sequence alignment), and the edges connect two symbols in different sequences. The edges are weighted and the goal is to compute an alignment maximizing the sum of the weights of the edges connecting two vertices in the same column. Even though the problem is $\mathcal{NP}$-complete, some algorithms for computing an optimal solution have been described [68, 69], but such algorithms still can require exponential time in the worst cases.

This formulation of multiple sequence alignment allows to model situations where alignments of *segments* are taken into account. A segment is a substring of a sequence, and we would like to have segments completely aligned with each other, while aligning only partially the two segments has the same cost as not aligning the two segments at all (see Fig. 3.3, in the ellipses there are the two segments we want to align perfectly). This can be viewed as a more general scoring scheme where the distance is not only between symbols, but between substrings of the given sequences. There is an increasing trend in using these score schemes [3, 77] for computing alignment

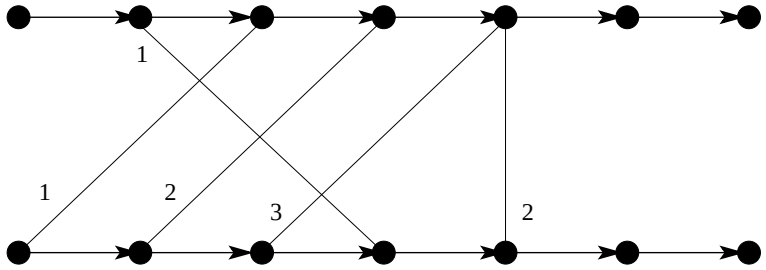

Figure 3.3: Example of trace alignment

of biological sequences.

In this section we focus on approximation complexity of the problem proving that Max Trace Alignment is $\mathcal{APX}$-hard, hence ruling out the possibility of polynomial time approximation scheme [9].

In the following we will introduce the formal definitions of the problems with which we will deal in this section.

The Max Weighted Trace Alignment problem (MTA, Pb. 19) [68] asks for maximum weight trace alignment, given an alignment graph $G = \langle V, E, F\rangle$. The Max Trace Alignment problem is the restriction of MTA when all edges have weight one.

The Max 2 Satisfiability B-Bounded problem (Max2Sat-B) is a restriction of Max Sat (Pb. 9), and has as instance a set of clauses $\{c_1, \ldots, c_k\}$ over the variables $v_1, \ldots, v_n$ such that each variable appears in at most B clauses and each clause contains 2 variables (possibly negated) and asks for a truth assignment of the variables that maximizes the number of satisfied clauses. The problem Max2Sat-B is $\mathcal{APX}$-hard for $B \geq 5$ [78, 79].

Given an instance of Max2Sat-B, we will build a corresponding instance $G =< V, E, F >$ of MTA.

In the following we will denote with $X = \{x_1, \ldots, x_n\}$ the set of the variables and with $C = \{c_1, \ldots, c_k\}$ the set of clauses of the instance of Max2Sat-B. Please note the in an instance $\mathcal{S}$ of the problem Max2Sat-B, $k \leq Bn$, and hence $\mathrm{Opt}(\mathcal{S}) \leq Bn$ from the boundedness of the problem. Moreover each clause c_l can be expressed as

$$x_{\mathrm{lvar}(l)}^{\mathrm{lexp}(l)} \vee x_{\mathrm{rvar}(l)}^{\mathrm{rexp}(l)}$$

with $1 \leq \mathrm{lvar}(l) < \mathrm{rvar}(l) \leq n$ and $\mathrm{lexp}(l), \mathrm{rexp}(l) \in \{\text{true}, \text{false}\}$, and where x_i^{false} stands for $\neg x_i$ and x_i^{true} stands for x_i, this means that we can describe a clause c_l as a fourtuple $\langle \mathrm{lvar}(l), \mathrm{rvar}(l), \mathrm{lexp}(l), \mathrm{rexp}(l)\rangle$. The graph obtained in the reduction has vertex set $\{T_i, F_i, X_i : 1 \leq i \leq n\} \cup \{D_{i,j} :$

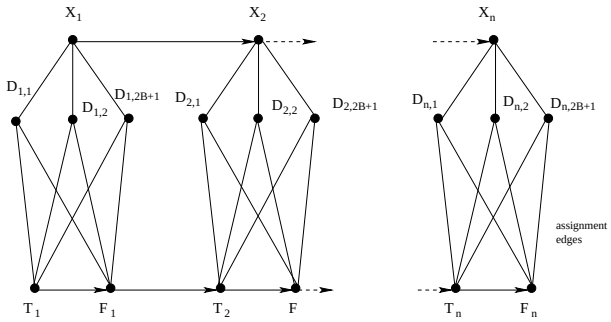

Figure 3.4: the encoding of a set of variables

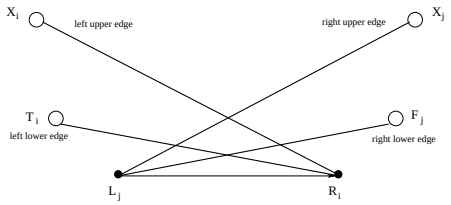

Figure 3.5: the encoding of the clause $x_i \vee \neg x_j$

$1 \leq j \leq n$, $1 \leq i \leq 2B+1\} \cup \{L_i, R_i : 1 \leq i \leq k\}$, moreover we will call a pair of vertices L_l, R_l the pair of *clause vertices* of c_l. The set F is $\{(T_i, F_i) : 1 \leq i \leq n\} \cup \{(F_i, T_{i+1}) : 1 \leq i \leq n-1\} \cup \{(X_i, X_{i+1}) : \leq i \leq n-1\} \cup \{(L_i, R_i) : 1 \leq i \leq k\}$.

The set E of undirected edges contains the *assignment* edges, $\{(D_{i,j}, T_i) : 1 \leq i \leq n,\ 1 \leq j \leq 2B+1\} \cup \{(D_{i,j}, T_i) : 1 \leq i \leq n,\ 1 \leq j \leq 2B+1\}$, the set of *internal edges* $\{(X_i, D_{i,j}) : 1 \leq i \leq n, 1 \leq j \leq 2B+1\}$. Moreover to each clause c_l we associate two *right* edges and two *left* edges, all of weight one. The right edges are $(L_l, X_{\mathrm{rvar}(l)})$ and $(L_l, P(\mathrm{rvar}(l), \mathrm{rexp}(l)))$, and the left edges are $(R_l, X_{\mathrm{lvar}(l)})$ and $(R_l, P(\mathrm{lvar}(l), \mathrm{lexp}(l)))$ where $P(i,j) = T_i$ if $j = \text{true}$, otherwise $P(i,j) = F_i$. The two edges $(L_l, X_{\mathrm{rvar}(l)})$ and $(R_l, X_{\mathrm{lvar}(l)})$ are called *upper* edges, while the other ones are called *lower* edges.

In Fig. 3.6 it is represented the encoding of the set of n variables, while in Fig. 3.6 it is represented the encoding of the clause $c_l = x_i \vee \neg x_j$.

The following results are fundamental in proving the correctness of our reduction.

Lemma 3.6.1. *Let $G = \langle V, E, F\rangle$ be a graph obtained reducing an instance of* Max2Sat-B*, let x_i be a variable of such instance and let $G_1 = \langle V, E_1, F\rangle$ be a trace alignment of G. Then in G_1 there cannot be both assignment edges*

$(D_{i,j}, T_i)$ *and* $(D_{i,j}, F_i)$

Proof. An immediate consequence of the fact that $\langle T_i, (T_i, F_i), F_i, (F_i, D_{i,j}), D_{i,j}, (D_{i,j}, T_i).T_i\rangle$ is a special cycle in G. □

Lemma 3.6.1 allows us to reconstruct an assignment from a trace alignment; in fact we can assign to x_i the value true if $D_{i,1}$ is adjacent to T_i in the trace alignment, while we assign to x_i the value false if and only if $D_{i,1}$ is adjacent to F_i. We still have to prove that $D_{i,1}$ is adjacent to T_i or to F_i, and to show how we can relate the cost of the trace alignment to the number of satisfied clauses.

Given a clause $c_l = (x_i^\alpha, x_j^\beta)$, $\langle X_j, (X_j, L_l), L_l, (L_l, R_l), R_l, (R_l, X_i,), X_i, (X_i, X_{i+1}), \ldots, (X_{j-1}, X_j), X_j\rangle$ and $\langle P(\text{rvar}(l), \text{rexp}(l)), (P(\text{rvar}(l), \text{rexp}(l)), L_l), L_l, (L_l, R_l), R_l, (R_l, P(\text{lvar}(l), \text{lexp}(l))), \ldots, T_{i+1}, (T_{i+1}, F_{i+1}), F_{i+1}, \ldots, (T_{j-1}, F_{j-1}), \ldots, P(\text{lvar}(l), \text{lexp}(l)), (P(\text{lvar}(l), \text{lexp}(l)), F_j), F_j\rangle$ are two cycles in G, hence the following Lemma holds:

Lemma 3.6.2. *Let $G = \langle V, E, F\rangle$ be a graph obtained reducing an instance of* Max2Sat-B*, let c_l be a clause of such instance and let $G_1 = \langle V, E_1, F\rangle$ be a trace alignment of G. Then in G_1 there are not both upper edges and there are not both lower edges.*

An immediate consequence is that in a trace alignment there are at most 2 edges adjacent to the vertices encoding each clause.

Lemma 3.6.3. *Let $G = \langle V, E, F\rangle$ a graph obtained reducing an instance of* Max2Sat-B*, and let $G_1 = \langle V, E_1, F\rangle$ be a trace alignment of G. Then it is possible to compute (in polynomial time) a trace alignment $G_2 = \langle V, E_2, F\rangle$ such that* $\text{weight}(G_1) \leq \text{weight}(G_2)$ *and for each variable x_i exactly $2B + 1$ internal edges and $2B + 1$ assignment edges $(D_{i,j}, T_i)$, $(D_{i,j}, F_i)$ are in E_2. Moreover for each i, either all $D_{i,j}$ are adjacent to T_i or all $D_{i,j}$ are adjacent to F_i.*

Proof. Let $G_1 = \langle V, E_1, F\rangle$ be a trace alignment, let $C(i)$ be the set $\{X_i\} \cup \{D(i, j) : 1 \leq j \leq 2B+1\}$, and let I be the set of all i such that $C(i)$ does not satisfy the Lemma. Then repeat the following procedure for each $i \in I$. Let k be an index such that the subgraph of G_1 induced by the set $\{X_i, D_{i,k}, T_i, F_i\}$ has the maximum number of edges among all $\{X_i, D_{i,j}, T_i, F_i\}$, for $1 \leq j \leq 2B + 1$. Then $(X_i, D_{i,j}) \in E_2$ iff $(X_i, D_{i,k}) \in E_1$, $(D_{i,j}, T_i) \in E_2$ if and only if $(D_{i,k}, T_i) \in E_1$, $(D_{i,j}, F_i) \in E_2$ iff $(D_{i,k}, F_i) \in E_1$ for $1 \leq j \leq 2B + 1$, that is in G_2 the sets $\{X_i, D_{i,j}, T_i, F_i\}$ are all isomorphic, while all other edges of G_2 are the same as in G_1. Note that G_2 cannot contain any special cycle, and that in G_2 there are either $2B+1$ or no internal edges while, by Lemma 3.6.1

there are either $2B+1$ or no assignment edges. If G_2 has $2B+1$ internal and $2B+1$ assignment edges then the lemma is proved, otherwise let E_2 be the set obtained from E_1 removing all edges incident on X_i, on F_i or on T_i, adding all internal edges and all edges $(D_{i,j}, T_i)$ for each $1 \leq j \leq 2B+1$. Note that all subgraphs induced by the sets $\{X_i, D_{i,j}, T_i, F_i\}$ for $1 \leq j \leq 2B+1$ are isomorphic, hence there is no path from X_i to T_i or to F_i in G_1, otherwise there would be $2B+1$ internal edges and $2B+1$ assignment edges.. The number of edges incident on X_i, F_i or T_i is at most $2B$ from the boundedness of Max2Sat-B, hence $\text{weight}(G_2) \geq \text{weight}(G_1)$. It is immediate to note that all cycles in $\langle V, E_2, F\rangle$ are also in $\langle V, E_1, F\rangle$, since in G_2 there cannot be any special cycle including edges in $C(i)$. This is due to the fact that when we add some internal or assignment edges, we remove all other edges incident on T_i or F_i. Since we have not created new cycles, the final graph G_2 is a trace alignment of $G = \langle V, E, F\rangle$. □

From Lemma 3.6.3 we can always get a truth assignment from a trace alignment.

Lemma 3.6.4. *Let $G = \langle V, E, F\rangle$ be a graph obtained reducing an instance of* Max2Sat-B*, and let $G_1 = \langle V, E_1, F\rangle$ be a trace alignment of G such that all vertices $D_{i,j}$ are adjacent to T_i or are all adjacent to F_i. Then it is possible to compute (in polynomial time) a trace alignment $G_2 = \langle V, E_2, F\rangle$ such that* $\text{weight}(G_1) \leq \text{weight}(G_2)$ *and for each clause c_l there is at least an undirected edge incident on one of the vertices encoding such clause.*

Proof. Let $G_1 = \langle V, E_1, F\rangle$ be a trace alignment, and pose initially E_2 equal to E_1. For each clause c_l such that the corresponding clause vertices are not incident on any undirected edges add the left lower edge to E_2. Since there is only one edge incident on L_i and no edge incident on R_i, there are no cycles including L_i or R_i. □

Lemmata 3.6.3, 3.6.4 allow us to take into account only trace alignments where all vertices $D_{i,j}$ are adjacent to T_i or to F_i, and each pair of vertices T_i, F_i has at least an undirected edge incident on such pair. In the following we will call such a trace alignment a *canonical* alignment. An immediate consequence of such Lemmata is that there exists an optimal alignment that is canonical. We have already stated that a canonical assignment encodes a truth assignment, in the following we will prove that a clause have two undirected edges incident on the vertices encoding it only if such clause is satisfied by the assignment. In order to prove that we need two useful properties of canonical alignments.

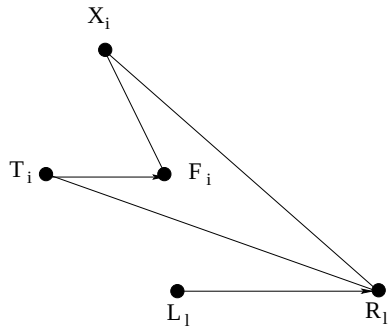

Figure 3.6: Cycle of the proof of Lemma 3.6.5

Proposition 3.6.5. *Let $G = \langle V, E, F\rangle$ be a graph obtained reducing an instance of* Max2Sat-B*, and let $G_1 = \langle V, E_1, F\rangle$ be a canonical trace alignment of G. Let c_l be a clause whose corresponding pair has two undirected edges incident on it in G_1. Then such edges are both left edges or are both right edges.*

Proof. Assume to the contrary that c_l is a clause such that L_l and R_l are both adjacent to an undirected edge in G_1. By Lemma 3.6.2 such edges cannot be both upper edges or both lower edges. W.l.o.g. we can assume that $(L_l, X_{\mathrm{rvar}(l)}), (R_l, T_{\mathrm{lvar}(l)}) \in E_1$. Since G_1 is a canonical alignment then $(D_{\mathrm{rvar}(l),1}, T_{\mathrm{lvar}(l)}) \in E_1$ or $(D_{\mathrm{rvar}(l),1}, F_{\mathrm{lvar}(l)}) \in E_1$. If $(D_{\mathrm{rvar}(l),1}, T_{\mathrm{lvar}(l)}) \in E_1$, $\langle L_l, (L_l, R_l), R_l, (R_l, T_{\mathrm{lvar}(l)}), T_{\mathrm{lvar}(l)}, (T_{\mathrm{lvar}(l)}, D_{\mathrm{lvar}(l),1}), D_{\mathrm{lvar}(l),1}, (D_{\mathrm{lvar}(l),1}, X_{\mathrm{lvar}(l)}), X_{\mathrm{lvar}(l)}, \ldots, X_{\mathrm{rvar}(l)}, (X_{\mathrm{rvar}(l)}, L_l), L_l\rangle$ is a cycle in G_1, otherwise $\langle L_l, (L_l, R_l), R_l, (R_l, T_{\mathrm{lvar}(l)}), T_{\mathrm{lvar}(l)}, (T_{\mathrm{lvar}(l)}, F_{\mathrm{lvar}(l)}), F_{\mathrm{lvar}(l)}, (F_{\mathrm{lvar}(l)}, D_{\mathrm{lvar}(l),1}), D_{\mathrm{lvar}(l),1}, (D_{\mathrm{lvar}(l),1}, X_{\mathrm{lvar}(l)}), X_{\mathrm{lvar}(l)}, \ldots, X_{\mathrm{rvar}(l)}, (X_{\mathrm{rvar}(l)}, L_l), L_l\rangle$ is a cycle in G_1; in both cases contradicting the assumption that G_1 is an alignment. □

Proposition 3.6.6. *Let $G = \langle V, E, F\rangle$ a graph obtained reducing an instance of* Max2Sat-B*, and let $G_1 = \langle V, E_1, F\rangle$ be a canonical trace alignment of G. Let c_l be a clause whose left edges are in G_1. Then $(D_{\mathrm{lvar}(l),1}, P(l, \mathrm{lexp}(l))) \in E_1$.*

Proof. Assume to the contrary that $(D_{\mathrm{rvar}(l),1}, P(l, \mathrm{rexp}(l))) \notin E_1$. Whenever $P(l, \mathrm{rexp}(l)) = \text{true}$, then $\langle R_l, (R_l, T_{\mathrm{lvar}(l)}), T_{\mathrm{lvar}(l)}, (T_{\mathrm{lvar}(l)}, F_{\mathrm{lvar}(l)}), F_{\mathrm{lvar}(l)}, (F_{\mathrm{lvar}(l)}, D_{\mathrm{lvar}(l),1}), D_{\mathrm{lvar}(l),1}, (D_{\mathrm{lvar}(l),1}, X_{\mathrm{lvar}(l)}), X_{\mathrm{lvar}(l)}, (X_{\mathrm{lvar}(l)}, R_l), R_l\rangle$ is a cycle, otherwise $P(l, \mathrm{rexp}(l)) = \text{false}$. In such case $\langle R_l, (R_l, X_{\mathrm{lvar}(l)}), X_{\mathrm{lvar}(l)}, (X_{\mathrm{lvar}(l)}, D_{\mathrm{lvar}(l),1}), D_{\mathrm{lvar}(l),1}, (X_{\mathrm{lvar}(l)}, T_{\mathrm{lvar}(l)}), T_{\mathrm{lvar}(l)}, (T_{\mathrm{lvar}(l)}, F_{\mathrm{lvar}(l)}), F_{\mathrm{lvar}(l)}, (F_{\mathrm{lvar}(l)}, R_l), R_l\rangle$ is a special cycle. □

Along the same lines it is immediate to prove that if both right edges of a clause c_l are in G_1, then $(X_{\mathrm{rvar}(l)}, P(l, \mathrm{rexp}(l))) \in E_1$.

Prop. 3.6.6 relates the cost of the trace alignment with the truth assignment. More precisely it says that for each clause that has two undirected edges incident on its vertices, since by Lemma 3.6.5 such edges are incident on one of the two clause vertices, such vertex encodes a variable that makes such clause satisfied.

From Prop. 3.6.6 and Lemma 3.6.1 follows a fundamental corollary:

Corollary 3.6.7. *Let x_i be a variable that appears positive and negative in an instance of* Max2Sat-B*, let $G = \langle V, E, F\rangle$ be the associated instance of* MTA *and let G_1 be trace alignment for G. Let c_1, c_2 be two clauses such that x_i appears positive in c_1 and negative in c_2. Then there cannot be two undirected edges incident on both vertices encoding the two occurrences of x_i.*

This allows us to state that the cost of the trace alignment implies an upper bound over the number of clauses that are satisfied by the assignment.

From Corollary 3.6.7 it is immediate to prove that

$$\mathrm{Opt}(G_1) = 2(2B+1)n + k + opt(\mathcal{S}) \leq (4B+9)\,\mathrm{Opt}(\mathcal{S}) \tag{3.1}$$

where the equality, together with the fact that G_1 is optimal, follows from Lemma 3.6.6 and the inequality follows from the fact that $\mathrm{Opt}(\mathcal{S}) \geq k/2$ and $n \leq k/2$.

The next step is to reconstruct a truth assignment α from a feasible solution of MTA. By Lemmata 3.6.3, 3.6.4 we can assume that such feasible solution is a canonical trace alignment. Let $G_1 = \langle V, E_1, F\rangle$ be a canonical trace alignment of $G = \langle V, E, F\rangle$, we will construct an assignment α for $\mathcal{S}$. Since G_1 is a canonical trace alignment $D_{i,1}$ is adjacent either to T_i or to F_i. Then $\alpha(x_i) =$ true if and only if $D_{i,1}$ is adjacent to T_i. By Lemma 3.6.6, it is immediate to prove that for each clause pair such that both left edges or both right edges are in E_1, the corresponding clause is satisfied by the assignment α. It follows directly that

$$\mathrm{Opt}(\mathcal{S}) - \mathrm{cost}(\alpha) \leq \mathrm{Opt}(G) - \mathrm{cost}(G_1) \tag{3.2}$$

To prove that our reduction is an L-reduction we have to prove that $\frac{\mathrm{Opt}(\mathcal{S})-\mathrm{cost}(\alpha)}{\mathrm{Opt}(\mathcal{S})} \leq \beta \frac{\mathrm{Opt}(G)-\mathrm{cost}(G_1)}{\mathrm{Opt}(G)}$ for a certain constant β.

