ALGORITHMS FOR MULTIPLE SEQUENCE ALIGNMENT, COMPARISON OF TREES, AND STEINER TREES

By

LUSHENG WANG, M. Sc. (University of Regina)

M. Eng. (Shandong University)

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfilment of the Requirements

for the Degree

DOCTOR OF PHILOSOPHY

McMaster University

March 1995

ALGORITHMS FOR MULTIPLE SEQUENCE ALIGNMENT, COMPARISON OF TREES, AND STEINER TREES

DOCTOR OF PHILOSOPHY (1995) MCMASTER UNIVERSITY

(Electrical and Computer Engineering) Hamilton, Ontario

TITLE: **Algorithms for Multiple Sequence Alignment, Comparison of Trees, and Steiner Trees**

AUTHOR: Lusheng Wang

M. Sc. (University of Regina)

M. Eng. (Shandong University)

SUPERVISOR: Dr. Tao Jiang

Associate Professor

Department of Computer Science and Systems

B.Sc. (Univ. of Sci. and Tech. of China, Hefei)

Ph.D. (Minnesota)

NUMBER OF PAGES: xi, 112

Abstract

Comparison of sequences and trees is an essential problem in computational biology. In this thesis, we investigate some algorithmic problems in sequence and tree comparison. Theoretical issues such as computational complexity, the efficiency of algorithms, and the performance of approximation algorithms are stressed.

The most popular approach for comparing a set of sequences is multiple sequence alignment. We show that two popular variants of multiple sequence alignment, multiple alignment with SP-score and tree alignment, are NP-complete. We also design a polynomial time approximation scheme (PTAS) for tree alignment, which is believed to be the first PTAS in computational biology.

Tree comparison has applications in the study of RNA secondary structures and evolutionary trees. We propose the notion of alignment of trees as a new method for comparing RNA secondary structure trees and give an efficient algorithm for computing the optimal alignment of such trees. Several other methods for comparing evolutionary trees are also considered.

Steiner trees have been studied extensively in the literature, and are closely related to tree alignment. We also design a PTAS for some planar Steiner tree problems.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Dr. T. Jiang for his expert guidance, continued assistant, and supervision throughout the course of this work. Thanks also go to Dr. G.B. Golding, Dr. Z-Q.T. Luo and Dr. D.L. Parnas, the members of the Supervisory Committee, for their useful suggestions.

I would also like to acknowledge the financial support provided by the National Science and Engineering Research Council of Canada and the Canadian Genome Analysis and Technology Program. Much appreciation goes to Dr. X. Chen for his valuable discussions, cooperation, and especially supplying the sequence analysis tool (SAT) which allows me to test Algorithm 4.1 with real data.

Thanks to my wife and parents, who have given me their constant support and encouragement.

Contents

List of Figures

Chapter 1

Introduction

The modern era of molecular biology began with the discovery of the double helical structure of DNA. Today, sequencing nucleic acids, the determination of genetic information at the most fundamental level, is a major tool of biological research [85]. This revolution in biology has created a huge amount of data at great speed by directly reading DNA sequences. The growth rate of data volume is exponential. For instance, the volume of DNA and protein sequence data is currently doubling every 22 months [59]. The large amount of data poses a serious challenge in storing, retrieving and analyzing biological information.

A new field, *computational biology*, is emerging to meet the rapidly increasing computational need. It consists of many important areas such as information storage and retrieval, biological sequence analysis, and protein structure prediction, and so on [25, 53]. Information storage and retrieval require efficient methods to accurately process experimentally gathered data, including the design of genomic databases and the detection of the homology (or similarity) between biological sequences. In sequence analysis, efficient and effective sequence comparison algorithms are crucial in dealing elegantly with the insertion, deletion

and replacement events that occur in biological sequences. Structural prediction is also of great interest to us. Today, one can predict where a helix will occur with about 60 or 70 percent accuracy by various methods. Almost all the prediction work is done on computers [25].

In many areas of computational biology, one often models the biological reality with some numeric criteria and expects a solution which minimizes or maximizes those criteria [50]. Thus, many new combinatorial optimization problems have been proposed, most of which are difficult computational problems. On the other hand, solutions to these optimization problems are needed desperately in biological research. In some cases, *e.g.* in sequence analysis, even approximate solutions can be very helpful. It can thus be expected that computer scientists will be facing more and more challenges to provide efficient algorithms and heuristics for these optimization problem.

This thesis investigates some optimization problems arising in several important areas of computational biology, including sequence analysis, phylogenetic reconstruction, and structure prediction. Theoretical issues such as computational complexity, the efficiency of algorithms, and the performance of approximation algorithms are stressed. The central theme of the thesis is the comparison of sequences and trees. An overview of the results is given below:

- **Sequence Comparison**

Multiple sequence alignment is one of the most popular approaches to sequence comparison and plays an essential role in finding conserved subregions among a set of sequences, and inferring the evolutionary history of some species [30]. Many methods have been proposed to measure the quality of an alignment and they result in several variants of multiple sequence alignment [10, 31, 44, 53, 66, 85]. The two most important variants are *multiple*

alignment with SP-score and *tree alignment.* A lot of efforts have been made in the design of algorithms for these two problems. However, the computational complexity status of them was left open. We show that both of the problems are NP-complete. This implies that there does not exist polynomial time algorithms to compute optimal solutions of these two problems.

In sequence analysis, approximate solutions are very useful since they may still convey important hints and clues to the function or structure of the sequences being compared [53]. We devise a polynomial time approximation scheme (PTAS) for tree alignment. Theoretically, the PTAS allows us to obtain an approximate solution arbitrarily close to the optimal solution in polynomial time. This is the first PTAS in computational biology to our knowledge. Moreover, we consider a generalized version of tree alignment, called *generalized tree alignment* and show that it is MAX SNP-hard. This implies that it is unlikely to have a PTAS for generalized tree alignment and [3] forms a sharp contrast with the PTAS for tree alignment.

- **Comparison of Labeled Trees**

RNA secondary structures can be expressed in terms of ordered labeled trees [72, 86]. The comparison of RNA secondary structures can help identify conserved structural motifs in an RNA folding process [54] and construct taxonomy trees [72]. Extending the notion of sequence alignment, we propose the *alignment of trees* as a new measure for comparing labeled trees. We design an efficient algorithm for computing the alignment distance of trees, which is faster than the best known algorithm for computing the edit distance of trees.

- **Comparison of Evolutionary Trees**

The evolutionary history of a set of species is described by an unordered partially labeled tree, which is called an *evolutionary tree* or a *phylogeny*. Reconstructing the evolutionary history for a set of species is a fundamental yet difficult task in evolutionary genetics. Different methods may lead to different evolutionary trees. Thus, it is interesting to design methods to compare different evolutionary trees for the same set of species We consider several methods such as the maximum agreement subtree, the maximum refinement subtree, and the subtree-transfer distance. The computational complexity of these problems are settled, and an approximation algorithm with performance ratio 3 for subtree-transfer distance is designed.

- **The Approximation of Steiner Trees**

The tree alignment problem can be viewed as a special case of the well-known *Steiner tree problem under a given topology*. The PTAS designed for tree alignment in fact works for the Steiner tree problem under a given topology in any metric space. We will also consider some planar Steiner tree problems in this thesis. A PTAS is given for the planar Steiner tree problem when the given set of regular points is *c-local*. The algorithm works for both Euclidean and rectilinear metrics.

The thesis is organized as follows: Chapter 2 includes some fundamental concepts such as sequence alignment, alignment of trees, Steiner trees, approximation algorithms and MAX SNP-hardness. Chapter 3 studies the computational complexity of multiple sequence alignment. The PTAS for tree alignment is given in Chapter 4. Chapter 5 discusses the alignment of labeled trees and Chapter 6 focuses on the comparison of evolutionary trees. In Chapter 7, some planar Steiner tree problems are considered. Finally, the main contributions of this thesis are summarized in Chapter 8.

Chapter 2

Some Fundamental Concepts and Notations

This chapter introduces some general concepts and notations needed in the thesis. First, we discuss two basic methods for the comparison of sequences, namely, string edit and sequence alignment. We then extend these methods to trees. A brief review of Steiner trees is also given. Finally, we include a brief exposition of approximation algorithms and the recently developed MAX SNP-hardness theory.

2.1 Comparison of Sequences

A *sequence* is a string over some alphabet Σ. The primary structures of deoyribonucleic acid (DNA), ribonucleic acid (RNA), and protein molecules are sequences of nucleotides or amino acid residues. For DNA sequences, the alphabet, Σ, contains four letters A, C, G, and T, representing four distinct nucleotides *adenine*, *cytosine*, *guanine* and *thymine*. For

RNA sequences, the nucleotide *uracil* (U) replaces thymine (T), *i.e.*, Σ contains four letters A, C, G, and U. For protein sequences, Σ contains 20 letters, each representing a unique amino acid [39].

To compare two sequences, one of the most popular methods is *string edit*, which uses three edit operations to transform one string into the other [66]. The three edit operations are *insertion*, *deletion* and *replacement* defined below. Let $s = s_1 s_2 \ldots s_n$ be a sequence. Inserting a letter a into s at the i-th position, we obtain a sequence $s_1 s_2 \ldots s_{i-1} a s_i \ldots s_n$. Deleting the letter at the i-th position, we obtain a sequence $s_1 s_2 \ldots s_{i-1} s_{i+1} \ldots s_n$. Replacing the letter at the i-th position with a letter a, we obtain a sequence $s_1 s_2 \ldots s_{i-1} a s_{i+1} \ldots s_n$. For example, we can obtain string $s_1 = ACGT$ from $s_2 = ACCGA$, by deleting a C and replacing the letter A with T. Each operation is assigned a *score*. The value of an editing sequence transforming one string into the other is the sum of the scores of all its edit operations. An optimal editing sequence is one with the smallest value. The *edit distance* between two strings is the value of an optimal editing sequence transforming one string into the other.

Another way of comparing two sequences is to align them [66]. In order to obtain an *alignment* of two sequences s_1 and s_2, we insert spaces into or at either end of s_1 and s_2 such that the two resulting sequences s_1' and s_2' are of the same length and then overlay them. Once we have the alignment, we can compare these two sequences column by column. Each column contains two letters, which are called *opposing* letters. A space is viewed as a new letter and is denoted as Δ throughout this thesis. Two opposing identical letters form a *match* and two opposing nonidentical letters form a *mismatch*, or a *replacement*. A space in one sequence opposite to a letter x in the other can be viewed as a *deletion* of x from the second sequence, or an *insertion* of x into the first sequence.

```
s1:    AC GT
s2:    ACCGA
```

Figure 2.1: An alignment with two mismatches.

Example 2.1. Let $s_1 = ACGT$ and $s_2 = ACCGA$. An alignment of s_1 and s_2 is shown in Figure 2.1.

Suppose that l is the length of the sequences s_1' and s_2'. The value of the alignment is defined as $\sum_{i=1}^{l} \mu(s_1'(i), s_2'(i))$, where $s_1'(i)$ and $s_2'(i)$ denote the two letters at the i-th column of the alignment, and $\mu(s_1'(i), s_2'(i))$ denotes the *score* of the two opposing letters under some given *score scheme* μ. There are several popular score schemes for amino acids and for nucleotides [47, 70]. A standard assumption about a score scheme μ is that it is a distance metric, namely,

1. $\mu(a, b) = 0$ if $a = b$.
2. $\mu(a, b) = \mu(b, a)$ for all a and b.
3. $\mu(a, c) \leq \mu(a, b) + \mu(b, c)$ for all a, b, c.

An *optimal alignment* of two sequences is one that *minimizes* the value over all possible alignments. The *alignment distance* between two sequences is defined as the minimum alignment value of the two sequences. The alignment distance can also be used as a measure of the similarity of two sequences. The smaller the alignment distance is, the more similar the two sequences are. It is well-known that the alignment distance between two sequences is equal to their edit distance [66, 85].

The concept of an alignment can be easily extended to more than two sequences. A *multiple alignment* $\mathcal{A}$ of $k \geq 2$ sequences is obtained as follows: spaces are inserted into each sequence so that the resulting sequences s_i' $(i = 1, 2, \ldots k)$ have the same length l, and

```
s1:  CGUCGUUACC
s2:  C UCGUUACA
s3:  CGU GUU CC
s4:  CG CGUU CG
```

Figure 2.2: A multiple alignment.

the sequences are arranged in k rows of l columns each.

Example 2.2. Let $s_1 = CGUCGUUACC$, $s_2 = CUCGUUACA$, $S_3 = CGUGUUCC$ and $s_4 = CGCGUUCG$. A multiple alignment is given in Figure 2.2.

The value of the multiple alignment $\mathcal{A}$ is defined as

$$\sum_{i=1}^{l} \mu(s'_1(i), s'_2(i), \ldots s'_k(i)),$$

where $s'_l(i)$ denotes the i-th letter in the resulting sequence s'_l, and $\mu(s'_1(i), s'_2(i), \ldots s'_k(i))$ denotes the score of the i-th column. The multiple sequence alignment problem is to construct a multiple alignment minimizing its value. There are many ways to define $\mu(s'_1(i), s'_2(i), \ldots s'_k(i))$. Two popular ones are listed below.

1. *SP-score* (Sum-of-the-Pairs): The score of each column is defined as:

$$\mu(s'_1(i), s'_2(i), \ldots s'_k(i)) = \sum_{1 \le j < l \le k} \mu(s'_j(i), s'_l(i)),$$

   where $\mu(s'_j(i), s'_l(i))$ is the score of the two opposing letters $s'_j(i)$ and $s'_l(i)$. The SP-score has previously been studied extensively. See, *e.g.*, [1, 4, 9, 31, 62].

2. *Tree score*: In order to define the score $\mu(s'_1(i), s'_2(i), \ldots s'_k(i))$ of the i-th column, an *evolutionary* (or *phylogenetic*) tree $T = (V, E)$ with k leaves is assumed, each leaf j corresponding to a sequence s_j. (Here V and E denote the sets of nodes and edges in T, respectively.) Let $k+1, k+2, \ldots, k+m$ be the internal nodes of T. For each internal

node j, reconstruct a letter (possibly a space) $s'_j(i)$ such that $\sum_{(p,q)\in E}\mu(s'_p(i), s'_q(i))$ is minimized. The score $\mu(s'_1(i), s'_2(i), \ldots s'_k(i))$ of the i-th column is thus defined as

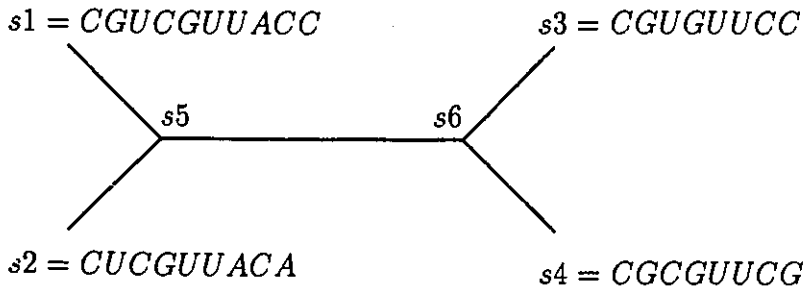
$$\mu(s'_1(i), s'_2(i), \ldots s'_k(i)) = \sum_{(p,q)\in E} \mu(s'_p(i), s'_q(i)).$$

This measure has been discussed in [1, 5, 63, 65, 66]. Multiple sequence alignment with tree score is often referred to as *tree alignment* in the literature.

Note that, a tree alignment induces a set of *reconstructed* sequences, each corresponding to an internal node. Thus, it is convenient to reformulate tree alignment as follows: Given a set X of k sequences and an evolutionary tree T with k leaves, where each leaf is associated with a given sequence, reconstruct a sequence for each internal node to minimize the *cost* of T. Here, the cost of T is the sum of the edit distance of each pair of (given or reconstructed) sequences associated with an edge. Observe that, once a sequence for each internal node has been reconstructed, a multiple alignment can be obtained by optimally aligning the pair of sequences associated with each edge of the tree. Moreover, the tree score of this induced multiple alignment equals the cost of T. In this sense, the two formulations of tree alignment are equivalent. From now on, we will always use the second formulation.

Example 2.3. Consider the alignment given in Example 2.2. Suppose that the score of a pair of letters is 0 for a match and 1 for a mismatch. The SP-scores of the ten columns are 0, 3, 3, 3, 0, 0, 0, 4, 0, and 5, respectively. Thus, the SP-score of the multiple alignment is 18. This is actually an optimal alignment with respect to SP-score.

Example 2.4. Let us consider tree alignment. Again, we assume that the score of a pair of letters is 0 for a match and 1 for a mismatch. Suppose that an evolutionary tree T with four leaves is given as in Figure 2.3 (a). The set of given sequences and the score scheme are the same as in Example 2.2. The reconstructed sequences are $s_5 = CUCGUUACC$ and



(a)

```
s1:CGUCGUUACC  s2:CUCGUUACA  s5:CUCGUUACC  s3:CGUGUUCC  s4:CGCGUUCG
s5:C UCGUUACC  s5:CUCGUUACC  s6:CGCGUU CC  s6:CGCGUUCC  s6:CGCGUUCC
```

(b)

```
                    s1:   CGUCGUUACC
                    s2:   C UCGUUACA
                    s3:   C GUGUU CC
                    s4:   C GCGUU CG
reconstructed       s5:   C UCGUUACC
  sequences         s6:   C GCGUU CC
```

(c)

Figure 2.3: (a) The given evolutionary tree T. (b) The pairwise alignments. (c) The optimal tree alignment.

$s_6 = CGCGUUCC$, which minimize the cost of T. The pairwise alignments corresponding to each edge of T are illustrated in Figure 2.3 (b). Note that, in order to construct a multiple alignment consistent with all pairwise alignments, we have to introduce some columns of spaces in some pairwise alignments. The multiple alignment induced from the pairwise alignments is illustrated in Figure 2.3 (c). The tree score of this alignment is 6, which is the same as the cost of T. The optimal tree alignment obtained here is different from the optimal SP-score alignment in Example 2.3.

In practice, we often face a more difficult problem called *generalized tree alignment.* Suppose we are given a set of sequences. The problem is to construct an evolutionary tree as well as a set of sequences (called reconstructed sequences) such that each leaf of the evolutionary tree is assigned a given sequence, each internal node of the tree is assigned a reconstructed sequence, and the cost of the tree is minimized over all possible evolutionary trees and reconstructed sequences.

2.2 Comparison of Trees

A single-strand RNA molecule folds to form base pairs (A and U form a pair, and G and C form a pair). Those bases of RNA which cannot be paired form loops. The *secondary structure* of an RNA describes its folding conformation. (The *primary* structure of an RNA is its nucleotide sequence.) Figure 2.4 illustrates such a structure. In general, we can decompose an RNA secondary structure into components of five types: stem (S), hairpin loop (H), bulge loop (B), interior loop (I), and multi-branch loop (M). The secondary structure can be conveniently expressed as a tree in which each node is labeled by a letter S, H, B, I, or M, and the left to right order among siblings is significant [55, 71, 72, 86]. For example, the RNA secondary structure in Figure 2.4(a) can be represented as an *ordered labeled* tree as shown in Figure 2.4(b). The comparison of RNA secondary structure trees can help identify conserved structural motifs in an RNA folding process [54] and construct taxonomy trees [72]. On the other hand, the comparison of *unordered trees* has applications to the morphological problems arising in genetics and other fields [73, 74, 77].

Similar to sequences, we can edit one tree into another. Again, the operations used are insertion, deletion and replacement [76, 86]. Let T be an ordered (or unordered) labeled tree. Inserting a node u into T means that for some node v in T, we make u the parent

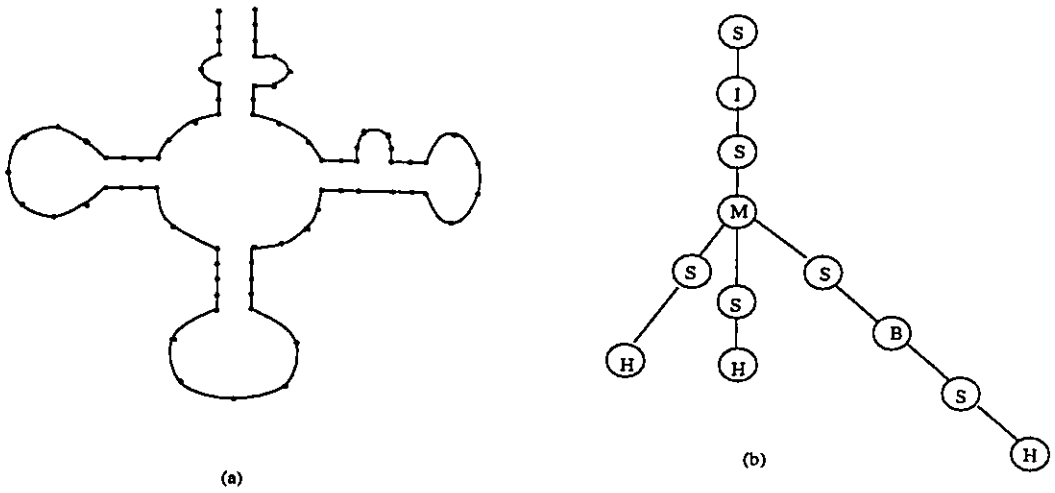


Figure 2.4: An RNA secondary structure and its tree representation.

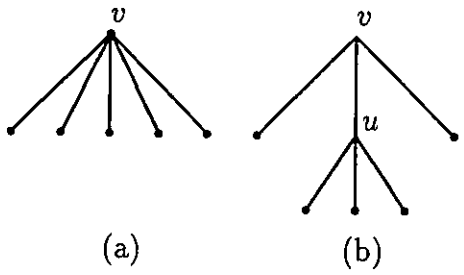


Figure 2.5: (a) The original tree. (b) The tree after insertion.

of a consecutive subsequence (or a subset, respectively) of the children of v and then v the parent of u. Figure 2.5 gives an example. A deletion is just the opposite of an insertion. The replacement operation replaces the label of a node with another label. Again, the edit distance between two trees is the value of an optimal editing sequence transforming one into the other.

Similar to sequence alignment, we can define the alignment of two labeled trees. Let T_1 and T_2 be two labeled trees. An alignment $\mathcal{A}$ of T_1 and T_2 is obtained by first inserting nodes labeled with *spaces* into T_1 and T_2 such that the two resulting trees T_1' and T_2' have the same structure, *i.e.*, they are identical if the labels are ignored, and then *overlaying* T_1'

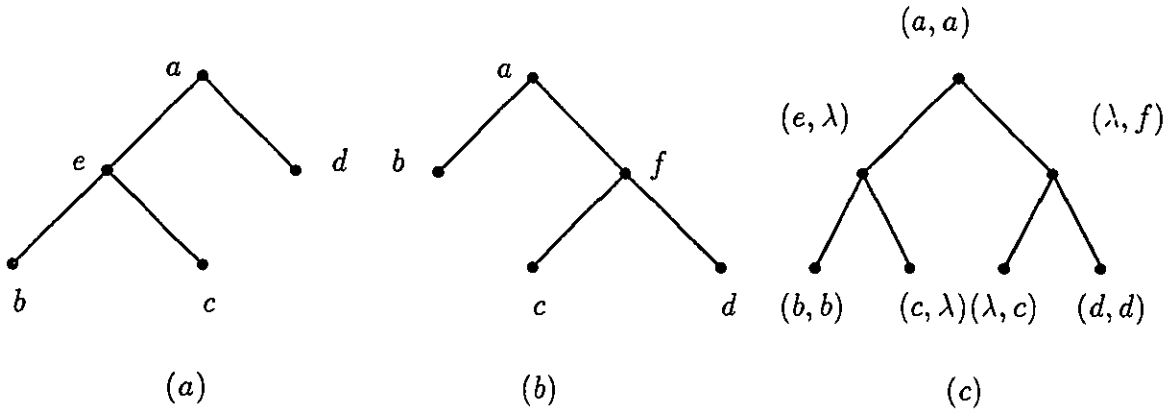


Figure 2.6: (a) Tree T_1. (b) Tree T_2. (c) The optimal alignment of T_1 and T_2.

on T_2'. An example alignment is shown in Figure 2.6. The *alignment distance* between T_1 and T_2 is the value of an optimal alignment of T_1 and T_2. Unlike the case of sequences, we will see that alignment distance and edit distance for trees are actually very different in Chapter 5.

Trees can also be used to describe the relationships among objects. In the analysis of molecular evolution, the evolutionary history of a set of species is described by an *evolutionary tree*. Let S be a set of species. An evolutionary tree T on S is a (rooted) [1] *unordered* tree such that the leaves of T are uniquely labeled with the elements in S. The internal nodes are unlabeled and the order among siblings is *insignificant*. Usually we require that each internal node has at least two children. The root can be viewed as the common ancestor of the set of species. Figure 2.7 gives an evolutionary tree containing four species. There are many ways to construct evolutionary trees. Different methods may obtain different evolutionary trees for the same set of species, and there is thus the need to compare different evolutionary trees defined on the same set of species. Many approaches for comparing evolutionary trees have been proposed We will discuss some of them in Chapter 6.