From (3.1) and (3.2) posing $\beta = 4B + 9$ suffices.

Chapter 4

Comparing Sequences: Approximating the Alignment

4.1 Introduction

In Chapter 3 we have shown some negative results on the possibility of designing efficient exact or approximate solutions to the Multiple Sequence Alignment problem. In this chapter we will instead focus our attention to a restricted version of the problem which has strong biological relevance and we will describe a polynomial-time approximation scheme for such problem. In [58], Jiang *et al.* ask whether a particular restriction of the Multiple Sequence Alignment problem, namely the case of metric scoring matrix that we proved to be $\mathcal{NP}$-hard, is in $\mathcal{PTAS}$. In this chapter we will give a positive answer to a closely related question by showing that a restriction of Space-L Multiple Sequence Alignment does admit a polynomial-time approximation scheme; more precisely, such a polynomial-time approximation scheme exists when the ratio of the costs between each pairwise alignment is in a fixed interval.

Since Theorem 3.5.1 states that the Space-L Multiple Sequence Alignment is $\mathcal{APX}$-hard, we need to introduce a restriction in order to describe a $\mathcal{PTAS}$.

Definition 4.1.1 (Variability). Let $I = \langle\langle t_1, \ldots, t_k\rangle, (d_M, g)\rangle$ be an instance of the Space-L Multiple Alignment problem, then the *variability* of I, denoted by $v(I)$, is

$$v(I) = \max\{\frac{n\alpha(M) + Lg}{d^{opt}_{M,L}(t_i, t_j)} : 1 \leq i < j \leq k\}$$

where $\alpha(M)$ is the maximum value $d_M(a_1, a_2)$ between two different symbols $a_1, a_2 \in \Sigma \cup \{\Delta\}$.

Please note that the value $v(I)$ of the instance I can be computed in polynomial time. The SPACE-L MULTIPLE ALIGNMENT(σ) problem is the restriction of the SPACE-L MULTIPLE ALIGNMENT problem to instances I with $s(I) \leq \sigma$.

A few comments are in order. The most common multiple alignment problem in Molecular Biology is the alignment of homologous protein sequences from different species. For a pair $\langle t_i, t_j \rangle$ of such sequences, the corresponding aligned sequences $\langle a(i,j)t_i, a(i,j)t_j \rangle$ will be small only if the original sequences t_i, t_j are very similar, which usually happens only if the two species of origin have a relatively recent (in the time scale of evolution) common ancestor, and will be close to the average distance of random sequences if the species diverged a long time ago, or if the optimal alignment requires more than L spaces. For scoring matrices used in practice, the average distance of random sequences is usually a number of about the same order of magnitude as $n\alpha(M)$. The algorithms used in practice for multiple sequence alignment tend to perform well if all sequences are closely related to each other, while our first theorem covers one of the cases that are difficult in practice and quite common, namely the case where *none* of the sequences are closely related to each other.

4.2 The Approximation Scheme

The main results of this section is the following:

Theorem 4.2.1. *The* SPACE-L MULTIPLE ALIGNMENT*(σ) problem has a ptas for all constants σ.*

Note that in the above theorem, the scoring scheme (d_M, g) is considered part of the input, thus the theorem works for all affine scoring schemes, no matter whether the scoring function is a metric and the gap penalties are fixed or variable. This does not contradict the results about $\mathcal{APX}$-hardness from [60] though, since the variability of the instances used to obtain the latter results was not bounded.

Theorem 4.2.1 will be proved by reformulating it as a kind of facility location problem. To see the connection, suppose a communication network is to be set up in a country that consists of k regions. In each region, there should be one switchboard of the network, and each switchboard is to be connected by expensive, high quality cable to every other switchboard. If in each region there are several possible locations for the switchboard that are equally

good for the operation of the network within this region, then the locations of switchboards should be chosen in such a way as to minimize overall cost of cable between them. The question of choosing optimal locations for the switchboards can then be formalized as follows. The SWITCHBOARD LOCATION problem [Pb. 25] has as instance some disjoint sets $R_1, \ldots, R_k$ called *regions*, as well as a distance function d between all pairs of points x_i, x_j in $R_1 \cup \cdots \cup R_k$. The distance function gives strictly positive values whenever the two points are distinct. A feasible solution is a set $\langle x_1 \ldots, x_k \rangle$ of points such that $x_i \in R_i$ for $1 \leq i \leq k$. The problem asks for a feasible solution that minimizes $\sum_{1 \leq i < j \leq k} d(x_i, x_j)$.

While facility location problems with objective functions similar to those of SWITCHBOARD LOCATION have been studied for regions of the real line (see e.g. [93], [6]), we are not aware of any published results concerning the general formulation of SWITCHBOARD LOCATION given above.

We will discuss later how instances of SPACE-L ALIGNMENT(σ) can be mapped to suitable instances of SWITCHBOARD LOCATION in order to have a $(1+\epsilon)$ approximation algorithm. But first we have to introduce a restriction of SWITCHBOARD LOCATION similar to the one introduced for SPACE-L ALIGNMENT. Let $I = \{R_1, \ldots, R_k, d\}$ be an instance of the SWITCHBOARD LOCATION problem. We define the *spread* $s(I)$ of I as

$$s(I) = \frac{\max\{d(x_i, x_j) : 1 \leq i < j \leq k,\, x_i \in R_i, x_j \in R_j\}}{\min\{d(x_i, x_j) : 1 \leq i < j \leq k,\, x_i \in R_i, x_j \in R_j\}}.$$

It is immediate from the definition that $s(i) \geq 1$. For any pair of constants P, σ, the SWITCHBOARD LOCATION$_P(\sigma)$ problem is the SWITCHBOARD LOCATION problem restricted to instances of spread at most σ and where each region contains at most P points.

Theorem 4.2.2. *The* SWITCHBOARD LOCATION$_P(\sigma)$ *problem admits a ptas for all constants* P, σ.

Proof. Now let σ be a fixed constant, and suppose we have an instance I of the SWITCHBOARD LOCATION$_P(\sigma)$ problem, where $\{R_i : 1 \leq i \leq k\}$ are the regions of I, and $R_i = \{x_{i,j} : 1 \leq j \leq P\}$. (Since one can always add dummy points to the regions, we do not lose generality by assuming the regions to be *exactly* of cardinality P.) Let D be the value $\min\{d(x_{i,j}, x_{h,\ell}) : 1 \leq i < h \leq k,\, 1 \leq j, \ell \leq k\}$. Now we can formulate the SWITCHBOARD LOCATION

problem as PIP:

$$\begin{aligned} \text{minimize} \quad & \sum_{1\le h<i\le k,\ 1\le j,\ell\le k} \frac{d(x_{i,j},x_{h,\ell})}{D} y_{i,j}y_{h,\ell} && (4.1)\\ \text{subject to} \quad & \sum_j k y_{i,j} = k && i=1,\dots,k \\ & y_{i,j}\in\{0,1\} && i=1,\dots,k;\ j=1,\dots,P \end{aligned}$$

Please note that the total number of variables is at most kP. Since $s(I)\le\sigma$, all coefficients of the objective functions are between 1 and σ. Thus the PIP is σ-smooth.

Now suppose we want to find a solution to the SWITCHBOARD LOCATION problem that is within a factor of $(1+\varepsilon)$ of minimum. Setting $\delta=\frac{\varepsilon}{2P^2}$, and running the algorithm of [8] on the PIP defined above, we find a 0/1 solution that satisfies all constraints within an additive error of $O(\delta\sqrt{kP\log kP})$. Since for 0/1 solutions the left hand sides of our side constraints are multiples of k; for sufficiently large k we can assume that these side constraints are satisfied *exactly*. But then for each region R_i, exactly one of the numbers $y_{i,j}$ is equal to 1. Thus the corresponding $x_{i,j}$'s form a feasible solution of instance I of the SWITCHBOARD LOCATION problem, and the sum of the distances is within an additive error of $D\varepsilon\binom{k}{2}$. By the choice of D, the minimum value for the sum of all distances in any feasible solution of instance I of the SWITCHBOARD LOCATION problem cannot be less than $D\binom{k}{2}$, and thus we have found, in polynomial time, an approximation within a factor of $(1+\varepsilon)$. □

Now let us show how Theorem 4.2.2 implies Theorem 4.2.1. Suppose we are given an instance $I=\langle\langle t_1,\dots,t_k\rangle,(d_M,g)\rangle$ of the SPACE-L ALIGNMENT(σ) problem, and let $\varepsilon>0$. We want to find a space-L multiple alignment of these sequences that scores within $(1+\varepsilon)$ of optimum. Let $N=\lceil\frac{4L\sigma}{\varepsilon}\rceil$ and note that N is a constant. Let n be the length of the longest among the sequences $t_1,\dots,t_k$, and let $K=\lceil 2N+\frac{gN}{\alpha(M)}\rceil$.

First assume that $n\le K$. In this case we let R_i be the set of all sequences $x_{i,j}$ that are obtainable by inserting L spaces into t_i (at the beginning, end, or between symbols). This set contains at most $\binom{K+L}{L}$ elements. Note that $\binom{K+L}{L}$ is a constant that does not depend on the number of sequences k. Thus the family $\{R_i : 1\le i\le k\}$ together with the distances $d(x_{i,j},x_{i',j'})$ defined by the scoring scheme is an instance of the SWITCHBOARD LOCATION problem where the cardinality of all regions is bounded by the constant $\binom{K+L}{L}$. Feasible solutions of the SWITCHBOARD LOCATION problem are exactly all space-L alignments of our sequences, and the objective function of

the SWITCHBOARD LOCATION problem is exactly the SP-score of the alignment. Since the variability of the SPACE-L ALIGNMENT problem is bounded by σ, the spread of the corresponding SWITCHBOARD LOCATION problem that we constructed is also bounded by σ. Thus the ptas for SWITCHBOARD LOCATION$_{\binom{K+L}{L}}(\sigma)$ finds a solution within $(1+\varepsilon)$ of optimum.

Now assume that $n > K$. In this case we partition each sequence t_i into consecutive chunks $\langle s_{i,h} : 1 \le h \le N\rangle$, where the length of each chunk differs from $\frac{n}{N}$ by no more than 1. With each function $f : \{1,\ldots,N+1\} \to \mathbb{N}$ such that $\sum_{1\le i\le N+1} f(i) \le L$ we associate a sequence $t_{i,f}$ by inserting $f(h)$ space symbols to the left of each chunk $s_{i,h}$. In other words,

$$t_{i,f} = \Delta^{f(1)} s_{i,1} \Delta^{f(2)} s_{i,2} \ldots \Delta^{f(N)} s_{i,N} \Delta^{f(N+1)}$$

Now we let R_i be the set of all $t_{i,f}$ for functions $f : \{1,\ldots,N+1\} \to \mathbb{N}$ that satisfy $\sum_{1\le i\le N+1} f(i) \le L$. We run the approximation algorithm for SWITCHBOARD LOCATION$_{N+1}(\sigma)$ that finds a solution within $(1+\frac{\varepsilon}{3})$ on the instance $\langle R_1,\ldots,R_k,(d_M,g)\rangle$.

The algorithm returns a space-L multiple alignment $\langle t_{1,f_1},\ldots,t_{k,f_k}\rangle$ of the sequences $\langle t_1,\ldots,t_k\rangle$. We want to prove that the alignment $\langle t_{1,f_1},\ldots,t_{k,f_k}\rangle$ scores within $(1+\varepsilon)$ of optimum. Let $\langle at_1,\ldots,at_k\rangle$ denote the space-L multiple alignment with optimal SP-score. For each i, let $g_i : \{1,\ldots,N+1\} \to \mathbb{N}$ be the function such that for each $1 \le i \le k$ and $1 \le h \le N$, g_i is equal to the number of spaces in at_i inserted immediately to the left of the chunk $s_{i,h}$ or between characters of $s_{i,h}$. Instead of t_{i,g_i} we will write bt_i. Since $bt_i \in R_i$ for each i, we have

$$SP(\langle t_{1,f_1},\ldots,t_{k,f_k}\rangle) \le (1+\frac{\varepsilon}{3}) SP(\langle bt_1,\ldots,bt_k\rangle).$$

Since $1+\varepsilon > (1+\varepsilon/2)(1+\varepsilon/3)$ whenever $\varepsilon < 1$, it now suffices to show that

$$SP(\langle bt_1,\ldots,bt_k\rangle) \le (1+\frac{\varepsilon}{2}) SP(\langle at_1,\ldots,at_k\rangle).$$

Let us split the sequences at_i, bt_i into $N+1$ chunks $at_{i,h}$, $bt_{i,h}$ for $1 \le h \le N+1$ where $bt_{i,h} = \Delta^{g_i(h)} s_{i,h}$, $s_{i,N+1}$ is the empty string, and $|bt_{i,h}| = |at_{i,h}|$, so that $at_i = at_{i,1} at_{i,2} \cdots at_{i,N+1}$ and $bt_i = bt_{i,1} bt_{i,2} \cdots bt_{i,N+1}$. From the definition of g_i, whenever $g_i(h) = g_j(h) = 0$, the pairwise alignment $\langle at_{i,h}, at_{j,h}\rangle$ is the same as $\langle bt_{i,h}, bt_{j,h}\rangle$. Since at most L spaces are inserted into each sequence t_i, and since the maximum penalty on each chunk (excluding the newly inserted spaces) is equal to the length of the chunk (i.e. at most $\frac{n}{N}+1$) multiplied by $\alpha(M)$, and there are globally only L extra spaces, we get the inequality

$$d_M(bt_i, bt_j) \le d_M(at_i, at_j) + \alpha(M) L (2 + \frac{n}{N}) + Lg.$$

Since $n > 2N + \frac{gN}{\alpha(M)}$ and $\alpha(M)\frac{Ln}{N} \geq 2L\alpha(M) + Lg$, we get

$$d_M(bt_i, bt_j) \leq d_M(at_i, at_j) + \alpha(M)n\frac{2L}{N}.$$

By the choice of N, the latter yields

$$d_M(bt_i, bt_j) \leq d_M(at_i, at_j) + \frac{\alpha(M)n\varepsilon}{2\sigma}.$$

Since the spread of our instance was assumed to be at most σ, we have $d_M(at_i, at_j) \geq \frac{\alpha(M)n}{\sigma}$, and we get

$$d_M(bt_i, bt_j) \leq d_M(at_i, at_j)(1 + \frac{\varepsilon}{2}),$$

as required.

The following results on hardness of Switchboard Location problems are not covered by Theorem 3.5.1.

Theorem 4.2.3. *The* Switchboard Location$_2(\sigma)$ *problem is* $\mathcal{NP}$*-hard for each fixed constant* $\sigma > 1$.

Proof. Let $\sigma > 1$. Since the number of instances of Switchboard Location$_2(\sigma)$ increases with σ, we may without loss of generality assume that $\sigma \leq 2$. We prove the theorem by reducing the Max-Cut problem to Switchboard Location$(_2(\sigma)$. Given a graph $G = \langle V, E\rangle$ with vertices $V = \{v_1, \ldots, v_k\}$, construct a metric space $X = \{x_1, \ldots, x_k, y_1, \ldots, y_k\}$ as follows: For $i \neq j$, we let $d(x_i, x_j) = d(y_i, y_j) = 1$. If $\{v_i, v_j\} \in E$, then $d(x_i, y_j) = \sigma$; if $\{v_i, v_j\} \notin E$, then $d(x_i, y_j) = 1$. (Note that for our choice of σ, the distance function is actually a metric.) For $1 \leq i \leq k$, the region R_i is defined as $\{x_i, y_i\}$. This gives us an instance I of the Switchboard Location$(_2(\sigma))$ problem. Every solution $\bar{x}$ of I induces a partition $\langle V_x, V_y\rangle$, where $V_x = \{v_i : x_i \in \bar{x}\}$ and $V_y = \{v_i : y_i \in \bar{x}\}$. If $c_{\bar{x}}$ denotes the size of the cut induced by the partition $\langle V_x, V_y\rangle$, then the measure of $\bar{x}$ is equal to $\binom{k}{2} + (\sigma - 1)(|E| - c_{\bar{x}})$, and the theorem follows from $\mathcal{NP}$-hardness of the Max-Cut problem. □

Theorem 4.2.4. *The* Switchboard Location$_2$ *problem is* $\mathcal{APX}$*-hard.*

In view of our observation that Gap-0-1 Alignment is a special case of Switchboard Location, Theorem 4.2.4 is a corollary of Theorem 3(c) of [60].

Chapter 5

Comparing Sequences: Approximating Sub- and Supersequences

5.1 Introduction

The problems of computing the Longest Common Subsequence (LCS) and the Shortest Common Supersequence (SCS) of a set of sequences are two well-known $\mathcal{NP}$-hard problem [75]. The problem of computing the longest common subsequence of two sequences has been deeply investigated (see the survey in [80]), and a number of algorithms have been proposed in order to improve the running time for typical instances [5, 81, 56, 83], but all these algorithms still have a $O(n^2)$ time complexity in the worst case. The only algorithm that has broken this barrier is the one by Masek and Paterson [76] based on the Four Russians' technique [7]; their algorithm has $O(n^2/\log n)$ time complexity.

Let us now consider the general problem of computing the longest common subsequence of k sequences of length n; even this problem has been studied by several authors, proposing algorithms based on the dynamic programming technique [54, 49], but they were not able to substantially improve the $O(n^k)$ time and space cost in the worst case. These requirements however are unacceptable even for small k in most situations, since the parameter n is usually extremely large in practice (e.g. text-editing, analysis of biological sequences, where n can be greater than 500). Consequently people have moved their interest towards the search for heuristic algorithms to find an approximate solution for the LCS problem. But, also in this framework, negative results have been provided for the LCS problem over an arbitrary

alphabet: Jiang and Li [59] proved that the problem has no polynomial time approximation algorithm with performance ratio n^{δ}, for any constant $\delta < 1$, unless $\mathcal{P} = \mathcal{NP}$. Despite the discouraging results, it has been proved that the LCS problem over a fixed alphabet can be indeed very well approximated on the average by using a simple algorithm called Long Run [59]: it gives as a solution of the LCS for a set S of sequences, the sequence σ^l, such that σ^l is the longest common subsequence of S of the form σ^l, for $\sigma \in \Sigma$. The Long Run algorithm works quite well in practice since it can provide a solution which has a length close to the optimum. More precisely, in [59], Jiang and Li proved that given n input sequences generated randomly according to the uniform probability distribution, all of the same length n and over fixed alphabet, then the Long Run algorithm approximates the longest common subsequence with an $O(n^{1/2+\epsilon})$ expected additive error (the additive error is given as the difference between the length of the optimum and of the approximation) for any $\epsilon > 0$. Anyway this does not imply that Long Run performs well on instances made of a relatively small number (e.g. less than 30) of long sequences (more than 100 characters each). Moreover, even though Long Run gives a solution whose length is a good approximation of the optimal one, it is quite evident that this algorithm presents some deep shortcomings that are more relevant in the molecular biology setting. In fact the Longest Common Subsequence problem is used to model the comparison of biological sequences [48] and the actual longest common subsequence of a set of biological sequences should represent some regions that are common to all sequences, hence the longest common subsequence is likely to point out some highly conserved regions. Anyway it is not hard to realize that the subsequence given by Long Run seldom has any biological meaning, as it contains only one distinct symbol.

Another drawback of using Long Run to compute a subsequence is that it does not give the optimal solution even when the instance contains only one sequence (the optimal solution is trivially the sequence in the instance). Moreover it is possible to prove that there are instances consisting of one sequence where Long Run gives a subsequence whose length is $1/|\Sigma|$ of that of the actual longest common subsequence, where $|\Sigma|$ is size of the alphabet Σ. Actually, it is not difficult to prove that $|\Sigma|$ is also the guaranteed performance ratio of Long Run.

The Shortest Common Supersequence (SCS, in short) allows to model different computational problems and hence it has several applications in various areas, mainly scheduling [43], data compression [91], query optimization [86], text-editing and DNA sequence assembly in Computational Biology [85].

The problem SCS of a set $\mathcal{S}$ of sequences consists in the computation

of a common supersequence of $\mathcal{S}$ that has minimum length; whenever the instances can contain at most k sequences of maximum length n, then the problem can be solved in $O(n^k)$ time and space by dynamic programming. Anyway this approach is not feasible in practice, except for small values of n and k. Just as for Longest Common Subsequence such approach can be improved by exploiting the Four Russians' technique, resulting in a $O(\log n)$ speedup.