[1] Note that, evolutionary trees are also often viewed as *unrooted* trees in the literature, especially when the position of the common ancestor is unknown.

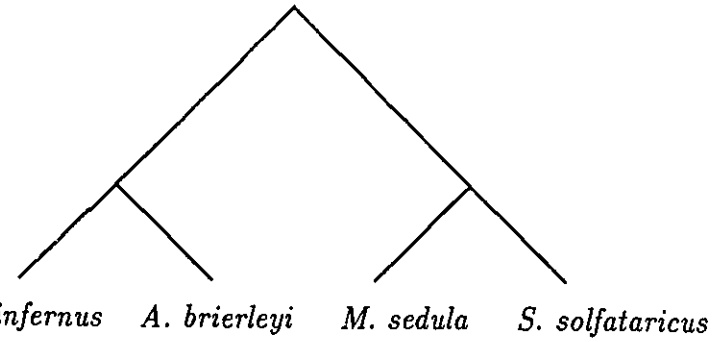


Figure 2.7: The evolutionary tree for the sulfolobales group of archaea obtained by the parsimony method.

2.3 Steiner Trees

The tree alignment and the generalized tree alignment problems discussed in Section 2.1 can be viewed as variants of the well-known Steiner tree problem. Let V be a set of points in some space, S a subset of V and d a real function $V \times V \longrightarrow R$ specifying the distance between every pair of points. A *Steiner tree* for S is any free tree T, *i.e.*, an acyclic graph, whose vertex set contains S as a subset. The cost on an edge is the distance between the two end points of the edge. The *cost* of T is the sum of the costs on all edges in T. The points in S are called *regular* points and the vertices of T which are not in S are called the *Steiner* points. The Steiner tree problem is defined as the problem of computing a *Steiner minimal tree* for S, *i.e.*, a Steiner tree for S with the minimum cost.

There are many variants of the Steiner tree problem [42]. It was first studied for the Euclidean plane, where the distance between two points a and b in the Euclidean plane is defined as $\sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$, where (x_1, y_a) and (x_b, y_b) are the coordinates of a and b in the plane, respectively. Another well-known variant is the *rectilinear Steiner tree* problem, where the distance between points a and b is $|x_a - x_b| + |y_a - y_b|$. The Euclidean Steiner tree problem and the rectilinear Steiner tree problem have been studied extensively

due to their applications in practice such as network design, location of facilities, component placement on circuit board, etc [42]. The generalized tree alignment problem defined in Section 2.1 is in fact a Steiner tree problem in the space of sequences, where each point is a sequence, and the distance between two sequences is simply their edit distance.

There is a related problem called the *Steiner tree problem under a given topology*, where the Steiner tree topology is given, and the problem is to locate all the Steiner points of the given topology such that the cost of the tree is minimized [42]. The tree alignment problem can be viewed as the Steiner tree problem under a given topology in the space of sequences.

2.4 Approximation Algorithms and MAX SNP-hardness

Since this thesis contains several approximation algorithms and some non-approximability results, we give a brief review of the concepts concerning approximation algorithms and the recently developed MAX SNP-hardness theory. The MAX SNP-hardness theory is closely related to the non-approximability results.

If an optimization problem is NP-hard, then it is unlikely to have a polynomial time algorithm that yields the optimal solutions of this problem. Thus, approximation algorithms are of interest for NP-hard problems. Usually, we use the *performance ratio* to measure the "goodness" of an approximation algorithm. Suppose that A is an approximation algorithm for some minimization problem. For any instance, I, of the problem, let $opt(I)$ be the size of an optimal solution and $A(I)$ the size of the solution obtained by A on I. The performance ratio of algorithm A is defined as $\max_I \frac{A(I)}{OPT(I)}$. [2] An *approximation scheme* for a minimization problem is an algorithm A, which takes as part of its input an error

[2] It is defined as $\min_I \frac{OPT(I)}{A(I)}$ for maximization problems.

bound ϵ and has the performance ratio $1+\epsilon$. Such an algorithm can also be viewed as a family of algorithms $\{A_\epsilon | \ \epsilon > 0\}$, where each algorithm A_ϵ is an approximation algorithm with ratio $1+\epsilon$. A *polynomial-time approximation scheme* (PTAS) is an approximation scheme $\{A_\epsilon | \ \epsilon > 0\}$ where algorithm A_ϵ runs in time polynomial in the size of the instance I for each fixed ϵ. For more details, see [28].

In the development of approximation algorithms for NP-hard optimization problems, it has been found that some simple algorithms can achieve constant approximation ratio for many problems such as Vertex Cover, Traveling Salesman, Maximum Cut, etc. [60]. However, it seems extremely hard to improve these ratios, or, even more ambitiously, to devise a PTAS for these problems. To further study these problems, Papadimitriou and Yannakakis defined a special reduction, called *linear reduction*, that preserves certain approximability properties [60]. Based on Fagin's syntactic definition [18] of the class NP, they introduced a class of natural optimization problems, MAX SNP, which includes Independent Set, Vertex Cover, Dominating Set, Maximum Cut, etc. It is known that every problem in this class can be approximated within some constant ratio, and moreover, they showed that a PTAS for any problem that is MAX SNP-hard under linear reduction would imply one for every other problem in the class.

Now, we give the definition of linear reduction introduced by Papadimitriou and Yannakakis [60], used to show the MAX SNP-hardness of a problem. Suppose that Π and Π' are two minimization problems (the definition for maximization problem is analogous). We say that Π *linearly reduces* (or L-reduces) to Π' if there are polynomial-time algorithms f and g and constants $\alpha, \beta > 0$ such that, for any instance I of Π,

1. $OPT(f(I))) \leq \alpha \cdot OPT(I)$.

2. Given any solution of $f(I)$ with cost c', algorithm g produces in polynomial time a

solution of I with cost c satisfying $|c - OPT(I)| \leq \beta|c' - OPT(f(I))|$.

It follows from the above definition that (i) the composition of two L-reductions is an L-reduction and (ii) if problem Π L-reduces to problem Π' and Π' can be approximated in polynomial time within a factor of $1+\epsilon$, then Π can be approximated within factor $1+\alpha\beta\epsilon$. In particular, if Π' has a PTAS, so does Π. A problem Π is *MAX SNP-hard* if for any problem Π' in MAX SNP there is an L-reduction from Π' to Π. In other words, if a MAX SNP-hard problem has a PTAS, so does every problem in MAX SNP. Several problems have been proved to be MAX SNP-hard in [60].

The definition of MAX SNP-hardness allows us to study the non-approximability of many optimization problems in a unified way. It has been shown that it is impossible for an MAX SNP-hard problem to have a PTAS unless NP=P [3].

Theorem 2.1 *A MAX SNP-hard problem does not have a PTAS unless NP=P.*

Chapter 3

The Complexity of Multiple Sequence Alignment

3.1 Introduction

Multiple sequence alignment is one of the most important and challenging problems in computational biology [50, 53]. Many papers have been written on effective and efficient methods for constructing multiple sequence alignment. For a comprehensive survey, see [10, 85].

Many methods have been suggested to measure the quality of a multiple alignment. Among them, *SP-score* seems to be very sensible and has received a lot of attention [4, 9, 69]. The best algorithm to compute an optimal alignment under SP-score measure is based on dynamic programming and requires a running time which is in the order of the product of the lengths of input strings [1]. Gusfield first proposed a polynomial-time approximation algorithm for this problem that achieves ratio $2 - \frac{2}{k}$ on k input sequences [31]. Pevzner

improved Gusfield's algorithm to obtain a ratio of $2 - \frac{3}{k}$ [62]. Recently Bafna, Lawler and Pevzner pushed the ratio to $2 - \frac{l}{k}$ [5] for any fixed l. However, it was not known if multiple alignment with SP-score is NP-complete [62]. Here we show that this problem (actually, the decision version of it) is NP-complete.

Tree score is another important measure [1, 33, 62, 63, 65, 67]. Many algorithms have been proposed for tree alignment [1, 33, 67], which all run in exponential time in the worst case. Again it was not known if this problem is NP-hard. Among the many possible structures, binary tree and star are the most common ones [1, 67]. We will prove that tree alignment is NP-complete even when the tree is binary. Furthermore, we show that the problem is MAX SNP-hard if the evolutionary tree is a star. That is, a PTAS does not exist in this case unless P=NP. In contrast, when the given evolutionary tree is of bounded degree, tree alignment is *not* MAX SNP-hard, for a PTAS exists in this case. The PTAS will be given in Chapter 4.

A more challenging problem is the generalized tree alignment problem [31]. Foulds and Graham proved that a variation, where the distance between two sequences is defined as Hamming distance, is NP-complete [24]. Recently, Sweedyk and Warnow proved that generalized tree alignment is NP-complete [75]. Several approximate methods have been proposed in the literature [32, 33, 65, 67]. Gusfield showed that a minimum-cost spanning tree of the input sequences has a cost that is at most twice the optimum An interesting question is whether one can find efficient algorithms with performance ratio better than 2. It is easy to see that the recent results of Zelikovsky [89], and Berman and Ramaiyer [6] on the approximation of Steiner minimal trees imply that generalized tree alignment can be approximated within a factor of 1.747 in polynomial time [45]. But can we make the approximation ratio arbitrarily close to 1? In Section 3.4, we will answer this negatively

by showing that generalized tree alignment is MAX SNP-hard. Thus, our result implies that the problem does not have a PTAS. In other words, the approximation ratio cannot be made arbitrarily close to 1.

3.2 The NP-completeness of Multiple Sequence Alignment with SP-score

In this section, we prove that the following decision version of multiple sequence alignment with SP-score is NP-complete.

INSTANCE: Set of sequences $S = \{s_1, s_2, \ldots, s_k\}$, and positive integer c.

QUESTION: Is there a multiple alignment of S with SP-score c or less?

The reduction is from the *shortest common supersequence* problem

INSTANCE: Finite set S of sequences over alphabet Σ and positive integer m.

QUESTION: Is there a sequence s with $|s| \leq m$ such that each $t = t_1 t_2 \cdots t_r \in S$ is a subsequence of s, *i.e.*, $s = s_0 t_1 s_1 t_2 s_2 \cdots t_r s_r$, for some sequences $s_0, s_1, \ldots, s_r$?

The problem is NP-complete even if $|\Sigma| = 2$ [58].

Theorem 3.1 *Multiple sequence alignment with SP-score is NP-complete.*

Proof. Obviously, multiple sequence alignment is in NP. We reduce the shortest common supersequence problem to multiple alignment with SP-score. Given a set S of sequences over alphabet $\{0, 1\}$, and a positive integer m, we construct a collection of sets $X = \{X_{i,j} | i, j \geq 0, i + j = m\}$, where $X_{i,j} = S \cup \{a^i,\ b^j\}$ and a and b are two new letters. Here we can

Table 3.1: Score scheme I.

S	0	1	a	b	Δ
0	2	2	1	2	1
1	2	2	2	1	1
a	1	2	0	2	1
b	2	1	2	0	1
Δ	1	1	1	1	0

assume that each sequence in S has length at most m. The score scheme is shown in Table 1. Clearly the score scheme satisfies triangle inequality. The positive integer c is defined as $c = (k-1)||S|| + (2k+1)m$, where $||S||$ is the total length of all sequences in S.

To show that multiple alignment with SP-score is NP-hard, it is sufficient to show that: S has a supersequence s of length m if and only if some $X_{i,j}$ has an alignment with value at most c.

(if) Suppose that we have an alignment $\mathcal{A}$ of the $k+2$ sequences in $X_{i,j}$ with value at most c, for some i, j. Consider the induced alignment of the k sequences in S. No matter what the alignment is, its score is always $(k-1)||S||$. Thus, in $\mathcal{A}$, the total contribution of the pairwise alignments involving sequences a^i and/or b^j, is at most $(2k+1)m$. Therefore, every 0 must be aligned with an a and every 1 must aligned with a b in $\mathcal{A}$. We can obtain a supersequence s for S by assigning 0 to the columns containing a's and 1 to the other columns. The length of s is $i + j = m$.

(only if) Let s be a supersequence for S with length m. Let i be the number of 0's in s and j the number of 1's in s. Consider set $X_{i,j}$. For each sequence $t \in S$, there exists an alignment of t and s such that each 0 (or 1) in X_i matches a 0 (or 1, respectively) in s. Some 0's and 1's in s may correspond to spaces. To obtain the desired multiple alignment, we align each t in S with s as above and then align the a's in the sequence a^i with the 0's

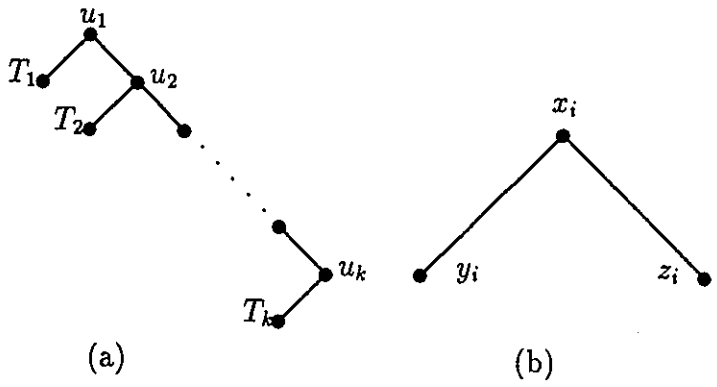


Figure 3.1: (a) The tree T. (b) The subtree T_i.

in s and the b's in b^j with the 1's in s. Obviously, in this alignment, the letters in a column are either $0, a, \Delta$, or $1, b, \Delta$. The value of the alignment (with sequence s removed) is c.

Therefore, by checking the value of an optimal alignment of $X_{i,j}$, $i + j = m$, we can answer if there is a supersequence s for X with length m in polynomial time. ■

3.3 The Complexity of Tree Alignment

In this section, we study the complexity of tree alignment. Recall that the tree alignment problem is that given a set X of k sequences and an evolutionary tree T with k leaves, where each leaf is associated with a given sequence, reconstruct a sequence for each internal node to minimize the *cost* of T. Two important evolutionary structures are considered: binary trees and stars.

Theorem 3.2 *Tree alignment is NP-complete even when the given evolutionary tree is binary.*

Table 3.2: Score scheme III.

	0	1	a	b	Δ
0	0	1	1	1	1
1	1	0	1	1	1
a	1	1	0	2	2
b	1	1	2	0	2
Δ	1	1	2	2	0

Proof. The reduction is again from the shortest common supersequence problem. Let $S = \{s_1, s_2, \ldots s_k\}$ be a set of sequences on $\{0, 1\}$. Now, we construct a binary tree T as in Figure 3.1(a), where each T_i is a subtree shown in Figure 3.1(b). Every leaf in T is associated with a sequence over $\Sigma = \{0, 1, a, b\}$. We define $y_i = s_i$ for $i = 1, 2, \ldots, k$, $z_i = a^{i-1}b^{m-i+1}$, for $i = 1, 2, \ldots, k-1$, and $z_k = a^m$.

The score scheme is defined in Table 3, which again satisfies the triangle inequality.

Let $M = \sum_{i=1}^{k} 2m - |s_i|$. We will show that there is a common supersequence s with $|s| = m$ if and only if there is a tree alignment with cost M.

(*only if*) Assume that there is a supersequence s with $|s| = m$. To obtain the desired tree alignment, we assign sequence s to every x_i and u_i, $i = 1, \ldots, k$.

(*if*) Suppose that there exists an assignment of sequences to the internal nodes of T such that the cost of the resulting tree alignment is M. For each i, let t_i be the sequence assigned to x_i. For any node v in T, let $T(v)$ denote the subtree rooted at v. First observe that, because of the triangle inequality, for each i the optimal cost of $T(x_i)$ is at least the edit distance between the sequences at the nodes y_i and z_i, which is $2m - |s_i|$. Hence, the cost of $T(x_i)$ in this tree is exactly $2m - |s_i|$. Since all the edges between (x_i, u_i) and (u_i, u_{i+1}) must cost 0, $t_1 = \ldots = t_k$. The sequences b^m and a^m assigned to z_1 and z_k force t_i not to contain any a or b. Hence, in order for $T(x_i)$ to achieve score $2m - |s_i|$, the sequence

t_i must be a supersequence of s_i and $|t_i| = m$. Therefore, we have a common supersequence for S with length m. ■

Note that, most definitions of evolutionary trees require that the nodes in the tree be labeled with distinct sequences. In this case, Theorem 3.2 still holds. We can modify the above proof by identifying nodes x_k and u_k in the tree, and adding a suffix w_i to each y_i and z_i, where $w_k = (1)^k(01)^k$ and $w_i = (1)^k(11)^{k-i-1}10(01)^i$ for any $1 \leq i < k$. Therefore, in the optimal alignment, each $x_i = sw_i$, and each $u_i = s(1)^k(11)^{k-i}(01)^i$ for $1 \leq i < k$ and $x_k = sw_k$.

To prove the MAX SNP-hardness of tree alignment when the given evolutionary tree is a star, we begin with the Max Cut-B problem.

Max Cut-B: Given a graph $G = (V, E)$ with degree bounded by B, find a partition of V which divides V into disjoint sets V_0 and V_1 such that the number of edges that go from V_0 to V_1 is the largest.

Max Cut-B is shown to be MAX SNP-complete in [60]. Now, we can prove our next result.

Theorem 3.3 *It is MAX SNP-hard to construct an optimal tree alignment when the given evolutionary tree is a star.*

Proof. The reduction is from Max Cut-B. Let $G = (V, E)$ be a graph with degree bounded by constant k, where $E = \{v_1, v_2, \ldots, v_n\}$. Define $\Sigma = \{0, 1, a, b, \#, *, \$\}$. The letters $\#, \$$ will serve as delimiters and $*$ will be a kind of "wild card". For each $v_i \in V$, we construct a sequence $s_i = z_{i,1}z_{i,2}\cdots z_{i,n}\$$, where

$$z_{i,j} = \begin{cases} D'_j 0 *^k D_j 1 *^k & \text{if } j \neq i,\ v_i \text{ and } v_j \text{ are adjacent} \\ D'_j *^{k+1} D_j *^{k+1} & \text{if } j \neq i,\ v_i \text{ and } v_j \text{ are not adjacent} \\ D'_j 1^{k+1} D_j 0^{k+1} & \text{if } j = i, \end{cases}$$

where $D'_1 = \$$, $D_1 = \#$, and $D'_i = D_i = \#$ for $i = 2, \ldots, n$.

Observe that, in general s_i has the form

$$(\$x^{k+1}\#y^{k+1})(\#x^{k+1}\#y^{k+1})\cdots(\#x^{k+1}\#y^{k+1})(\#x^{k+1}\#y^{k+1}\$).$$

It contains n blocks of x^{k+1} and n blocks y^{k+1} in the sequence, where the i-th x^{k+1} is 0^{k+1}, i-th y^{k+1} is 1^{k+1}, and the rest of x^{k+1}'s and y^{k+1}'s are either $*^{k+1}$, $0*^k$, or $1*^k$. [1] Similarly, let $t_i = u_{i,1}u_{i,2}\cdots u_{i,n}\$$, where

$$u_{i,j} = \begin{cases} D'_j *^{k+1} D_j *^{k+1} & \text{if } j \neq i \\ D'_j 1^{k+1} D_j 0^{k+1} & \text{if } j = i. \end{cases}$$

Finally, define

$$\begin{aligned} X_0 &= \{s_i | i = 1, 2, \ldots, n\}, \\ X_1 &= \{a^i(\# *^{k+1} \#)^n | i = 0, 1, \ldots, 5n(k+1)\}, \\ X_2 &= \{a^i(\#b^{k+1}\#)^n | i = 1, 2, \ldots, 3k\}, \end{aligned}$$

[1] The following construction forces the internal sequence to be of the form

$$*^{k+1}(\#x^{k+1}\#*^{k+1})(\#x^{k+1}\#*^{k+1})\cdots(\#x^{k+1}\#*^{k+1}),$$

where there are n blocks of x^{k+1} in the sequence, each of them is either 0^{k+1} or 1^{k+1}. Since the degree of each node in G is bounded by k, the segment $z_{i,i}$ in each s_i dominates the alignment of s_i and the internal sequence. The optimal score for the alignment of s_i and the internal sequence is the number of edges (with v_i as an end) not being cut.

Table 3.3: score scheme

	#	$	0	1	*	Δ	a	b
#	0	$2k$	$4k$	$4k$	$4k$	$4k$	$4k$	$4k$
$	$2k$	0	$4k$	$4k$	$4k$	$4k$	$4k$	$4k$
0	$4k$	$4k$	0	1	0	1	1	1
1	$4k$	$4k$	1	0	0	1	1	1
*	$4k$	$4k$	0	0	0	0	0	2
Δ	$4k$	$4k$	1	1	0	0	0	2
a	$4k$	$4k$	1	1	0	0	0	2
b	$4k$	$4k$	1	1	2	2	2	2

and

$$X_3 = \{a^i t_j | i = 1, 2, \ldots, k,\ j = 1, 2, \ldots, n\}.$$

The set of given sequences X is as follows:

$$X = X_0 \cup X_1 \cup X_2 \cup X_3.$$

The score scheme is given in Table 4. Note that the scores do not satisfy the triangle inequality. The evolutionary tree is a star (*i.e.*, a tree with only one internal node) with $|X|$ leaves, each is associated with a sequence in X.

First, we show that the internal sequence in an optimal tree alignment for X should be in the form

$$*^{k+1}(\#x^{k+1}\#*^{k+1})(\#x^{k+1}\#*^{k+1})\cdots(\#x^{k+1}\#*^{k+1}),$$

where there are n blocks of x^{k+1}, each is either 0^{k+1} or 1^{k+1}. This is due to the following reasons.

1. The sequences in $X_1 = \{a^i(\# *^{k+1} \#)^n | i = 0, 1, \ldots, 5n(k+1)\}$ force the internal sequence to contain exactly n #'s and no $. Otherwise, the $5n(k+1)$ sequences in X_1 contribute a cost of $5n(k+1) \cdot 2k = 10k(k+1)n$ or more. However, if the internal sequence

is in the form $(\#1^{k+1}\#*^{k+1})^n$, the total cost of the tree is less than $5k^2n+6kn < 10k(k+1)n$. (See the analysis below.)

2. The sequences in $X_2 = \{a^i(\#b^{k+1}\#)^n | i = 1, 2, \ldots, 3k\}$ force the internal sequence to contain none of $0, 1, b$ at positions between the $2i$-th # and $(2i+1)$-th #. Otherwise, the existence of such a letter would make the $3k$ sequences in X_2 contribute an extra cost of $3k$, while the contribution from the sequences in X_0 decreases by at most $k+1$ and the contribution from the sequences in X_3 decreases by at most k. Thus, we can always delete such letters without increasing the total cost.

3. The score scheme allows us to delete an a from the internal sequence without increasing the cost.

4. Since $\mu(b,b) = \mu(b,\Delta) = 2$ and $\mu(b,1) = \mu(b,0) = 1$, it is advantageous for the internal sequence to be of form

$$*^{k+1}(\#\{0,1\}^{k+1}\#*^{k+1})(\#\{0,1\}^{k+1}\#*^{k+1})\cdots(\#\{0,1\}^{k+1}\#*^{k+1}),$$

where $\{0,1\}^{k+1}$ denotes any binary string of length $k+1$. The leading and trailing $*$'s are used to absorb the 0's and 1's in the beginning or end of an s_i or an t_i. (See Figure 3.2.)

5. The kn sequences in $X_3 = \{a^i t_j | i = 1, 2, \ldots, k,\ j = 1, 2, \ldots, n\}$ allow us to modify the internal sequence into the form

$$*^{k+1}(\#x^{k+1}\#*^{k+1})(\#x^{k+1}\#*^{k+1})\cdots(\#x^{k+1}\#*^{k+1}),$$

where there are n blocks of x^{k+1} in the sequence, each of them is either 0^{k+1} or 1^{k+1}.

Now, we prove condition (1) of L-reduction. Suppose that there is a partition (V_0, V_1) of V, which cuts c edges. The internal sequence can be constructed as

$$*^{k+1}(\#x^{k+1}\#*^{k+1})(\#x^{k+1}\#*^{k+1})\cdots(\#x^{k+1}\#*^{k+1}),$$

where there are n blocks of x^{k+1}, the i-th block is 1^{k+1} if v_i is in V_1, and 0^{k+1} otherwise. In this case, the sequences in X_1 contributes no cost. Each sequence in X_2 contributes a cost of $(k+1)n$ and thus X_2 totally contributes $3k(k+1)n$. Each sequence in X_3 contributes a cost of $2k$ and totally X_3 contributes $2k^2n$.