When the instances are not required to contain only a constant number of sequences, the problem is intractable even in the restricted case of binary alphabet, as proved by Maier [75]: this result has shifted the attention towards the investigation of approximation algorithms for the problem. In this direction, Irving and Fraser [55] analyzed the quality of the solutions returned by two approximation algorithms called Tournament and Greedy, proving some negative results. More precisely they have shown that such algorithms cannot match the worst-case approximation ratio of a trivial algorithm which achieves $|\Sigma|$-approximation ratio over a fixed alphabet Σ. In fact, a $|\Sigma|$-approximation for the SCS problem over an instance consisting of sequences of length at most n over the fixed alphabet $\Sigma = \{\sigma_1, \cdots, \sigma_q\}$ can be achieved simply by producing the common supersequence $(\sigma_1\sigma_2\cdots\sigma_q)^n$: such algorithm is called Alphabet. More recent results by T. Jiang and M. Li point out that the SCS problem is also hard to approximate whenever the alphabet is not fixed, while the same authors proved that the average performance ratio of the Alphabet algorithm is quite satisfactory asymptotically. After the work by Timkovsky [94] that proposed a small improvement to the trivial approximation factor $|\Sigma|$ achieved by Alphabet, to our knowledge no new approximation algorithm with a guaranteed factor has been proposed. Getting any improvement in this direction appears to be a challenging question, mainly because the known approximation algorithms are not very useful in practice for two reasons: the answer that Alphabet produces depends only on the alphabet size and on the maximum length of a sequence in the instance, and it is not really dependent on the set $\mathcal{S}$ of sequences of the instance, consequently it is unable to get close to an optimum solution in all cases. On the other hand the heuristics proposed by Irving and Fraser can produce a solution whose size is very far from the the optimal one (more precisely there are instances where Tournament and Greedy produce a solution whose approximation ratio over binary alphabet is not bounded by any constant). In this chapter we will describe an approximation algorithm for LCS and one for SCS. Both algorithms are improvements of an algorithm over binary alphabet that has been described in [23]. The main idea is that the optimal common supersequence can be thought as built of blocks, where each block is a substring containing only one distinct symbol of the alphabet. If we knew

the alphabet symbol in each block of the optimal solution then it is likely that some strategy for computing the approximate lengths of the blocks may lead to a good solution. In particular in the case of binary alphabet it is possible to restrict the possible choices for such alphabet symbols to a set whose size is polynomial in the size of the input, moreover such set can be easily computed. How to compute such set and the strategy to expand or reduce the lengths of the blocks is the core of our algorithms.

The algorithm for the LCS problem is called Expansion, and has a guaranteed performance ratio $|\Sigma|$, hence matching the one of Long Run, even though such bound is not tight for our algorithm. Moreover we will show experimentally that the average performance of our algorithm is definitely better than the one of Long Run. The algorithm is based on a technique similar to the one initially proposed for the Shortest Common Supersequence problem in [23].

The experiments have been executed on two main groups of instances: one consisting of sequences of length between 90 and 100 and the other consisting of sequences of length between 400 and 500.

The goals of the two experiments are different. In the first one we have instances containing random sequences and we compare the approximate solution with the optimum one, so that we can compute exactly the performance ratio of the algorithm on these instances. The instances of the second experiment contain a greater number of longer sequences, moreover the sequences are fairly homologous in order to point out the behavior of the algorithm over biological sequences. In fact each instance contains sequences generated from a random sequence simulating an evolution according to the Jukes-Cantor [74] model of evolution, hence are sufficiently representative of the sequences usually found in practice. Instances generated in this way contain sequences of at least 400 symbols, thus ruling out the possibility of comparing the approximate solution with the exact solution, since a dynamic programming algorithm for such instances would not be feasible. Nonetheless we are able to provide an upper bound on the performance ratio of the algorithm, based on the fact that a common subsequence of a set $\mathcal{S}$ of sequences cannot be longer than the shortest sequence in $\mathcal{S}$. Since such bound is trivial we expect that the performance ratio of our algorithm is definitely better than the one we have obtained.

A comparison between the results obtained from the two main experiments has allowed to confirm that the Expansion algorithm achieves an average performance ratio less than 1.08, while the Long Run has an average performance ratio which is at least 1.34.

In this chapter, we propose a new approximation algorithm (Reduce-Expand) for the SCS problem over a fixed alphabet (binary and of arbitrary

size) that has the same guaranteed approximation of the trivial one, but overcomes its negative limits. Since the practical use of such an algorithm is the main issue, we compare experimentally our new algorithm with Alphabet, giving particular attention to instances representing real-world cases.

In this direction one of the objectives is to check if the performance of the algorithm does not degrade for sequences that have more than 100 characters, as this is the usual size of sequences occurring in practice when we have to deal with biological sequences. In fact we tested our algorithm for sequences up to 500 characters, and we have got quite satisfactory results.

In all our experiments it has not been possible to compute the exact solutions; in fact, as pointed out previously, the dynamic programming approach is not feasible in practice due to time and space constraints. Consequently we had to use a lower bound on the length of the optimal solution, in place of the optimum, in order to compute an estimated approximation ratio achieved by the algorithms examined.

5.2 Preliminaries

In this section we will introduce some fundamental notions that will be used in this chapter.

Definition 5.2.1 (Basic Sequence). A *basic sequence* of length k over Σ is a sequence $\sigma_1 \cdots \sigma_k$, with $\sigma_i \in \Sigma$ for all $1 \leq i \leq k$ and $\sigma_i \neq \sigma_{i+1}$ for all i, $1 \leq i < k$.

Please note that there are $|\Sigma|(|\Sigma|-1)^{k-1}$ basic sequences of length k over an alphabet Σ.

Definition 5.2.2 (Substream). Let $\mathcal{S}$ be a set of sequences, then a *substream* of $\mathcal{S}$ is a basic sequence that is a common subsequence of $\mathcal{S}$.

Analogously we define a *superstream* as a basic sequence that is a common supersequence of $\mathcal{S}$. When $\mathcal{S}$ contains only a sequence s, by substream (superstream) of s we mean the substream (superstream) of $\{s\}$.

Definition 5.2.3 (Factorization). Let s be a sequence, then *factorization* into blocks of s is the $2k$-tuple $\langle \sigma_1, j_1, \cdots, \sigma_k, j_k \rangle$, where $\sigma_1 \cdots \sigma_k$ is a substream of s, $j_l > 0$, for $1 \leq l \leq k$ and $\sigma_1^{j_1} \cdots \sigma_k^{j_k} = s$.

The sequence $\sigma_i^{j_i}$ introduced in Def. 5.2.3 is called the ith *block* of s, while the *run* of a sequence is the maximum length of one of its blocks.

The following set of instances will be used as an example. Let $\mathcal{S}$ be the set {`aabbaabcbc`, `abbbbcbbabbaa`, `bcabbbab`}, then the factorizations

of the sequences in $\mathcal{S}$ are respectively $\{(a,2)(b,2)(a,2)(b,1)(c,1)(b,1)(c,1)$, $(a,1)(b,4)(c,1)(b,1)(a,1)(b,2)(a,2)$, $(b,1)(c,1)(a,1)(b,3)(a,1)(b,1)$.

The basic sequences of length 2 over the alphabet $\{\mathtt{a},\mathtt{b},\mathtt{c}\}$ are `ab`, `ac`, `ba`, `bc`, `ca`, `cb`, moreover `ab` is a substream of $\mathcal{S}$ while `ca` is not. Note that the sequences in $\mathcal{S}$ have run 2, 4 and 3 respectively.

We introduce now the formal definitions of the two problems studied in this chapter.

Problem 4 (Longest Common Subsequence).
Instance: a sets $\mathcal{S}$ of sequences.
Solution: a *common subsequence* of $\mathcal{S}$, that is a sequence s such that s can be obtained from each sequence in $\mathcal{S}$ by removing some characters of such sequence.
Goal: to maximize the length of the subsequence.

Problem 5 (Shortest Common Supersequence).
Instance: a sets $\mathcal{S}$ of sequences.
Solution: a *common supersequence* of $\mathcal{S}$, that is a sequence s such that s can be obtained from each sequence in $\mathcal{S}$ by inserting some characters of such sequence.
Goal: to minimize the length of the subsequence.

It is possible to solve in polynomial time both LCS and SCS problems for instances containing a constant number of sequences via dynamic programming [85], but it would require $O(n^k)$ time and space, where n is the length of the longest sequence in $\mathcal{S}$. Hence this algorithm is feasible only for small values of n and k. An improvement of such algorithm based on the Four Russians' technique [7] is possible and the time complexity would be $O(n^k/\log n)$, but the hidden constants would not make such an algorithm more appealing than the one in [85] for practical cases.

In [53] Hirschberg described a linear space algorithm for computing the length of an optimal solution to the LCS problem over 2 sequences: such algorithm can be generalized to compute the optimum value of the objective function our problems, leading to an $O(n^k)$ time and $O(n^{k-1})$ space algorithm, but with a time complexity which is roughly twice as much as the one of the dynamic programming algorithm described in [85].

When the number k of sequences is not fixed, but it is a part of the instance, the time complexities of the algorithms before mentioned are not polynomial. In fact it is known from [75, 82] that in such case the decision version of the LCS problem is $\mathcal{NP}$-complete also over binary alphabet. Hence the subsequent step is to look for efficient approximation algorithms; an example of such an algorithm is the Long Run.

It is rather straightforward to describe the Long Run algorithm. Given a set $\mathcal{S}$ of sequences over the alphabet Σ, let $\text{occur}_\sigma(s)$ be the number of occurrences of σ in a sequence s. Then, for each symbol $\sigma \in \Sigma$, let c_σ be the minimum value of $\text{occur}_\sigma(s)$ over all sequences $s \in \mathcal{S}$. Long Run returns α^{c_α} where α is the symbol of Σ maximizing c_α.

While the Long Run algorithm over fixed alphabet gives an approximate common subsequence with a guaranteed performance ratio of $|\Sigma|$, it is easy to note that such subsequence contains only one symbol of the alphabet Σ, so it is rather useless in practice.

Now we can describe the Alphabet for the SCS problem. Let $\mathcal{S}$ be a set of sequences of maximum length n over the alphabet $\Sigma = \{\sigma_1, \ldots, \sigma_q\}$ then a common supersequence of $\mathcal{S}$ is $(\sigma_1\sigma_2 \cdots \sigma_q)^n$. The algorithm that returns such a common supersequence is called Alphabet, it is immediate to note that it achieves a $|\Sigma|$ approximation ratio.

Two more approximation algorithms for the SCS problem have been presented in literature and are used in this chapter, even though such algorithms cannot have a constant approximation ratio on fixed alphabet: these are called Greedy and Tournament. Let $\mathcal{S} = \{s_1, \ldots, s_m\}$ be a set of sequences, then Greedy returns the sequence $\text{Greedy}(\text{SCS}(s_1, s_2), s_3, \ldots, s_m)$ computed recursively until there is only one sequence in the set. Let $\mathcal{S} = \{s_1, \ldots, s_m\}$ be a set of sequences, where we assume that $m = 2^p$, with p integer. Then Tournament returns the sequence $\text{Tournament}(\bigcup_{i=i}^{i=m/2} \text{SCS}(s_{2i-1}, s_{2i}))$, computed recursively until there is only one sequence in the set.

5.3 The Algorithm for the LCS

In this section we propose a new approach for computing an approximate solution of the LCS problem. Our algorithm is based on the following main observation: a longest common subsequence of a set $\mathcal{S}$ of sequences has a number b of blocks, with $1 \leq b \leq B$, for B the length of a substream for $\mathcal{S}$ of maximum length. A strategy to obtain a good approximation consists of expanding substreams of different lengths, so that we obtain a solution with factorization similar to the one of a longest common subsequence. Since the number of possible expansions of a stream (that is assigning an exponent to each symbol of the stream) is exponential in n, we develop a polynomial time strategy which takes into account the variation in the size of the blocks inside the factorizations of the sequences in $\mathcal{S}$: this behavior is realized by the Expand procedure.

The Expand procedure receives as input a set $\mathcal{S}$ of sequences and a stream e. It scans e from left to right, block by block, and at each time it tries

to obtain a new common subsequence of $\mathcal{S}$ by doubling the length of the examined block. Then, it continues to expand the sequence from left to right in this way, until no block of the sequence can be doubled. Finally, it examines again the sequence from left to right, trying each time to enlarge the size of each block, until the sequence cannot be further expanded, since otherwise the property of being a common subsequence of $\mathcal{S}$ is violated.

Expand$(R_0, \ldots, R_m, t)$
Input: A set $\mathcal{S}$ of sequences and a basic sequence $e = \sigma_1 \cdots \sigma_{|e|}$
For $1 \leq j \leq |e|$ do
$k_i := 1$
EndFor
Repeat
  test := false
  For $1 \leq j \leq |e|$ do
    If $\sigma_1^{k_1} \cdots \sigma_j^{2k_j} \cdots \sigma_{|e|}^{k_{|e|}}$ is a subsequence of S then
      $k_j := 2 * k_j$
      test := true
    EndIf
  EndFor
Until test = false
k_j:= max$\{\alpha\}$ such that $\sigma_1^{k_1} \cdots \sigma_j^{\alpha} \cdots \sigma_{|e|}^{k_{|e|}}$ is a common subsequence of S
Return$(\sigma_1^{k_1} \cdots \sigma_j^{\alpha} \cdots \sigma_{|e|}^{k_{|e|}})$.

Figure 5.1: An outline of the Expand algorithm

Computing the longest size of a block can be implemented with a binary search. A simple example will help the reader in understanding the procedure. Let $\mathcal{S}$ be the set $\{\mathtt{a}^4\mathtt{b}^3\mathtt{a}^4\mathtt{b}^2\mathtt{a},\ \mathtt{a}^3\mathtt{b}^4\mathtt{a}^4\mathtt{b}^3\}$, then the sequences of the expansions of the stream abab computed inside Expand is: abab, $\mathtt{a}^2$bab, $\mathtt{a}^2\mathtt{b}^2$ab, $\mathtt{a}^2\mathtt{b}^2\mathtt{a}^2$b, $\mathtt{a}^2\mathtt{b}^2\mathtt{a}^2\mathtt{b}^2$, $\mathtt{a}^2\mathtt{b}^2\mathtt{a}^4\mathtt{b}^2$, $\mathtt{a}^3\mathtt{b}^2\mathtt{a}^4\mathtt{b}^2$, $\mathtt{a}^3\mathtt{b}^3\mathtt{a}^4\mathtt{b}^2$. The latter sequence is the one returned by the procedure.

The Expand procedure gives a way to obtain a common subsequence from a stream. The basic idea on which the Expansion algorithm relies is computing the longest common subsequence of a set $\mathcal{S}$ of sequences from a set T of streams of $\mathcal{S}$, where each sequence x in T is expanded by the Expand procedure. At the end, a set C of common subsequences for $\mathcal{S}$ is obtained: then the sequence of maximum length in C is the solution returned by the algorithm.

Computing a set of streams requires two different approaches depending on the size of the alphabet; in fact the set of all streams over a binary alphabet is polynomial in the size of the instance, hence it is possible to compute all of them. This is not true in the case of arbitrary alphabets, where we have to use some sort of heuristic to compute a subset of all possible substreams.

5.3.1 Binary alphabet

Let us first illustrate the main body of the algorithm in the case of binary alphabet: all substreams of the instance are computed, successively the Expand procedure expands such substreams. The best sequence among all returned by the various calls to Expand is the output of the algorithm.

ExpansionBinary($\mathcal{S}$)
Input: A set $\mathcal{S}$ of sequences.
Let B be the minimum length of a stream of $\mathcal{S}$.
For $1 \leq t \leq B$ do
  $z_t :=$ Expand$(\mathcal{S}, \sigma_1\sigma_2\cdots\sigma_t)$, where $\sigma_1 = 0$ and $\sigma_1\cdots\sigma_t$ is a stream of $\mathcal{S}$
  $w_t :=$ Expand$(\mathcal{S}, \sigma_1\sigma_2\cdots\sigma_t)$, where $\sigma_1 = 1$ and $\sigma_1\cdots\sigma_t$ is a stream of $\mathcal{S}$
EndFor
Let cs be the longest sequence in $\{z_t : 1 \leq t \leq B\} \cup \{w_t : 1 \leq t \leq B\}$
Return(cs)

Figure 5.2: An outline of the Expand algorithm over binary sequences

Note that given the instance $\mathcal{S} = \{a^4b^3a^4b^2a,\ a^3b^4a^4b^3\}$ of the example in the previous section, the Expansion algorithm computes on input $\mathcal{S}$ an approximate solution of length at least 12, while Long Run returns the subsequence a^7.

5.3.2 Arbitrary alphabet

As stated previously, in the case of an arbitrary alphabet there is an additional difficulty with respect to the binary case in applying the technique of the Expand procedure, which is given by the fact that the number of the streams of the instance may be exponential in the length of the sequences in the instance. Thus we need to develop a heuristic to choose a subset of streams that will be expanded by the Expand procedure. The heuristic we give consists of two steps: given a set $\mathcal{S}$ of sequences, we initially compute all streams of $\mathcal{S}$ of maximum length 2. Then we apply a *greedy* algorithm to S to obtain a stream st of $\mathcal{S}$. All substrings of st are streams of $\mathcal{S}$ that are

expanded by the Expand procedure. The algorithm is stated below, where we assume that an exact longest common subsequence of two sequences is computed by the dynamic programming algorithm in [30].

```
Greedy(𝕊)
Input: A set 𝕊 = {s_1, ..., s_|S|} of sequences.
If 𝕊 contains only one sequence then
   Returnthe sequence in 𝕊
EndIf
If |𝕊| = 2 then
   lcs := LCS(s_1, s_2)
   Returnthe longest stream of lcs
EndIf
For each s_i ∈ 𝕊 do
   Replace s_i with the longest stream of {s_i}
EndFor
For each i, j such that 1 ≤ i < j ≤ |𝕊| do
   s_{i,j} = |Greedy({s_i, s_j})|
   temp[i, j] := |s_{i,j}|
EndFor
Let i, j be the pair of indices that maximizes temp[i, j]
Return(Greedy(𝕊 − {s_i, s_j} ∪ s_{i,j})
```

Figure 5.3: An outline of the Greedy algorithm

5.4 Theoretical Analysis

Given a set $\mathbb{S}$ of k sequences of length n, testing if a sequence is a common subsequence of $\mathbb{S}$ can be done in $O(nk)$ time. A careful implementation of Expand requires exactly 1 unsuccessful test and at most $O(\log n)$ successful tests for each block of the stream e. Consequently the total time to compute the exponent of each block is $O(\log n)$, as the last step is a binary search. Since there are at most n blocks, the time complexity of Expand is $O(kn^2 \log n)$.

The ExpansionBinary algorithm contains at most $2n$ calls to Expand, consequently the algorithm has $O(kn^3 \log n)$ time complexity.

The analysis of ExpansionArbitrary is slightly more involved. A careful implementation of the Greedy procedure has $O(k^2n^2)$ time complexity when it receives as input the set $\mathbb{S}$ of sequences, while it requires $O(n^2)$ when it receives 2 sequences of maximum length n as input. The substrings of

```
ExpansionArbitrary(𝕊)
Input: A set 𝕊 of sequences.
Streams := {s : s is a stream of 𝕊 of length ≤ 2}
Add to Streams all substrings of Greedy(𝕊)
Initially cs is the empty word;
For each z ∈ Streams do
  w = Expand(𝕊, z)
  If w is longer than cs then
      cs := w
  EndIf
EndFor
Return(cs)
```

Figure 5.4: An outline of the Expand algorithm over arbitrary alphabet

the output of Greedy($\mathcal{S}$) are at most n^2, as such output must be a common subsequence of $\mathcal{S}$, consequently the streams that are expanded are at most $|\Sigma|^2 + n^2$. It follows that the time complexity of the algorithm is $O((|\Sigma|^2 + n^2)(kn^2 \log n) + n^2k^2) = O(((|\Sigma|^2 + n^2) \log n + k)kn^2)$, for $|\Sigma| > 2$. Consequently when $n > k$ and $n > |\Sigma|$, as in the instances of our experiments the time complexity is $O(kn^4 \log n)$.

Observe that we can describe the Long Run algorithm as a restricted case of the Expansion algorithm, thus showing that the solution computed by such algorithm over a set $\mathcal{S}$ of sequences can never be better that the solution computed by our Expansion algorithm.

```
Long Run(𝕊)
Input: A set 𝕊 of sequences.
Initially lcs is the empty word;
For each z ∈ Σ do
  w = Expand(𝕊, z)
  If w is longer than cs then
      cs := w
  EndIf EndFor
Return(cs)
```

Figure 5.5: An outline of the Long Run algorithm

Since the set of streams expanded by Long Run is a subset of those expanded by Expansion, it is immediate to note that the value of the solution

returned by Expansion is always at least as good as the one returned by Long Run.