Let $c(v)$ denote the number of edges incident upon v that are cut by the partition. For each $v_i \in V$, s_i contributes $2k + d(v_i) - c(v_i)$. This can be observed as follows. Since there are $2n$ #'s in the internal sequence and $2n-1$ #'s and 2 \$'s in each s_i, the delimiters in s_i always contribute a cost of $6k$. For each i, if the i-th block of x^{k+1} of the internal sequence is 1^{k+1} (*i.e.*, $v_i \in V_1$), we align s_i with the internal sequence as in Figure 3.2(a), *i.e.*, the right end delimiter of the s_i is matched with a space, and if the i-th block of x^{k+1} of the internal sequence is 0^{k+1} (*i.e.*, $v_i \in V_0$), we align s_i with the internal sequence as in Figure 3.2(b). If $v_i \in V_1$, then for each v_j adjacent to v_i, the segment $z_{j,i}$ of s_j, which is of the form $D_i'0*^k D_i 1*^k$, will contribute 1 towards the cost if and only if $v_j \in V_1$. (See Figure 3.2(a).) Similarly, if $v_i \in V_0$, then for each v_j adjacent to v_i, the segment $z_{j,i}$ of s_j will contribute 1 towards the cost if and only if $v_j \in V_0$. (See Figure 3.2(b).) That is, all the edges that are not cut by the partition are counted here.

Therefore, the total cost of the tree is

$$3k(k+1)n + 2k^2n + \sum_{i=1}^{n} 2k + d(v_i) - c(v_i) = 5k^2n + 5kn + 2|E| - 2c.$$

Recall that the optimal c is at least $|E|/2$. Since the degree of G is bounded, condition (1) of L-reduction holds.

By the same argument, it is not hard to show that, given a tree alignment for X with cost $c' = 5k^2n + 5kn + 2|E| - 2c$, we can easily construct a partition of G which cuts c edges, by looking at the 0/1 assignment to the x-blocks in the internal sequence. Thus, condition (2) of L-reduction also holds (with $\beta = 1/2$). ■

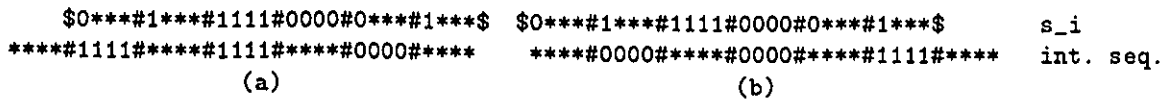


Figure 3.2: (a) v_i is in V_1. (b) v_i is in V_0.

3.4 The MAX SNP-hardness of Generalized Tree Alignment

In this section, we show that constructing an optimal generalized tree alignment is MAX SNP-hard. This implies that there is no PTAS for the problem, unless P=NP, by the result of [3]. First, let us give a formal definition of generalized tree alignment. Suppose we are given a set X of k sequences. Let Y be a set of hypothetical sequences, where $Y \cap X = \emptyset$. A *loaded tree* $T_{X,Y}$ for X is a *weighted* tree of $|X \cup Y|$ nodes, where each leaf is uniquely labeled with a sequence in X and each internal node is labeled with a sequence in $X \cup Y$ [30, 31]. The *cost* of an edge is the edit distance between the two sequences associated with the ends of the edge. The cost $c(T_{X,Y})$ of the tree $T_{X,Y}$ is the total cost of all edges in $T_{X,Y}$. Given sequences X, the problem is to find a set of sequences Y as well as a loaded tree $T_{X,Y}$ for X which minimizes $c(T_{X,Y})$ over possible sets Y and trees $T_{X,Y}$. In most cases, one requires that the given sequences be assigned to the leaves in the tree [19].

In order to prove the MAX SNP-hardness of tree alignment, we first prove a sequence of auxiliary MAX SNP-hardness results. We begin with the Vertex Cover-B problem, which is proved to be MAX SNP-complete in [60].

Vertex Cover-B: Given a graph $G = (V, E)$ with degree bounded by B, find the smallest vertex cover, *i.e.*, a smallest subset $V' \subseteq V$ such that, for each edge $(u, v) \in E$, at least one of u and v belongs to V'.

We then L-reduce Vertex Cover-B to the following more restricted version of itself:

Triangle-free Vertex Cover-B: Given a triangle-free graph $G = (V, E)$ with degree bounded by B, find the smallest vertex cover.

Now we L-reduce Triangle-free Vertex Cover-B to a restricted version of generalized tree alignment:

Restricted Generalized Tree Alignment: Given two sets of sequences X and Y, find a subset $Y' \subseteq Y$ and a loaded tree $T_{X,Y'}$ with the smallest cost.

Finally this problem is L-reduced to the generalized tree alignment problem, stated again below:

Generalized Tree Alignment: Given a set of sequences X, find a set of sequences Y and a loaded tree $T_{X,Y}$ with the smallest cost.

Now, we describe the required reductions.

Lemma 3.4 *Triangle-free vertex cover-B is MAX SNP-hard.*

Proof. For each edge (v_i, v_j) in the given graph G, we insert two vertices $u_{i,j}$ and $u_{j,i}$ into the edge. This should remove all the triangles. Call this new graph G'. Clearly, G has a vertex cover of size c if and only if G' has a vertex cover of size $c + |E|$. This is an L-reduction because $|E| \leq B \cdot c$. ■

Lemma 3.5 *Restricted generalized tree alignment is MAX SNP-hard.*

Proof. Let $G = (V, E)$ be a triangle-free graph with degree bounded by B, where $V = \{1, 2, \ldots, n\}$. Without loss of generality, we also assume that G is connected. Let 0_i

Table 3.4: Score scheme II.

	0	1	Δ
0	0	1	2
1	1	0	2
Δ	2	2	0

denote the binary sequence of length n with a 0 at the i-th position and 1's at the rest, and $0_{i,j}$ denote the binary sequence of length n with 0's at the i-th and j-th positions and 1's at the rest. We construct sets $X = \{0_{i,j} | (i,j) \in E\}$ and $Y = \{1^n\} \cup \{0_i | i = 1, 2, \ldots, n\}$. The score scheme is defined in Table 2, which also satisfies the triangle inequality.

Seven types of edges may appear in a restricted loaded tree. Their costs are:

1. $c(1^n, 0_i) = 1$.
2. $c(1^n, 0_{i,j}) = 2$.
3. $c(0_i, 0_j) = 2\mu(1,0) = 2$.
4. $c(0_i, 0_{k,l}) = \mu(1,0) = 1$, if $i = k$ or $i = l$.
5. $c(0_i, 0_{k,l}) = 3\mu(1,0) = 3$, if $i \neq k$ and $i \neq l$.
6. $c(0_{i,j}, 0_{k,l}) = 2\mu(1,0) = 2$, if $\{i,j\} \cap \{k,l\} \neq \emptyset$.
7. $c(0_{i,j}, 0_{k,l}) = 4\mu(1,0) = 4$, if $\{i,j\} \cap \{k,l\} = \emptyset$.

Now, we want to show that the reduction is indeed an L-reduction. Suppose that G has a vertex cover U of size c. We can connect each sequence $0_{i,j} \in X$ to some 0_k, where $k = i$ or j, and $k \in U$, and then connect the sequences $\{0_i | i \in U\}$ to 1^n. Each connection costs 1. This gives us a restricted loaded tree with cost $|E| + c$. Since the degree of G is bounded by B, $|E| \leq B \cdot c$. So condition (1) of L-reduction holds.

To see that condition (2) of L-reduction also holds, we need the following claim.

Claim 3.6 *Given a restricted loaded tree with cost c', we can find a loaded tree with cost not greater than c' in polynomial time such that all the edges are of type (1) or (4).*

Proof. Each edge $(1^n, 0_{i,j})$ can be replaced by two edges $(1^n, 0_i)$ and $(0_i, 0_{i,j})$. An edge $(0_i, 0_j)$ can be replaced by two edges $(0_i, 1^n)$ and $(1^n, 0_j)$. An edge $(0_i, 0_{k,l})$ of type (5), where $i \neq k$ and $i \neq l$, can be replaced by the edges $(0_k, 0_{k,l})$ and $(0_k, 0_i)$ with the same cost 3. An edge $(0_{i,j}, 0_{k,l})$ of type (6), where $\{i,j\} \cap \{k,l\} = \{m\}$, can be replaced by the edges $(0_{i,j}, 0_m)$ and $(0_{k,l}, 0_m)$ with the same cost 2. An edge $(0_{i,j}, 0_{k,l})$ of type (7), where $\{i,j\} \cap \{k,l\} = \emptyset$, can be replaced by the edges $(0_{i,j}, 0_i)$, $(0_{k,l}, 0_k)$ $(1^n, 0_i)$ and $(1^n, 0_k)$ with the same cost 4. ■

Given a loaded tree with cost c', we can construct a new loaded tree with the same cost c' using edges of types (1) and (4) only. The number of sequences of form 0_i in the new tree is at most $c' - |E|$. This implies a vertex cover of G of size at most $c' - |E| + 1$. Therefore, setting $\beta = 1$ makes condition (2) hold. This completes the proof. ■

Theorem 3.7 *Generalized tree alignment is MAX SNP-hard.*

Proof. By Lemma 3.5, it suffices to show that given a loaded tree T for X with cost c, where X is the same as in the proof of Lemma 3.5, there is a polynomial-time algorithm to construct a restricted loaded tree for X and $Y = \{1^n\} \cup \{0_i | 1 \leq i \leq n\}$, with cost c or less. Observe that here X has the "triangle-free" property, *i.e.*, X does not simultaneously contain the sequences $0_{i,j}$, $0_{i,k}$, and $0_{j,k}$ for any i, j, k. We will give a method to modify the tree T so that every sequence not in X is of form 1^n or 0_i.

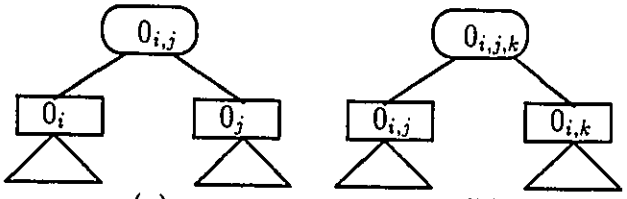


Figure 3.3: (a) Bad sequence of type (a). (b) Bad sequence of type (b).

A sequence that is not in X or of form 1^n or 0_i is a *bad sequence* and a node that is associated with a bad sequence is a *bad node*. Two sequences $0_{i,j}$ and $0_{k,l}$ are *adjacent* if $\{i,j\} \cap \{k,l\} \neq \emptyset$. Note that, as we have seen before, two adjacent sequences $0_{i,j}$ and $0_{i,k}$ can be connected through the sequence 0_i using two edges, each costing 1.

For convenience, we make (without loss of generality) a few assumptions about T. Here we view T as a rooted tree. We can assume that each edge in the tree T has cost 1. This is because we can delete any edge of cost 2 or more, find two adjacent sequences $0_{i,j}$ and $0_{i,k}$, one from each disconnected component, and reconnect them through 0_i. Since X is constructed from a connected graph G, such adjacent sequences always exist. Note that, this implies that all the sequences in the tree are of length n, because the score between Δ and other letters is 2. Moreover, we can assume that every bad node in T has two or more children. Otherwise, we can delete the bad node and reconnect the two disconnected components as above without increasing the cost. Lastly, we assume that each node in T is labeled by a unique sequence.

We will delete the bad nodes in T iteratively from the bottom to the top. Below we describe the steps involved in one iteration, which removes at least one bad node. Consider a bad node at the lowest level of the tree. The sequence, denoted s_1, associated with the node must be of form either $0_{i,j}$ (type (a)) or $0_{i,j,k}$ (type (b)), as shown in Figure 3.3. Note that, in case (b), s_1 must have exactly two children due to the triangle-free property of X.

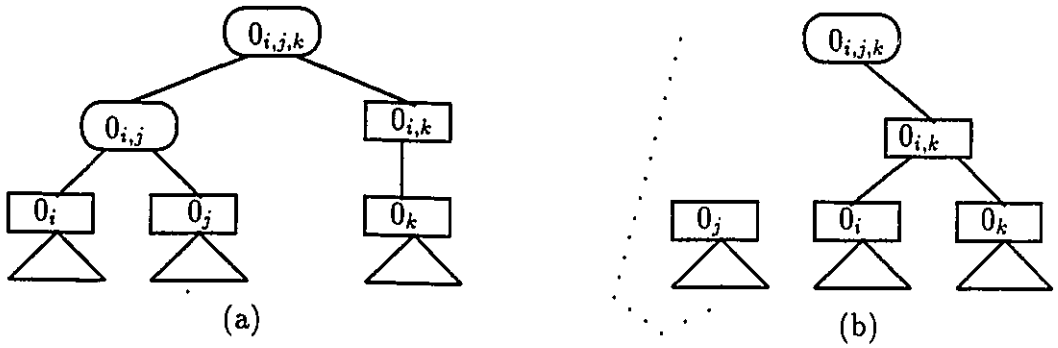


Figure 3.4: When $s_1 = 0_{i,j}$.

In the figure, an ellipse denotes a bad node, a rectangle denotes a *good* node, and a triangle denotes a subtree containing no bad nodes.

Suppose that s_1 is of type (a), say, $0_{i,j}$. Since s_1 has two children 0_i and 0_j, the parent of s_1 must have three 0's, say, $0_{i,j,k}$. Because each of 0_i and 0_j can appear at most once in T, a sibling, say, $0_{i,k}$, of s_1 can not be a bad node. See Figure 3.4(a). In fact, $0_{i,k} \in X$ because it has at most one child. Thus, we can delete s_1, move the subtree under 0_i to $0_{i,k}$ and relink the subtree under 0_j to the tree through some appropriate adjacent sequences (one from the subtree and one from the rest of the tree), as shown in Figure 3.4(b). This will not increase the cost.

Now suppose that s_1 is of type (b), say, $0_{i,j,k}$. Its parent, denoted s_2, contains either two 0's or four 0's. Suppose that s_2 contains two 0's, say, $s_2 = 0_{i,j}$. The assumptions made above force s_1 to have exactly two children $0_{i,k}$ and $0_{j,k}$. s_1 has a sibling of form 0_i or 0_j, then we can link $0_{i,k}$ (or $0_{j,k}$) to 0_i (or to 0_j, respectively) with cost 1, and thereby get rid of s_1. Thus, we assume that s_1 has a sibling s_3 with three 0's, say $0_{i,j,l}$, which is also a bad sequence of type (b). Similarly, s_3 must have two children $0_{i,l}$ and $0_{j,l}$. We can modify the two subtrees rooted at s_1 and s_3 to get rid of the bad nodes s_1 and s_3, as shown in Figure 3.5

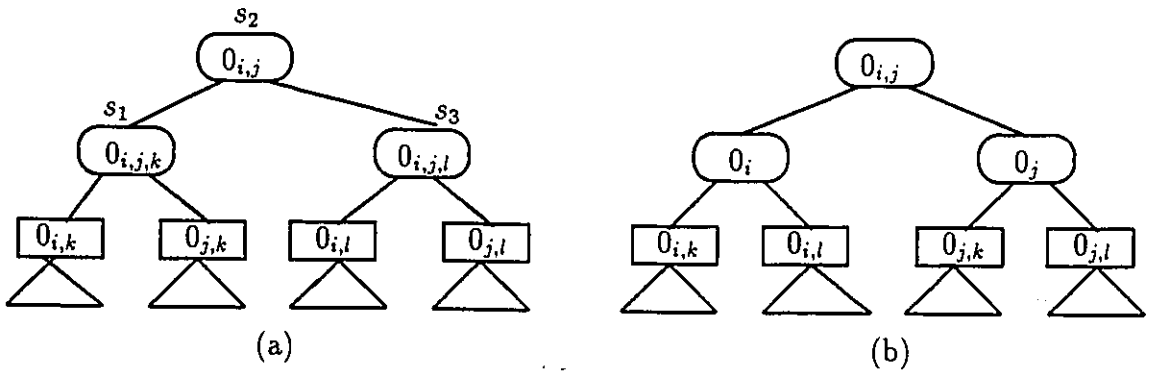


Figure 3.5: When $s_1 = 0_{i,j,k}$ and $s_2 = 0_{i,j}$.

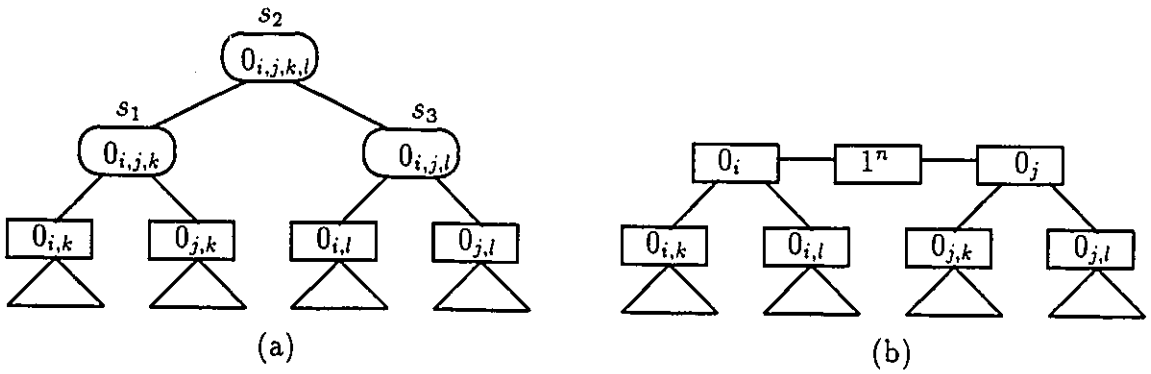


Figure 3.6: When $s_1 = 0_{i,j,k}$ and $s_2 = 0_{i,j,k,l}$.

Now suppose that s_2 contains four 0's, say, $s_2 = 0_{i,j,k,l}$. Without loss of generality, we assume that s_1 has a sibling $s_3 = 0_{i,j,l}$. We temporarily modify the subtrees under s_1 and s_2 as in Figure 3.6.

Let s_4 be a sibling of s_2. Since the parent of s_2 has either three 0's or five 0's, s_4 has either four 0's, say, $0_{i,j,k,m}$, or two 0's, say, $0_{k,l}$. We consider two cases.

Case 1: $s_4 = 0_{i,j,k,m}$. Similar to $s_2 = 0_{i,j,k,l}$, s_4 has four descendants of form $0_{p,q}$, where $p, q \in \{i, j, k, l\}$. One of the descendants has to involve i, *e.g.*, it is of form $0_{i,k}$. Thus, we can link 0_i to $0_{i,k}$ with cost 1. This reconnects the component in Figure 3.6(b) to the tree.

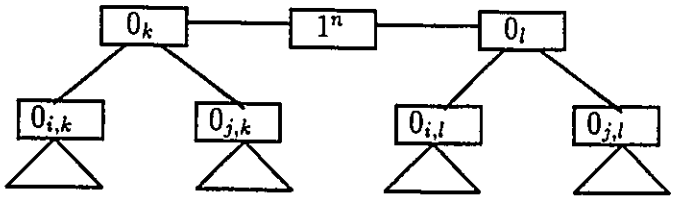


Figure 3.7: Rearranging the component.

Then we delete node s_2 and reduce the cost by 1.

Case 2: $s_4 = 0_{k,l}$. The component in Figure 3.6 can be reorganized as in Figure 3.7 without extra cost. Then we can link 0_k to s_4 with cost 1, and delete s_2 which reduces the cost by 1.

Therefore, we can gradually remove all the bad nodes from T. This completes the proof. ■.

3.5 Concluding Remarks

It remains an interesting open question if the score scheme in the proof of Theorem 3.3 can be made to satisfy the triangle inequality. If so, then the result in Chapter 4 that tree alignment with a tree of bounded degree has a PTAS implies that the degree of the tree makes a difference in the approximability of tree alignment.

Chapter 4

Approximation Algorithms for Tree Alignment

4.1 Introduction

In Chapter 3, we have shown that tree alignment is NP-complete even when the given evolutionary tree is binary. In this chapter, we design a PTAS for tree alignment when the given evolutionary tree is of bounded degree. In fact, the evolutionary tree is often binary in practice This forms a sharp contrast with the MAX SNP-hardness of the generalized tree alignment problem.

4.1.1 Previous Results

Sankoff proposed an algorithm to compute an optimal tree alignment using dynamic programming [63, 67]. The time complexity of this algorithm is $O(m(2n)^k)$, where n is the

length of an input sequence, m is the number of internal nodes, and k is the number of leaves. The algorithm also assumes that the score between two letters is either 1 or 0. An algorithm that can handle more general score schemes is reported in [33]. Some heuristic algorithms have also been considered in the past. Altschul and Lipman tried to cut down the computation volume required by dynamic programming [1]. Sankoff, Cedergren and Lapalme gave an iterative improvement method to speed up the computation [65, 67]. It is claimed that the algorithm usually produces a reasonable alignment in 5 iterations. Waterman and Perlwitz devised a heuristic method when the sequence are related by a binary tree [84]. Their method computes an "average" sequence for each pair of related input sequences, from bottom to top, and then constructs an overall alignment by aligning each input sequence against the "average" sequence constructed at the root. The running time of this algorithm is $O(kn^2)$. Hein proposed an efficient algorithm based on the concept of a *sequence graph* [33]. Nevertheless, none of these algorithms have a guaranteed performance bound. In Section 3.3, it has been shown that tree alignment is NP-hard even if the given evolutionary tree is a binary tree.

4.1.2 Our Results

In this chapter, efficient approximation algorithms with guaranteed performance ratio for tree alignment are presented for the first time. We first give a simple algorithm which produces a loaded (*i.e.*, fully labeled) tree by elaborately lifting the input sequences from the leaves to their ancestors. It is shown that the loaded tree has a cost at most twice the optimum. The time complexity of this algorithm is $O(k^3 + k^2n^2)$. Augmenting the construction with a local optimization technique, we then extend this algorithm to a PTAS. More precisely, we devise an algorithm which, for each $t > 1$, has a performance ratio $1 + \frac{3}{t}$ and runs in time $O(k^{d^{t-1}+2}M(d, t-1, n))$, where $M(d, t-1, n)$ is the time needed

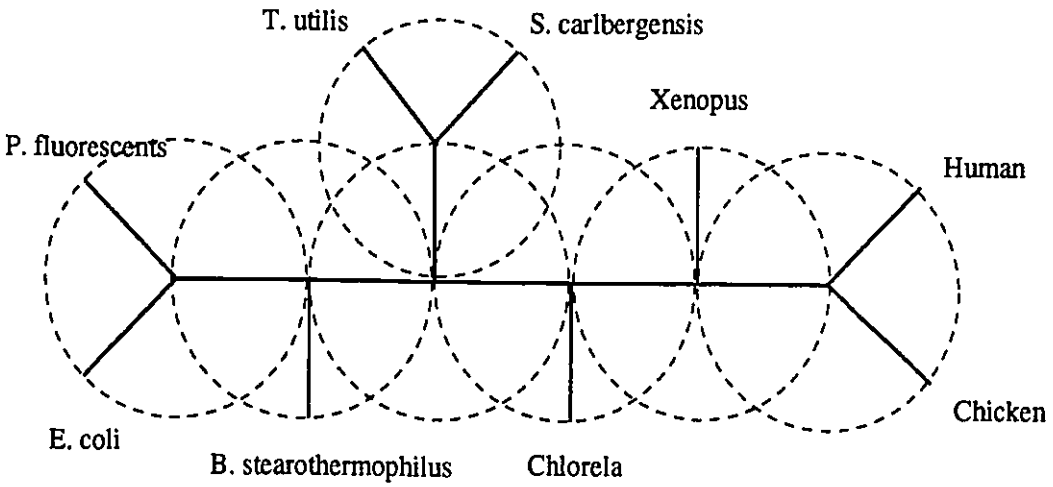


Figure 4.1: An evolutionary tree with 9 species, which is divided into seven 3-components.

to optimally align a tree of depth $t-1$ and is upper-bounded by $O(n^{d^{t-1}+1})$. The result is interesting since (i) to our knowledge, this is the first PTAS in the field of computational biology and (ii) it shows a great contrast with the MAX SNP-hardness of generalized tree alignment.

4.1.3 Applications in the Study of Molecular Evolution

Our algorithms may also be practical for the analysis of molecular evolution in the following sense: they can help set up a good initial alignment for the iterative method of Sankoff, Cedergren and Lapalme [65]. The combined algorithm will always produce an alignment that has a cost not greater than $1+\epsilon$ times the optimum and is (we hope) satisfactory to geneticists.

To illustrate the iterative method in [65], consider the evolutionary tree in Figure 4.1, which contains 9 given species on its 9 leaves. To construct a loaded tree, we first assign the given sequences to each internal node (arbitrarily). Then we divide the evolutionary tree

into 7 3-*components* as shown in Figure 4.1, where a 3-component is a *star* with a *center* and at most 3 terminals. Local optimization is done for every 3-component as follows. From the labels of the three terminals, we can compute a label of the center using dynamic programming to minimize the cost of the component [32, 63]. The new center label can then be used to update the center label of an overlapping 3-component. The algorithm converges since each local optimization reduces the cost of the tree by at least one. Thus, if the process is repeated long enough, every 3-component will become optimal. However, this does not necessarily result in an optimal loaded tree. Nonetheless, it seems the algorithm can produce a reasonably good loaded tree after 5 iterations [65].