Corollary 5.4.1. *The Expansion algorithm has $|\Sigma|$ guaranteed performance ratio.*

5.5 Experimental Analysis

In this section, we describe the results of two different groups of experiments we have developed to study the average case behavior of our algorithm. The first group contains instances with 4 random sequences of length between 90 and 100, where the runs of the sequences are generated according to the uniform distribution. For these experiments we have been able to compare the approximate solution computed by the Expand algorithm with the one returned by Long Run and an exact longest common subsequence computed by the dynamic programming algorithm. Besides a natural comparison based on the lengths of the solutions, we propose a measure representing how much the approximate solution resembles an optimal one. To achieve this goal we introduce a new parameter, called *similarity*, which is defined by the following formula relating the number $N(\mathcal{S})$ of blocks of a longest common subsequence over the set $\mathcal{S}$ of sequences to the number $A(\mathcal{S})$ of blocks of the approximate solution: $\sqrt{E((N(\mathcal{S}) - A(\mathcal{S}))^2)}$, where $E()$ is the expectation. Please note that it is desirable to have an algorithm which achieves a small similarity index.

The second group of experiments consists of instances with 5, 10 or 20 sequences whose lengths range from 400 to 500. Moreover the sequences in each set $\mathcal{S}$ are generated from a random sequence $base(\mathcal{S})$ on which we simulated an evolution process according to the Jukes-Cantor model [74]. Moreover in our simulation only deletions and substitutions were allowed. In this way we can easily generate instances that are representative of the ones usually found in practice. It has not been possible to compute the exact solution of the LCS over such instances, due to both time and space constraints[1], hence we have compared the length of our approximate solution with that of the shortest sequence in $\mathcal{S}$, which is an upper bound on the length of a longest common subsequence. Consequently the ratios stated in Tables 5.5, 5.5, 5.5 are upper bounds of the actual ones. Since we did not compute an actual longest common subsequence, it did not make sense to compute the similarity

[1]The space constraints are especially demanding, as a careful implementation of Hirschberg's algorithm for instances of 5 sequences of length 400 still requires at least 16Gbytes of memory

index, hence in this case we only dealt with the performance ratio. A fundamental parameter in all experiments is the maximum run of the sequences in the instances.

The results of the first group of experiments are summarized in Table 5.1, where the the average performance ratio, the standard deviation of the performance ratio and the similarity index achieved by Expansion algorithm and Long Run are represented. The sequences of the experiments are over binary alphabet and are obtained by generating sequences of integer values according to a uniform distribution in the range between 1 and the maximum run: each of such sequence gives the lengths of the blocks. In particular this group of experiments contains input sequences with length between 90 and 100.

	Max Run	2	6	12	18
Expansion	Average R_A	1.0715	1.063	1.0496	1.0416
	Std. Dev.	0.0205	0.0299	0.0374	0.0412
	Similarity	6.2914	3.3651	2.2081	1.531
Long Run	Average R_A	1.6224	1.4176	1.3717	1.3456
	Std. Dev.	0.0428	0.0742	0.1083	0.1366
	Similarity	50.779	15.9346	7.3655	4.5989

Table 5.1: Experiments over sequences of length between 90 and 100

In Table 5.1 it is possible to note that the Expansion algorithm has outperformed the Long Run algorithm for each value of the maximum run parameter giving, on the average, a better solution both in terms of the length of the approximate solution and in terms of the number of blocks of the approximate solution with respect to the number of blocks of an actual longest common subsequence. In fact the Long Run algorithm has an average performance ratio which is always at least 1.34, while the Expansion algorithm has never achieved an average performance ratio greater than or equal to 1.08. The analysis of the similarity index shows clearly that the Expansion algorithm compute an approximate solution which is more similar to an actual longest common subsequence, as the similarity index of Long Run is always at least three times as the one of Expansion.

The second group of experiments have been run over sequences of maximum length 500 and alphabets of sizes 4 and 20, that is using the alphabet of DNA and protein sequences respectively. The results that we have obtained are very encouraging, since the Expansion algorithm has never had an average performance ratio larger than 1.16.

Studying how the performance of our algorithm depends on the size of

Min. length	Max Run 8	16	32
400	1.15024	1.08671	1.05411
450	1.04483	1.0266	1.0171
480	1.00787	1.00409	

20 sequences, alphabet size 4

Min. length	Max Run 8	16	32
400	1.15664	1.08485	1.05187
450	1.04413	1.02766	1.01634
480	1.00757	1.00397	1.00209

20 sequences, alphabet size 20

Table 5.2: Results of the experiment over 20 sequences of maximum length 500

Min. length	Max Run 8	16	32
400	1.13848	1.07975	1.04457
450	1.03437	1.0206	1.0117
480	1.00565	1.00333	1.00124

10 sequences, alphabet size 4

Min. length	Max Run 8	16	32
400	1.13308	1.08091	1.04814
450	1.03421	1.02097	1.01232
480	1.00508	1.00231	1.00129

10 sequences, alphabet size 20

Table 5.3: Results of the experiment over 10 sequences of maximum length 500

Min. length	Max Run 8	16	32
400	1.11052	1.05749	1.03417
450	1.01994	1.01274	1.00692
480	1.00361	1.00154	1.00074

5 sequences, alphabet size 4

Min. length	Max Run 8	16	32
400	1.10212	1.05383	1.03377
450	1.02046	1.01149	1.00786
480	1.00271	1.05383	1.00067

5 sequences, alphabet size 20

Table 5.4: Results of the experiment over 5 sequences of maximum length 500

the alphabet has been one of the goals of this thesis. While it is obvious that the algorithm should perform better on alphabets of smaller size, we found out that the performance of the algorithm smoothly get worse when alphabet size increases from 4 to 20 (see Table 5.5).

Another goal of our experiments has been determining how the performance ratio is influenced by the number of sequences in each instance. In this case the degradation of the performance with respect to the size of the instances is noticeable, but it is still smooth. The Fig. 5.6 reports only part of the results shown in Table 5.5, pointing out the dependence of the performance of the algorithm with respect to the minimum length of the sequences and the number of sequences in the instance. Such results are from experiments over sequences with maximum run 16 and alphabet size 4 are represented. Anyway, the trends of the results we have obtained for different maximum runs and different alphabet sizes are similar to the ones reported in such figure.

The third goal of the experiments has been to determine how the performance ratio of the algorithm depends on the maximum run of the sequences. An interesting fact that it is possible to devise from the results of the experiments is that the performance of the algorithm improves as the maximum run increases. This may seem quite surprising, but the Expand procedure is designed so that it is able to adapt its behavior according to the distribution of the runs in the sequences. In Table 5.5, which summarizes the results of this group of experiments we have not stated the standard deviation, since

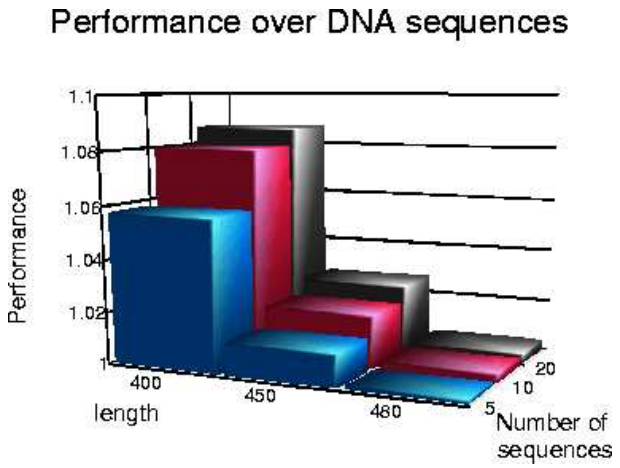

Figure 5.6: Experiments with maximum run 16

the values found are along the same lines as those in the first experiment.

5.6 The Algorithm for the SCS

In this section we are going to describe our algorithm called Reduce-Expand, shortly RE. The algorithm contains four procedures: Half Reduce, AuxiliarySet and Expand. The first procedure is called Half, it receives as input a set $\mathcal{S}$ of sequences, and for each $s \in \mathcal{S}$ with factorization $\sigma(1)^{b(1)} \cdots \sigma(k)^{b(k)}$ it returns a sequence $\sigma(1)^{\lceil b(1)/2 \rceil} \cdots \sigma(k)^{\lceil b(k)/2 \rceil}$.

Half($\mathcal{S}$)
Input: a set $\mathcal{S}$ of sequences
Let $R := \emptyset$.
For each $s \in \mathcal{S}$ do
  Let $\sigma(1)^{b(1)} \cdots \sigma(k)^{b(k)}$ be the factorization of s.
  $R := R \cup \{\sigma(1)^{\lceil b(1)/2 \rceil} \cdots \sigma(k)^{\lceil b(k)/2 \rceil}\}$.
EndFor
Return(R).

Figure 5.7: An outline of the Half procedure

The procedure Reduce receives as input a set of sequences and exploits Half to compute a set of instances of the SCS problem.

Reduce($\mathcal{S}$)
Input: a set $\mathcal{S}$ of sequences
Let $maxrun$ be $max\{run(s) : s \in \mathcal{S}\}$;
$m := \lceil \log_2(maxrun) \rceil$;
$R_m := \mathcal{S}$;
For$i := m$ downto 1 do
R_{i-1} :=Half(R_i);
EndFor
Return($R_0, \ldots, R_m$).

Figure 5.8: An outline of the Reduce procedure

Please note that all sequences in R_0 are basic sequences, hence it is likely that we are able to give a good approximate solution for such instance (in fact we can solve exactly the SCS problem for basic sequences over binary alphabet). The main idea of our algorithm is to exploit an approximate solution for the instance R_{i-1} to compute an approximate solution of the problem with instance R_i. Such computation is guided by an additional sequence (called *auxiliary* sequence) that is added to all sets R_i. In our algorithm all auxiliary sequence will be basic sequences and common supersequences of R_0. Since the computation of the approximated solution is dependent on the auxiliary sequence, a fundamental question is whether trying different auxiliary sequences may help in getting a better solution to the problem. In fact one of the main steps of the algorithm, the procedure AuxiliarySet, computes a set of auxiliary sequences, while the procedure Expand uses each such auxiliary sequence to compute an approximate solution to the problem.

Assume now that there is only one auxiliary sequence t for a given instance $\mathcal{S}$ of the SCS problem, and we recall that t is a basic sequence and a common supersequence of R_0 as computed by Reduce. The main property that holds for each call to Expand is that the output of such procedure is a sequence whose factorization into blocks has t as basic sequence.

The main property of the Expand procedure, together with the fact that Expand returns a common supersequence of R_m, follows from the construction of the Reduce procedure. We are now able to state the whole RE algorithm, which basically consists of applying Reduce and AuxiliarySet to compute a set of instances and a set of auxiliary sequences, to call Expand for each auxiliary sequence in the set and to return the shortest supersequence returned by the various calls to Expand.

We have not described the procedure AuxiliarySet yet, because more than one strategy can be identified to compute a set of auxiliary sequences. From

```
Expand(R_0, ..., R_m, t)
Input: a set R_0, ..., R_m computed by Reduce and an auxiliary sequence t.
Let σ(1)^{b(1)} ··· σ(|t|)^{b(|t|)} be the factorization of t;
For 1 ≤ i ≤ m do
  For1 ≤ i ≤ |t| do
  b(i):=2b(i);
  EndFor
  For1 ≤ j ≤ |t| do
    Whileσ(1)^{b(1)} ··· σ(|t|)^{b(|t|)} is a common supersequence
    of R_i do
      b(j) := b(j) − 1;
    EndWhile
  b(j) := b(j) + 1;
  EndFor
EndFor
Return(σ(1)^{b(1)} ··· σ(|t|)^{b(|t|)});
```

Figure 5.9: An outline of the Expand algorithm

one hand it is desirable to have a fast procedure even though this implies that a small number of auxiliary sequences are computed, on the other hand computing a larger set is likely to lead to better solution; consequently it does not seem possible to find an implementation of the procedure that is definitely the best solution to the problem.

Over the binary alphabet $\Sigma = \{a, b\}$ it is possible to compute all auxiliary sequences whose factorization into blocks is the same as that of an optimal solution in polynomial time. In fact the only two basic sequences of length l are $(ab)^{\lfloor l/2 \rfloor}a^{l-2\lfloor l/2 \rfloor}$ and $(ba)^{\lfloor l/2 \rfloor}b^{l-2\lfloor l/2 \rfloor}$ and $|\Sigma|n$ is an upper bound on the length of any shortest common supersequence of a set of sequences of maximum length n over the alphabet $|\Sigma|$. The procedure AuxiliarySetBinary is stated as follows:

In the case of arbitrary alphabet the set of all possible auxiliary sequences is not polynomial in the dimension of the input, hence it is necessary to design some heuristics to compute a set of auxiliary sequences. As stated previously each auxiliary sequence must be a common supersequence of R_0 and a basic sequence. The heuristic we have developed consists mainly of computing a supersequence s of R_0, to compute the basic sequence of s and pose the set of all auxiliary sequences as the set of sequences that can be obtained from such basic sequence by inserting up to k symbols, for a certain constant k. It is immediate to note that the set of auxiliary sequences computed in this

Reduce-Expand($\mathcal{S}$)
Input: a set $\mathcal{S}$ of sequences.
$R_0, \ldots, R_m$:=Reduce($\mathcal{S}$);
T :=AuxiliarySet(R_0);
$T := T \cup \{$Alphabet($\mathcal{S}$)$\}$;
For each$t \in T$ do
  $temp(t)$:= Expand($R_0, \ldots, R_m, t$);
EndFor
Returnthe sequence $temp(t)$ of minimum length;

Figure 5.10: The Reduce-Expand algorithm

AuxiliarySetBinary($R_0, \ldots, R_m$)
Input: the set $R_0, \ldots, R_m$ computed by Reduce.
Let l be the maximum length of a sequence in R_0;
$T := \emptyset$;
For$l \leq i \leq l+1$ do
  $T := T \cup \{(ab)^{\lfloor i/2 \rfloor} a^{i-2\lfloor i/2 \rfloor}\}$;
  $T := T \cup \{(ba)^{\lfloor i/2 \rfloor} b^{i-2\lfloor i/2 \rfloor}\}$;
EndFor
Remove from T each sequence that is not common supersequence of R_0;
Return(T).

Figure 5.11: An outline of the AuxiliarySetBinary algorithm

way is polynomial in the dimension of the instance.

5.7 The Experiments

The main goal of the experiments has been to evaluate the performance (represented by the approximation ratio) of RE with respect to the length and the number of the sequences, the maximum ratio between the lengths of the sequences (variability) and the maximum run of the sequences.

In order to compute precisely the approximation ratio achieved by our algorithm it would have been necessary to compute the exact solution on all instances considered by using one of the dynamic programming algorithms presented in literature: this cannot be done in practice due to time and space constraints. Consequently we had to compute a lower bound on the length of the SCS of a set $\mathcal{S}$ of sequences. Given a subset $\mathcal{S}_1 \subset \mathcal{S}$, clearly the length of

AuxiliarySet($R_0, \ldots, R_m$)
Input: the set $R_0, \ldots, R_m$ computed by Reduce.
Parameters: *Heuristic*: the heuristic used to compute a supersequence of R_0, can be Greedy or Tournament. α: the maximum number of symbols that are inserted.
s :=Heuristic(R_0);
$T = \{$ basic sequence of $s\}$;
Add to T each basic sequence that can be obtained from t by inserting at most α symbols of Σ;
Return(T);

Figure 5.12: An outline of the AuxiliarySet algorithm

$SCS(\mathcal{S}_1)$ is a lower bound on that of $SCS(\mathcal{S})$. The lower bound on $|SCS(\mathcal{S})|$ that we have exploited is obtained by determining all subsets of $\mathcal{S}$ on which we can compute the exact solution by dynamic programming and taking the maximum over the lengths of such exact solutions. From some experiments it turned out that the lower bound is about 10% to 20% less than the optimum.

All experiments have been run on some Pentium II/ Linux workstations owned by the bioinformatics group of our department.

5.8 The Results

In this section we will present the results of our experimental analysis. Our first experiment was aimed at determining the relation among the number of sequences in the instance, the variability of the instance (that is the ratio between the minimum and maximum lengths of the sequences in the instance), the maximum run of the sequences in the instance and the approximation ratio achieved by the algorithm. For each value of the parameters we have generated 1000 random instances containing sequences of maximum length 300 over binary alphabet: we have run our algorithm on such instances and we have compared the result with the lower bound described in the previous section. The results of such experiment are represented in Table 5.5.

The experiment has shown that the average error of the algorithm slightly worsens as the number of sequences increases, but for a moderate number of sequences the average error ratio is still very encouraging. The variability parameter does not seem to be very relevant, while the algorithm outputs better approximate solutions as the maximum run increases. The analysis of the worst-case error ratio is basically the same, except for the fact that the

Alphabet size: 2 — Maximum length of sequences: 300				
Number of sequences	Max run	Variability	Average error	Worst error
5	64	.9	1.197	1.346
		.95	1.205	1.4
	128	.9	1.152	1.344
		.95	1.159	1.366
10	64	.9	1.293	1.425
		.95	1.301	1.421
	128	.9	1.229	1.401
		.95	1.231	1.422
20	64	.9	1.354	1.48
		.95	1.365	1.481
	128	.9	1.272	1.465
		.95	1.273	1.461

Table 5.5: Results of experiment on binary alphabet

number of sequences does not seem to affect significantly the output of the algorithm.

The results of Table 5.5 suggested us to extend our algorithm to deal with sequences over arbitrary alphabet and to extend our experimental analysis. Since one of the main applications of the SCS problem is in the field of DNA sequence assembly, it is of fundamental relevance that the algorithm is able to cope with instances representing biological sequences; hence we have concentrated our experiments on alphabets of size 4 and 20 (modeling respectively DNA and protein sequences). Since our study on binary alphabet has pointed out that variability does not seem the influence significantly the quality of the solution returned by the algorithm, we have decided to fix the variability parameter to the value 0.95 for all of the following experiments. One additional goal of our experiments is to determine if the algorithm can effectively deal with biological sequences which, in practice, can have some hundreds of characters, hence we decided to analyze also the behavior of the algorithm with respect to the lengths of the sequences in the instance and to enlarge to 500 the maximum length considered.

In the following by RE+g we will denote the RE algorithm where Greedy has been used in the AuxiliarySetBinary procedure. Analogously RE+t will denote the RE algorithm where Tournament has been used in the AuxiliarySetBinary procedure.

The results of the experiments over sequences of length 4 (DNA sequences) are represented in Table 5.6. Each such experiment consisted of 100 instances of random sequences.

Alphabet size: 4						
Number of sequences	Max run	Max length	Min length	Average error		
				RE+g	RE+t	Alphabet
5	32	300	285	2.205	2.176	2.448
	64	300	285	1.509	1.499	2.444
		400	380	1.525	1.528	2.519
		500	475	1.527	1.536	2.546
10	32	300	285	1.908	1.907	2.508
	64	300	285	1.871	1.856	2.353
		400	380	1.872	1.859	2.409
20	32	300	285	2.205	2.176	2.448
	64	300	285	2.137	2.095	2.267

Table 5.6: Results of experiment on alphabet size 4

Both versions of our algorithm clearly outperform Alphabet on all cases, moreover we have got another evidence that the quality of the solution output by RE improves as the maximum run increases, while the improvement of Alphabet is not as large as the one achieved by RE.

For instances with 5 sequences and maximum run 64 it has been possible to push the maximum length of the sequences to 500 characters, showing that the performance of the algorithm degrades very slowly as the maximum length increases. As analyzing the behavior of our algorithm in this case was one of the main goals of our experiments, we summarizes such result in Fig. 5.13, where it is immediate to note that the two variants of RE considered are basically indistinguishable, and definitely better than Alphabet.

The last experiment performed has been on sequences over alphabet of size 20 (protein sequences). Each experiments consists of 50 instances containing 5 random sequences: the results of such experiment are summarized in Table 5.7.

The results for the case of alphabet size 20 are along the same lines as the ones for smaller alphabet. In this case it is immediate to note that the gap between the approximation ratio achieved by RE and that of Alphabet is even larger than the one for alphabet size 4, pointing out the practicality of our algorithm for such sequences.