4.1.4 Applications in Steiner Trees

Tree alignment can be viewed as a special case of the problem of Steiner trees under a given topology [42, 64]. It is known that a Steiner minimal tree can be computed in polynomial time for a given topology in the rectilinear (*i.e.*, Manhattan) space [22, 64]. This implies that the variant of tree alignment where the space is Σ^n and the distance is the Hamming distance is solvable in polynomial time. The same result trivially holds if the space is a graph. (Here the graph is part of the input and the running time is polynomial in the size of the graph.) For the Euclidean plane, the Steiner minimal tree can be found in $O(n^2)$ time if the given topology is a Steiner topology [42, 43]. For arbitrary topology, no exact algorithm is known; an iterative dynamic programming algorithm is given in [64].

The construction of our approximation algorithms in fact works for Steiner trees under a given topology in any metric space. As a corollary, we obtain a PTAS for the problem in the Euclidean plane for arbitrary topology. It is worth mentioning that, when the topology is not given, the best we know is that planar Steiner minimal trees can be approximated with ratio

$\frac{2}{\sqrt{3}} - \epsilon$, for some $\epsilon > 0$ [16]. At the moment, it is open whether the planar Steiner problem has a PTAS. It has been observed that Karp's *probabilistic ϵ-approximation scheme* for the traveling salesman problem can be modified to work for the planar Steiner problem [42, 49]. We note in passing that the Steiner problem on graphs is MAX SNP-hard [8].

The ratio 2 approximation algorithm and the PTAS are presented in Sections 4.2 and 4.3, respectively.

4.2 An Approximation Algorithm with Ratio 2

As mentioned earlier, our basic idea is to construct a loaded tree by elaborately lifting the given sequences from the leaves to their ancestors. Call a loaded tree a *lifted* tree if the label of each internal node equals the label of some child of the node. Below, we first show that there is a lifted tree of small cost and then compute the minimum-cost lifted tree in $O(k^3 + k^2n^2)$ time.

Let $X = \{s_1, \ldots, s_k\}$ be a set of sequences and T an evolutionary tree for X (*i.e.*, the leaves of T are uniquely labeled with the sequences $s_1, \ldots, s_k$). The *degree* of a node is its number of children. The degree of a tree is the maximum degree of its nodes. As in all earlier work in this area, we only consider trees with degrees bounded by some small constant d. To simplify the presentation, we will further assume that each internal node in T has exactly d children in our discussion. The extension of our results to the general case is fairly straightforward.

For a tree T, let $r(T)$ be the root of T, $L(T)$ the set of the leaves of T, and $I(T)$ the set of the internal nodes in T. The parent of a node v is represented as $p(v)$. For each node v in tree T, T_v denotes the subtree of T rooted at v. A leaf that is a descendant of

node v is called a *descendant leaf* of v. Define $S(v)$ to be the set of sequence labels of all descendant leaves of v. Note that the label of a leaf never changes during an alignment process. Define the cost of an edge (u, v) as the edit distance between the sequence labels of nodes u and v. The cost $c(T)$ of the tree T is the sum of the cost of every edge in T. Let T^{min} denote an optimal loaded tree for T. For each node v in T^{min}, the *closest descendant leaf* of v, denoted $l(v)$, is a descendant leaf of v such that the path from v to $l(v)$ is the shortest (*i.e.*, with the minimum-cost) among all descendant leaves of v. For convenience, let $sl(v)$ denote sequence label of $l(v)$. Define a loaded tree T^l as follows: for each internal node v in T, assign the sequence $sl(v)$ to v. Clearly, the tree T^l can be made a lifted tree if we break ties carefully while assigning $l(v)$. Throughout the analysis, we assume that the score scheme is a distance metric. (See Section 2.1.)

Lemma 4.1 $c(T^l) \leq 2(1 - \frac{1}{k})c(T^{min})$, *where k is the number of leaves in T.*

Proof. Consider a counter-clockwise walk along the outside of the optimal tree T^{min} that travels twice, once in each direction, through all the edges except those in the two boundary (leftmost and rightmost) paths. Since the order of the children does not matter, we can choose the two boundary paths such that their cost is the greatest. Therefore, the total cost of this walk is $c(T^l) \leq 2(1 - \frac{1}{k})c(T^{min})$, where k is the number of leaves in T. The walk can be thought of as a path that links all the leaves into a chain, from left to right. Take an arbitrary node v and consider the subtree T_v^{min} rooted at v. Let $v_1, \ldots, v_d$ be the children of v. These children induce d subtrees $T_{v_1}^{min}, \ldots, T_{v_d}^{min}$. For a node v, denote the rightmost and leftmost descendent leaves of v by $f(v)$ and $g(v)$, respectively. For each $i = 1, \ldots, d-1$, to connect the two leaves $f(v_i)$ and $g(v_{i+1})$, the walk uses a path

$$P_{v,i} : f(v_i) \rightarrow v_i \rightarrow v \rightarrow v_{i+1} \rightarrow g(v_{i+1}),$$

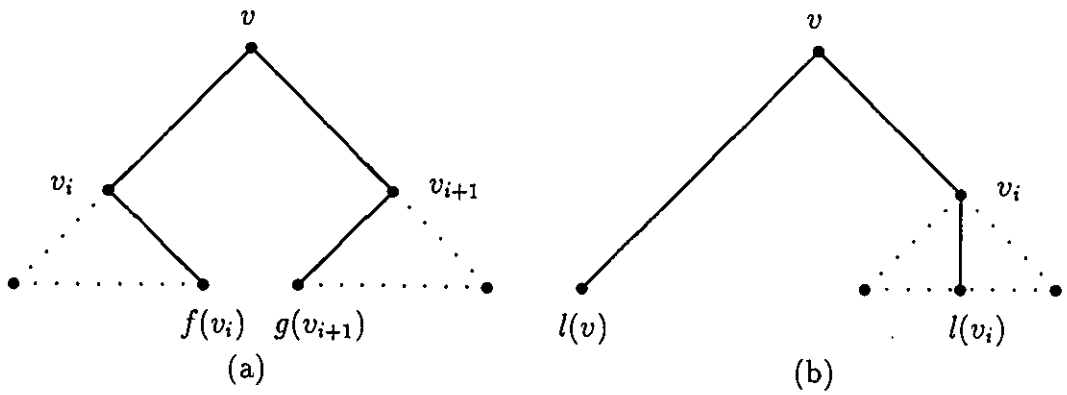


Figure 4.2: (a) The path $P_{v,i}$ in the walk. (b) The path $P'_{v,i}$.

(See Figure 4.2(a).) It is easy to see that

$$\sum_{v \in T} \sum_{i=1}^{d-1} c(P_{v,i}) = c(T^l) \leq 2(1 - \frac{1}{k})c(T^{min}).$$

Define $d-1$ new paths $P'_{v,i} : l(v) \rightarrow v \rightarrow v_i \rightarrow l(v_i)$, where $1 \leq i \leq d$ and $l(v_i) \neq l(v)$. (See Figure 4.2(b).) Let $P'^{(1)}_{v,i}$ and $P'^{(2)}_{v,i}$ denote the subpaths of $P'_{v,i}$: $l(v) \rightarrow v$ and $v \rightarrow v_i \rightarrow l(v_i)$, respectively. $P^{(1)}_{v,i}$ and $P^{(2)}_{v,i}$ denote the subpaths of $P_{v,i}$: $f(v_i) \rightarrow v_i \rightarrow v$ and $v \rightarrow v_{i+1} \rightarrow g(v_{i+1})$, respectively. By the definition of $l(v)$, we have:

$$c(P'^{(1)}_{v,i}) \leq c(P^{(2)}_{v,i}), \quad c(P'^{(2)}_{v,i}) \leq c(P^{(1)}_{v,i}),$$

and

$$c(P'^{(1)}_{v,i}) \leq c(P^{(1)}_{v,i-1}), \quad c(P'^{(2)}_{v,i}) \leq c(P^{(2)}_{v,i-1}).$$

Thus,

$$c(P'_{v,i}) \leq c(P_{v,i}), \quad \text{and} \quad c(P'_{v,i}) \leq c(P_{v,i-1}).$$

Therefore,

$$\sum_{\substack{1\le i\le d \\ l(v_i)\neq l(v)}} c(P'_{v,i}) \le \sum_{i=1}^{d-1} c(P_{v,i}). \tag{4.1}$$

Now consider the tree T^l. The total cost of the edges between v and its d children is

$$\sum_{i=1}^{d} dist(sl(v), sl(v_i)).$$

By the triangle inequality, we have $dist(sl(v), sl(v_i)) \le c(P'_{v,i})$. It follows from the above that

$$\begin{aligned} c(T^l) &= \sum_{v\in T}\sum_{i=1}^{d} dist(sl(v), sl(v_i)) \\ &\le \sum_{v\in T}\sum_{\substack{1\le i\le d \\ l(v_i)\neq l(v)}} c(P'_{v,i}) \\ &\le \sum_{v\in T}\sum_{i=1}^{d-1} c(P_{v,i}). \end{aligned}$$

Therefore, $c(T^l) \le 2(1+\frac{1}{k})c(T^{min})$. ■

Since T^l is actually a lifted tree, we can immediately conclude:

Corollary 4.2 *There exists a lifted tree with cost at most $2c(1-\frac{1}{k})T^{min}$, where k is the number of leaves in T.*

Computing T^l is not easy since it is derived from the optimal tree T^{min}. However, in the following we describe a simple algorithm that constructs an optimal lifted tree T^*, *i.e.*, one that has the smallest cost among all the lifted trees. From the above corollary,

$$c(T^*) \le c(T^l) \le 2c(1-\frac{1}{k})T^{min}.$$

The idea behind the algorithm is to use dynamic programming.

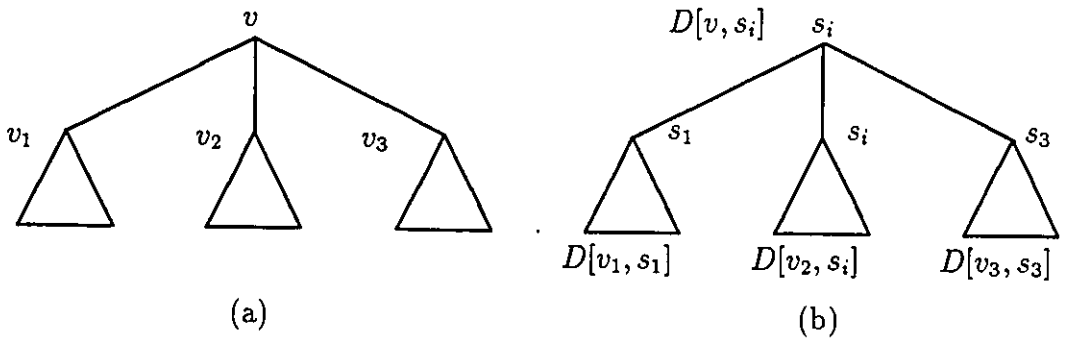


Figure 4.3: (a) The subtree T_v. (b) The lifted subtree, where $s_1 \in S(v_1)$, $s_i \in S(v_2)$, and $s_3 \in S(v_3)$.

For each $v \in T$, $i = 1, \ldots, k$ such that $s_i \in S(v)$, let $D[v, s_i]$ denote the cost of an optimal lifted tree for T_v with v being assigned the sequence s_i. It is possible to compute $D[v, s_i]$ recursively. For each leaf v, we define $D[v, s_i] = 0$ if the label of v is s_i. Let v be an internal node, and $v_1, \ldots, v_d$ its children. Suppose that $s_i \in S(v_p)$, where $1 \leq p \leq d$. Clearly p is unique. Then $D[v, s_i]$ can be computed as follows: For each $q = 1, \ldots, d$ and $q \neq p$, find a j_q such that $s_{j_q} \in S(v_q)$ and $D[v_q, s_{j_q}] + dist(s_i, s_{j_q})$ is minimized. Then

$$D[v, s_i] = D[v_p, s_i] + \sum_{\substack{1 \leq q \leq d \\ q \neq p}} (D[v_q, s_{j_q}] + dist(s_i, s_{j_q}))$$

(See Figure 4.3.) The full algorithm, is described in Figure 4.4.

Let n be the maximum length of any sequence in X. Line 3 of Algorithm 4.1 takes $O(k^2n^2)$ time. Each execution of line 7 takes $O(k)$ time. Since line 7 is executed $O(k^2)$ times, Algorithm 4.1 requires $O(k^3 + k^2n^2)$ time in the worst case. Hence, we have the following theorem.

Theorem 4.3 *Algorithm 4.1 outputs a loaded tree with cost at most* $2(1 - \frac{1}{k})c(T^{min})$ *in time* $O(k^3 + k^2n^2)$.

```
1. begin
2. for each pair (i, j), 1 ≤ i < j ≤ k, do
3.     compute dist(s_i, s_j).
4. for each level of T, with the bottom level first, do
5.     for each node v at the level do
6.         for i = 1 to k
7.             if s_i ∈ S(v) then compute D[v, s_i].
8. Select an s ∈ X such that D[r(T), s] is minimized.
9. Compute the lifted tree with cost D[r(T), s] by back-tracing.
10. end.
```

Figure 4.4: **Algorithm 4.1**

4.3 A Polynomial-Time Approximation Scheme

We extend Algorithm 4.1 to a PTAS by considering constant-size components of the tree and augmenting the "lifting" technique with local optimization which is also used in the iterative improvement method in Section 4.1.3. An overview of the approach is given below.

Recall that, if we label each node v in the tree T with the sequence $sl(v)$, we obtain a loaded tree T^l with cost at most twice the cost of the optimal loaded tree T^{min}. Let $t > 0$ be an integer. For each $v \in T$, define the *depth-t component* $T_{v,t}$ as the subtree of T_v containing only the top $t+1$ levels. Clearly, $T_{v,t}$ has at most d^t leaves and at most $(d^t-1)/(d-1)$ internal nodes. For a subtree $T_{v,t}$, $v \neq r(T)$, we can obtain a loaded subtree $T'_{v,t}$ by assigning to each node $u \in L(T_{v,t}) \cup \{v\}$ the sequence assigned to u in the tree T^l, which is $sl(u)$, and constructing the sequences for the other nodes in $T_{v,t}$ such that the cost of the subtree is minimized by dynamic programming. Obviously, $c(T'_{v,t}) \leq c(T^l_{v,t})$, where $T^l_{v,t}$ is the depth-t subtree of T^l rooted at v. This suggests that we should partition the tree T into depth-t components, and construct an optimal loaded subtree $T'_{v,t}$ for each component $T_{v,t}$.

We partition T as follows. Let set V_i contain all nodes at level i of T. (The root is at level 0.) We can partition the internal nodes (excluding the root) of T into t groups $G_0, \ldots, G_{t-1}$, where

$$G_i = \bigcup_{j \equiv i(\bmod t)} V_j.$$

For each $i = 0, \ldots, t-1$, let $T'_{r(T),i}$ denote the loaded subtree obtained by assigning to each node $v \in L(T_{r(T),i})$ the sequence $sl(v)$ and each other node in $T_{r(T),i}$ a sequence so that the cost of $T'_{r(T),i}$ is minimized. Clearly, for each i, $0 \leq i \leq t-1$, the union of subtrees $\cup_{v \in G_i} T'_{v,t} \cup T'_{r(T),i}$ forms a loaded tree, denoted as T'_i. We will first show that the total cost of all these loaded subtrees is bounded by $t+3$ times the cost of the optimal loaded tree, *i.e.*,

$$\sum_{i=0}^{t-1} T'_i = \sum_{v \neq r(T)} c(T'_{v,t}) + \sum_{i=0}^{t-1} c(T'_{r(T),i}) \leq (t+3) c(T^{min}). \tag{4.2}$$

Therefore, there exists a loaded tree T'_i with cost $c(T'_i) \leq (1 + \frac{3}{t}) c(T^{min})$.

Again, computing T'_i is not easy since it relies on the optimal tree T^{min}. Call a loaded tree a *depth-t component tree* with respect to some fixed G_i if every node $v \in G_i$ is assigned a sequence which labels some node in $L(T_{v,t})$. In other words, the labels of the nodes in G_i are all lifted from the leaves. Since in the tree T'_i every node $v \in G_i$ is assigned the sequence $sl(v)$, T'_i is a depth-t component tree with respect to G_i. Hence, we can design a dynamic programming algorithm to identify the optimal depth-t component tree with respect to G_i, for each i, and select the best tree as our output which costs at most $(1 + \frac{3}{t}) c(T^{min})$.

Now, let us first prove the inequality (4.2). For each node v, let $\rho(v)$ be the length of the path from v to $l(v)$, $h(v)$ the total length of the edges from v to its children, and $s(v)$ be the sequence label assigned to v, in the optimal tree T^{min}. In particular, if v is a leaf, then $\rho(v) = 0$ and $h(v) = 0$. The following lemma holds.

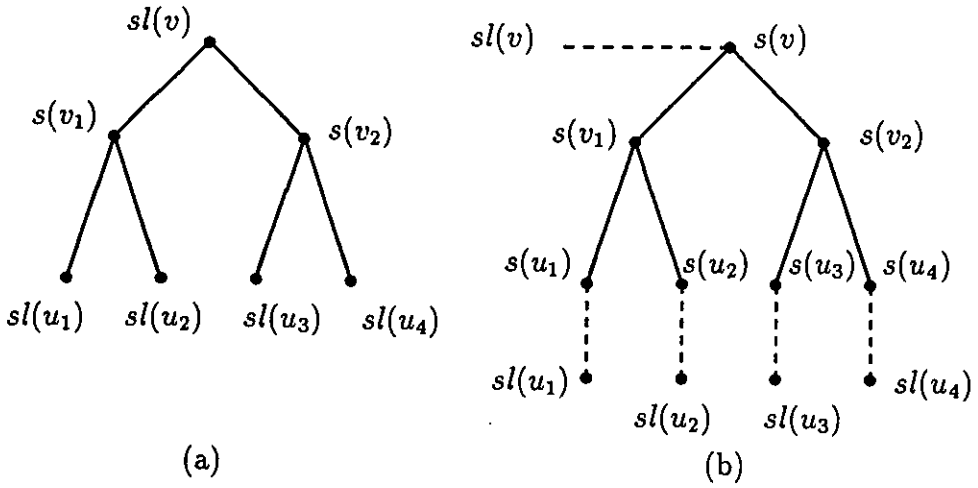


Figure 4.5: (a) The subtree $T''(v,2)$. (b) The subtree T_v^{min}.

Lemma 4.4 *Let $v \neq r(T)$ be a node in T and have children $v_1, \ldots, v_d$, then*

$$c(T'_{v,t}) \leq \sum_{u \in I(T_{v,t})} h(u) + \sum_{u \in L(T_{v,t})} \rho(u) + h(v) + \sum_{i=1}^{d} \rho(v_i). \tag{4.3}$$

Proof. Let $T''_{v,t}$ be the loaded subtree obtained by labeling each node $u \in L(T_{v,t}) \cup \{v\}$ with the sequence $sl(u)$ and each other node u in $T_{v,t}$ with the sequence $s(u)$. (See Figure 4.5.) By the triangle inequality, we have for each node $u \in L(T_{v,t})$,

$$dist(s(p(u)), sl(u)) \leq dist(s(p(u)), s(u)) + \rho(u).$$

Thus,

$$c(T''_{v,t}) \leq \sum_{i=1}^{d} dist(sl(v), s(v_i)) + \sum_{u \in I(T_{v,t}) - \{v\}} h(u) + \sum_{u \in L(T_{v,t})} \rho(u). \tag{4.4}$$

An explanation of this inequality is given in Figure 4.5(b). Again, by the triangle inequality,

$$dist(sl(v), s(v_i)) \leq dist(s(v_i), s(v)) + \rho(v).$$

Moreover,

$$\sum_{i=1}^{d} dist(s(v_i), s(v)) = h(v).$$

Hence, we can conclude

$$c(T''_{v,t}) \leq \sum_{u \in I(T_{v,t})} h(u) + \sum_{u \in L(T_{v,t})} \rho(u) + d\rho(v).$$

Since $\rho(v) \leq dist(s(v_i), s(v)) + \rho(v_i)$ for each $i = 1, \ldots, d$,

$$\begin{aligned} c(T''_{v,t}) &\leq \sum_{u \in I(T_{v,t})} h(u) + \sum_{u \in L(T_{v,t})} \rho(u) + \sum_{i=1}^{d} dist(s(v_i), s(v)) + \rho(v_i) \\ &\leq \sum_{u \in I(T_{v,t})} h(u) + \sum_{u \in L(T_{v,t})} \rho(u) + h(v) + \sum_{i=1}^{d} \rho(v_i). \end{aligned}$$

Since the cost of $T'_{v,t}$ is minimized, we have $c(T'_{v,t}) \leq c(T''_{v,t})$. ■

Note that, if the node v is near the bottom level of T, the real depth t' of $T_{v,t}$ could be less than t. In this case, the above inequality (4.3) becomes

$$c(T'_{v,t'}) \leq \sum_{u \in I(T_{v,t'})} h(u) + \sum_{u \in L(T_{v,t'})} \rho(u) + h(v) + \sum_{i=1}^{d} \rho(v_i). \tag{4.5}$$

Similarly, let $T''_{r(T),i}$ denote the loaded subtree obtained by assigning to each node $v \in L(T_{r(T),i})$ the sequence $sl(v)$ and each other node v the sequence $s(v)$. Note that, now the root $r(T)$ is labeled with $s(r(T))$ instead of $sl(r(T))$ in the subtree $T''_{r(T),i}$. Then inequality (4.4) becomes

$$c(T'_{r(T),i}) \leq c(T''_{r(T),i}) \leq \sum_{u \in I(T_{r(T),i})} h(u) + \sum_{u \in L(T_{r(T),i})} \rho(u). \tag{4.6}$$

Sum up inequalities (4.3) or (4.5) for all $v \in T$, and inequality (4.6) for $i = 0, \ldots, t-1$, we have

$$\sum_{v \neq r(T)} c(T'_{v,t}) + \sum_{i=0}^{t-1} c(T'_{r(T),i}) \leq (t+1) \sum_{v \in T} h(v) + 2 \sum_{v \in T} \rho(v). \tag{4.7}$$

The coefficients in the above inequality are established from the following observations:

1. The term $h(v)$ in inequality (4.3) is counted once for every v.

2. The term $\sum_{u \in I(T_{v,t})} h(u)$ in inequalities (4.3), (4.5), and (4.6) totally contributes at most t $h(v)$'s for each v.

3. For each v in the top t levels of T, there is a $\rho(v)$ contribution from (4.6) but at most a contribution of 1 from inequality (4.3) or (4.5). For each v below level t, there is no $\rho(v)$ contribution from inequality (4.6) but at most two $\rho(v)$'s from inequality (4.3) or (4.5). Therefore, each v contributes at most two $\rho(v)$'s.

Now, we want to upper-bound the right-hand side of inequality (4.7) in terms of $c(T^{min})$. Obviously,

$$\sum_{v \in T} h(v) = c(T^{min}). \tag{4.8}$$

To establish the relation between $\sum_{v \in T} \rho(v)$ and $c(T^{min})$, we need the following lemma, which is a variation of Lemma 3.2 in [16].

Lemma 4.5 *Let U be a tree such that every internal node has at least two children. There exists a one-to-one mapping e from the internal nodes of U to the leaves such that for each internal node v (i) $e(v)$ is a descendant of v and (ii) the paths from internal nodes v to $e(v)$ are edge-disjoint.*

Proof. We prove it by induction on the depth of the tree. First, we strengthen the lemma by adding (iii) there is another leaf $e'(r(U))$ such that the path from $r(U)$ to $e'(r(U))$ is edge-disjoint from the paths in (ii). The lemma is trivial for trees with depth 1. Suppose that (i)-(iii) hold for any tree with depth $i > 0$. Consider a tree U with depth $i + 1$. Let the root of U have $d \geq 2$ children: $v_1, \ldots, v_d$. To construct the mapping e for the internal

nodes in U, we preserve the mappings for the subtrees $U_{v_1}, \ldots, U_{v_d}$, and assign $e'(v_1)$ as $e(r(U))$ and $e'(v_2)$ as $e'(r(U))$. ■

Let e be such a mapping for tree T. For each node v, let $\tau(v)$ denote the length of the path from v to $e(v)$ in T^{min}. ($\tau(v) = 0$ if v is a leaf.) It follows from the above lemma that

$$\sum_{v \in T} \tau(v) \leq c(T^{min}).$$

By the definition of $\rho(v)$,

$$\sum_{v \in T} \rho(v) \leq \sum_{v \in T} \tau(v) \leq c(T^{min}). \tag{4.9}$$

The above inequality is crucial for our result. The inequalities (4.7), (4.8), and (4.9) immediately imply the following lemma:

Lemma 4.6

$$\sum_{i=0}^{t-1} T_i = \sum_{v \neq r(T)} c(T'_{v,t}) + \sum_{i=0}^{t-1} c(T'_{r(T),i}) \leq (t+3)c(T^{min}).$$

By Lemma 4.6, there exists an i, $0 \leq i \leq t-1$, such that

$$t \cdot c(T'_i) \leq (t+3)c(T^{min}).$$

Thus, the following lemma holds.