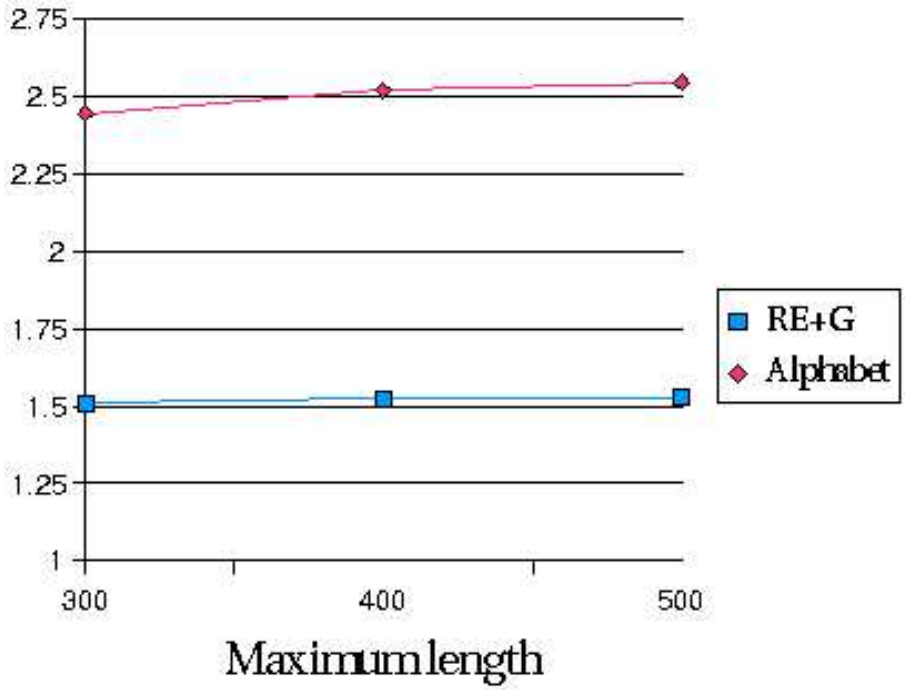

Figure 5.13: Performance with respect to the maximum length of the sequences

Alphabet size: 20						
Number of sequences	Max run	Max length	Min length	Average error		
				RE+g	RE+t	Alphabet
5	32	300	285	2.018	2.013	10.693
	64			1.979	1.981	10.204

Table 5.7: Results of experiment on alphabet size 20

Chapter 6

Comparing Phylogenies

6.1 Introduction

Evolutionary trees are trees where each leaf is labeled by a distinct element in a set S of species and where all internal nodes have degree at least three. They are frequently used by biologists to represent classifications of species. More precisely, each edge is weighted with the estimated (temporal) distance between the two species represented by its endpoints. A number of methods to infer evolutionary trees have been proposed [32, 39, 25, 37, 38, 42, 84, 13, 12, 67, 32, 92], so it is quite usual to compute more evolutionary trees over the same set of species applying different methods, hence obtaining various trees, motivating the compelling need to compare different trees in order to extract a common history. The Maximum Agreement Subtree method is a basic approach that allows to reconciliate different evolutionary trees over the same set of species: it computes a subset of the extant species about which all trees "agree". A general way to define an agreement subtree from a set $T_1, \cdots, T_k$ of S-labeled trees has been formalized in [4]. This method assumes that each edge is labeled by an interval weight (a range of time to measure the duration of the evolution process) and looks for a subset S^* of the extant species S such that:

- each edge of the subtree induced in each tree of the given set is labeled by a value belonging to the given interval,
- for each pair of extant species in S^*, the distance between them is the same in all trees.

The problem stated above is the Maximum Interval Weight Agreement Subtree (MIWT), and is a very general formulation of the problem

of comparing phylogenies. In order to obtain more efficient algorithmic solutions, some restrictions have been introduced to MIWT. A first natural restriction requires that each interval reduces to a single value; such problem is called Maximum Weight Agreement Subtree (MWT). A different restriction of MIWT is the one where an agreement subtree is homeomorphic to a subtree of each tree in the instance, since it is equivalent to require all intervals to be of the form $[1, n-1]$, where n is the number of extant species considered. This problem is called Maximum Homeomorphic Agreement Subtree (MHT). Note that this problem is sometimes referred to as Maximum Agreement Subtree and is abbreviated by (MAST). A third restriction of MIWT is the one where all intervals are of the form $[1, 1]$, and is called Maximum Isomorphic Agreement Subtree (MIT), as all subtrees induced by a feasible solution must be isomorphic. The MIT problem is also a restricted case of the maximum isomorphic subgraph problem, investigated in [63]. Since MIT and MHT are the two more restricted problems among the ones we have mentioned, most of the efforts to develop efficient algorithms have been concentrated on them.

Efficient algorithms for the MHT problem for instances of two trees have been widely investigated in literature. While some heuristics have been found [40, 70], the first polynomial time algorithm has been described only in 1993 by Steel and Warnow [90]. Afterwards further improvements have appeared in literature [34, 64, 71]. To our knowledge the most efficient algorithms for the problem are due to Farach and Thorup which developed a $O(n^{3/2} \log n)$ algorithm for rooted trees of bounded degree [35, 36], to Cole and Hariharan [29, 28] for the case of rooted trees of unbounded degree, which gave a $O(n \log n)$ algorithm, and to Kao, Lam, Przytycka, Sung and Ting [65] which described a technique allowing to match the time complexity of the two previously cited algorithms also in the case of unrooted trees. The problems MHT and MIT over a set of trees, where at least one of the trees has bounded degree, can be solved in polynomial time [4], even though the time complexity is exponential in the bound for the degree. Moreover both problems are $\mathcal{NP}$-hard for instances containing three trees of unbounded degree, hence it is necessary to focus on designing polynomial time approximation algorithms. The approximation complexity of the MHT problem has been deeply investigated in [51], where some strong negative results have been obtained. Since the MIT is a different restriction of the MIWT, it seems natural to investigate if the negative results for MHT hold also for MIT or the latter problem is easier to approximate than the former one. In this chapter we show that the negative results of [51] hold also for the MIT problem, as a consequence of a nontrivial application of the self-improvement technique. Applying self-improvement usually leads to a result of the form "either problem P admits a

ptas or P cannot be approximated within a constant factor unless $\mathcal{NP}$=$\mathcal{P}$ (or another unlikely collapse between complexity classes occur). This idea has been exploited in [44] to prove that Max Independent Set either has a ptas or no constant factor polynomial-time approximation algorithm (unless $\mathcal{NP}$=$\mathcal{P}$) and has been successively pushed further by Karger et. al. in [66] to prove that the Longest Path cannot be approximated within $O(\log n)$ unless $\mathcal{P}$=$\mathcal{NP}$. In the latter paper the inapproximability result has been obtained by combining the self-improvement technique and an L-reduction (to prove that the problem is $\mathcal{APX}$-hard). Consequently the "easy" way to prove that MIT is hard to approximate seems to rely on the $\mathcal{APX}$-hardness proof by Amir and Keselman [4] and on the application of the self-improvement technique to the problem. Unfortunately the MIT problem seems to lack some of the properties that in [51] have been exploited implicitly in the proofs. Hence in this chapter we deal with a restriction of MIT, called R-MIT, that has the desired properties, but before applying self-improvement to R-MIT we have to prove that the latter problem is $\mathcal{APX}$-hard.

As a consequence of our results achieving a constant error ratio is an $\mathcal{NP}$-hard problem, even for instances consisting of only three trees. Moreover we have strengthened such negative results in the case of instances containing an arbitrary number of trees (in the restricted case where each tree in the instance has depth 2), as we show that MIT shares exactly the same inapproximability properties of Max Clique [50], implying that there cannot exist a polynomial time $n^{1-\epsilon}$ ratio approximation algorithm for each $\epsilon > 0$, unless $\mathcal{NP}$=$\mathcal{ZPP}$ (see [79] for a definition of $\mathcal{ZPP}$). A similar, but slightly weaker, negative result on the approximability of MHT has been obtained in [45], showing that such problem cannot be approximated within factor n^{ϵ} for any $0 \leq \epsilon < \frac{1}{9}$, since approximating MHT within factor n^{ϵ} in polynomial time implies a polynomial-time approximation algorithm for Max Clique with $n^{3\epsilon+o(1)}$ guaranteed approximation ratio.

6.2 Preliminaries

All trees we will deal with in this chapter are rooted, that is we distinguish a special vertex of the tree T and we call such a vertex *root*, denoted by $r(T)$. All results presented in this chapter are referred to rooted trees, but they can be generalized to the unrooted case.

Let $S = \{s_1, \ldots, s_n\}$ be a set of labels. An S-labeled tree has n leaves, each one labeled with a distinct element of S; since each label identifies unambiguously a leaf of the tree, in the following of the chapter we will write a label x meaning the leaf of the tree with label x. The Maximum Isomor-

PHIC AGREEMENT SUBTREE Problem (shortly MIT) is defined formally as follows:

Problem 6 (MIT).
Instance: a set $\mathcal{T} = \{T_1, \ldots, T_m\}$ of S-labeled trees.
Solution: an S^*-labeled tree T^*, with $S^* \subseteq S$, such that T^* is isomorphic to a subtree of all trees in $\mathcal{T}$.
Goal: to maximize $|S^*|$.

The MAXIMUM HOMEOMORPHIC AGREEMENT SUBTREE (MHT) is:

Problem 7 (MHT).
Instance: a set $\mathcal{T} = \{T_1, \ldots, T_m\}$ of S-labeled trees.
Solution: an S^*-labeled tree T^*, with $S^* \subseteq S$, such that T^* does not contain any internal node (with the possible exception of the root) of degree 2 and, for each tree $T_i \in \mathcal{T}$, T^* is homeomorphic to a subtree of T_i.
Goal: $|S^*|$, to be maximized.

It is immediate to note that the $\mathcal{NP}$-completeness proof given by Amir and Keselman in [4] is an L-reduction for the MHT problem, as pointed out in [51]. Similarly it is possible to prove that MIT is $\mathcal{APX}$-hard, that is there is no polynomial time approximation scheme for it, unless $\mathcal{P}$=$\mathcal{NP}$.

Anyway, differently from [51], we have to deal with a restricted version of the problem in order to prove our inapproximability results, hence such $\mathcal{APX}$-hardness proof is not adequate to our purposes. More precisely we consider only instances consisting of trees having leaves all at the same depth in every tree. Formally $d_{T_i}(a, r(T_i)) = d_{T_j}(b, r(T_j))$ for all $a, b \in S$ and every pair of trees T_i, T_j in the instance. We will say that trees in such instances are *restricted*. This new problem will be called R-MIT. Clearly all inapproximability results for this problem hold also for MIT.

The following Lemma, proved in [88], characterizes all feasible solutions of each instance of R-MIT.

Lemma 6.2.1. *Let $\mathcal{T}$ be a set of S-labeled trees, and let $S^* \subseteq S$. Then there exists a S^*-labeled tree T^* that is isomorphic to a subtree of each tree in $\mathcal{T}$ if and only if for each pair of labels $a, b \in S^*$, a and b have the same distance in all trees in $\mathcal{T}$.*

As a consequence we can identify a feasible solution of an instance of MIT as a subset of its label set. The following property of trees, whose straightforward proof is omitted, will be used in the remaining of the chapter.

Proposition 6.2.2. *Let a, b be two leaves of a S-labeled tree with root r. Then $d_T(a, b) = d_T(a, r) + d_T(b, r) - 2\, d_T(r, lca_T(a, b))$.*

6.3 R-MIT is $\mathcal{APX}$-hard

In this section we are going to prove that the R-MIT problem is $\mathcal{APX}$-hard. This results is necessary to prove that MIT is hard to approximate even on instances consisting of only three trees.

The first step is to rule out the possibility of having a ptas for R-MIT, by describing an L-reduction (see Def. 2.1.6) from the Tridimensional Bounded Matching problem (shortly 3DM-B) to R-MIT, defined as follows:

Instance: three pairwise disjoint sets $\langle X_1, X_2, X_3\rangle$ and a set M of distinct triples where $M \subseteq X_1 \times X_2 \times X_3$ and every element in $X_1 \cup X_2 \cup X_3$ occurs in at least one and at most B triples of M.

Solution: a subset M_1 of M, such that no two triples in M_1 share a common element.

Measure: $|M_1|$, to be maximized.

The general 3DM-B problem is $\mathcal{APX}$-hard [62].

Let $\mathcal{M} = \langle X_1, X_2, X_3, M\rangle$ be an instance of the 3DM-B problem, with $M \subseteq X_1 \times X_2 \times X_3$, $X_i = \{x_{i,1}, x_{i,2}, \ldots x_{i,|X_i|}\}$. Then we will associate to $\mathcal{M}$ an instance $\langle T_1, T_2, T_3\rangle$ of MIT. Each tree T_i consists of the following nodes and edges: a root labeled r_i, a node connected to the root for each element $x_{i,j}$ of X_i, and each triple $(x_{1,j}, x_{2,j}, x_{3,j}) \in \mathcal{M}$ is a leaf of T_i connected to $x_{i,j}$. Consequently each tree T_i is M-labeled.

In Fig. 6.1 it is represented the instance of R-MIT associated to the instance of 3DM-B where $X_1 = \{x_{1,1}, x_{1,2}, x_{1,3}\}$, $X_2 = \{a, c\}$, $X_3 = \{b, d\}$ and $M = \{(x_{1,1}ab), (x_{1,1}cd), (x_{1,2}cd), (x_{1,3}cd)\}$.

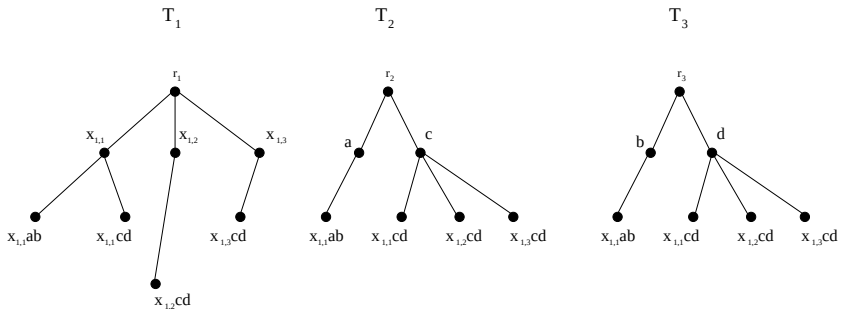

Figure 6.1: Example of instance of R-MIT associated to an instance of 3DM-B

Since the distance from each node to the root is 2 in all trees of the instance of MIT associated to an instance of 3DM-B, such set of trees is an

instance of R-MIT. The following Lemma is an immediate consequence of such fact.

Lemma 6.3.1. *Let* $\mathcal{M} = \langle X_1, X_2, X_3, M\rangle$ *be an instance of* 3DM-B *and let* $\langle T_1, T_2, T_3\rangle$ *be the associated instance of* MIT. *Given a tree* T_i *with* $1 \leq i \leq 3$, *and given two distinct leaves* s, t *of* T_i, *then the distance between* s *and* t *in* T_i *is* 2 *or* 4 .

Note that the distance of two leaves s and t in a tree T_i is 2 if and only if s and t are labeled by triples of M that share the same element in the set X_i.

Lemma 6.3.2. *Let* $\mathcal{M} = \langle X_1, X_2, X_3, M\rangle$ *be an instance of* 3DM-B, *let* $\langle T_1, T_2, T_3\rangle$ *be the associated instance of* R-MIT *and let* $S \subseteq M$. *Then* S *is a feasible solution of* $\langle T_1, T_2, T_3\rangle$ *if and only if each pair* s, t *of distinct triples in* S *has distance* 4 *in all trees* T_i.

Proof. By Lemma 6.2.1 S is a feasible solution if and only if each distinct pair s, t of triples in S have the same distance in all trees T_i, which is, by Lemma 6.3.1 either 2 or 4. Let us assume that there exists a pair s, t that has distance 2 in all trees. Then by construction s is equal to t, contradicting the fact that all triples in M are distinct, hence for each pair the distance must be 4, as stated. The other direction follows immediately by Lemma 6.2.1. □

From Lemmata 6.3.1 and 6.3.2 the reduction from 3DM-B to R-MIT that we have described can be thought as a polynomial-time computable function r that associates to each instance $\mathcal{M}$ of 3DM-B an instance $r(\mathcal{M})$ of R-MIT and a polynomial-time computable function s that associates to each feasible solution Apx of $r(\mathcal{M})$ a feasible solution $s(Apx)$ such that the costs of Apx and of $s(Apx)$ are the same and the optima of $\mathcal{M}$ and of $r(\mathcal{M})$ are the same. This implies that our reduction is an L-reduction, hence R-MIT is $\mathcal{APX}$-hard. The following theorem follows from the results by Arora et. al. given in [9].

Theorem 6.3.3. *There does not exists a ptas for* R-MIT *unless* $\mathcal{P} = \mathcal{NP}$.

6.4 Product of Trees

The inapproximability result over instances of three trees is obtained by means of the *self-improvement* technique. In [51] such technique has been exploited to prove a similar result for the MHT problem. Such technique requires a careful definition of a product between instances of the problem, which is defined as follows:

Definition 6.4.1. Let T_1 be a S_1-labeled tree, T_2 a S_2-labeled tree and, for a given leaf s of T_1, $T_{2\cdot s}$ is the tree obtained from T_2 relabeling each leaf s_2 with the sequence ss_2. Then the product $T_1 \cdot T_2$ is the tree obtained from T_1 replacing each leaf s with the tree $T_{2\cdot s}$.

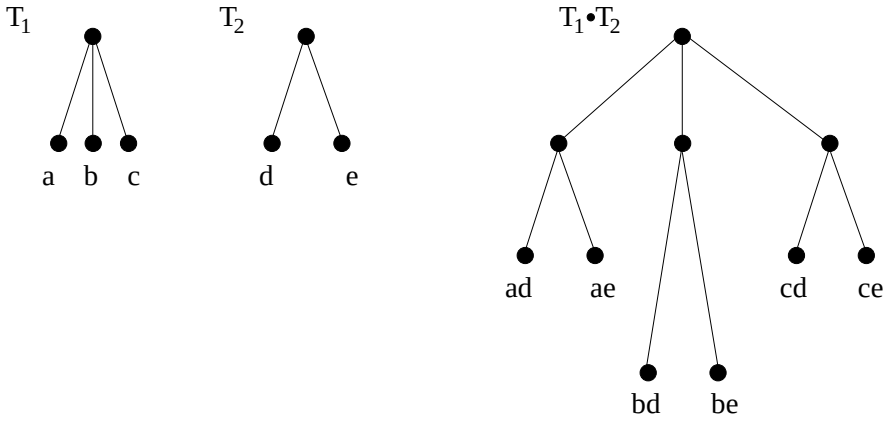

Figure 6.2: Product of trees

Let T be a S-labeled tree, then $T^2 = T \cdot T$ and $T^i = T^{i-1} \cdot T$, $i > 2$. Note that the label of a leaf of the tree T^k is a sequence $s_1 \dots s_k$ of k symbols over the alphabet S. An immediate property of the product of trees is stated below:

Proposition 6.4.1. *Let T_1, T_2 be two restricted trees. Then $T_1 \cdot T_2$ is also a restricted tree.*

The following Lemma points out the motivation for our definition of product.

Lemma 6.4.2. *Let T_1, T_2 be two restricted S-labeled trees, let a, b be two labels in S and let α, β be two strings of $k-1$ symbols over S. Then $d_{T_1^k}(\alpha a, \beta b) = d_{T_2^k}(\alpha a, \beta b)$ if and only if $d_{T_1}(a, b) = d_{T_2}(a, b)$ and $d_{T_1^{k-1}}(\alpha, \beta) = d_{T_2^{k-1}}(\alpha, \beta)$*

Proof. Since T_1 and T_2 are restricted trees (that is in T_1 and T_2 all leaves have the same depth), and by Prop 6.2.2, in order to prove the lemma, it is sufficient to show that $d_{T_1^k}(\mathrm{lca}_{T_1^k}(\alpha a, \beta b), r(T_1^k)) = d_{T_2^k}(\mathrm{lca}_{T_2^k}(\alpha a, \beta b), r(T_2^k))$ if and only if $d_{T_1}(\mathrm{lca}_{T_1}(a, b), r(T_1)) = d_{T_2}(\mathrm{lca}_{T_2}((a, b), r(T_2))$ and $d_{T_1^{k-1}}(\mathrm{lca}_{T_1^{k-1}}(\alpha, \beta), r(T_1^{k-1})) = d_{T_2^{k-1}}(\mathrm{lca}_{T_2^{k-1}}(\alpha, \beta), r(T_2^{k-1}))$. Assume initially that $\alpha = \beta$, then, by definition of product, $d_{T_1^k}(\mathrm{lca}_{T_1^k}(\alpha a, \beta b), r(T_1^k)) = d_{T_{1\cdot\alpha}}(\mathrm{lca}_{T_{1\cdot\alpha}}(\alpha a, \alpha b),$

$r(T_{1\cdot\alpha})) + \text{depth}(T_1^{k-1})$ and $d_{T_2^k}(\text{lca}_{T_2^k}(\alpha a, \beta b),\, r(T_2^k)) = d_{T_{2\cdot\alpha}}(\text{lca}_{T_{2\cdot\alpha}}(\alpha a, \alpha b),$ $r(T_{2\cdot\alpha})) + \text{depth}(T_2^{k-1})$, hence $d_{T_1^k}(\text{lca}_{T_1^k}(\alpha a, \beta b), r(T_1^k)) = d_{T_2^k}(\text{lca}_{T_2^k}(\alpha a, \beta b),$ $r(T_2^k))$ if and only if $d_{T_1}(\text{lca}_{T_1}(a, b), r(T_1)) = d_{T_2}(\text{lca}_{T_2}((a, b), r(T_2))$. Assume now that $\alpha \neq \beta$, then $\text{lca}_{T_1^k}(\alpha a, \beta b)) = \text{lca}_{T_1^{k-1}}(\alpha, \beta)$ and $\text{lca}_{T_2^k}(\alpha a, \beta b) =$ $\text{lca}_{T_2^{k-1}}(\alpha, \beta)$. Consequently $d_{T_1^k}(\text{lca}_{T_1^k}(\alpha a, \beta b), r(T_1^k)) = d_{T_2^k}(\text{lca}_{T_2^k}(\alpha a, \beta b),$ $r(T_2^k))$ iff $d_{T_1^{k-1}}(\text{lca}_{T_1^{k-1}}(\alpha, \beta),\, r(T_1^{k-1})) = d_{T_2^{k-1}}(\text{lca}_{T_2^{k-1}}(\alpha, \beta), r(T_2^{k-1}))$. This suffices to prove the Lemma. □

The following lemma relates a feasible solution of $\langle T_1, T_2, T_3\rangle$ with a feasible solution of $\langle T_1^k, T_2^k, T_3^k\rangle$.

Lemma 6.4.3. *Let $\langle T_1, T_2, T_3\rangle$ be an instance of* R-MIT*, and let F be a feasible solution with cost $cost(F)$ of such an instance. Then it is possible to compute in polynomial time a solution of $\langle T_1^k, T_2^k, T_3^k\rangle$ whose cost is $cost(F)^k$.*

Proof. Let F_k be the set of strings of labels $\{f_1 \cdots f_k : f_i \in F,\ 1 \leq i \leq k\}$, then for each pair of strings of labels $f_{\alpha_1} \cdots f_{\alpha_k}$, $f_{\beta_1} \cdots f_{\beta_k}$ in F_k, their distance is the same in all trees T_1^k, T_2^k, T_3^k, since for each $1 \leq i \leq k$ $d_{T_1}(f_{\alpha_i}, f_{\beta_i}) = d_{T_2}(f_{\alpha_i}, f_{\beta_i}) = d_{T_3}(f_{\alpha_i}, f_{\beta_i})$, as all $f_{\alpha_i}, f_{\beta_i}$ are in S and from Prop. 6.2.2. □

Lemma 6.4.4. *Let $\langle T_1^k, T_2^k, T_3^k\rangle$ be an instance of* R-MIT *and let S_k be a feasible solution of $\langle T_1^k, T_2^k, T_3^k\rangle$, then it is possible to compute in polynomial time a feasible solution S_1 of $\langle T_1, T_2, T_3\rangle$ such that $cost(S_k) \leq cost(S_1)^k$.*

Proof. Let $S_k = \{f_{\alpha_1}, \ldots, f_{\alpha_k}\}$ be a feasible solution of $\langle T_1^k, T_2^k, T_3^k\rangle$. By applying Lemma 6.4.2 iteratively we can obtain k feasible solutions F_i of $\langle T_1, T_2, T_3\rangle$, where each solution F_i contains exactly the symbols f_{α_i} of S that are in the i-th position of a string in S_k. Let F^* be the largest such F_i and let F_k^* be the set of strings $\{f_1 \ldots f_k : f_j \in F^*,\ 1 \leq j \leq k\}$. Just as in the proof of Lemma 6.4.3 it is possible to prove that F_k^* is a feasible solution of $\langle T_1^k, T_2^k, T_3^k\rangle$. An immediate counting argument and the fact that F^* is the F_i of maximum cardinality imply that $|S_k| \leq |F_k^*| = |F^*|^k$. □

In the following, given an instance $\langle T_1, T_2, T_3\rangle$ of R-MIT and an approximation algorithm for R-MIT, we will denote by $\text{apx}(\langle T_1, T_2, T_3\rangle)$ the solution returned by such algorithm for the instance $\langle T_1, T_2, T_3\rangle$, while the optimum solution is denoted by $\text{Opt}(\langle T_1, T_2, T_3\rangle)$. The basic idea of the proofs of the main results in this section is sketched in the following: given an instance $\langle T_1, T_2, T_3\rangle$ of R-MIT we expand it (by Lemma 6.4.3) to another instance $\langle T_1^k, T_2^k, T_3^k\rangle$ to which we apply an hypothetical approximation algorithm. By Lemma 6.4.4 we are able to infer from the approximate solution of

$\langle T_1^k, T_2^k, T_3^k\rangle$ an approximate solution of $\text{apx}(\langle T_1, T_2, T_3\rangle)$ whose approximation factor is "much better" than the one we have got for $\text{apx}(\langle T_1^k, T_2^k, T_3^k\rangle)$.

We now state our main results, where all logarithms have natural bases.

Theorem 6.4.5. *There does not exists a polynomial-time constant-ratio approximation algorithm for* R-MIT *unless* $\mathcal{NP}$=$\mathcal{P}$.

Proof. Assume that there exists a ϵ-approximation algorithm with polynomial time complexity for R-MIT. Pose $k = \lceil \log \epsilon \rceil$, consequently $e^k \geq \epsilon$. Then, by Lemmata 6.4.3, 6.4.4,

$$\left(\frac{\text{Opt}(\langle T_1, T_2, T_3\rangle)}{\text{apx}(\langle T_1, T_2, T_3\rangle)}\right)^k = \frac{\text{Opt}(\langle T_1^k, T_2^k, T_3^k\rangle)}{\text{apx}(\langle T_1^k, T_2^k, T_3^k\rangle)} \leq \epsilon \leq e^k$$

hence $(\frac{\text{Opt}(\langle T_1,T_2,T_3\rangle)}{\text{apx}(\langle T_1,T_2,T_3\rangle)}) \leq e$. Please note that computing $\langle T_1^k, T_2^k, T_3^k\rangle$ from $\langle T_1, T_2, T_3\rangle$ can be done in $O(n^{\lceil \log \epsilon \rceil})$ time, hence we have described a ptas for R-MIT. By Theorem 6.3.3 $\mathcal{NP} = \mathcal{P}$. □

Corollary 6.4.6. *There exists* $\delta > 0$ *such that* R-MIT *cannot be approximated within factor* $\log^\delta n$ *in polynomial time, unless* $\mathcal{NP} \subseteq \mathcal{DTIME}[2^{polylog\, n}]$.

Proof. Assume that for all $\delta > 0$ there exists a $\log^\delta n$-approximation polynomial-time algorithm for R-MIT, we will prove that there exists an e-approximation polynomial-time algorithm. Pose $k = \lceil \log(\log^\delta n)\rceil$, consequently $e^k \geq \log^\delta n$. Just as in the proof of Theorem 6.4.5 we will denote with $\text{apx}(\langle T_1, T_2, T_3\rangle)$ the solution returned by the approximation algorithm for the instance $\langle T_1, T_2, T_3\rangle$, while $\text{Opt}(\langle T_1, T_2, T_3\rangle)$ denotes the optimum solution. Then, by Lemmata 6.4.3, 6.4.4,

$$\left(\frac{\text{Opt}(\langle T_1, T_2, T_3\rangle)}{\text{apx}(\langle T_1, T_2, T_3\rangle)}\right)^k = \frac{\text{Opt}(\langle T_1^k, T_2^k, T_3^k\rangle)}{\text{apx}(\langle T_1^k, T_2^k, T_3^k\rangle)} \leq \log^\delta n$$

taking the logarithms of both sides

$$k \log \left(\frac{\text{Opt}(\langle T_1, T_2, T_3\rangle)}{\text{apx}(\langle T_1, T_2, T_3\rangle)}\right) \leq \log(\log^\delta n)$$

Consequently

$$\lceil \log(\log^\delta n)\rceil \log \left(\frac{\text{Opt}(\langle T_1, T_2, T_3\rangle)}{\text{apx}(\langle T_1, T_2, T_3\rangle)}\right) \leq \log(\log^\delta n)$$

implying that $\log(\frac{\text{Opt}(\langle T_1,T_2,T_3\rangle)}{\text{apx}(\langle T_1,T_2,T_3\rangle)}) \leq 1$. Hence $\frac{\text{Opt}(\langle T_1,T_2,T_3\rangle)}{\text{apx}(\langle T_1,T_2,T_3\rangle)} \leq e$. It is immediate to note that that computing $\langle T_1^k, T_2^k, T_3^k\rangle$ from $\langle T_1, T_2, T_3\rangle$ can be done in $O(n^{\lceil \log \log^\delta n\rceil}) = 2^{polylogn}$ time. Thus the claim follows from Thm. 6.4.5. □

6.5 MIT over Unbounded Number of Trees

The inapproximability result proved in the previous section can be strengthened when instances are not required to contain exactly three trees, but can contain an arbitrary number of trees (even in the case of all trees of depth 2). This can be proved by a simple L-reduction from Max Clique (Pb. 11). Since such reduction preserves the optimum and the cost of approximate solutions, MIT with unbounded number of trees inherits the same inapproximability results of Max Clique, which cannot be approximated within $n^{1-\epsilon}$ for each $\epsilon > 0$, unless $\mathcal{ZPP}$=$\mathcal{NP}$[50]. Informally the Max Clique problem asks for a maximum cardinality subset K of the vertices of a graph F, such that K induces a complete subgraph of G.

The reduction is quite simple: let $G = \langle V, E \rangle$ be a graph with $E \neq \emptyset$. The instance of MIT contains the V-labeled trees in the set $\{T_{edge}\} \cup \{T_{ij} : i, j \in V,\ (i,j) \notin E\}$, where T_{edge} has root r and each leaf v of T_{edge} has p_v as parent and p_v is a child of r. Each tree T_{ij} consists of a root r, a node p_{ij} that is the parent of both leaves v_i, v_j a node p_z for each $z \in V - \{v_i, v_j\}$ and each p_z is the parent of the leaf v_z. Moreover p_{ij} and all p_z with $z \in V - \{i, j\}$ are the children of the root.

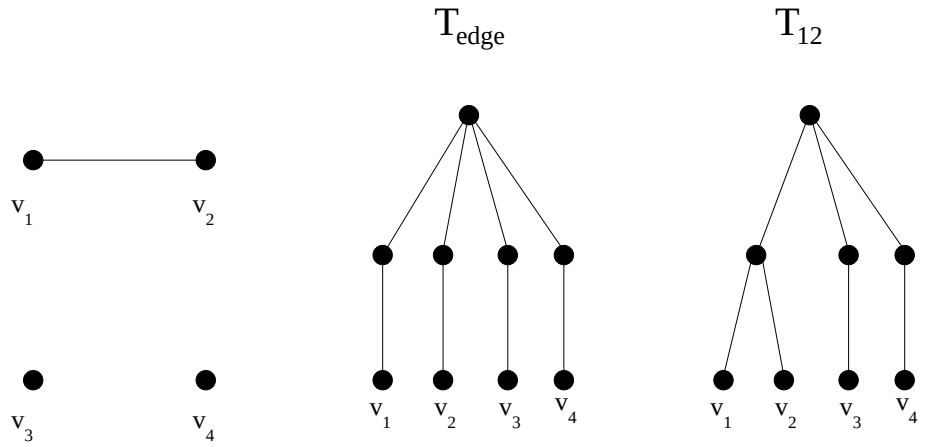

Figure 6.3: Example of reduction from Max Clique.

An example of application of such reduction is represented in Fig. 6.3. The following Lemma points out the structure of all feasible solutions considered in our reduction.

Lemma 6.5.1. *Let $G = \langle V, E \rangle$ be an instance of* Max Clique, *let $\mathcal{T}$ be the set of V-labeled trees associated to G, and let T be a feasible solution of*

MIT($\mathfrak{T}$). *Let v_1, v_2 be two distinct leaves of T. Then the distance between v_1 and v_2 in T is four.*

Proof. Let v_1, v_2 be two distinct leaves of T. Since v_1 and v_2 are both in a feasible solution T of $\mathfrak{T}$, by Lemma 6.2.1 their distance must be the same in all trees in $\mathfrak{T}$. Since $d_{T_{edge}}(v_1, v_2) = 4$ then $d_T(v_1, v_2) = 4$. □

We will show how to compute a feasible solution of MIT from a feasible solution of Max Clique and vice versa, so that the costs of both solutions are the same.

Let $\mathfrak{T}$ be the instance of MIT, associated to the instance $G = \langle V, E\rangle$ of Max Clique, and let $V_1 \subset V$ be a feasible solution of $\mathfrak{T}$. Please note that, by Lemmata 6.2.1 and 6.5.1 a subset $V_1 \subseteq V$ is a feasible solution of $\mathfrak{T}$ if and only if $d_T(v_1, v_2) = 4$ for each pair of distinct elements $v_i, v_j \in V_1$ and each tree $T \in \mathfrak{T}$. We will prove that V_1 is a clique of G. Assume to the contrary that V_1 is not a clique of G, that is there exist two vertices $v_i, v_j \in V_1$ such that $(v_i, v_j) \notin E$. By construction in $\mathfrak{T}$ there is the tree T_{ij}, and $d_{T_{ij}}(v_i, v_j) = 2$. Consequently by Lemma 6.5.1 v_i and v_j cannot both be in a feasible solution of $\mathfrak{T}$. To compute a feasible solution of $\mathfrak{T}$ from a clique of G is trivial, hence the following theorem follows:

Theorem 6.5.2. MIT *over an unbounded number of trees cannot be approximated within $n^{1-\epsilon}$ for each $\epsilon > 0$, unless $\mathcal{ZPP}$=$\mathcal{NP}$.*

Chapter 7

Reconstructing Phylogenies

7.1 Introduction

Recent advances in Molecular Biology [16] has lead to the availability of a huge amount of biological sequences (DNA, RNA, proteins) allowing to solve some of the most intriguing problems in evolution [46, 95]. As stated in the previous chapter, biologists now agree to represent the evolution of a set of species by means of an *evolutionary tree*, which is an edge-weighted rooted tree whose leaves are labeled by sequences of extant species in such tree, the internal nodes represent the (possibly unknown) ancestors and the weight on the edges represent the distance in the evolution between species. Anyway the size of the datasets involved makes exhaustive methods, such as Maximum Likelihood [38] and Maximum Parsimony [42], not feasible in practice.

In order to overcome such infeasibility a number of *quartet based* methods have been proposed in literature, gaining much popularity [12, 13, 24, 32, 67, 92] among researchers in the Computational Biology community thanks to its capability of giving, in most of the cases, sensible evolutionary trees in a short time. In this chapter a quartet is a set of four species and a quartet topology is a partition of a quartet into two subsets, where in each set there are the two species that are more closely related.

The quartet paradigm divides the phylogeny inference process into two steps:

1. Given the sequences of the datasets apply a *quartet topology inference* method [42, 38, 84, 67] to reconstruct the (supposed) quartet topologies;

2. Given all quartet topologies apply a *quartet recombination* method [13, 32, 92] to reconstruct an evolutionary tree topology.

At the end of this two-step process the tree is rooted and weights are assigned to the edges.

The second step is better understood than the former one, but still the possibility of reconstructing an erroneous topology, if the quartet topologies inferred in the first step contains even a small amount of errors, can lead to disappointing results, and it is considered one of the main drawbacks of such method.

Hence a natural problem may be to ask for the evolutionary tree that is more likely to "agree over" a given set of quartet topologies. This can be formalized as an optimization problem (Maximum Quartet Consistency), whose decision version is, unfortunately, $\mathcal{NP}$-complete [14].

So alternative ways, such as Quartet Cleaning [57], to improve the accuracy of quartet based method are to be sought. The main idea is to modify the two-step method for reconstructing evolutionary trees into three steps: more precisely a new step is introduced between the two mentioned above. This new step exploits the fact that quartet errors may be sparse with respect to the number of quartet topologies, hence it may be possible to find such errors and replace them with the correct topologies. Consequently in the reconstruction process the topology inference gives a set of raw topologies that are cleaned by a quartet cleaning algorithm to obtain a set of "error free" topologies. Finally a quartet recombination method is applied to reconstruct an evolutionary tree topology.

More precisely the main observation is that each error affects only a certain part of the (still unknown) tree, in such case we will write that a quartet is *across* a vertex (or an edge): see Fig. 2.3 for an example. It is possible to compute the exact number of quartets that are across an edge or a vertex, and the quartet cleaning technique allows to correct a number of errors that is bounded by fixed fraction of the number of quartets that are across a vertex (or an edge). The quartet cleaning algorithms presented in literature [14] can be split into two classes: global and local algorithms. The former ones assume that the bound on the number of errors holds for all vertices (or edges) of the tree, while the local algorithms allow the presence of a large number of errors across some vertices (edges) and still reconstruct all vertices (edges) for which the error bound is satisfied.

In this chapter we will describe two optimal local vertex quartet cleaning algorithms, that is we will describe how to reconstruct all vertices of the tree for which a certain error bound holds. The only local vertex algorithm previously presented is in [14], and it allows to clean a number of errors that is at most $1/4$ of the total number of quartets across a vertex, and has time complexity $O(n^7)$, where n is the number of species. Our first algorithm runs in linear time, while the second one cleans an optimal number of errors in

$O(n^7)$ time.

Since the methods described in [13, 32, 92] are known to output the correct phylogeny provided that they are given a consistent set of quartet topologies, the efficiency of the reconstruction process is greatly improved. Experimental evidence of such improvement is given in this thesis where we summarize the results of some simulations.

In the chapter the bounds on the quartet errors across a vertex v will always be of the form $\frac{1}{\alpha}((|A_v|-1)(|B_v|+|C_v|-1) + (|B_v|-1)(|A_v|+|C_v|-1) + (|C_v|-1)(|A_v|+|B_v|-1))$

In the following we will describe some algorithms for local vertex cleaning [57, 14], that is we will show how to compute a phylogeny which contains all vertices v such that the number of errors across v is bounded by a given value.

Definition 7.1.1 (α-bounded bipartition). Let T be an evolutionary tree over a set S, and let $\langle A, B\rangle$ be a bipartition of S. Then $\langle A, B\rangle$ is *α-bounded* if the number of quartet errors across such bipartition is strictly less than $(|A|-1)(|B|-1)/\alpha$.

Definition 7.1.2 (α-bounded tripartition). Let T be an evolutionary tree over a set S, and let $\langle A, B, C\rangle$ be a tripartition of S. Then $\langle A, B, C\rangle$ is called *α-bounded* if all bipartitions $\langle A, B\cup C\rangle$, $\langle B, A\cup C\rangle$, $\langle C, A\cup B\rangle$ are α-bounded.

7.2 Local vertex cleaning algorithm

A quartet cleaning algorithm has *local vertex bound alpha* if it corrects all quartet errors across any vertex whose induced tripartition is α-bounded [14]. Note that a local vertex algorithm does not require the bound to hold for all vertices, hence such algorithms are more robust than those that require the bound to hold for all edges of the tree.

In our paper we will describe two cleaning algorithms with different local vertex bounds. A local vertex cleaning algorithm has been initially presented in [14], where the following fundamental result has been proved:

Lemma 7.2.1. *Let $\langle A_1, B_1, C_1\rangle$ and $\langle A_2, B_2, C_2\rangle$ be two 2-bounded tripartition of a set S. Then $\langle A_1, B_1, C_1\rangle$ and $\langle A_2, B_2, C_2\rangle$ are compatible.*

The proof stated in that paper is for the case of 4-bounded tripartitions, but it holds also for the case of F-bounded tripartitions, with $F \geq 2$. Hence, to describe a local vertex cleaning algorithm, it suffices to compute a set of 2-bounded tripartitions containing all F-bounded tripartitions.

Both algorithms that are described in our paper rely on the following procedure AssociatePartition, which associates a tripartition to a *triplet*, that is a set of three species $a, b, c \in S$.

AssociatePartition(a, b, c)
Input: Three species in S.
Let $A := \{a\}$; $B := \{b\}$; $C := \{c\}$.
For each $s \in S - \{a, b, c\}$ do
    If $(a, s|b, c) \in Q$ then $A := A \cup \{s\}$
    If $(s, b|a, c) \in Q$ then $B := B \cup \{s\}$
    If $(a, b|c, s) \in Q$ then $C := C \cup \{s\}$
EndFor
Return the tripartition $\langle A, B, C\rangle$

Given a tripartition $\langle A, B, C\rangle$ associated to the triplet a, b, c we will say that all quartets (a, b, c, s) with $s \in S - \{a, b, c\}$ *witness* the tripartition $\langle A, B, C\rangle$; clearly if the number of witnesses of a tripartition is large, then it is likely that such tripartition is correct.

In particular if there are no quartet errors in Q then the set of witnesses of a tripartition $\langle A, B, C\rangle$ is exactly the set of quartets across $\langle A, B, C\rangle$, which contains $|A||B||C|(|S|-3)/2$ elements, consequently correct tripartitions are likely to have a number of witnesses that is close to the maximum value.

Lemma 7.2.2. *Let (a, b, c, d) be a quartet of species in S. Then (a, b, c, d) may witness at most 4 tripartitions.*

Proof. By construction (a, b, c, d) may witness only tripartitions associated to a triplet in the set $\{a, b, c, d\}$. Since there exist exactly 4 such triplets the claim follows. □

Note that Lemma 7.2.2 implies that there are at most $O(n^4)$ tripartitions that have at least one witness.

Lemma 7.2.3. *Let $\langle A, B, C\rangle$ be a tripartition of the set S induced by the removal of a vertex in the tree, and let (a, b, c, s) be a quartet error with $a, s \in A$, $b \in b$, $c \in C$. Then at most $2|S| - 7$ missing witnesses of $\langle A, B, C\rangle$ are due to such error.*

Proof. By construction the set of witnesses is constructed from a triplet where each specie belongs to a distinct set of the tripartition. Moreover the only triplets whose associated partition may be incorrectly computed due to an error (a, b, c, s) are the triplets contained in the set $\{a, b, c, s\}$. W.l.o.g. we

can assume that $a, s \in A$, $b \in B$, $c \in C$, then only the partitions associated to the triplets (a, b, c) and (s, b, c) may be incorrectly different from $\langle A, B, C\rangle$ due to the error (a, b, c, s). The quartets considered in the computation of the partitions associated to such triplets are exactly $2|S|-7$ since (a, b, c, s) is common to both computations and the witnesses for the triplet (a, b, c) is the set $\{(a, b, c, z) : z \in S - \{a, b, c\}\}$. The same ideas apply also for (s, b, c). □

Lemma 7.2.3 suggests that a α-bounded partition must have a sufficiently large number of witnesses; in the next section we will prove that result for the case of $\alpha = 9$.