Lemma 4.7 *There exists a depth-t component tree with cost at most* $(1 + \frac{3}{t})c(T^{min})$.

Although T'_i approximates T^{min} with the desired bound, the trouble is again that it is not easy to compute T'_i. Below, we describe an algorithm to construct a *minimum-cost* depth-t component tree $\hat{T}$, *i.e.*, the cost of $\hat{T}$ is the smallest among all the depth-t component

trees (with respect to some G_i). The basic idea is to combine dynamic programming with local optimization.

Consider a $T_{v,t}$. Let u be a child of v and $L(T_{u,t-1}) = \{w_1, \ldots, w_m\}$ be the set of leaves in $T_{u,t-1}$. (Note that the component $T_{v,t}$ may not be full.) For each $i = 1, \ldots, m$, let $s_i \in S(w_i)$ be a sequence. Then for each sequence s, $\hat{T}(v, t, u, s, s_1, \ldots, s_m)$ denotes the loaded subtree obtained from $T_{u,t-1} \cup \{(v, u)\}$ by labeling v with sequence s and each node w_i with sequence s_i, and constructing a sequence for each other node in $T_{u,t-1}$ so that the cost of $\hat{T}(v, t, u, s, s_1, \ldots, s_m)$ is minimized.

The top subtree $T_{r(T),i}$, $0 \leq i \leq t-1$, is treated similarly and the resulting loaded subtree is denoted $\hat{T}(r(T), i, u, s, s_1, \ldots, s_m)$.

Let v be a node at level i of T, and $s \in S(v)$. Define $\hat{T}(v, s)$ as the minimum-cost loaded subtree obtained from T_v such that the node v is labeled with the sequence s and the loaded subtree itself forms a depth-t component tree with respect to G_j, where $j \equiv i(\bmod t)$. We use $D[v, s]$ to denote the cost of $\hat{T}(v, s)$. Similar to the previous section, $D[v, s]$ can be computed recursively from bottom to top.

If v is a leaf, $D[v, s(v)] = 0$. Let v be an internal node and $u_1, \ldots, u_d$ the children of v. For each $i = 1, \ldots, d$, let $L(T_{u_i,t-1}) = \{w_{i,1}, \ldots, w_{i,m}\}$. Suppose that $s \in S(w_{p,q})$, where $1 \leq p \leq d$ and $1 \leq q \leq m$ are unique. Then $D[v, s]$ can be computed as follows: For each $i = 1, \ldots, d$ and each $j = 1, \ldots, m$, find an $s_{i,j}$ such that $s_{i,j} \in S(w_{i,j})$ if $i \neq p$ or $j \neq q$, and $s_{i,j} = s$ otherwise, and moreover

$$c(\hat{T}(v, t, u_i, s, s_{i,1}, \ldots, s_{i,m})) + \sum_{j=1}^{m} D[w_{i,j}, s_{i,j}]$$

is minimized. Then

$$D[v, s] \;=\; \sum_{i=1}^{d} c(\hat{T}(v, t, u_i, s, s_{i,1}, \ldots, s_{i,m})) \;+$$

```
1. begin
2. for each level, with the bottom level first, do
3.    for each node v at the level do
4.       for each s ∈ S(v) do
5.          compute D[v, s].
6. for i = 0 to t − 1 do
7.    compute D[i].
8. Select an i to minimize D[i].
9. Compute the loaded tree T̂ from D[i] by back-tracing.
10. end.
```

Figure 4.6: **Algorithm 4.2**

$$\sum_{i=1}^{d}\sum_{j=1}^{m} D[w_{i,j}, s_{i,j}].$$

For each $i = 0, \ldots, t-1$, let $D[i]$ be the cost of the optimal depth-t component tree with all the roots of all their depth-t components in G_i. $D[i]$ can be computed from the top subtree $T_{r(T),i}$ and the values $D[v,s]$ of the nodes at level i of T in a way similar to above. Clearly, $\min\{D[i] | 0 \leq i \leq t-1\} = c(\hat{T})$.

The actual algorithm for computing $\hat{T}$ is given in Figure 4.6. Suppose that computing each $c(\hat{T}(v,t,u,s,s_1,\ldots,s_m))$ or $c(\hat{T}(r(T),i,u,s,s_1,\ldots,s_m))$ (*i.e.*, local optimization) requires at most $M(d,t-1,n)$ time. Clearly $M(d,t-1,n)$ is upper-bounded by $O(n^{d^{t-1}+1})$ [63]. Thus an execution of line 5 takes $k^{d^{t-1}}M(d,t-1,n)$ time. Line 5 is repeated a total of $O(k^2)$ times. Line 7 is executed t times, each taking at most $k^{d^{t-2}+1}M(d,t-2,n)$ time. Therefore, the time complexity of Algorithm 4.2 is $O(k^{d^{t-1}+2}M(d,t-1,n))$. (In fact, it is easy to show that the algorithm runs in time $O((k/d^t)^{d^{t-1}+2}M(d,t-1,n))$.)

Theorem 4.8 *For any t, Algorithm 4.2 has a performance ratio $R_t \leq 1 + \frac{3}{t}$ and runs in time $O(k^{d^{t-1}+2}M(d,t-1,n))$.*

Corollary 4.9 *Tree alignment has a PTAS when the given evolutionary tree is of bounded degree.*

4.4 Preliminary Experiment

Some preliminary experimenting has been done to test Algorithm 4.1 with real data, using an interactive sequence alignment tool (SAT) developed by X. Chen [11]. Algorithm 4.1 is implemented in the C programming language as a function of SAT. A set of real data is taken from [65]. The given evolutionary tree is shown in Figure 4.1, where there are nine leaves. Each leaf is assigned a sequence. The nine RNA sequences for the leaves are given in Figure 4.4.

For a comparison, we use the same score scheme as in [65], *i.e.*, $\mu(A,C) = 1.75$, $\mu(A,G) = 1.0$, $\mu(A,U) = 1.75$, $\mu(C,G) = 1.75$, $\mu(C,U) = 1.0$, $\mu(G,U) = 1.75$, and 2.25 for insertion/deletion. In the computation, we select a root between the two dotted internal nodes as shown in Figure 4.8. Algorithm 4.1 lifts the leaves to the internal nodes and ends up with a lifted tree as shown in Figure 4.8. The total cost of the lifted tree is 382.25. Then the lifted tree is used as an initial assignment for the iterative method developed by Sankoff et al. [65]. Using the subroutine for optimally aligning a 3-sequence component provided by SAT, we do a local optimization for every internal node as described in Section 4.1. The value of the tree is reduced to 305.25 after one iteration. The value is further reduced to 304.25 after the second iteration and it remains the same after the third iteration. This comes fairly close to 295.50, which is the best value obtained in [65]. The corresponding multiple sequence alignment is shown in Figure 4.9.

At present, we are not able to test Algorithm 4.2 for $t \geq 3$, since computing an optimal

E.coli
UGCCUGGCGG CCGUAGCGCG GUGGUCCCAC CUGACCCCAU GCCGAACUCA GAAGUGAAAC
GCCGUAGCGC CGAUGGUAGU GUGGGGUCUC CCCAUGCGAG AGUAGGGAAC UGCCAGGCAU

P.fluorescens
UGUUCUUUGA CGAGUAGUAG CAUUGGAACA CCUGAUCCCA UCCCGAACUC AGAGGUGAAA
CGAUGCAUCG CCGAUGGUAG UGUGGGGUUU CCCCAUGUCA AGAUCUCGAC CAUAGAGCAU

S.carlbergensis
GGUUGCGGCC AUACCAUCUA GAAAGCACCG UUCUCCGUCC GAUAACCUGU AGUUAAGCUG
GUAAGAGCCU GACCGAGUAG UGUAGUGGGU GACCAUACGC GAAACCUAGG UGCUGCAAUCU

Human
GUCUACGGCC AUACCACCCU GAACGCGCCC GAUCUCGUCU GAUCUCGGAA GCUAAGCAGG
GUCGGGCCUG GUUAGUACUU GGAUGGGAGA CCGCCUGGGA AUACCGGGUG CUGUAGGCUU

Xenopus
GCCUACGGCC ACACCACCCU GAAAGUGCCC GAUCUCGUCU GAUCUCGGAA GCCAAGCAGG
GUCGGGCCUG GUUAGUACUU GGAUGGGAGA CCGCCUGGGA AUACCAGGUG UCGUAGGCUU

Chlorella
AUGCUACGUU CAUACACCAC GAAAGCACCC GAUCCCAUCA GAACUCGGAA GUUAAACGUG
GUUGGGCUCG ACUAGUACUC GGUUGGGAGG AUUACCUGAG UGGGAACCCC GACGUAGUGU

Chichen
GCCUACGGCC AUCCCACCCC UGUAACGCCC GAUCUCGUCU GAUCUCGGAA GCUAAGCAGG
GUCGGGCCUG GUUAGUACUU GGAUGGGAGA CCUCCUGGGA AUACCGGGUG CUGUAGGCUU

B.stearothermophilus
CCUAGUGACA AUAGCGAGGA GAGAAACACC CGUCUCCAUC CCGAACACGA AGGUUAAGCU
CUCCCAGCGC CGAUGGUAGU UGGGGCCAGC GCCCCUGCAA GAGUAGGUUG UCGCUAGGC

T.utilis
GGUUGCGGCC AUAUCUAGCA GAAAGCACCG UUCUCCGUCC GAUCAACUGU AGUUAAGCUG
CUAAGAGCCU GAUCGAGUAG UGUAGUGGGU GACCAUACGC GAAACUCAGG UGCUGCAAUCU

Figure 4.7: The nine sequences.

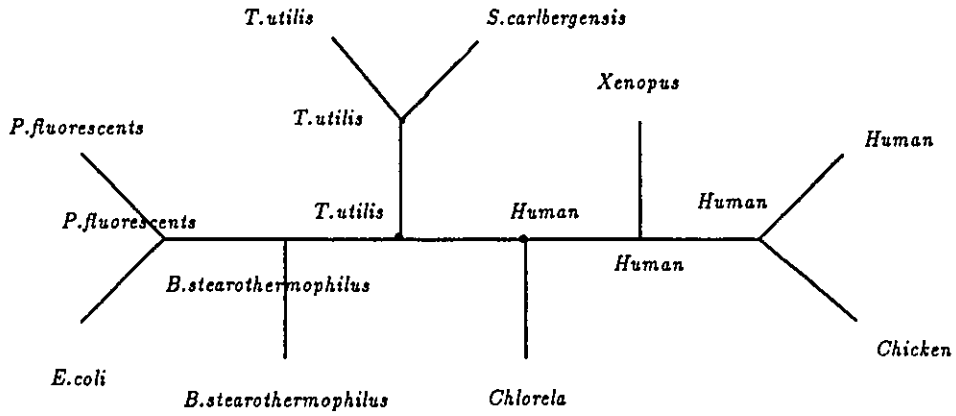


Figure 4.8: The lifted tree obtained by Algorithm 4.1.

```
0: UGCC*UGGCGGC*CGUA*GC***GC*G*GUGGUCCCACCUGACCCCAUGCCGA*ACUCAGAAGUGAAAC*GC*
1: UGUU*CUUUGACGAGUA*GU*A*GC*A*UUGG*AACACCUGAUCCCAUCCCGA*ACUCAGAGGUGAAAC*GA*
2: *G*G*UUGCGGC*CAUA*CC*A*UCUA*GA*A*AGCACCGUUCUCCGU*CCGAUAACCUGUAGUUAAGC*UG*
3: *G*U*CUACGGC*CAUA*CC*A*CC*CUGA*A*CGCGCCCGAUCUCGU*CUGA*UCUCGGAAGCUAAGCAGG*
4: *G*C*CUACGGC*CACA*CC*A*CC*CUGA*A*AGUGCCCGAUCUCGU*CUGA*UCUCGGAAGCCAAGCAGG*
5: *A*UGCUACGUU*CAUA**C*A*CC*ACGA*A*AGCACCCGAUCCCAU*CAGA*ACUCGGAAGUUAAACGUG*
6: *G*C*CUACGGC*CAUC*CC*ACCC*CUGU*A*A*CGCCCGAUCUCGU*CUGA*UCUCGGAAGCUAAGCAGG*
7: *C*C*UAGUGAC*AAUA*GCGA*GG*A*GAGA*AACACCCGUCUCCAUCCCGA*ACACGAAGGUUAAGC*UCU
8: *G*G*UUGCGGC*CAUAUCU*A*GC*A*GA*A*AGCACCGUUCUCCGU*CCGAUCAACUGUAGUUAAGC*UG*

0: CGUAGCGCC*GAU*G*GUAGUGU*G*GGGU*CU*C*CCCAUG*C*GAGAGUAGGGAACUGCCAG*GCAU
1: UGCAUCGCC*GAU*G*GUAGUGU*G*GGGU*UU*C*CCCAUGUC*AAGA*UCUCG*ACCA*UAGAGCAU
2: GUAAGAGCCUGACCGAGUAGUGUAGUGGGU*GA*C*CAUACG*C*GAAACCUAGGUGCUG*CAA*UC*U
3: GUCGG*GCCUGGU*UAGUACUUGGAUGGGA*GA*C*CGCCUG*G*GAAUACCGGGUGCUG*UAG*GCUU
4: GUCGG*GCCUGGU*UAGUACUUGGAUGGGA*GA*C*CGCCUG*G*GAAUACCAGGUGUCG*UAG*GCUU
5: GUUGG*GCUCGAC*UAGUACUGGGUUGGGAGGA*U*UACCUGAGUGGGAACC**CCGACG*UAG*UG*U
6: GUCGG*GCCUGGU*UAGUACUUGGAUGGGA*GA*C*CUCCUG*G*GAAUACCGGGUGCUG*UAG*GCUU
7: CCCAGCGCC*GAU*G*GUAGU*U*G*GGGC*CAGCGCCCCUG*C*AAGAGUAGGUUGUCGCUAG*GC**
8: CUAAGAGCCUGAUCGAGUAGUGUAGUGGGU*GA*C*CAUACG*C*GAAACUCAGGUGCUG*CAA*UC*U
```

Figure 4.9: The multiple sequence alignment obtained by our algorithm. The numbers 0-8 stand for *E.coli*, *P.fluorescens*, *S.carlbergensis*, *Human*, *Xenopus*, *Chlorella*, *Chichen*, *B.stearothermophilus*, and *T.utilis*, respectively.

tree alignment of five 110-letter sequences requires too much space and time.

4.5 Remarks

It is unknown whether Corollary 4.9 holds for evolutionary trees with unbounded degrees. Interestingly, we have shown that for a score scheme that does not satisfy the triangle inequality, constructing an optimal loaded tree is MAX SNP-hard even when the given evolutionary tree is a star.

Also, the high complexity of Algorithm 4.2 excludes even moderate t from consideration in practice. It would be of great interest to improve the efficiency of the algorithm.

Chapter 5

A New Measure for Comparing Labeled Trees

5.1 Introduction

In many fields such as RNA secondary structures comparison, syntactic pattern recognition, image clustering, genetics, and chemical structure analysis, one often faces the problem of finding the similarity of two *labeled trees* [54, 55, 57, 66, 71, 73, 74, 77]. As we have mentioned in Chapter 2, the comparison of *ordered* labeled trees is very useful in the study of RNA secondary structures as an RNA secondary structure can be conveniently expressed as an ordered labeled tree [55, 71, 72, 86]. On the other hand, the comparison of *unordered labeled trees* has applications to the morphological problems arising in genetics (*e.g.*, determining genetic diseases based on ancestry tree patterns) and other fields [73, 74, 77].

As in the case of sequence comparison, there are many ways to measure the similarity between two trees. For instance, one could use the *largest common sub-tree*, the *smallest*

common super-tree, *tree edit distance*, and the *transferable ratio* between two trees to describe the degree of similarity [51, 55, 66, 72, 76, 88]. Although edit distance and transferable ratio are both sensible measures of the distance between RNA secondary structures [55, 66], each of them only represents a certain approximation of the true functional similarity. Thus, more realistic and feasible measures would always be of interest.

The notion of *alignment of trees* was introduced in Section 2.2 as a new measure of similarity of labeled trees. Here we present an algorithm for computing the alignment distance between ordered trees. The time complexity of this algorithm is $O(|T_1| \cdot |T_2| \cdot (\deg(T_1) + \deg(T_2))^2)$, where $|T_i|$ and $\deg(T_i)$ are the size and degree of T_i, respectively. (Again, the degree of a tree is the maximum number of children of any node in the tree.) We also show that the alignment distance between two unordered trees can be computed in polynomial time if the trees have bounded degrees and becomes NP-hard if one of the trees is allowed to have an arbitrary degree. The following summarizes the main differences between alignment of trees and tree edit.

- **Alignment of trees vs tree edit.**

It is well known that edit and alignment are two equivalent notions for sequences. In particular, for any two sequences s_1 and s_2, the edit distance between s_1 and s_2 equals their alignment distance. However, edit and alignment turn out to be very different for trees. The following are some interesting comparisons between alignment of trees and tree edit.

1. The edit distance and alignment distance between two trees can be different. For example, assume that each edit operation (*i.e.*, insertion, deletion, or replacement) costs 1 and consequently each pair of distinct letters has a score 1. Consider the two ordered trees shown in Figure 2.6. To optimally edit T_1 into T_2, we simply delete

e from T_1 and insert f into the new tree. Thus, the edit distance between T_1 and T_2 is 2. The optimal alignment of the two trees is unique and is shown in 2.6(c), with a value 4. The difference between edit distance and alignment distance can be made arbitrarily large by adding subtrees below nodes b, c, d in both trees. It is easy to see that in general the edit distance is smaller than the alignment distance for trees. This is because each alignment of trees actually corresponds to a restricted tree edit in which all the insertion precede all the deletions. Note that, the order of edit operations is not important for sequences. Also, it seems that alignment charges more for the structural dissimilarity at the top levels of the trees than at the lower levels, whereas edit treats all the levels the same.

2. The best algorithm for computing the edit distance between ordered trees runs in time $O(|T_1| \cdot |T_2| \cdot \min\{\text{depth}(T_1), \text{leaves}(T_1)\} \cdot \min\{\text{depth}(T_2), \text{leaves}(T_2)\})$, where $\text{depth}(T_i)$ and $\text{leaves}(T_i)$ are the depth and number of leaves of tree T_i, $i = 1, 2$ [86]. Clearly, $\deg(T_i) \leq \text{leaves}(T_i)$. In practice (*e.g.*, RNA secondary structures), $\deg(T_i) \ll \text{leaves}(T_i)$ and $\deg(T_i) \ll \text{depth}(T_i)$. Hence, the result above shows that it is easier (faster) to align ordered trees than to edit. In particular, we can align trees with bounded degrees in time $O(|T_1| \cdot |T_2|)$.

3. The difference in time complexity is even bigger for unordered trees. As mentioned earlier, unordered trees with bounded degrees can be aligned in polynomial time (in fact, in time $O(|T_1| \cdot |T_2|)$). On the other hand, editing unordered trees with bounded degrees is NP-hard [87]. (In fact, it is MAX SNP-hard [88].)

4. The alignment of trees can be easily generalized to more than two trees as in the case of sequences. This provides a way to compare several trees simultaneously. Although it is also possible to compare several trees based on tree edit distance, as reported in [72],

the method seems to be only applicable to situations where clustering is required (*e.g.*, in the construction of a taxonomy tree).

In section 5.2, we present the algorithm for aligning ordered trees. Section 5.3 contains some results on unordered trees. We briefly discuss multiple alignment of trees with SP-score in section 5.4.

5.2 An Efficient Algorithm for Aligning Ordered Trees

We need some definitions. The notion of alignment can be easily extended to *ordered forests*. (Each ordered forest is a sequence of ordered trees.) The only change is that it is now possible to insert a node (as the root) to join a consecutive subsequence of trees in a forest. Denote the alignment distance between forests F_1 and F_2 as $D(F_1, F_2)$. Let θ denote the empty tree, λ denote space, and $\mu(a, b)$ denote the score of the opposing letters a and b. The nodes in an ordered tree of size n are numbered 1 through n according to the *postorder*. Let T_1 and T_2 be two fixed ordered labeled trees throughout this section. Denote the label of node j in tree T_i as $l_i[j]$ and the subtree of T_i rooted at node j as $T_i[j]$.

In the following, let i be a node of T_1 and j a node of T_2. Suppose that the degrees of i and j are m_i and n_j, respectively. Denote the children of i as $i_1, \ldots, i_{m_i}$ and the children of j as $j_1, \ldots, j_{n_j}$. For any s, t, $1 \leq s \leq t \leq m_i$, let $F_1[i_s, i_t]$ represent the forest consisting of the subtrees $T_1[i_s], \ldots, T_1[i_t]$. For convenience, $F_1[i_1, i_{m_i}]$ is also denoted $F_1[i]$. Note that $F_1[i] \neq F_1[i, i]$. $F_2[j_s, j_t]$ and $F_2[j]$ are defined similarly.

5.2.1 Properties of the Alignment Distance

The following lemmas below form the basis of our algorithm. The first lemma is trivial.

Lemma 5.1 $D(\theta,\theta) = 0$;

$$D(F_1[i],\theta) = \sum_{k=1}^{m_i} D(T_1[i_k],\theta); \qquad D(T_1[i],\theta) = D(F_1[i],\theta) + \mu(l_1[i],\lambda);$$

$$D(\theta,F_2[j]) = \sum_{k=1}^{n_j} D(\theta,T_2[j_k]); \qquad D(\theta,T_2[j]) = D(\theta,F_2[j]) + \mu(\lambda,l_2[j]).$$

Lemma 5.2

$$D(T_1[i],T_2[j]) = \min \begin{cases} D(\theta,T_2[j]) + \min_{1\le r\le n_j}\{D(T_1[i],T_2[j_r]) - D(\theta,T_2[j_r])\} \\ D(T_1[i],\theta) + \min_{1\le r\le m_i}\{D(T_1[i_r],T_2[j]) - D(T_1[i_r],\theta)\} \\ D(F_1[i],F_2[j]) + \mu(l_1[i],l_2[j]) \end{cases}$$

Proof: Consider an optimal alignment (tree) $\mathcal{A}$ of $T_1[i]$ and $T_2[j]$. There are four cases: (1) $(l_1[i],l_2[j])$ is a label in $\mathcal{A}$, (2) $(l_1[i],\lambda)$ and $(l_1[k],l_2[j])$ are labels in $\mathcal{A}$ for some k, (3) $(l_1[i],l_2[k])$ and $(\lambda,l_2[j])$ are labels in $\mathcal{A}$ for some k, (4) $(l_1[i],\lambda)$ and $(\lambda,,l_2[j])$ are labels in $\mathcal{A}$. We actually need not consider Case 4 since in that case we can delete the two nodes and then add $(l_1[i],l_2[j])$ as the new root, resulting in a better alignment.

Case 1. The root of $\mathcal{A}$ must be labeled as $(l_1[i],l_2[j])$. Clearly, $D(T_1[i],T_2[j]) = D(F_1[i],F_2[j]) + \mu(l_1[i],l_2[j])$.

Case 2. The root of $\mathcal{A}$ must be labeled as $(l_1[i],\lambda)$. In this case k must be a node in $T_1[i_r]$ for some $1 \le r \le m_i$. Therefore,

$$D(T_1[i],T_2[j]) = D(T_1[i],\theta) + \min_{1\le r\le m_i}\{D(T_1[i_r],T_2[j]) - D(T_1[i_r],\theta)\}$$

Case 3. Similar to Case 2. □

Note, the above implies that $D(F_1[i], F_2[j])$ is required for computing $D(T_1[i], T_2[j])$.

Lemma 5.3 *For any s, t such that $1 \leq s \leq m_i$ and $1 \leq t \leq n_j$,*

$$D(F_1[i_1, i_s], F_2[j_1, j_t]) = \min \begin{cases} D(F_1[i_1, i_{s-1}], F_2[j_1, j_t]) + D(T_1[i_s], \theta) \\ D(F_1[i_1, i_s], F_2[j_1, j_{t-1}]) + D(\theta, T_2[j_t]) \\ D(F_1[i_1, i_{s-1}], F_2[j_1, j_{t-1}]) + D(T_1[i_s], T_2[j_t]) \\ \mu(\lambda, l_2[j_t]) + \min_{1 \leq k < s} \{D(F_1[i_1, i_{k-1}], F_2[j_1, j_{t-1}]) + D(F_1[i_k, i_s], F_2[j_t])\} \\ \mu(l_1[i_s], \lambda) + \min_{1 \leq k < t} \{D(F_1[i_1, i_{s-1}], F_2[j_1, j_{k-1}]) + D(F_1[i_s], F_2[j_k, j_t])\} \end{cases}$$

Proof: Consider an optimal alignment (forest) $\mathcal{A}$ of $F_1[i_1, i_s]$ and $F_2[j_1, j_t]$). The root of the rightmost tree in $\mathcal{A}$ is labeled by either $(l_1[i_s], l_2[j_t])$, $(l_1[i_s], \lambda)$, or $(\lambda, l_2[j_t])$.