7.3 Properties of α-bounded tripartitions ($\alpha \geq 9$)

In this section we will describe a linear (i.e. $O(n^4)$) time algorithm that computes all α-bounded tripartitions for $d \geq 9$. This algorithm exploits the fact that α-bounded tripartitions must have a certain number of witnesses. The following lemma establishes the relationship between α and the number of witnesses.

Lemma 7.3.1. *Let $\langle A, B, C\rangle$ be a α-bounded tripartition, $\alpha \geq 9$, e is a constant such that $e \geq \frac{4\alpha}{\alpha-8}$, and $|A| + |B| + |C| \geq (\frac{12}{\alpha} - \frac{3e-6}{2e})e$, then there are at least $|A||B||C|(|S| - 3)/e$ witnesses of this tripartition*

Proof. Let $Q_{err(\alpha)} = \frac{1}{\alpha}((|A| - 1)(|B| + |C| - 1) + (|B| - 1)(|A| + |C| - 1) + (|C| - 1)(|A| + |B| - 1))$ be the maximum number of possible errors across a α-bounded tripartition for $\alpha \geq 2$. By Lemma 7.2.3 the number of witnesses of a α-bounded partition is at least

$$\frac{1}{2}|A||B||C|(|S| - 3) - Q_{err(\alpha)}(2|S| - 7). \tag{7.1}$$

In order to show that a α-bounded partition has at least $|A||B||C|(|S|-3)/e$ witnesses for some constant e, we need to prove the following inequality holds when $|A| \geq 1$, $|B| \geq 1$, $|C| \geq 1$:

$$Q_{err(d)}(2|S| - 7) \leq (\frac{1}{2} - \frac{1}{e})|A||B||C|(|S| - 3) \tag{7.2}$$

When all terms are expanded, this inequality becomes

$$\frac{1}{d}(4|A|^2(|B|+|C|)+4|B|^2(|A|+|C|)+4|C|^2(|A|+|B|)+12|A||B||C|+$$
$$27(|A|+|B|+|C|)-(26(|A||B|+|A||C|+|B||C|)+6(|A|^2+|B|^2+|C|^2)+21)\leq$$
$$(\frac{1}{2}-\frac{1}{e})(|A|^2|B||C|+|A||B|^2|C|+|A||B||C|^2-3|A||B||C|) \quad (7.3)$$

Since $27(|A|+|B|+|C|)-(26(|A||B|+|A||C|+|B||C|)+5(|A|^2+|B|^2+|C|^2)+21)$ is always less than zero, it suffices to prove that the following inequality holds:

$$\frac{1}{d}(4|A|^2(|B|+|C|)+4|B|^2(|A|+|C|)+4|C|^2(|A|+|B|)+$$
$$12|A||B||C|-(|A|^2+|B|^2+|C|^2))\leq$$
$$(\frac{1}{2}-\frac{1}{e})(|A|^2|B||C|+|A||B|^2|C|+|A||B||C|^2-3|A||B||C|) \quad (7.4)$$

We will prove (7.4) by determining when the following four inequalities hold:

$$\frac{4}{\alpha}(|A|^2(|B|+|C|)-|A|^2)\leq(\frac{1}{2}-\frac{2}{e})|A|^2|B||C| \quad (7.5)$$

$$\frac{4}{\alpha}(|B|^2(|A|+|C|)-|B|^2)\leq(\frac{1}{2}-\frac{2}{e})|B|^2|A||C| \quad (7.6)$$

$$\frac{4}{\alpha}(|C|^2(|A|+|B|)-|C|^2)\leq(\frac{1}{2}-\frac{2}{e})|C|^2|A||B| \quad (7.7)$$

$$\frac{12}{d}|A||B||C|\leq\frac{1}{e}|A||B||C|(|A|+|B|+|C|)-(\frac{1}{2}-\frac{1}{e})(3|A||B||C|) \quad (7.8)$$

Note that the first three of these inequalities cannot hold when if $\alpha<8$; hence, $\alpha\geq 9$. Given a valid value for α, these inequalities imply that $e\geq\frac{\alpha}{\alpha-8}$ by the following reasoning:

$$\frac{4}{\alpha}\leq\frac{1}{2}-\frac{2}{e}\Rightarrow\frac{4}{\alpha}-\frac{1}{2}\leq-\frac{2}{e}\Rightarrow\frac{\alpha-8}{2\alpha}\geq\frac{2}{e}\Rightarrow e\leq\frac{4\alpha}{\alpha-8} \quad (7.9)$$

Given valid values α and e, the fourth inequality is true under the assumptions that $|A|\geq 1$, $|B|\geq 1$, $|C|\geq 1$ and $|A|+|B|+|C|\geq e(\frac{12}{d}+\frac{3e-6}{2e})$ by the following reasoning:

$$\begin{aligned}
\tfrac{12}{\alpha}|A||B||C| &\leq \tfrac{1}{e}|A||B||C|(|A|+|B|+|C|)-(\tfrac{1}{2}-\tfrac{1}{e})(3|A||B||C|)\Rightarrow\\
\tfrac{12}{\alpha} &\leq \tfrac{1}{e}(|A|+|B|+|C|)-3(\tfrac{1}{2}-\tfrac{1}{e})\Rightarrow\\
\tfrac{12}{\alpha}+\tfrac{3e-6}{2e} &\leq \tfrac{1}{e}(|A|+|B|+|C|)\Rightarrow\\
(\tfrac{12}{\alpha}+\tfrac{3e-6}{2e})e &\leq |A|+|B|+|C|
\end{aligned} \quad (7.10)$$

This completes the proof. □

The reader can verify that lower values of α weaken the bounds on both the number of required witnesses and the size of S. For example, if $d = 9$, $e = 36$ and $|S| = 99$; however, if $d = 11$, $e = 12$ and $|S| = 29$.

The first step of our algorithm is to compute the tripartition associated to each triplet (a, b, c), by abuse of language with AssociatePartition(a, b, c) we will also denote the partition returned by the algorithm. Please note that after the first step is completed, it is possible to access to AssociatePartition(a, b, c) in constant time.

Since we can assume that there exists an ordering $s_1, \ldots, s_n$ of the elements in S, then we can represent a tripartition with a sequence of n integers $\langle p_1, \ldots, p_n \rangle$ where $p_i = 1$ if $s_i \in A$, $p_i = 2$ if $s_i \in B$, and $p_i = 3$ if $s_i \in C$. For instance the tripartition $\{a_1, a_4\}, \{a_2, a_5\}, \{a_3\}$ is stored as $\langle 1, 2, 3, 1, 2 \rangle$, while the partition $\{\{a_1, a_2\}, \{a_3, a_5\}, \{a_4\}\}$ is stored as $\langle 1, 1, 2, 3, 2 \rangle$. Such encoding allows to examine a partition in $O(n)$ time.

By Lemma 7.2.2 each quartet can witness at most 4 tripartitions, moreover in our algorithm all tripartitions considered are associated so some triplets. Consequently we will use a data structure, called $w(q)$ such that $w(q) = \{(a, b, c) : q \text{ witnesses AssociatedPartition}(a, b, c)\}$.

Since each call to AssociatePartition requires $O(n)$ time, the total time spent in the first For loop is $O(n^4)$. Computing the number of witnesses of a partition P can be done in $O(nt)$ time, where t is the number of tripletes such that P=AssociatePartition(t), by exploiting the array w and keeping a list of the witnesses of the partition P. Since $\sum_P |\{t : P = \text{AssociatePartition}(t)\}| = O(n^3)$, the total time spent in the second For loop is $O(n^4)$. As for the final loop, note that the set P_L prior to that loop contains only partitions with the property that the number of its witnesses is at least $|A||B||C|(|S| - 3)/e$ times the number of quartets across each partition. By Lemma 3.2., the total number of witnesses summed over all partitions is $O(n^4)$, Hence the total number of quartets across partitions that need to be examined in that loop is $O(n^4)$ and this final loop runs in $O(n^4)$ time. This leads to a $O(n^4)$ time complexity of the whole ConstructPartitions procedure.

To derive the tree associate with set P_L produced above, note that by Lemma 3.1, there is a tree compatible with the tripartitions in P_L, and moreover, this tree can be computed in $O(n^4)$ time by a standard algorithm [25].

7.4 Properties of 2-bounded partitions

A simple algorithm to compute a set of tripartitions from a set of quartets is the following:

ConstructPartitions(Q, d)
Input: A set Q of quartets over the species S.
output: The list P_L of α-bounded tripartitions of S induced by Q.
$P_L := \emptyset$
$e := d/(d-8)$
$size := e((12/d) + ((3e-6)/(2e)))$
If $|S| < size$ Then
    Get P_L from lookup table
    Return(P_L)
EndIf
Initialize partition-table P_T to be an empty table with $|S|^3/6$ rows, indexed by each triplet.
For each triplet $a, b, c \in S$ do
    $P_T(a, b, c)$:=AssociatePartition(a, b, c)
    For each $s \in S - \{a, b, c\}$ do
        $w(a, b, c, s)$:=$w(a, b, c, s) \cup \{P_T(a, b, c)\}$
    EndFor
EndFor
Sort the set of triplets $\{(a, b, c)\}$ using the $P_T(a, b, c)$ as keys.

/* In the sorted list, only streams of consecutive triplets can have the same associated tripartition.*/
Scan the sorted list of triplets and For each stream $t_i, \ldots t_{i+k}$ of consecutive triplets that have the same associated tripartition do
    Add P to the set P_L of computed partitions
    Compute the number of witnesses of P.
EndFor
For each partition $P = \langle A, B, C \rangle$ in P_L do
    Remove P from P_L if P has less than
$|A||B||C|(|A| + |B| + |C| - 3)/e$ witnesses.
EndFor
For each partition $P = \langle A, B, C \rangle$ in L do
    Remove P from P_L if P is not 2-bounded
EndFor
Return (P_L)

```
Clean(Q, S)
Input: A set Q of quartets over the species S.
Output: A set P of compatible tripartitions.
L := ∅
For each triplet a, b, c ∈ S do
    P := AssociatePartition(a, b, c)
    underlineIf P is 2-bounded then
        L := L ∪ {P}
    EndIf
EndFor
Return L
```

If there exists a triplet $(a, b, c,)$ with P=AssociatedPartition(a, b, c), the Clean algorithm is guaranteed to output a list L of partitions including P. Consequently we only have to ensure that we include also the 2-bounded partitions that are not associated to any triplet. exaclty P. Consequently we now have to deal only with the case where all tripartitions associated to the triplets $a \in A$, $b \in B$, $c \in C$ are incorrect.

Let us consider a 2-bounded tripartition $\langle A, B, C\rangle$, by Def. 7.1.2 the number of quartet errors across such tripartition is less than $\frac{1}{2}(2|A||B| + 2|A||C| + 2|B||C| - 3|A| - 3|B| - 3|C| + 3)$. Let $err(a, b, c)$ be the set $(Q_T - Q) \cap \{\{a, b, c, w\} : w \in S - \{a, b, c\}\}$; intuitively $err(a, b, c)$ consists of all quartet errors among the quartets considered inside a call to the procedure AssociatePartition(a, b, c). Since a single quartet error influences only the tripartitions associated to two distinct triplets, (a proof of this result can be found in [14]), for a 2-bounded tripartition $\langle A, B, C\rangle$ the sum of $|err(a, b, c)|$ over all triplets (a, b, c) with $a \in A$, $b \in B$, $c \in C$ is less than $2|A||B| + 2|A||C| + 2|B||C| - 3|A| - 3|B| - 3|C| + 3$. Since the number of triplets (a, b, c) is $|A||B||C|$ we can study the average number of errors per triplet which is equal to the following expression:

$$\frac{2|A||B| + 2|A||C| + 2|B||C| - 3|A| - 3|B| - 3|C| + 3}{|A||B||C|} \tag{7.11}$$

Clearly whenever such average value is strictly less than an integer avg there is a triplet (a, b, c) whose associated tripartition has at most $avg - 1$ quartet errors across it. Assume initially that $|C| = 1$, then the number of errors is $2|A||B| - |A| - |B|$, hence

$$\frac{2|A||B| - |A| - |B|}{|A||B|} < 2 \tag{7.12}$$

Assume now that $|C| = 2$, then

$$\frac{2|A||B| + |A| + |B| - 3}{2|A||B|} < 2 \tag{7.13}$$

Consequently if $|C| < 3$ there is a triplet whose associated tripartition has at most one error. Since the above inequalities hold for all $|A|, |B|$, and for simmetry of (7.11), we can now assume that $|A| > 2$, $|B| > 2$, $|C| > 2$. We will prove that the average number is less than 2 also in this new case.

$$\begin{gathered} 2|A||B| + 2|A||C| + 2|B||C| - 3|A| - 3|B| - 3|C| + 3 < 2|A||B||C| \Rightarrow \\ 2|A||B|(1 - \tfrac{|C|}{3}) + 2|A||C|(1 - \tfrac{|B|}{3}) + 2|B||C|(\\ 1 - \tfrac{|A|}{3}) < 3|A| + 3|B| + 3|C| - 3 \end{gathered}$$

Since $|A|, |B|, |C|$ are all greater than or equal to three, all terms in the left hand side of (7.14) are not greater than zero, while the right hand side is strictly positive.

Now assume that all sets A, B, C contain at least six elements, in this case we will prove that the average number of errors is less than one.

$$\begin{gathered} 2|A||B| + 2|A||C| + 2|B||C| - 3|A| - 3|B| - 3|C| + 3 < |A||B||C| \Rightarrow \\ |A||B|(2 - \tfrac{|C|}{3}) + |A||C|(2 - \tfrac{|B|}{3}) + |B||C|(\\ 2 - \tfrac{|A|}{3}) < 3|A| + 3|B| + 3|C| - 3 \end{gathered}$$

Just as in the previous case since $|A|, |B|, |C|$ are all greater than or equal to 6, then the left hand side cannot be greater than zero, while the right hand side is strictly positive. Consequently all tripartitions $\langle A, B, C\rangle$ such that all sets contain at least 6 elements are already computed by the procedure Clean described at the beginning of this section, while all other 2-bounded tripartitions have at most one error across it. A brute force procedure can correct such error, giving a set of tripartitions including the correct one.

The algorithm computing the set of all 2-bounded tripartitions follows directly.

To determine the time complexity of the algorithm note that the most expensive step is to check all tripartitions for 2-boundedness. The algorithm computes the $O(n^3)$ tripartitions associated to any triplet. Hence $O(n^7)$ time suffices to check all such tripartitions. An additional set of tripartitions can be computed by the procedure CorrectError. Such procedure receives a tripartition as input and can output $O(n)$ tripartitions, hence the total number of additional tripartitions is $O(n^4)$, since for each triplet at most one

CorrectError($\langle A, B, C\rangle$)
Input: A tripartition $\langle A, B, C\rangle$ of S has less than 7 elements.
$L := \emptyset$
For each $s \in S$ do
    L:=$L \cup \{\langle A \cup \{s\}, B, C\rangle\}$
    L:=$L \cup \{\langle A\cup, B \cup \{s\}, C\rangle\}$
    L:=$L \cup \{\langle A\cup, B, C \cup \{s\}\rangle\}$
EndFor
For each partition P in L do
    Remove P from L if P is not 2-bounded
EndFor
Return(P)

Clean(Q, S)
Input: A set Q of quartets over the species S.
$L := \emptyset$, $P := \emptyset$
For each triplet $a, b, c \in S$ do
    P:=AssociatePartition(a, b, c)
    If $|A| < 6$ or $|B| < 6$ or $|C| < 6$ then
        $L := L \cup$CorrectErrors(P)
    else
        $L := L \cup \{P\}$
    EndIf
EndFor
For each partition $P = \langle A, B, C\rangle$ in L do
    Remove P from L if P is not 2-bounded
EndFor
Return (L)

call to CorrectError is made. Anyway these tripartitions are not completely general, since the tripartitions given as input are all such that one set contains at most 6 elements; by construction of CorrectError the tripartitions returned as output differ from the one given as input by at most one element, this implise that all tripartitions returned by CorrectError have one set containing at most 7 elements. Consequently the number of quartets across a tripartition returned by CorrectError is $O(n^3)$. Hence the algorithm Clean computes all 2-bounded tripartitions in $O(n^7)$ time.

Appendix A

List of Problems

In the following we will give the formal definitions of all problems that have been mentioned in the thesis, stating the most relevant known results, prior to this thesis, regarding the computational complexity of such problems.

Problem 8 (Max Cut).
Instance: a edge-weighted undirected graph $G = \langle V, E \rangle$.
Solution: a bipartition (V_1, V_2) of V.
Goal: to maximize the total weight of the edges with exactly one endpoint in V_1.

This problem is $\mathcal{APX}$-hard [78], even on unweighted cubic graphs, that is over graphs where each vertex is incident on exactly 3 edges and all edges have weight 1 [2].

Problem 9 (Max Sat).
Instance: a weighted set of clauses of the form: $\bigvee x_i^{\alpha_i}$ where x_i^1 stands for x_i and x_i^0 stands for $\not{x}_i$.
Solution: a truth assignment to the variables x_i.
Goal: to maximize the total weight of the clauses that are satisfied by the assignment.

In [78] it has been proved that Max Sat is $\mathcal{APX}$-hard even when all clauses have weight 1, all clauses contain 2 variables and all variables appear exactly 5 times in the clauses in the instance. Both Max Cut and Max Sat admit a ptas on dense instances [8].

Problem 10 (Min Vertex Cover).
Instance: an unoriented graph $G\langle V, E \rangle$.
Solution: a cover of G, that is a subset $C \subseteq V$ such that C contains at least one of the endpoints of each edge $e \in E$.
Goal: to minimize the cardinality of the cover.

The problem is $\mathcal{APX}$-hard even on bounded degree graphs [78], and in particular on cubic graphs [62].

Problem 11 (Max Clique).
Instance: an unoriented graph $G\langle V, E\rangle$.
Solution: a clique, that is a subset $K \subseteq V$ such that the subgraph of G induced by K is complete (all pairs of vertices are an edge).
Goal: to maximize the cardinality of the clique.

Problem 12 (Min Independent Set).
Instance: an unoriented graph $G\langle V, E\rangle$.
Solution: an independent set, that is a subset $I \subseteq V$ such that the subgraph of G induced by I contains no edge.
Goal: to minimize the cardinality of the independent set.

Since a clique of a graph G is an independent set of the complement graph of G, the two last problems share the same (non-)approximability results. They cannot be approximated within factor $O(n^{1-\epsilon})$ for any $\epsilon > 0$ in polynomial time, unless $\mathcal{NP} = \mathcal{ZPP}$.

Problem 13 (Max Tridimensional Matching).
Instance: three disjoint sets X_1, X_2, X_3 and a set $M \subseteq X_1 \times X_2 \times X_3$ of triplets.
Solution: a *matching*, that is a subset $M_1 \subseteq M$ such no triplets in M_1 share a common element.
Goal: to maximize the cardinality of the matching.

The problem is $\mathcal{APX}$-hard [62].

Problem 14 (Longest Common Subsequence).
Instance: a sets $\mathcal{S}$ of sequences.
Solution: a *common subsequence* of $\mathcal{S}$, that is a sequence s such that s can be obtained from each sequence in $\mathcal{S}$ by removing some characters of such sequence.
Goal: to maximize the length of the subsequence.

Problem 15 (Shortest Common Supersequence).
Instance: a sets $\mathcal{S}$ of sequences.
Solution: a *common supersequence* of $\mathcal{S}$, that is a sequence s such that s can be obtained from each sequence in $\mathcal{S}$ by inserting some characters of such sequence.
Goal: to minimize the length of the subsequence.

Both problems are $\mathcal{NP}$-hard on binary alphabet and $\mathcal{APX}$-hard on general alphabet. Moreover LCS is as hard to approximate as Max Clique, while SCS cannot be approximated within $O(\log^{\delta} n)$ factor in polynomial time, unless $\mathcal{NP} \subseteq \mathcal{DTIME}[2^{polylog\, n}]$.

Problem 16 (Multiple Sequence Alignment with SP-score).
Instance: a sets $\mathcal{S}$ of sequences over alphabet Σ, a scoring function $d : (\Sigma \cup \{\Delta\}) \times (\Sigma \cup \{\Delta\}) \to \mathbb{N}$
Solution: an *alignment* of $\mathcal{S}$, that is a matrix of $|\mathcal{S}|$ rows, where in the rows there are the sequences in $\mathcal{S}$, eventually with spaces (Δ) inserted.
Goal: the cost of a column of the alignment is the sum of $d(a, b)$ over all pairs (a, b) of distinct cells in the column. The cost of an alignment is the sum of the costs of all columns, to be minimized.