Case 1: the label is $(l_1[i_s], l_2[j_t])$. In this case, the rightmost tree must be an optimal alignment of $T_1[i_s]$ and $T_2[j_t]$. Therefore

$$D(F_1[i_1, i_s], F_2[j_1, j_t]) = D(F_1[i_1, i_{s-1}], F_2[j_1, j_{t-1}]) + D(T_1[i_s], T_2[j_t])$$

Case 2: the label is $(l_1[i_s], \lambda)$. In this case, there is a k, $0 \leq k \leq t$, such that $T_1[i_s]$ is aligned with the subforest $F_2[j_{t-k+1}, j_t]$. A key observation is here that we can assume the subtree $T_2[j_{t-k+1}]$ is not split by the alignment with $T_1[i_s]$. There are three subcases.

2.1 $(k = 0)$ *I.e.*, $F_2[j_{t-k+1}, j_t] = \theta$. Therefore,

$$D(F_1[i_1, i_s], F_2[j_1, j_t]) = D(F_1[i_1, i_{s-1}], F_2[j_1, j_t]) + D(T_1[i_s], \theta)$$

2.2 $(k = 1)$ *I.e.*, $F_2[j_{t-k+1}, j_t] = T_2[j_t]$. This is the same as in Case 1.

```
1. begin
2. Input F1[i_s, i_{m_i}] and F2[j_t, j_{n_j}].
3. D(F1[i_s, i_{s-1}], F2[j_t, j_{t-1}]) := 0.
4. for p := s to m_i
5.     D(F1[i_s, i_p], F2[j_t, j_{t-1}]) := D(F1[i_s, i_{p-1}], F2[j_t, j_{t-1}]) + D(T1[i_p], θ).
6. for q := t to n_j
7.     D(F1[i_s, i_{s-1}], F2[j_t, j_q]) := D(F1[i_s, i_{s-1}], F2[j_t, j_{q-1}]) + D(θ, T2[j_q]).
8. for p := s to m_i
9.     for q := t to n_j
10.        Compute D(F1[i_s, i_p], F2[j_t, j_q]) as in Lemma 5.3.
11. Output {D(F1[i_s, i_p], F2[j_t, j_q]) | s ≤ p ≤ m_i, t ≤ q ≤ n_j}.
12. end
```

Figure 5.1: **Procedure 5.1:** Computing $\{D(F_1[i_s, i_p], F_2[j_t, j_q]) | s \leq p \leq m_i, t \leq q \leq n_j\}$ for fixed s and t.

2.3 ($k \geq 2$) This is the most general case. It is easy to see that

$$D(F_1[i_1, i_s], F_2[j_1, j_t]) = \mu(l_1[i_s], \lambda) + \min_{1 \leq k < t} \{D(F_1[i_1, i_{s-1}], F_2[j_1, j_{k-1}]) + D(F_1[i_s], F_2[j_k, j_t])\}.$$

Case 3: the label is $(\lambda, l_2[j_t])$. Similar to Case 2. □

5.2.2 The Algorithm

It follows from the above discussion that, for each pair of subtrees $T_1[i]$ and $T_2[j]$, we have to compute $D(F_1[i], F_2[j_s, j_t])$ for all $1 \leq s \leq t \leq n_j$, and $D(F_1[i_s, i_t], F_2[j])$ for all $1 \leq s \leq t \leq m_i$. That is, we need to align $F_1[i]$ with each subforest of $F_2[j]$, and conversely to align $F_2[j]$ with each subforest of $F_1[i]$.

For each fixed s and t, where $1 \leq s \leq m_i$ and $1 \leq t \leq n_j$, the procedure in Figure 5.1 computes $\{D(F_1[i_s, i_p], F_2[j_t, j_q]) | s \leq p \leq m_i, t \leq q \leq n_j\}$, assuming that all $D(F_1[i_k], F_2[j_p, j_q])$ have already been computed, where $1 \leq k \leq m_i$ and $1 \leq p \leq q \leq n_j$, and all $D(F_1[i_p, i_q], F_2[j_k])$ are known, where $1 \leq p \leq q \leq m_i$ and $1 \leq k \leq n_j$.

```
1. begin
2. Input T_1 and T_2.
3. D(θ, θ) := 0.
4. for i := 1 to |T_1|
5  Initialize D(T_1[i], θ) and D(F_1[i], θ) as in Lemma 5.1.
6. for j := 1 to |T_2|
7.   Initialize D(θ, T_2[j]) and D(θ, F_2[j]) as in Lemma 5.1.
8. for i := 1 to |T_1|
9.   for j := 1 to |T_2|
10.       begin
11.         for s := 1 to m_i
12.             Call Procedure 5.1 on F_1[i_s, i_{m_i}] and F_2[j].
13.         for t := 1 to n_j
14.             Call Procedure 5.1 on F_1[i] and F_2[j_t, i_{n_j}].
15.         Compute D(T_1[i], T_2[j]) as in Lemma 5.2.
16.       end
17. Output: D(T_1[|T_1|], T_2[|T_2|]).
18. end
```

Figure 5.2: **Algorithm 5.1:** Computing $D(T_1, T_2)$.

We can obtain $D(F_1[i], F_2[j_s, j_t])$ for all $1 \leq s \leq t \leq n_j$ by calling Procedure 5.1 n_j times, and $D(F_1[i_s, i_t], F_2[j])$ for all $1 \leq s \leq t \leq m_i$ by calling Procedure 5.1 m_i times. Our algorithm to compute $D(T_1, T_2)$ is given in Figure 5.2.

5.2.3 The Time Complexity

For an input $F_1[i_s, i_{m_i}]$ and $F_2[j_t, j_{n_j}]$, the running time of Procedure 5.1 is bounded by

$$O((m_i - s) \cdot (n_j - t) \cdot (m_i - s + n_j - t)) = O(m_i \cdot n_j \cdot (m_i + n_j)).$$

So, for each pair i and j, Algorithm 5.1 spends $O(m_i \cdot n_j \cdot (m_i + n_j)^2)$ time. Therefore, the time complexity of Algorithm 5.1 is

$$\sum_{i=1}^{|T_1|} \sum_{j=1}^{|T_2|} O(m_i \cdot n_j \cdot (m_i + n_j)^2) \leq \sum_{i=1}^{|T_1|} \sum_{j=1}^{|T_2|} O(m_i \cdot n_j \cdot (\deg(T_1) + \deg(T_2))^2)$$

$$\leq O((\deg(T_1)+\deg(T_2))^2 \cdot \sum_{i=1}^{|T_1|} m_i \cdot \sum_{j=1}^{|T_2|} n_j)$$
$$\leq O(|T_1| \cdot |T_2| \cdot (\deg(T_1)+\deg(T_2))^2).$$

If both T_1 and T_2 have degrees bounded by some constant, the time complexity becomes $O(|T_1| \cdot |T_2|)$. Note that the algorithm actually computes $D(T_1[i], T_2[j])$, $D(F_1[i], F_2[j])$, $D(F_1[i_s, i_t], F_2[j])$ and $D(F_1[i], F_2[j_s, j_t])$. With these data, an actual optimal alignment can be easily found using a simple back-tracking technique. The complexity will remain the same.

Whether the above complexity can be improved is a rather hard question. A direct approach would be to prove that the alignment distance between two forests satisfies either quadrangle or inverse quadrangle inequality. If this is true, then one can reduce the complexity for computing $D(F_1[i], F_2[j_s, j_t])$ and $D(F_1[i_s, i_t], F_2[j])$, using a matrix search technique. Unfortunately, we can show that neither of these inequalities hold.

5.3 The Alignment of Unordered Trees

Here we consider unordered labeled trees, *i.e.*, the order among the siblings is insignificant. It is known that computing the edit distance for unordered trees is MAX SNP-hard even if both trees have bounded degrees [88]. We will show that aligning unordered trees with bounded degrees can be done in polynomial time, and give a simple algorithm to align unordered binary trees. Finally, we prove that aligning unordered trees becomes NP-hard if one of the input trees can have an arbitrary degree.

1. **begin**
2. Input T_1 and T_2
3. **for** $i := 1$ **to** $|T_1|$
4. **for** $j := 1$ **to** $|T_2|$
5. $$D(F_1[i], F_2[j]) := \min\{ \begin{array}{l} \mu(l_1[i_2], \lambda) + D(F_1[i_2], F_2[j]) + D(T_1[i_1], \theta), \\ \mu(l_1[i_1], \lambda) + D(F_1[i_1], F_2[j]) + D(T_1[i_2], \theta), \\ \mu(\lambda, l_2[j_2]) + D(F_1[i], F_2[j_2]) + D(\theta, T_2[j_1]), \\ \mu(\lambda, l_2[j_1]) + D(F_1[i], F_2[j_1]) + D(\theta, T_2[j_2]), \\ D(T_1[i_1], T_2[j_1]) + D(T_1[i_2], T_2[j_2]), \\ D(T_1[i_1], T_2[j_2]) + D(T_1[i_2], T_2[j_1]) \}. \end{array}$$
6. $$D(T_1[i], T_2[j]) := \min\{ \begin{array}{l} \mu(l_1[i], l_2[j]) + D[F_1[i], F_2[j]), \\ \mu(l_1[i], \lambda) + D(T_1[i_1], T_2[j]) + D(T_1[i_2], \theta), \\ \mu(l_1[i], \lambda) + D(T_1[i_2], T_2[j]) + D(T_1[i_1], \theta), \\ \mu(\lambda, l_2[j]) + D(T_1[i], T_2[j_1]) + D(\theta, T_2[j_2]), \\ \mu(\lambda, l_2[j]) + D(T_1[i], T_2[j_2]) + D(\theta, T_2[j_1]) \}. \end{array}$$
7. Output $D(T_1[|T_1|], T_2[|T_2|])$.
8. **end**

Figure 5.3: **Algorithm 5.2:** Aligning unordered binary trees.

5.3.1 Unordered Trees with Bounded Degrees

When the degrees are bounded, we can compute the alignment distance using a modified version of Algorithm 5.1. Lemmas 5.1 and 5.2 still work. The only difference is in the computation of $D(F_1[i], F_2[j])$. We have to revise the recurrence relation in Lemma 5.3 as follows: for each (forest) $A \subseteq \{T_1[i_1], \ldots, T_1[i_{m_i}]\}$ and each (forest) $B \subseteq \{T_2[j_1], \ldots, T_2[j_{n_j}]\}$,

$$D(A,B) = \min \begin{cases} \min_{T_1[i_p] \in A, T_2[j_q] \in B} D(A - \{T_1[i_p]\}, B - \{T_2[j_q]\}) + D(T_1[i_p], T_2[j_q]), \\ \min_{T_1[i_p] \in A, B' \subseteq B} D(A - \{T_1[i_p]\}, B - B') + D(F_1[i_p], B') + \mu(l_1[i_p], \lambda), \\ \min_{A' \subseteq A, T_2[i_q] \in B} D(A - A', B - \{T_2[j_q]\}) + D(A', F_2[j_q]) + \mu(\lambda, l_2[j_q]). \end{cases}$$

Since m_i and n_j are bounded, $D(A,B)$ can be computed in polynomial time. If T_1 and T_2 are both in fact binary trees, the algorithm can be much simplified, as shown in Figure 5.3. It is easy to see that the time complexity of this algorithm is $O(|T_1| \cdot |T_2|)$.

5.3.2 The Hardness of Aligning Unordered Trees

Theorem 5.4 *Computing the alignment distance between ordered trees is NP-hard if one of the trees can have an arbitrary degree.*

Proof: The reduction is from exact cover by 3-sets (X3C), which is NP-hard [28].

INSTANCE: A set $A = \{a_1, \ldots a_m\}$ with $m = 3q$ and a collection C of 3-element subsets of A.

QUESTION: Does C contain an exact cover of A, *i.e.*, a subcollection $C' \subseteq C$ such that every element of A occurs in exactly one member of C'?

Let $A = \{a_1, \ldots, a_m\}$, and $C = \{c_1, \ldots c_n\}$, where $c_i = \{c_{i,1}, c_{i,2}, c_{i,3}\}$ and $c_{i,j} \in A$, be an instance of X3C. We construct two trees as in Figure 5.4. The alphabet of labels is $A \cup \{r, s, p\}$, assuming letters r, s, p are not contained in A. Let $\mu(a, b) = 0$ if $a = b$ or 2 if $a, b \neq \lambda$ and $a \neq b$, $\mu(a, \lambda) = 1$, and $\mu(\lambda, b) = 1$. The degree of T_1 is bounded by 3. Each $T_{1,i}$ is a subtree, which corresponds to the subset c_i in C. The sequence of nodes with label s in each $T_{1,i}$ is called the *upper segment* and the three branches are called the *lower segment.* The root of T_2 has $n + m$ children.

Now, we want to show that C contains an exact cover of A if and only if the alignment distance between T_1 and T_2 is $(n-2)+5q+6(n-q)$. Observe that for each subtree $T_{1,i}$, either its entire upper segment or its entire lower segment is "matched" with the corresponding parts of T_2 in an optimal alignment. Matching the upper segment saves a cost of 5, whereas matching the lower segment saves a cost of 6. Thus, an optimal alignment matches as many lower segments as possible.

If C contains an exact cover of A, say, C', then we can align the two trees such that

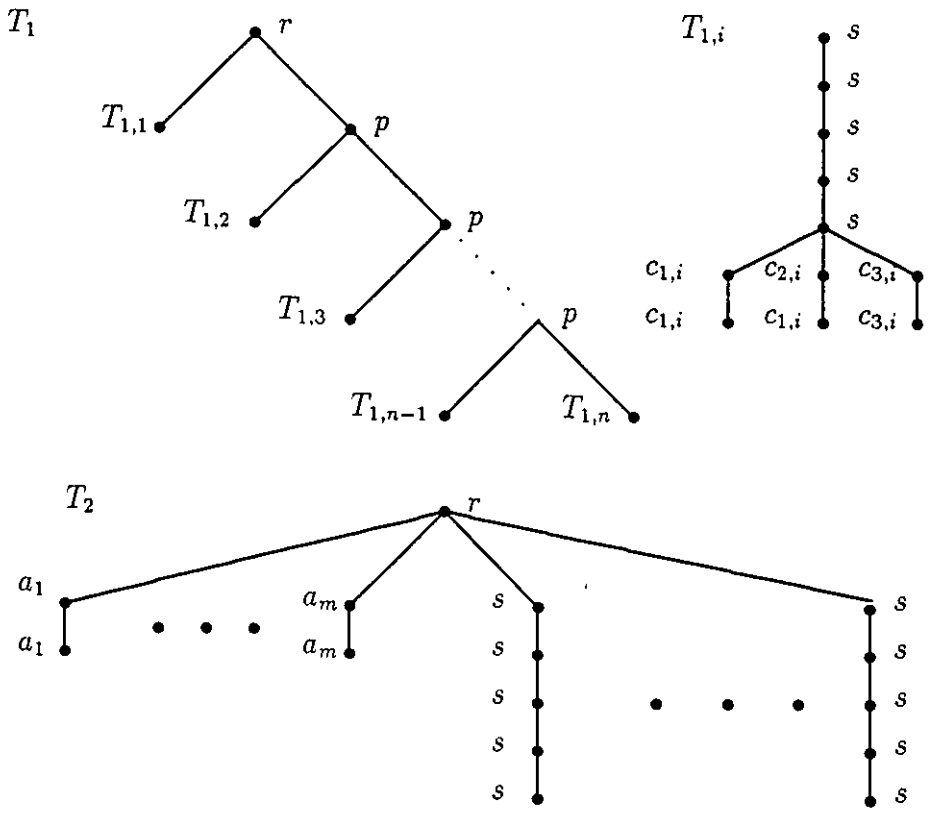


Figure 5.4: The reduction.

the q lower segments given by C' are all matched. This yields an alignment with cost $(n-2)+5q+6(n-q)$, since the q $T_{1,i}$'s, where $c_i \in C'$, contribute a cost of 5 each, the rest $n-q$ $T_{1,i}$'s contribute a cost of 6 each, and $n-2$ p's contribute a cost of 1 each. Conversely, if the alignment distance between T_1 and T_2 is $(n-2)+5q+6(n-q)$, there must exist q $T_{1,i}$'s each contributing a cost of 5. These $T_{1,i}$'s induce an exact cover of A. □

5.4 Multiple Alignment of Ordered Trees

For the purpose of finding highly conserved structural motifs, the comparison of multiple labeled ordered trees is required [54, 72]. The alignment of two trees can be easily extended to multiple trees. Many existing formulations of multiple sequence alignment are also applicable to multiple alignment of trees. For example, one can define the SP-score of a multiple alignment $\mathcal{A}$ of trees as the sum of the values of pairwise alignments induced by $\mathcal{A}$. (Here, SP stands for sum of all pairs.)

Since multiple sequence alignment with SP-score is NP-hard (See Section 3.2.), clearly multiple alignment of ordered trees with SP-score is also NP-hard. So it would be of interest to investigate approximation algorithms for multiple alignment of ordered trees. It is known that the *center star method* approximates a multiple sequence alignment with ratio 2[31]. The method first finds a sequence (the center) X_c among the k given sequences, such that

$$\sum_{j \neq c} D(X_c, X_j) \leq \sum_{j \neq i} D(X_i, X_j),$$

for any $1 \leq i \leq k$. Every sequence is then optimally aligned with the center X_c. Finally, a multiple alignment $\mathcal{A}_c$ of all the given sequences consistent with the pairwise alignments between X_c and the other sequences is derived. The value of $\mathcal{A}_c$ is at most twice the optimum [31]. Unfortunately, the construction does not work for trees since such a multiple

alignment $\mathcal{A}_c$ consistent with all pairwise alignments cannot be found in general for trees.

5.5 Conclusion

It is interesting to observe that the *tree inclusion* problem defined in [51] is actually a special case of alignment of trees. Again, let $\mu(a,b) = 0$ if $a = b$ or 2 if $a, b \neq \lambda$ and $a \neq b$, $\mu(a,\lambda) = 1$, and $\mu(\lambda,b) = 1$. Then T_1 is included in T_2 if and only if $D(T_1,T_2) = |T_2| - |T_1|$. The complexity of Algorithm 5.1 is just slightly higher than that for ordered tree inclusion [51]. Also, under the above cost/score scheme, an optimal edit from T_1 to T_2 yields a largest common sub-tree of T_1 and T_2, and an optimal alignment yields a smallest common super-tree [88]. So, edit and alignment form a "dual" in this sense.

Chapter 6

The Complexity of Comparing Evolutionary Trees

6.1 Introduction

In the analysis of molecular evolution, the evolutionary history of a set of species is described by an *evolutionary tree*. (See Section 2.3 for the definition.) Reconstructing the correct evolutionary tree for a set of species is one of the fundamental yet difficult problems in evolutionary genetics. Many methods have been proposed based on various criteria. However, these methods do not always produce the same answer. Therefore, it is interesting to design measures and automatic methods for the comparison of different evolutionary trees on the same set of species. A fruitful approach is to compute a tree that can somehow express the "intersection" of these evolutionary trees.

The notion of a *maximum agreement subtree* (MAST) was first proposed by Finden and Gordon [23]. Given an evolutionary tree T on set S and a subset $A \subseteq S$, the *restriction*

of T on A, denoted $T|A$, is an evolutionary tree on set A obtained from T by eliminating the species outside A and the internal nodes with only single child. The latter operation, called *forced contraction*, is illustrated in Figure 6.1(a). For any two evolutionary trees T_1 and T_2 on set S, an *agreement subtree* (AST) of T_1 and T_2 is a tree T such that for some $A \subseteq S$, $T = T_1|A = T_2|A$. We call $A \subseteq S$ the set of the *agreed* species and $S - A$ the set of the *disagreed* species. A maximum agreement subtree (MAST) of T_1 and T_2 is an AST with the largest number of leaves (*i.e.* the largest number of species have been agreed upon). The notion of AST and MAST can be easily extended to more than two evolutionary trees on the same set of species [2].

The first polynomial-time algorithm for MAST on two trees was given by Steel and Warnow [75]. Their algorithm runs in $O(n^2)$ time for bounded-degree trees and $O(n^{4.5} \log n)$ for unbounded-degree trees, where n is number of species. Farach and Thorup recently improved the running time to $O(n^{1.5} \log n)$ for unbounded-degree trees and to $O(nc^{\sqrt{\log n}})$ for bounded-degree trees [20, 21]. Amir and Keselman [2] considered MAST for several trees. They showed that MAST is polynomial-time solvable for multiple bounded-degree trees and is NP-hard for three trees with unbounded degrees. An algorithm to approximate the *complement* of MAST on multiple unbounded-degree trees with ratio 4 was given. (The complement is to minimize the number of disagreed species instead of the agreed species). They also raised two questions: the approximability of MAST on multiple unbounded-degree trees and, given two trees T_1 and T_2 on set S, how to compute a tree with the largest number of *edges* which is obtainable through a sequence of *edge contractions* from both restrictions $T_1|A$ and $T_2|A$ for some subset $A \subseteq S$. An edge contraction is shown in Figure 6.1(b) and is also referred to as *deletion of an internal node* in tree edit [86]. Let's call the second problem maximum agreement subtree with edge contractions (MAST-EC). Here we settle these two problems by showing that (i) MAST for three unbounded-degree trees cannot

be approximated within a constant ratio unless $P = NP$ and that this problem cannot be approximated within ratio n^{ϵ}, for some $0 < \epsilon \leq 1$, unless $NP \subseteq DTIME(2^{polylogn})$. (ii) MAST-EC is NP-hard.

The *refinement* of trees is another approach towards the "intersection" of trees, and was originally introduced in the study of the *compatibility* of evolutionary trees [17, 30, 82]. Tree T is said to be a refinement of trees T_1 and T_2 if both T_1 and T_2 can be derived from T through a sequence of edge contractions. Two trees are *compatible* if they have a refinement. Polynomial-time algorithms for the tree compatibility problem have been known for a long time (*e.g.* [30, 82]). It is natural to consider the optimization version of this problem for trees which are not compatible with each other, namely, given trees T_1 and T_2 on set S, find the largest subset $A \subseteq S$ such that $T_1|A$ and $T_2|A$ are compatible [83]. Let's call a refinement of $T_1|A$ and $T_2|A$ a *maximum refinement subtree* (MRST) of T_1 and T_2 and this problem the MRST problem. One can view MRST as a natural counterpart of MAST. We show that MRST can be solved in polynomial time if T_1 and T_2 have degrees bounded by some constant and it becomes NP-hard if one of the trees is allowed to have an arbitrary degree.

When recombination of DNA sequences occurs in an evolution, the history of the evolution cannot be adequately described by a single tree. A recent proposal in attempt to solve this problem is to use a list of evolutionary trees [34, 35]. Each tree corresponds to a *region* of the DNA sequences, and each tree can be obtained from the preceding tree on the list by transferring some subtrees from one place to another. Figure 6.1(c) shows a subtree-transfer operation, which moves T_2. Each such operation corresponds to a recombination event. A model for reconstructing the list of trees based on parsimony has been proposed in [34, 35]. The model requires the calculation of the subtree-transfer distance between

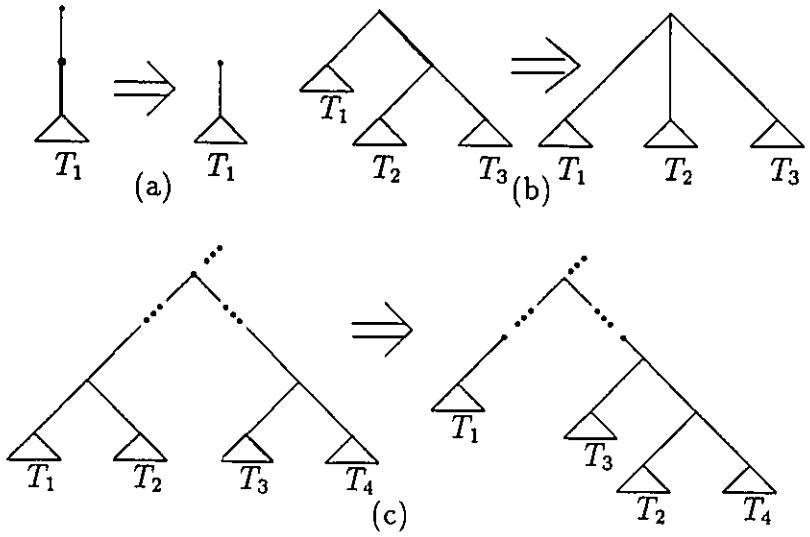


Figure 6.1: The operations.

two trees (*i.e.* the minimum number of subtrees we need to transfer). It was left open how to compute this distance. Unfortunately, we can show that computing the distance is NP-hard. We will also give a simple approximation algorithm achieving ratio 3. It turns out that this distance is also connected to the notion of agreement between trees.

The non-approximability of MAST on multiple unbounded-degree trees is given in Section 6.2. Sections 6.3 and 6.4 discuss the complexity of MAST-EC and MRST. Finally, the subtree-transfer distance is considered in Section 6.5.

6.2 Non-approximability of MAST on 3 Unbounded-Degree Trees

In this section, we show that MAST for three trees of unbounded-degrees cannot be approximated within any constant ratio in polynomial time unless $P = NP$, and that there is a

constant $0 < \epsilon \leq 1$ such that MAST for three trees of unbounded-degrees cannot be approximated within ratio n^ϵ in polynomial time, unless NP is contained in $DTIME(2^{polylogn})$.

INSTANCE: Three trees T_1, T_2 and T_3 of arbitrary degrees on set $S = \{s_1, s_2, \ldots s_n\}$.

QUESTION: Find a largest subset $A \subseteq S$ such that $T_1|A = T_2|A = T_3|A$.

The idea of the proof is the self-improvement technique as used in [45]. We first prove that the problem is MAX SNP-hard. Thus it cannot be approximated within ratio $1 + \epsilon$ for some $\epsilon > 0$ unless P = NP [3]. Then we define a product of trees and show that any approximation ratio r for the problem can be improved to $r^{1/k}$. Taking an appropriate k should give the desired bound.

Lemma 6.1 *The above problem is MAX SNP-hard.*

Proof. It can be easily verified that Amir and Keselman's construction for the NP-hardness [2] is in fact an L-reduction [60]. For the completeness, we include the construction here. The reduction is from the following variant of 3-Dimensional Matching which is MAX SNP-hard [48].

INSTANCE: A set $M \subseteq W \times X \times Y$ of ordered triples where W, X and Y are disjoint. Each element of $W \cup X \cup Y$ appears in at most B triples of M.

QUESTION: Find the largest subset $M' \subseteq M$ such that no two elements of M' agree in any coordinate.

Given an instance of 3DM-B, $M \subseteq W \times X \times Y$, we construct three evolutionary trees T_1, T_2 and T_3. Let $|W| = |X| = |Y| = q$. Each T_i has $2q + 1$ children. The first $q + 1$ children are leaves labeled with new symbols $a_1, a_2, \ldots, a_{q+1}$. Each of the last q children of T_1 corresponds to an element $w \in W$, and has as its children leaves labeled with the triples

of the form (w, x, y), where $x \in X$ and $y \in Y$. T_2 and T_3 are constructed in a similar way. It is easy to see that there is a 3-dimensional matching of size k if and only if there is an agreement subtree of size $k + q + 1$, and that this reduction is actually an L-reduction. ■

Now, we need to define the *product* of two evolutionary trees. Let T_1 and T_2 be two evolutionary trees on sets S_1 and S_2 respectively, where S_1 and S_2 are the sets of labels for these two trees. For each label $s \in S_1$, let $T_{2,s}$ denote the tree obtained from T_2 by replacing each label $s' \in S_2$ with a new label (s, s'). The product of T_1 and T_2, denoted $T_1 \times T_2$, is obtained from T_1 by replacing each leaf labeled s with the tree $T_{2,s}$. For any tree T, we define $T^2 = T \times T$ and $T^{k+1} = T \times T^k$. The following lemma allows us to improve an approximation ratio for MAST by taking the product.

Lemma 6.2 *Let T_1, T_2 and T_3 be the three evolutionary trees and $c(T_1, T_2, T_3)$ denote the size of an MAST for T_1, T_2, T_3. Then $c(T_1^{k+1}, T_2^{k+1}, T_3^{k+1}) = c(T_1, T_2, T_3) \cdot c(T_1^k, T_2^k, T_3^k)$. Moreover, given an AST of size c for $T_1^{k+1}, T_2^{k+1}, T_3^{k+1}$, we can find in polynomial time an AST of size c_1 for T_1, T_2, T_3 and an AST of size c_2 for T_1^k, T_2^k, T_3^k such that $c_1 \cdot c_2 \geq c$.*

Proof. We give a proof for $k = 2$. The general case follows from the same argument. Let S be the set of labels in T_1, T_2, T_3. Obviously, $c(T_1^2, T_2^2, T_3^2) \geq c(T_1, T_2, T_3) \cdot c(T_1, T_2, T_3)$. Suppose that we are given an AST T of size c for T_1^2, T_2^2, T_3^2. For each label $s \in S$ such that (s', s) appears in T for some $s' \in S$, we can identify an agreement subtree of $T_{1,s}$, $T_{2,s}$, and $T_{3,s}$ in T. Let c_1 be the number of such subtrees. Without loss of generality, assume that all such subtrees have the same size c_2 (otherwise we can improve c). Then, we have $c_1 \cdot c_2 = c$. Moreover, every such subtrees gives an AST of T_1, T_2, T_3 of size c_2, and replacing each subtree in T with a single leaf labeled with an appropriate element of S also results in an AST of T_1, T_2, T_3 of size c_1. This completes the proof. ■

Note that, if the size of the tree T_i $(i = 1, 2, 3)$ constructed in Lemma 6.1 is n, then the size of T_i^k is $O(n^k)$. Now, we give the main result in this section.

Theorem 6.3 *(i) MAST for three trees of unbounded-degrees can not be approximated within any constant ratio unless $P = NP$. (ii) There is a constant $0 < \epsilon \leq 1$ such that MAST for three trees of unbounded-degrees cannot be approximated within ratio n^ϵ in polynomial time, unless NP is contained in $DTIME(2^{polylogn})$.*

Proof. The argument is the same as in [45]. We only prove (i). (i) The idea is to show that if a constant ratio approximation algorithm exists, then PTAS exists. Suppose that MAST for three unbounded-degree trees has a polynomial time approximation algorithm with performance ratio c. For any given $\epsilon > 0$, let $k = log_{1+\epsilon}c$. Then by Lemma 6.2, we have an approximation algorithm for MAST for three unbounded-degree tree with ratio $c^{\frac{1}{k}} \leq 1 + \epsilon$. The algorithm runs in time n^k, thus polynomial in n. This implies a PTAS for MAST for three unbounded-degree tree. ■

6.3 Agreement Subtrees with Edge Contractions

A natural extension of the agreement subtree approach is to allow the application of edge contractions in the formation of an "agreement" of the given trees. Intuitively, an edge contraction "loosens" the structure of a tree and thus increases the chance of having an agreement. For example, in the extreme case, we can contract all the edges in all trees to end up with a star. However, the obtained star contains little information about the evolutionary history. Therefore, it is desirable to contract a small number of edges yet to have a large number of the agreed species. This is the intuition behind the MAST-EC problem first proposed in [2]. Here, we show that MAST-EC on bounded-degree trees is

NP-hard by a reduction from a restricted version of Exact Cover by 3-Sets given below [28].

INSTANCE: A collection C of subsets of a finite set S where every $c \in C$ contains three elements and every element $s \in S$ is contained in at most 3 subsets in C.

QUESTION: Find an exact covering $C' \subseteq C$ of S, *i.e.* a collection of mutually disjoint sets whose union equals S.

Theorem 6.4 *MAST-EC is NP-hard even if the given tree have bounded degree.*

Proof. Given an instance of Exact Cover by 3-Sets, let the set $S = \{s_1, s_2, \ldots, s_m\}$, where $m = 3q$ and $C = \{C_1, C_2, \ldots, C_n\}$, where each $C_i = \{t_{i,1}, t_{i,2}, t_{i,3}\}$, $t_{i,j} \in S$. Without loss of generality, we assume that $n > q$.

Two trees T and $\hat{T}$ are constructed as in Figure 6.2 and Figure 6.3. The top part is a binary tree whose actual structure is insignificant. (We need this part to get around the degree bound.) In order to make this part insignificant in the following calculation, introduce a large enough factor $f = 2(n + m)$. Each element $c_{i,j}$, $j = 1, 2, 3$, of C_i corresponds to a subtree $T_{i,j}$ as shown in Figure 6.4. In tree T, each element $s_i \in S$ corresponds to a subtree T_i as shown in Figure 6.5. Every triple of subtrees $T_{i,j}$ ($j = 1, 2, 3.$) is connected to the top part by a path of length $5f$, which has $5f$ internal nodes (not including the root of the subtree) and $10f$ edges (including the edges connecting $x_{i,j}$'s.). Call such a path a long chain, denoted by P_i. In tree $\hat{T}$, there are n corresponding long chains each of which contains $5f - 1$ internal nodes, and $10f - 2$ edges. (See Figure 6.3.) We will show that C has an exact cover of S if and only if there is a tree T' with at least $3(2f-2)q + (10f-2)(n-q) + 5fq$ edges such that, for some subset A of labels, T' can be obtained using edge contractions from both restrictions $T|A$ and $\hat{T}|A$.

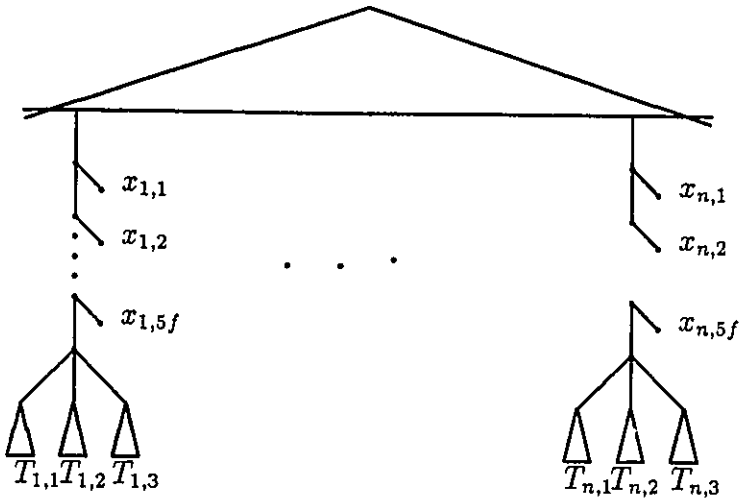


Figure 6.2: The tree T constructed from $C = \{C_1, C_2, \ldots, C_n\}$, where each subtree $T_{i,j}$ corresponds to a $c_{i,j} \in C_i$, $j = 1, 2, 3$.

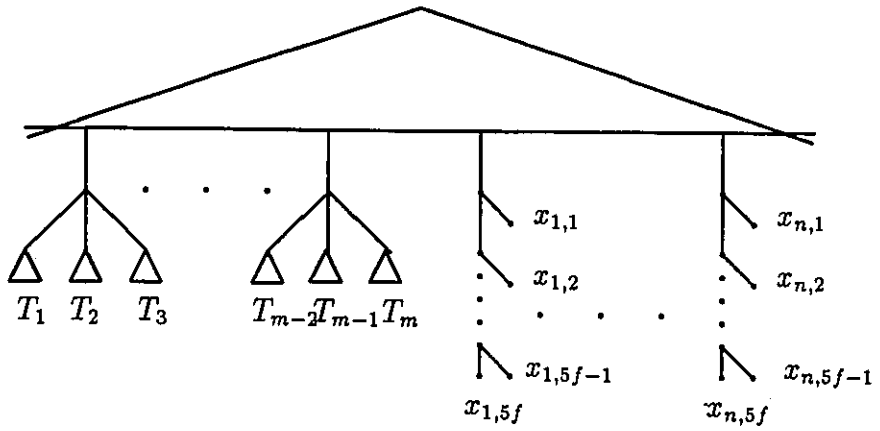


Figure 6.3: The tree $\hat{T}$. Each subtree T_i corresponds to an element $s_i \in S$ and each of the n chains of length $5f$ corresponds to a subset $C_i \in C$.

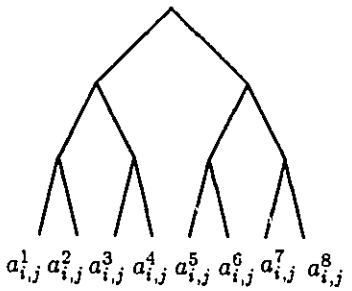


Figure 6.4: The subtree $T_{i,j}$ corresponding to $c_{i,j} \in C_i$ ($f = 8$ in this case).

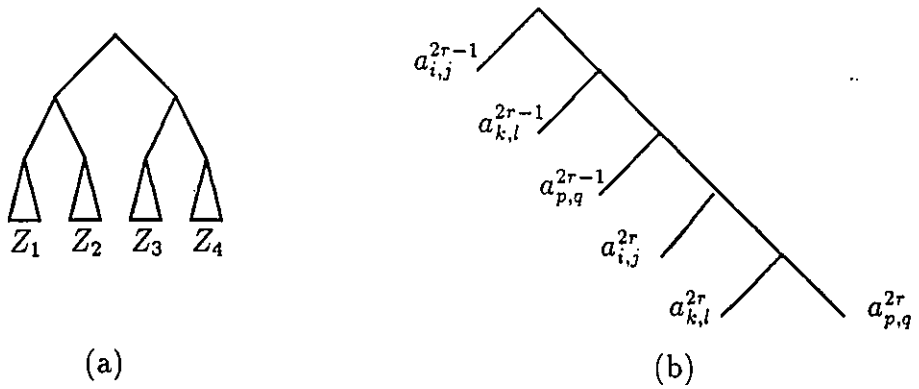


Figure 6.5: Suppose that $c_{i,j}$, $c_{k,l}$ and $c_{p,q}$ are the three occurrences of s_i. (a) The subtree T_i corresponding to s_i. (b) The subtree Z_r. (Again, $f = 8$.)

(if) Given an exact cover $\{C_{i_1}, \ldots C_{i_q}\} \subseteq C$ of S, there is a corresponding set of subtrees $\{T_{i_l,j} \mid 1 \leq l \leq q,\ 1 \leq j \leq 3\}$ in T. We can obtain an agreement subtree with edge contraction (AST-EC) T' which contains the subtrees $\{T_{i_l,j} \mid 1 \leq l \leq q,\ 1 \leq j \leq 3\}$ and the long chains P_i $(i \in \{1,2,\ldots,n\} - \{i_1,\ i_2,\ \ldots i_q\})$. Note that, each $T_{i_l,j}$ contributes $2f-2$ edges, and each P_i $(i \in \{1,2,\ldots,n\} - \{i_1,\ i_2,\ \ldots i_q\})$ contributes $10f-2$ edges. Moreover, we eliminate the internal nodes of the q long chains $P_{i_1}, \ldots, P_{i_q}$, each of which contains $5f$ labels and each label contributes one edge in the AST-EC T'. Thus, T' has at least $3(2f-2)q + (10f-2)(n-q) + 5fq$ edges.

(only if) Suppose we have an AST-EC T' of size at least $3(2f-2)q+(10f-2)(n-q)+5fq$. Note that, the constructions of $T_{i,j}$'s in T and T_i's in $\hat{T}$ imply that without decreasing the number of edges, we can modify T' such that if a label in a $T_{i,j}$ appears in T', then the whole subtree $T_{i,j}$ appears in T'. Furthermore, in order to keep $T_{i,j}$ in T', we have to eliminate $5f$ internal nodes in the long chain P_i. Thus, we can assume that once $T_{i,j}$ is in T', then the other two subtrees $T_{i,k}$ $(k = \{1,2,3\} - \{j\})$ are also in T'. Finally, the $5f \cdot n$ labels $x_{i,j}$ $(i = 1,\ldots,n, j = 1,\ldots,5f)$ contribute $5f \cdot n$ edges. Preserving a chain contributes another $5f$ edges, and preserving a triple of subtrees $T_{i,j}$ $(j = 1,2,3)$ contributes $3(2f-2)$ edges. Moreover, we cannot keep both P_i and the triple $T_{i,j}$ $(j = 1,2,3)$ in T' simultaneously. Thus, in order to obtaining $3(2f-2)q + 5fq + 10f(n-q)$ edges, we have to preserve q triples in T', which should give an exact cover of S. ■

A variant of MAST-EC is to construct a tree maximizing the the number of internal nodes instead of edges. Unfortunately, a simple modification of the above proves that this variant is also NP-hard.

6.4 Maximum Refinement Subtrees

The MRST problem can be reformulated as *alignment of unordered trees* described in Chapter 5. Every internal node is labeled null and each inserted node is also labeled null here. To formulate MRST as an alignment of trees problem, we need define the scores as follows: 1 for an identical pair of (non-null) labels and 0 otherwise. It is easy to see that in any alignment $\mathcal{A}$ of T_1 and T_2, the set of nodes with an identical pair of opposing labels induces a refinement subtree T of T_1 and T_2. The value of $\mathcal{A}$ is exactly the subset of labels appearing in T. Therefore, an optimal alignment of T_1 and T_2 gives an MRST of T_1 and T_2.

In Section 5.3, a polynomial-time algorithm to align unordered trees with bounded degree has been given. We hence have the following theorem.

Theorem 6.5 *MRST can be computed in polynomial time if both trees are of bounded degree.*

On the other hand, it is shown in Section 5.3 that alignment of unordered trees is NP-hard when one of the trees is allowed to have an arbitrary degree. A slight modification of the proof of this result gives the next theorem.

Theorem 6.6 *MRST is NP-hard if one of the given trees can have an arbitrary degree.*

6.5 The Subtree-Transfer Distance

Before we prove the results, it is again convenient to reformulate the problem. Let T_1 and T_2 be two evolutionary trees on set S. An *agreement forest* of T_1 and T_2 is any forest which can be obtained from both T_1 and T_2 by cutting k edges (in each tree) for some k and

applying forced contractions in each resulting component trees. Define the size of a forest as the number of components it contains. Then the *maximum agreement forest* (MAF) problem is to find an agreement forest with the *smallest* size. The following lemma shows that MAF is really equivalent to computing the subtree-transfer distance.

Lemma 6.7 *The size of an MAF of T_1 and T_2 is one more than their subtree-transfer distance.*

The lemma can be proved by a simple induction on the number of leaves. Intuitively, the lemma says that the transfer operations can be broken down into two stages: first we cut off the subtrees to be transferred from the rest in T_1 (not worrying where to put them), then we assemble them appropriately to obtain T_2. This separation will simplify the proofs.

6.5.1 The NP-hardness

Theorem 6.8 *It is NP-hard to compute the subtree-transfer distance between two binary trees.*

Proof. The reduction is again from Exact Cover by 3-Sets. Let $S = \{s_1, s_2, \ldots s_m\}$ be a set and $C_1, \ldots, C_n$ be an instance of this problem. Assume $m = 3q$.

The tree T_1 is formed by inserting n subtrees $A_1, \ldots, A_n$ into a chain containing $2n+2m$ leaves $x_1, \ldots, x_{2n}, y_1, \ldots, y_{2m}$ uniformly. (See Figure 6.6(a).) Each A_i corresponds to $C_i = \{c_{i,1}, c_{i,2}, c_{i,3}\}$, and has 9 leaves as shown in Figure 6.6(b). Suppose that $c_{j,j'}$, $c_{k,k'}$ and $c_{l,l'}$ are the three occurrences of an $s_i \in S$ in C. Then in T_2, we have a subtree B_i as shown in Figure 6.7(a). For each C_i, we also have a subtree D_i in T_2 as shown in Figure 6.7(b). The subtrees are arranged as a linear chain as shown in Figure 6.7(c).

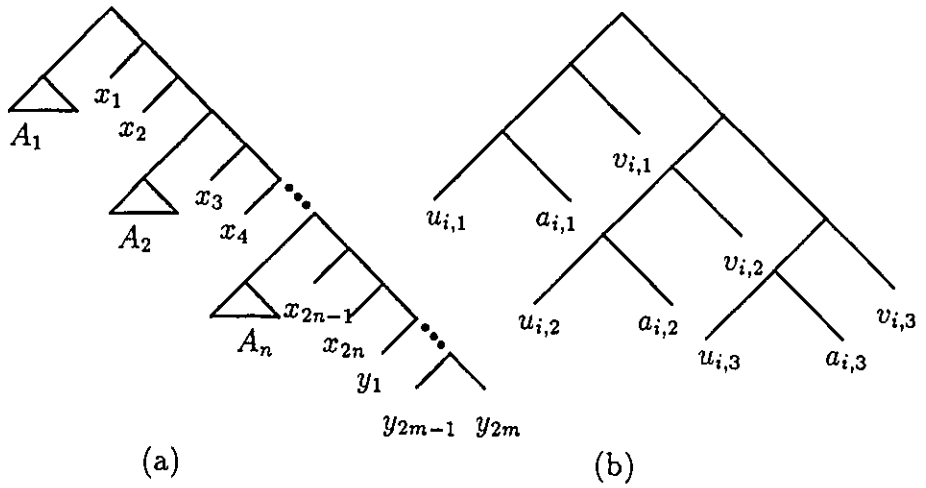


Figure 6.6: (a) The tree T_1. (b) The subtree A_i.

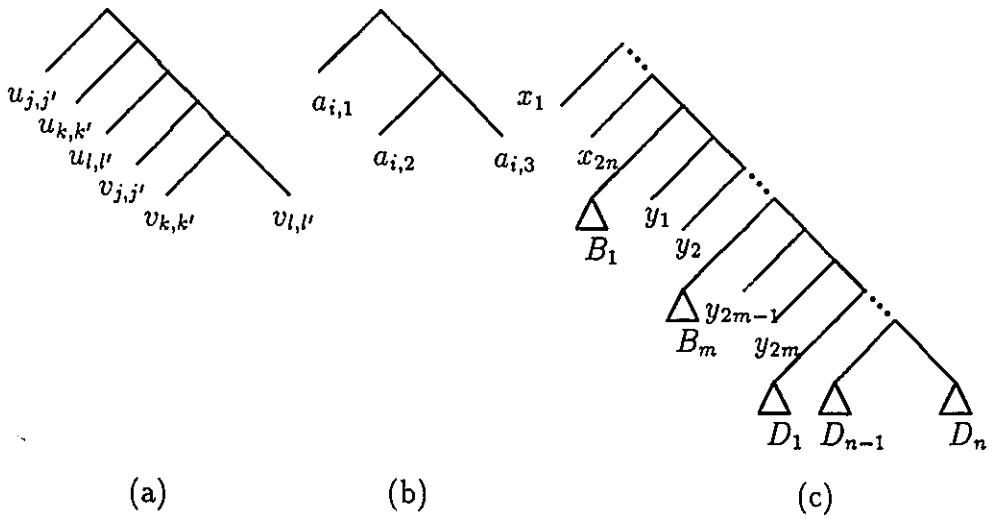


Figure 6.7: (a) The subtree B_i. (b) The subtree D_i. (c) The tree T_2.

Note that, each adjacent pair of subtrees A_i and A_{i+1} in T_1 is separated by a chain of length 2 which also appears in T_2. Thus, to form an MAF of T_1 and T_2, our best strategy is clearly to cut off $A_1, A_2, \ldots, A_n$ in T_1 and similarly cut off $B_1, B_2, \ldots, B_m$ in T_2. This then forces us to cut off $D_1, D_2, \ldots, D_n$ in T_2. Now in each A_i, we can either cut off the leaves $u_{i,1}, v_{i,1}, u_{i,2}, v_{i,2}, u_{i,3}, v_{i,3}$ to form a subtree containing three leaves $a_{i,1}, a_{i,2}, a_{i,3}$ (yielding $6 + 1 = 7$ components totally), or we can cut off $a_{i,1}$, $a_{i,2}$, and $a_{i,3}$. In the second case, we will be forced to also cut links between the three subtrees containing leaves $\{u_{i,1}, v_{i,1}\}$, $\{u_{i,2}, v_{i,2}\}$ and $\{u_{i,3}, v_{i,3}\}$ respectively, as the B_i's are already separated. Hence in this case the best we can hope for is $3 + 3 = 6$ components (if we can keep all three 2-leaf subtrees in the agreement forest).

We now formally show that C has an exact cover of S if and only if T_1 and T_2 have an agreement forest of size $1 + 6q + 7(n - q) = 7n - q + 1$.

(if) Suppose we are given an exact cover $\{C_{i_1}, C_{i_2}, \ldots, C_{i_q}\} \subseteq C$ of S. For each A_{i_l} $(l = 1, 2, \ldots q)$ in T_1, we cut off $a_{i_l,1}$, $a_{i_l,2}$, $a_{i_l,3}$, and the three subtrees containing leaves $\{u_{i_l,1}, v_{i_l,1}\}$, $\{u_{i_l,2}, v_{i_l,2}\}$ and $\{u_{i_l,3}, v_{i_l,3}\}$. Correspondingly, we can obtain the 6 components from T_2. Note that, each B_i in T_2 can contribute at most one subtree containing $\{u_{i_l,j}, v_{i_l,j}\}$ $(j = 1, 2, 3)$. In this way, each A_{i_l} creates 6 components in the agreement forest. For the rest of $n - q$ subtrees A_i $(i \in \{1, 2, \ldots n\} - \{1_1, \ldots i_q\})$, we can cut off the leaves $u_{i,1}, v_{i,1}, u_{i,2}, v_{i,2}, u_{i,3}, v_{i,3}$ and the subtree containing three leaves $a_{i,1}, a_{i,2}, a_{i,3}$. Correspondingly, we can obtain the 7 components from T_2. Therefore, the total number of components in the agreement forest is $1 + 6q + 7(n - q) = 7n - q + 1$.