This problem is $\mathcal{NP}$-hard for a score function that is not a metric [97].

Problem 17 (Gapped Multiple Sequence Alignment with SP-score).

Instance: a sets $\mathcal{S}$ of sequences over alphabet Σ, a scoring function $d : (\Sigma \cup \{\Delta\}) \times (\Sigma \cup \{\Delta\}) \to \mathbb{N}$
Solution: an *alignment* of $\mathcal{S}$, that is a matrix of $|\mathcal{S}|$ rows, where in the rows there are the sequences in $\mathcal{S}$, eventually with spaces (Δ) inserted.
Goal: the cost of a column of the alignment is the sum of $d(a, b)$ over all pairs (a, b) of distinct cells in the column. The cost of an alignment is the sum of the costs of all columns, to be minimized.

The version of Gapped Multiple Sequence Alignment with SP-score where all gaps are restricted to be either at the beginning or at the end of the aligned sequences is called Gap-0 Alignment. The restriction of the latter problem where only one space can be inserted is called Gap-0-1 Alignment.

Problem 18 (Min Space-L Multiple Alignment).
Instance: a set of sequences $\{t_1, \ldots, t_k\}$ and a scoring scheme (d_M, g).
Solution: a space-L multiple alignment $\mathcal{A}$ of $\langle t_1, \ldots, t_k \rangle$.
Goal: the cost of a column of the alignment is the sum of $d(a, b)$ over all pairs (a, b) of distinct cells in the column. The cost of an alignment is the sum of the costs of all columns, to be minimized.

Problem 19 (Maximum Weighted Trace Alignment).
Instance: an alignment graph $G = \langle V, E, F \rangle$ and a weight for each each edge in E.
Solution: a trace graph $G_1 = \langle V, E_1, F \rangle$, with $E_1 \subseteq E$, that is a graph with

no cycle containing an edge in F.
Goal: to maximize the sum of weights of all edges in E_1

The problem is $\mathcal{NP}$-hard [68].

Problem 20 (Maximum Quartet Consistency).
Instance: a set Q of quartet topologies over a species set S
Solution: an evolutionary tree T with leaves S.
Goal: to maximize the quartets induced from T that are also in Q.

The problem is $\mathcal{NP}$-hard, but admits a ptas, when Q contains all quartets over S [57, 14]. It is $\mathcal{APX}$-hard in the general case [89].

Problem 21 (Maximum Isomorphic Agreement Subtree).
Instance: a set $\mathcal{T} = \{T_1, \ldots, T_m\}$ of S-labeled trees.
Solution: an S^*-labeled tree T^*, with $S^* \subseteq S$, such that T^* is isomorphic to a subtree of all trees in $\mathcal{T}$.
Goal: to maximize $|S^*|$.

The problem is $\mathcal{APX}$-hard [4].

Problem 22 (Maximum Agreement Subtree).
Instance: a set $\mathcal{T} = \{T_1, \ldots, T_m\}$ of S-labeled trees.
Solution: an S^*-labeled tree T^*, with $S^* \subseteq S$, such that T^* does not contain any internal node (with the possible exception of the root) of degree 2 and, for each tree $T_i \in \mathcal{T}$, T^* is homeomorphic to a subtree of T_i.
Goal: to maximize $|S^*|$.

The problem cannot be approximated within $\log^{\delta} n$ factor in polynomial time, unless $\mathcal{NP} \subseteq \mathcal{DTIME}[2^{polylog\, n}]$ [51].

Problem 23 (Maximum Weighted Agreement Subtree).
Instance: a set $\mathcal{T} = \{T_1, \ldots, T_m\}$ of S-labeled trees, where each edge is labeled with a positive weight.
Solution: an S^*-labeled tree T^*, with $S^* \subseteq S$, such that, for each pair of leaves $s_1, s_2 \in S^*$, the distance (as sum of weights) between s_1 and s_2 is the same in all trees.
Goal: to maximize $|S^*|$.

Problem 24 (Maximum Interval Weighted Subtree).
Instance: a set $\mathcal{T} = \{T_1, \ldots, T_m\}$ of S-labeled trees, where each edge is labeled with an interval $[a, b]$.
Solution: an S^*-labeled tree T^*, with $S^* \subseteq S$, where each edge is labeled by a weight such that, for each pair of leaves $s_1, s_2 \in S^*$, the distance (as sum

of weights) between s_1 and s_2 belongs to the interval (as sum of extremal points of all intervals in the path) in all trees.
Goal: to maximize $|S^*|$.

The last two problems are more general versions of MIT and MHT, hence they inherit their inapproximability results.

Problem 25 (Switchboard Location).
Instance: a set of disjoint sets $\{R_1, \ldots, R_k\}$ and a function $d : (R_1 \cup \cdots \cup R_k) \times (R_1 \cup \cdots \cup R_k) \to \mathbb{Q}$ such that $x_i \neq x_j \Rightarrow d(x_i, x_j) > 0$.
Solution: a set $\langle x_1 \ldots, x_k \rangle$ of points such that $x_i \in R_i$ for $1 \leq i \leq k$.
Goal: to minimize $\sum_{1 \leq i < j \leq k} d(x_i, x_j)$.

Bibliography

[1] A. Aho, J. Hopcroft, and J. Ullman. *Data Structures and Algorithms*. Addison-Wesley, Reading, MA, 1983.

[2] P. Alimonti and V. Kann. Hardness of approximating problems on cubic graphs. In *CIAC97*, volume 1203 of *LNCS*, pages 288–298, 1997.

[3] S. Altschul and B. Erickson. Locally optimal subalignments using non-linear similarity functions. *Bulletin of Mathematical Biology*, 48:633–660, 1986.

[4] A. Amir and D. Keselman. Maximum agreement subtree in a set of evolutionary trees: Metrics and efficient algorithms. *SIAM Journal on Computing*, 26(6):1656–1669, 1997.

[5] A. Apostolico and C. Guerra. The longest common subsequence problem revisited. *Algorithmica*, 18(1):1–11, 1987.

[6] E. M. Arkin and M. Hassin. Minimum diameter covering problems. Technical report, 1997.

[7] V. Arlazarov, E. Dinic, M. Kronrod, and I. Faradzev. On the economic construction of the transitive closure of a direct graph. *Soviet. Math. Dokl.*, 11(5):1209–1210, 1970.

[8] S. Arora, D. Karger, and M. Karpinski. Polynomial time approximation schemes for dense instances of $\mathcal{NP}$-hard problems. *Journal of Computer and System Sciences*, 58:193–210, 2000.

[9] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. *Journal of ACM*, 45(3):501–555, 1998.

[10] V. Bafna, E. Lawler, and P. Pevzner. Approximation algorithms for multiple sequence alignment. *Theoretical Computer Science*, 182:233–244, 1997.

[11] P. Barone, P. Bonizzoni, G. Della Vedova, and G. Mauri. An approximation algorithm for the shortest common supersequence: An experimental analysis. In *Proceedings of the 15th ACM Symposium on Applied Computing (SAC200)*, 2001.

[12] A. Ben-Dor, B. Chor, D. Graur, R. Ophir, and D. Pellep. From four-taxon trees to phylogenies: The case of mammalian evolution. In *Proceedings of the 2nd Annual International Conference on Computational Molecular Biology*, pages 9–19, 1998.

[13] V. Berry and O. Gascuel. Inferring evolutionary trees with strong combinatorial evidence. In *Proceedings of the 3rd International Computing and Combinatorics Conference*, pages 111–120, 1997.

[14] V. Berry, T. Jiang, P. E. Kearney, M. Li, and T. Wareham. Quartet cleaning: Improved algorithms and simulations. In *Proceedings of the 7th European Symposium on Algorithm (ESA99)*, pages 313–324, 1999.

[15] H. Bodlaender, R. Downey, M. Fellows, M. Hallett, and H. Wareham. Parameterized complexity analysis in computational biology. *Computer Applications in the Biosciences (CABIOS)*, 11(1):49–57, 1995.

[16] M. Boguski. Bioinformatics - a new era. In S. Brenner, F. Lewitter, M. Patterson, and M. Handel, editors, *Trends Guide to Bioinformatics*, pages 1–3. Elsevier Science, 1998.

[17] P. Bonizzoni, M. D'Alessandro, G. Della Vedova, and G. Mauri. Experimenting an approximation algorithm for the LCS. In *Algorithms and Experiments (ALEX98)*, pages 96–102, 1998.

[18] P. Bonizzoni and G. Della Vedova. Multiple sequence alignment with score scheme that is a metric is NP-complete. *Theoretical Computer Science*, to appear.

[19] P. Bonizzoni, G. Della Vedova, and T. Jiang. Approximating maximum trace alignment and its complement. manuscript, 1999.

[20] P. Bonizzoni, G. Della Vedova, and G. Mauri. Approximating the maximum isomorphic agreement subtree is hard. In *Proceedings of the 11th Symposium on Combinatorial Pattern Matching (CPM2000)*, volume 1848 of *LNCS*, pages 119–128, 2000.

[21] P. Bonizzoni, G. Della Vedova, and G. Mauri. Approximating the maximum isomorphic agreement subtree is hard. *International Journal on the Foundations of Computer Science*, to appear.

[22] P. Bonizzoni, G. Della Vedova, and G. Mauri. Experimenting an approximation algorithm for the LCS. *Discrete Applied Mathematics*, to appear.

[23] P. Bonizzoni and A. Ehrenfeucht. Approximation of the shortest common supersequence with ratio less than the alphabet size. Technical Report TR 149-95, Dipartimento di Scienze Della Informazione, Universitá Degli Studi di Milano, 1996.

[24] D. Bryant. *Building Trees, Hunting for Trees, and Comparing Trees.* PhD thesis, University of Canterbury, 1997.

[25] P. Buneman. The recovery of trees from measures of dissimilarity. In *Mathematics in the Archaelogical and Historical Sciences.* Edinburgh University Press, 1971.

[26] H. Carrillo and D. Lipman. The multiple sequence alignment in biology. *SIAM Journal of Applied Mathematics*, 48:1073–1082, 1988.

[27] S. Chan, A. Wong, and D. Chiu. A survey of multiple sequence comparison methods. *Bulletin of Mathematical Biology*, 54(4):563–598, 1992.

[28] R. Cole, M. Farach, R. Hariharan, T. Przytycka, and M. Thorup. An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees. *SIAM Journal on Computing*, to appear.

[29] R. Cole and R. Hariharan. An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees. In *Proc. of the 7thAnnual ACM-SIAM Symposium on Discrete Algorithms (SODA96)*, pages 323–332, 1996.

[30] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[31] G. Della Vedova and H. T. Wareham. Optimal algorithms for local vertex quartet cleaning. manuscript, 1999.

[32] P. Erdös, M. Steel, L. A. Székely, and T. Warnow. Constructing big trees from short sequences. In *Automata, Languages and Programming, 24th International Colloquium (ICALP97)*, volume 1256 of *Lecture Notes in Computer Science*, pages 827–837, 1997.

[33] G. Even, J. Naor, and L. Zosin. An 8-approximation algorithm for the subset feedback edge set problem. In *Proc. of the 37th Symposium on the Foundations of Computer Science (FOCS 96)*, pages 310–319, 1996.

[34] M. Farach, T. M. Przytycka, and M. Thorup. On the agreement of many trees. *Information Processing Letters*, 55(6):297–301, 1995.

[35] M. Farach and M. Thorup. Fast comparison of evolutionary trees. *Information and Computation*, 123(1):29–37, 1995.

[36] M. Farach and M. Thorup. Sparse dynamic programming for evolutionary-tree comparison. *SIAM Journal on Computing*, 26(1):210–230, 1997.

[37] J. Felsenstein. Cases in which parsimony and compatibility will be positively misleading. *Systematic Zoology*, 27:401–410, 1978.

[38] J. Felsenstein. Evolutionary trees from DNA sequences: A maximal likelihood approach. *Journal of Molecular Evolution*, 17:368–386, 1981.

[39] J. Felsenstein. Numerical methods for inferring evolutionary trees. *Quart. Rev. Biol.*, pages 379–404, 1982.

[40] C. Finden and A. Gordon. Obtaining common pruned trees. *Journal of Classification*, 2:255–276, 1985.

[41] W. Fitch. Letter to the editor: Commentary on the letter by Ward C. Wheeler. *Mol. Biol. Evol.*, 10(3):713–714, 1993.

[42] W. M. Fitch. Toward defining the course of evolution: Minimal change for a specific tree topology. *Systematic zoology*, 20:406–441, 1971.

[43] D. Foulser, M. Li, and Q. Yang. Theory and algorithms for plan merging. *Artificial Intelligence*, 57:143–181, 1992.

[44] M. Garey and D. Johnson. *Computer and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman, 1979.

[45] L. Gasieniec, J. Jansson, A. Lingas, and A. Östlin. On the complexity of constructing evolutionary trees. *Journal of Combinatorial Optimization*, 3(2/3):183–197, 1999.

[46] R. Gupta and B. Golding. Evolution of hsp70 gene and its implications regarding relationships between archaebacteria, eubacteria and eukaryotes. *Journal of Molecular Evolution*, 37:573–582, 1993.

[47] D. Gusfield. Efficient methods for multiple sequence alignment with guaranteed error bounds. *Bulletin Mathematical Biology*, 55:141–154, 1993.

[48] D. Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, Cambridge, 1997.

[49] K. Hakata and H. Imai. The longest common subsequence problem for small alphabet size between many strings. In *Proc. 3rd International Symposium on Algorithms and Computation (ISAAC)*, volume 650 of *LNCS*, pages 469–478. Springer Verlag, 1992.

[50] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, to appear.

[51] J. Hein, T. Jiang, L. Wang, and K. Zhang. On the complexity of comparing evolutionary trees. *Discrete Applied Mathematics*, 71:153–169, 1996.

[52] D. M. Hillis, C. Moritz, and B. K. mable, editors. *Molecular Systematics*. Sinauer Ass., 2nd edition, 1996.

[53] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Communications of ACM*, 18(6):341–343, 1975.

[54] W. Hsu and M. Du. New algorithms for the LCS problem. *Journal of Computer and Systems Sciences*, 19:133–152, 1984.

[55] R. W. Irving and C. B. Fraser. On the worst-case behaviour of some approximation algorithms for the shortest common supersequence of k strings. In *Combinatorial Pattern Matching (CPM93)*, volume 684 of *LNCS*, pages 63–73, 1993.

[56] G. Jacobson and K.-P. Vo. Heaviest increasing/common subsequence problem. In *Proceedings of the 3rd Combinatorial Pattern Matching*, volume 644 of *LNCS*, pages 52–66, 1992.

[57] T. Jiang, P. Kearney, and M. Li. Orchestrating quartets: Complexity and approximation. In *Proc. of the 39th Conf. on the Foundations of Computer Science (FOCS98)*, 1998.

[58] T. Jiang, P. Kearney, and M. Li. Open problems in computational biology. *Journal of Algorithm*, 35(2):111–222, 2000.

[59] T. Jiang and M. Li. On the approximation of shortest common supersequences and longest common subsequences. *SIAM Journal on Computing*, 24(5):1122–1139, 1995.

[60] W. Just. Computational complexity of multiple sequence alignment with sp-score. submitted, 1999.

[61] W. Just and G. Della Vedova. Multiple sequence alignment as a facility location problem. In *Proceedings of the Prague Stringology Club Workshop 2000 (PSCW2000)*, 2000.

[62] V. Kann. Maximum bounded 3-dimensional matching is MAX SNP-complete. *Information Processing Letters*, 37(1):27–35, 1991.

[63] V. Kann. On the approximability of the maximum common subgraph problem. In *Proc. 9th Ann. Symp. on Theoretical Aspects of Comput. Sci. (STACS92)*, volume 577 of *LNCS*, pages 377–388, 1992.

[64] M.-Y. Kao. Tree contractions and evolutionary trees. *SIAM Journal on Computing*, to appear.

[65] M.-Y. Kao, T. W. Lam, T. M. Przytycka, W.-K. Sung, and H.-F. Ting. General techniques for comparing unrooted evolutionary trees. In *Proceedings of the 29th Symposium on the Theory of Computing (STOC97)*, pages 54–65, 1997.

[66] D. Karger, R. Motwani, and G. Ramkumar. On approximating the longest path in a graph. *Algorithmica*, 18(1):82–98, 1997.

[67] P. Kearney. The ordinal quartet method. In *Proceedings of the 2nd Annual International Comnference on Computational Molecular Biology (RECOMB98)*, pages 125–134, 1998.

[68] J. Kececioglu. The maximum weight trace problem in multiple sequence alignment. In *Proceedings of the 4th Symposium on Combinatorial Pattern Matching (CPM93)*, volume 684 of *LNCS*, pages 106–119, 1993.

[69] J. D. Kececioglu, H.-P. Lenhof, K. Mehlhorn, P. Mutzel, K. Reinert, and M. Vingron. A polyhedral approach to sequence alignment problems. *Discrete Applied Mathematics*, to appear.

[70] E. Kubicka, G. Kubicki, and F. McMorris. An algorithm to find agreement subtrees. *Journal of Classification*, 12(1):91–99, 1995.

[71] T. Lam, W. Sung, and H. Ting. Computing the unrooted maximum agreement subtree in subquadratic time. In *Proc. of the 5th Scandinavian Workshop on ALgorithms Theory (SWAT)*, LNCS, pages 124–135, 1996.

[72] A. Lesk. Computational molecular biology. In A. Kent and J. Williams, editors, *Encyclopedia of Computer Science and Technology*, volume 31, pages 101–165. Marcel Dekker, New York, 1994.

[73] V. Levenstein. Binary codes capable of correcting insertions and reversals. *Sov. Phys. Dokl.*, 10:707–710, 1966.

[74] W.-H. Li. *Molecular Evolution.* Sinauer Assoc., 1997.

[75] D. Maier. The complexity of some problems on subsequences and supersequences. *Journal of the ACM*, 25:322–336, 1978.

[76] W. Masek and M. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18–31, 1980.

[77] B. Morgenstern, A. Dress, and T. Werner. Multiple dna and protein sequence alignment based on segment-to-segment comparison. *Proceedings of the National Academy of Sciences*, 93:12098–12103, 1996.

[78] C. Papadimitriou and M. Yannakakis. Optimization, approximation and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

[79] C. H. Papadimitriou. *Computational Complexity.* Addison Wesley, 1993.

[80] M. Paterson and V. Dancik. Longest common subsequences. In *Mathematical Foundations of Computer Science, 19th International Symposium (MFCS)*, volume 841 of *LNCS*, pages 127–142, 1994.

[81] P. Pevzner. Matrix longest common subsequence problem, duality and Hilbert bases. In *Proceedings of the 3rd Combinatorial Pattern Matching*, volume 644 of *LNCS*, pages 79–89, 1992.

[82] K.-J. Räihä and E. Ukkonen. The shortest common supersequence problem over binary alphabet is NP-complete. *Theoretical Computer Science*, 16:187–198, 1981.

[83] C. Rick. A new flexible algorithm for the longest common subsequence problem. In *Proceedings of the 5rd Combinatorial Pattern Matching*, volume 937 of *LNCS*, pages 340–351, 1995.

[84] N. Saitou and M. Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4:406–425, 1987.

[85] D. Sankoff and J. Kruskal, editors. *Time Warps, String Edits and Macromolecules: the Theory and Practice of Sequence Comparisons*. Addison Wesley, 1983.

[86] T. Sellis. Multiple query optimization. *ACM Trans. Database Systems*, 13:23–52, 1988.

[87] T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[88] E. Smolenskii. *Jurnal Vicisl. Mat. i Matem. Fiz.*, 2(2):371–372, 1962.

[89] M. Steel. The complexity of reconstructing trees from qualitative characters and subtree. *Journal of Classification*, 9:91–116, 1992.

[90] M. Steel and T. Warnow. Kaikoura tree theorems: Computing the maximum agreement subtree. *Information Processing Letters*, 48(2):77–82, 1993.

[91] J. Storer. *Data Compression: Methods and Theory*. Computer Science Press, 1988.

[92] K. Strimmer and A. von Haeseler. Quartet puzzling: A quartet maximum-likelihood method for reconstructing tree topologies. *Molecular Biology and Evolution*, 13(7):964–969, 1996.

[93] A. Tamir. A distance constrained p-facility location problem on the real line. *Mathematical Programming*, 66:201–204, 1994.

[94] V. G. Timkovsky. On the approximation of shortest common non-subsequences and supersequences. Technical report, Dept. of Computer Science and Systems, McMaster University, Hamilton, 1993.

[95] L. Vigilant, M. Stoneking, H. Harpending, H. Hawkes, and A. Wilson. African populations and the evolution of human mitochondrial DNA. *Science*, 253:1503–1507, 1991.

[96] M. Vingron and P. Pevzner. Multiple sequence comparison and consistency on multipartite graphs. *Advances in Applied Mathematics*, 16:1–22, 1995.

[97] L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1(4):337–348, 1994.

[98] W. Wheeler. Letter to the editor: The triangle inequality and character analysis. *Mol. Biol. Evol.*, 10(3):707–712, 1993.

[99] N. Williams. Closing in on the complete yeast genome sequence. *Science*, 268:1560–1561, 1995.

List of Figures