(only if) Suppose we have an agreement forest of size $1 + 6q + 7(n - q) = 7n - q + 1$. From the previous discussion, we know that

1. Each A_i in T_1 contributes at least 6 components.

2. Among the n subtrees A_i's, at most q A_i's can contribute 6 components. Moreover, these A_i's correspond to disjoint C_i's in C.

Thus, if there is an agreement forest of size $7n - q + 1$, then there is an exact cover of S. ■

6.5.2 An Approximation Algorithm of Ratio 3

Our basic idea is to deal with a pair of sibling leaves a, b in the first tree T_1 at a time. If the pair a and b are siblings in the second tree T_2, we replace this pair with a new leaf labeled by (a, b) in both trees. Otherwise, we will cut T_2 until a and b become siblings or separated. Eventually both trees will be cut into the same forest. Five cases need be considered. Figure 6.8 illustrate the first four cases. The last case (Case (v)) is that a and b are also siblings in T_2.

The approximation algorithm is given in Figure 6.9. The variable N records the number of components (or the number of cuts plus 1).

Theorem 6.9 *The approximation ratio of the above algorithm is* 3, *i.e., it always produces an agreement forest of size at most three times the size of an MAF for* T_1 *and* T_2.

Proof. We consider the number of edges cut in an agreement forest and show that the above algorithm cuts at most three times as many the edges cut by an MAF. To establish the approximation bound, the basic idea is to consider an MAF and charge the edges cut by the algorithm to the edges cut by the MAF, and make sure that each MAF edge is charged at most three edges. For convenience, we will refer to an edge according to its lower end.

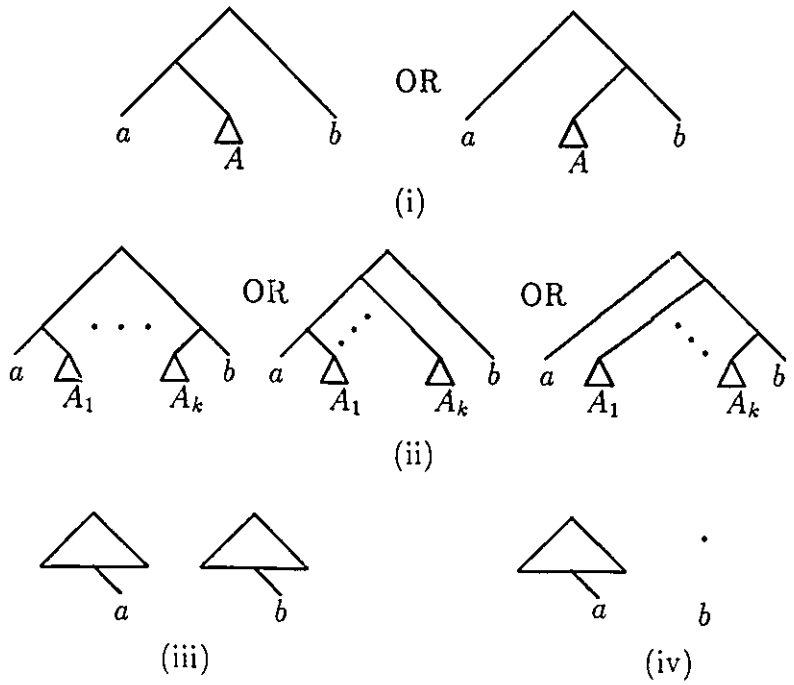


Figure 6.8: The first four cases of a and b in T_2.

1. **begin**
2. Input T_1 and T_2.
3. $N := 1$.
4. **For** a pair of sibling leaves a, b in T_1 **do**.
   Consider how they appear in T_2 and cut the trees:
   Case (i): Cut off the middle subtree A in T_2; $N := N + 1$.
   Case (ii): Cut off a and b in both T_1 and T_2; $N := N + 2$.
   Case (iii): Cut off a and b in both T_1 and T_2; $N := N + 2$.
   Case (iv): Cut off b in T_1.
   Case (v): Replace this pair with a new leaf labeled (a, b) in both T_1 and T_2.
5. If there exists any component in the forest for T_1 of size larger than 1, repeat Step 1.
6. Output the forest and N.
7. **end**

Figure 6.9: The approximation algorithm of ratio 3.

We need a lemma which establishes that the algorithm is always optimal in Case (i).

Lemma 6.10 *There exists an MAF F which cuts all the edges cut by the algorithm in Case (i).*

We only have to consider Cases (i)-(iii) because Case (iv) repeats an old cut. Let's fix the MAF F. We charge the edges as follows. In Case (i), each edge cut is simply charged to itself. Case (ii) is the most interesting. If at least one of the edges above a and b are cut by F, then we simply charge these two edges cut by the algorithms to the "correct" edge(s) cut by F. So each edge is charged with a cost of at most 2. Otherwise all the edges above the subtrees $A_1, \ldots, A_k$ ($k \geq 2$) must be cut by F and we charge the two edges above a and b to these edges (each charged a cost of $\frac{2}{k} \leq 1$). Note that, only edges cut in Case (i) create components of size bigger than 1. Thus, if Case (iii) occurs, then a and b must belong to different components in F and hence F must cut at least one of the edges above a and b in T_1 to disconnect them. So we can charge the two edges above a and b to the "correct" one(s) cut by F (each charged a cost of at most 2).

It is easy to see that each edge cut by F is charged at most twice. Moreover, if an edge is charged twice, the first time it is charged must be in the second subcase of Case (ii). The second time can be in either Case (i), Case (iii), or the first subcase of Case (ii). Hence, each such edge is charged a cost of at most $1 + 2 = 3$ edges cut by the algorithm. That is, the algorithm cuts at most three times as many edges cut by F. ∎

It would be interesting to improve the approximation ratio. On the other hand, the NP-hardness proof can be easily strengthened to work for MAX SNP-hardness. Thus, there is no hope for a PTAS for this problem.

Chapter 7

An Approximation Scheme for Some Planar Steiner Tree Problems

7.1 Introduction

The PTAS developed in Chapter 4 works for the problem of Steiner trees under a given topology. In this chapter, we consider the planar Steiner tree problem when the topology is not given. We devise a PTAS for a special case of the planar Steiner tree problem. Our algorithm works for both Euclidean and rectilinear metrics. Steiner minimal trees in Euclidean and rectilinear metrics are called Euclidean Steiner minimal trees (ESMT) and rectilinear Steiner minimal trees (RSMT), respectively.

The planar Steiner tree problem arises in many applications such as network design,

optimal location of facilities, component placement on circuit boards, etc [26]. Unfortunately, it is NP-hard for both Euclidean and rectilinear metrics [26, 27]. Considerable effort has been made to develop approximation algorithms [42]. A well-known strategy is to uses a minimum-cost spanning tree as an approximate solution. This method has a performance ratio of 2 for any metric. In particular, it achieves a ratio of $\frac{2}{\sqrt{3}}$ [15] for the Euclidean plane and a ratio of $\frac{3}{2}$ [40] for the rectilinear plane. Du, Zhang and Feng proved that an ESMT can be approximated in polynomial time with ratio $\frac{2}{\sqrt{3}} - \epsilon$, for some $\epsilon \geq 0$ [16]. Berman and Ramaiyer showed that for an RSMT, the ratio can be further improved to 1.33 [6].

There are also some probabilistic approximation methods for the ESMT and RSMT problems [49, 42]. Komlos and Shing extended Karp's partitioning algorithm to RSMT [52]. Assume that the set of regular points is uniformly distributed in the unit square. They obtained an algorithm with ratio $1 + O(\frac{1}{\sqrt{k}})$, with probability $1 - o(1)$, where k is the number of regular points considered in the algorithm to construct a local RSMT. Other probabilistic results can be found in [7, 41, 42].

In this chapter, we design a PTAS for the planar Steiner tree problem when the given set of regular points is *c-local* for some constant c. Here, a set of points X is c-local [1] if in the minimum-cost spanning tree for X, the length of the longest edge is at most c times the length of the shortest edge. For simplicity, we will assume that the length of the shortest edge is always 1. Hence, the restricted Steiner problem is approximable within ratio $1 + \epsilon$ for any $\epsilon > 0$ in polynomial time. Our algorithm works for both Euclidean and rectilinear metrics. It is worth pointing out that the ESMT and RSMT problems remain NP-hard when the set of regular points is c-local, since the original constructions in [26, 27] still work for this special case.

[1] The term c-local is borrowed from [78], where c-local graphs are considered. Local graphs are also called *civilized* graphs in [14].

An outline of our approach is given in next section. Section 7.3 contains the PTAS and its analysis. Some concluding remarks are given in section 7.4.

7.2 The Basic Idea and Partition Strategy

The basic idea of our algorithm is to combine the shifting technique in [37] with the local optimization method in [80]. First, we design a set of partitions, each of them partitions the whole area enclosing the regular points into many rectangular *cells* (mostly squares) of some *constant* size. (See Figure 7.1.) Each cell is further divided into *interior* and *boundary* areas as in Figure 7.2. Then, with respect to each partition, we organize the regular points contained in the interior area of each cell into several groups such that the "distance" between any two groups is greater than c, and construct a Steiner minimal tree (SMT) for each group. The collection of all the *local Steiner trees* in a cell form a *local Steiner forest* for the cell. After that, we connect all the local Steiner forests and the regular points contained in the boundary areas using the spanning tree method. Finally, we select a partition which yields the optimal Steiner tree among all the partitions.

We note in passing that a similar overall idea has recently (and independently) been applied to a number of graph problems including vertex cover, independent set, and dominating set [38].

In the rest of this section, we elaborate on the partitions. Without loss of generality, assume that the set of regular points X is contained in a rectangle R with corners $(0,0)$ $(a,0)$, $(0,b)$, and (a,b), as shown in Figure 7.1. For any positive integer k, a *partition* of size k is a grid in which adjacent horizontal/vertical lines are separated by a distance k. Clearly, there are k^2 different partitions of size k, depending on the positions of the top horizontal

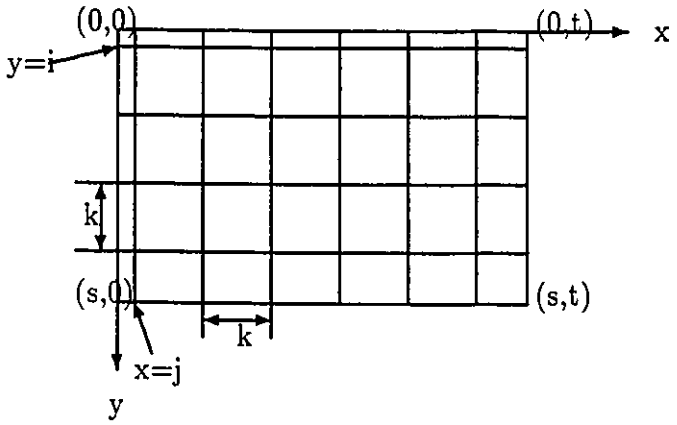


Figure 7.1: The rectangle with partition $P_{i,j}$ of size k.

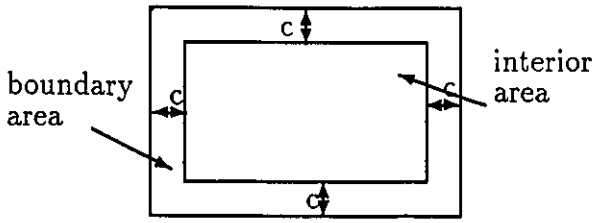


Figure 7.2: The interior and boundary areas.

line and the leftmost vertical line. Throughout this chapter, we use $P_{i,j}$, where $0 \leq i, j < k$, to denote the partition in which the top horizontal line and the leftmost vertical line are $x = i$ and $y = j$, respectively. The grid partitions the rectangle R into many cells, most of them are squares of size $k \times k$. Each cell is divided into an interior area and a boundary area, with a boundary of *width* c, as shown in Figure 7.2.

7.3 The Approximation Scheme and Analysis

In this section, we construct the approximation algorithm and analyze its performance for both Euclidean and rectilinear metrics.

Let X be the given set of regular points in the plane, P a fixed partition, $X_P \subseteq X$ the set of all regular points in the interior areas with respect to P. Call an edge (*i.e.*, line segment) *crossing* if it is not completely contained in the interior area of some cell. Throughout the chapter, we use T^{min} to denote an SMT for X, and T^P an SMT for X_P. For any tree T, $C(T)$ denotes the cost of the tree. Since X_P is a subset of X,

$$C(T^P) \leq C(T^{min}).$$

7.3.1 Constructing the Local Steiner Forests

We first develop some useful results. It should be emphasized that in our algorithm, we will deal with one cell at a time. Recall that the regular points in the interior area of a cell are divided into several groups and an SMT is constructed for each group. In order to show how to correctly group the regular points in an interior area, let's consider the SMT T^P. We need to modify T^P into a forest F^P such that each (local Steiner) tree in F^P is completely included in the interior area of some cell. Note that, the interior area of a cell may contain more than one tree of F^P. Define the *distance* between two (local Steiner) trees as the minimum distance between any two regular points in the two trees. We further require that the distance between any pair of trees in F^P is greater than c.

Lemma 7.1 *The SMT T^P can be modified into a forest F^P such that each tree in F^P is completely included in the interior area of a cell and the distance between any pair of trees in F^P is greater than c. Moreover, the total cost $C(F^P)$, which is the sum of the costs of all the trees in F^P, is at most $C(T^P)$. That is,*

$$C(F^P) \leq C(T^P) \leq C(T^{min}).$$

Proof. First, let us consider the case when no Steiner point of T^P is in the boundary areas of partition P. In this case, we can prune the tree T^P by simply deleting all the crossing edges. (Each end of a crossing edge could be either Steiner point or regular point.) Note that, in this case, the cost of any crossing edge is at least $2c$. Since T^P is optimal, the distance between any pair of trees in the resulting forest is greater than c. Otherwise, let T_1 and T_2 be two trees obtained above such that the distance between them is at most c. Since the width of a boundary is c, T_1 and T_2 must be in the interior area of a same cell. We can decrease the cost of T^P by adding an edge to connect T_1 and T_2 (which increases the cost by at most c) and deleting an appropriate crossing edge (which decreases the cost by at least $2c$).

Suppose that some Steiner points of T^P are in the boundary areas. There must be such a Steiner point f which directly connects two regular points in interior areas, say, d and e. Two subcases arise,

1. The points d and e are in the same interior area, as shown in Figure 7.3(a). We delete the edges (d, f) and (e, f), and connect the points d and e directly. This decreases the cost.

2. The points d and e belong in different interior areas, as shown in Figure 7.3(b). Now, we delete the crossing edges (d, f) and (e, f), saving a cost of at least $2c$. The deletion decomposes T^P into three trees. If the distance between two of the trees is at most c, we directly connect them with a cost of at most c. This will not increase the cost since we have to connect at most two pairs of trees.

Note that, in an ESMT, the degree of a Steiner point is 3 [16] and in an RSMT, the degree of a Steiner point is either 3 or 4 [6, 40]. Hence, after deleting the edges (e, f) and

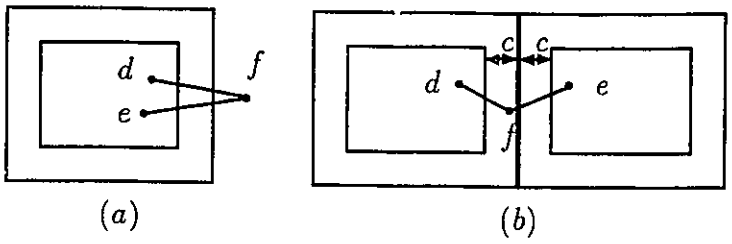


Figure 7.3: (a) The two points are in the same interior area. (b) They are not in the same interior area.

(d, f) in the above, the degree of f is at most 2 and we can eliminate f without any extra cost. In this way, we can remove all the Steiner points in the boundary areas and obtain a set of trees such that tne distance between any pair trees is greater than c. Then the first case applies. ■

It is difficult to compute the forest F^P, since the SMT T^P is unknown. Nevertheless, we can construct a forest that is similar to F^P in many ways. Consider the regular points in the interior area of some cell. By Lemma 7.1, if the distance of two regular points is at most c, then they must belong to a same tree of F^P. Thus, we can group the regular points by forming a minimum-cost spanning tree of these points and then deleting the edges longer than c. Therefore, we get a set of (spanning) trees $S_1, \ldots S_m$, consisting of edges of length at most c. We call these trees the *c-spanning trees*. Let Y_i, $i = 1, \ldots m$, be the set of regular points contained in the c-spanning tree S_i. Clearly, the regular points in the same group Y_i belong to a same tree of the forest F^P. The converse is not necessarily true. Namely, points in different groups Y_i's may also belong to a same tree of F^P. In other words, to find the best way of grouping the regular points, we have to consider all possible ways merging the groups $Y_1, \ldots, Y_m$. After each such possible merge, we obtain a local Steiner forest by constructing an SMT for every *new* group. We are interested in a local Steiner forest with the minimum cost among all possible merges.

1. **begin**
2. Let Y be the set of regular points in the interior area of the cell.
3. Construct a minimum spanning tree for Y.
4. Delete the edges longer than c.
5. Let $Y_1, \ldots, Y_m$ be the sets of regular points in the c-spanning trees.
6. **for** each possible way of merging the Y_i's
7. Merge the Y_i's and construct an SMT for each resulting group.
8. Select a local Steiner forest with the minimum cost.
9. **end**

Figure 7.4: **Procedure 7.1**

Let forest $\hat{F}^P$ denote the collection of the minimum-cost local Steiner forests, one for each cell. $\hat{F}^P$ has the following properties.

Lemma 7.2 *(i) Each tree in $\hat{F}^P$ is completely contained in the interior area of a cell; (ii) The distance between any pair of trees T_i and T_j in $\hat{F}^P$ is greater than c; and (iii) The total cost of the forest $\hat{F}^P$ is at most $C(F^P)$. Thus,*

$$C(\hat{F}^P) \leq C(F^P) \leq C(T^{min}).$$

The detailed procedure to construct an optimal local Steiner forest for a cell is given in Procedure 7.1. The number of possible ways of merging the groups is

$$\sum_{k=1}^{m} \left\{ \begin{matrix} m \\ k \end{matrix} \right\},$$

where $\left\{ \begin{matrix} m \\ k \end{matrix} \right\}$ is the Stirling number of second kind, which stands for the number of ways to partition a set of m items into k nonempty subsets [29]. Clearly,

$$\sum_{k=1}^{m} \left\{ \begin{matrix} m \\ k \end{matrix} \right\} \leq m!.$$

Thus, the time complexity of Procedure 7.1 is $O(m!M(|Y|))$, where $M(|Y|)$ is the time to construct an SMT for the set Y of regular points in the interior area of the cell. For both Euclidean and rectilinear metrics, $M(|Y|)$ is exponential in $|Y|$ [42, 52].

Clearly $m!$ dominates the time complexity of Procedure 7.1 in general. It is possible to lower it to 2^m using a straightforward dynamic programming method. For each $I \subseteq \{1, \ldots, m\}$, let $D[I]$ denote the cost of an SMT for the set $\cup_{i \in I} Y_i$ and $E[I]$ denote the minimum cost of any local Steiner forest for $\{Y_i | i \in I\}$. We have the following recurrence relation:

$$E[I] = \min_{I' \subseteq I} D[I'] + E[I - I']. \tag{7.1}$$

Thus $E[\{1, \ldots, m\}]$ gives the cost of a minimum-cost local Steiner forest for $Y_1, \ldots, Y_m$. The forest can be recovered by a simple backtracking. Let Procedure 7.2 be the modified version of Procedure 7.1 which uses Equation 7.1 to compute a minimum-cost local Steiner forest. Then the running time of Procedure 7.2 is $O(2^m M(|Y|) + 2^m \cdot 2^m) = O(2^m M(|Y|))$.

7.3.2 Connecting the Local Forests and Boundary Points

We can construct a Steiner tree for the set X from the forest $\hat{F}^P$ as follows. Fix a minimum-cost spanning tree T_S for X. Note that each edge in T_S has length at most c since X is c-local. Let E_P denote the set of crossing edges in T_S. Construct a graph G_P by adding all the crossing edges in E_P to $\hat{F}^P$.

Lemma 7.3 *G_P is connected.*

```
1. begin
2. Construct a minimum-cost spanning tree T_S for X.
3. for each possible partition P
4.   begin
5.     Find the set of crossing edges E_P.
6.     for each cell
7.       Use Procedure 7.2 to compute a minimum-cost local Steiner forest.
8.     Let F̂^P be the set of all local Steiner forests.
9.     Construct the graph G_P = E_P ∪ F̂^P.
10.  end.
11. Select a G_P with the smallest cost among all partitions.
12. Prune G_P into a tree.
13. end
```

Figure 7.5: **Algorithm 7.1**

Proof. Since T_S is connected, the graph $T_S \cup \hat{F}^P$ is connected. For each tree T_i in $\hat{F}^P$, let's call an edge in T_S connecting two regular points in T_i an *redundant* edge. Becasue T_i is connected, deleting any redundant edge will not affect the connectivity of the graph. Moreover, since each edge in T_S has length at most c and the distance between any pair of trees in $\hat{F}^P$ is greater than c, we can conclude that every non-redundant edge in T_S is a crossing edge. Therefore, G_P is connected. ■

Now, we are ready to introduce our algorithm, which in fact computes G_P for every possible partition P, selects the G_P with the smallest cost, and prunes the selected G_P into a tree. See Figure 7.5.

Theorem 7.4 *Algorithm 7.1 achieves an approximation ratio* $1 + \frac{14c}{\sqrt{3}k}$ *and* $1 + \frac{9c}{k}$ *for Euclidean and rectilinear metrics, respectively. Its runing time is* $O((|X|c)^2 2^{k^2} M(k^2)/k^2)$.

Proof. By Lemma 7.2,

$$C(G_P) = C(\hat{F}^P) + C(E_P) \leq C(T^{min}) + C(E_P). \tag{7.2}$$

Sum up Inequality 7.2 for all the partitions $P_{i,j}$, we have

$$\begin{aligned}\sum_{i=1}^{k}\sum_{j=1}^{k} C(G^{P_{i,j}}) &\leq k^2 C(T^{min}) + \sum_{i=1}^{k}\sum_{j=1}^{k} C(E_P) \\ &\leq k^2 C(T^{min}) + \mu ckC(T_s),\end{aligned}$$

where $\mu = 6$ in rectilinear metric and $\mu = 7$ in Euclidean metric. The last inequality is based on the observation that each edge in the minimum-cost spanning tree T_S is "cut" at most μck times by the boundary. The reason is as follows. Note that, the boundary of each cell consists of at most $4ck - 4c^2$ points. Each point could serve as one of the two ends of a crossing edge. Thus, a crossing edge can be cut at most $8ck - 8c^2$ times. In this counting, an edge that is completely contained in a boundary is considered being cut twice. Again note that the length of each edge in T_S is bounded by c. In rectilinear metric, this "double counting" happens at least $4 \cdot \frac{1}{2}ck = 2ck$ times for each edge. In Euclidean metric, double counting happens at least $4(1 - \frac{\sqrt{2}}{2})ck \geq ck$ times. Thus, each edge in T_S is cut at most $6ck$ in rectilinear metric and $7ck$ times in Euclidean metrics.

Since Algorithm 7.1 selects G_P with the smallest cost, we have

$$C(G_P) \leq C(T^{min}) + \frac{\mu ck}{k^2} C(T_S) = C(T^{min}) + \frac{\mu c}{k} C(T_S).$$

Therefore, the cost of the Steiner tree output by Algorithm 7.1 is at most $C(T^{min}) + \frac{\mu c}{k} C(T_S)$. In conjunction with the results given in [15, 40], we conclude that the performance ratio of Algorithm 7.1 is $1 + \frac{14c}{\sqrt{3}k}$ for Euclidean metric and $1 + \frac{9c}{k}$ for rectilinear metric.

There are totally k^2 distinct partitions. For each partition, there are at most $O(\frac{(|X|c)^2}{k^2})$ cells and each cell contains at most k^2 regular points. It requires at most $O(2^{k^2} M(k^2))$ time to construct a minimum-cost local Steiner forest for each cell. Therefore, the time complexity of Algorithm 7.1 is $O((|X|c)^2 2^{k^2} M(k^2)/k^2)$. ■

Corollary 7.5 *There is a PTAS for the ESMT and RSMT problems when the given set of regular points is c-local.*

7.4 Some Remarks

It remains an interesting open problem to prove or disprove the existence of a PTAS for the general ESMT or RSMT problems.

The performance ratio of our algorithm holds as long as the length of the longest edge in the minimum-cost spanning tree is at most c. We do not have to assume that the length of the shortest edge is 1. The only trouble is that there could be too many regular points in one cell of the partition, making the running time possibly exponential. However, if we assume that the regular points are "uniformly distributed", *i.e.*, the number of regular points in each cell of size $k \times k$ is at most αk^2 for some constant α, then our algorithm will still run in polynomial time. Note that our notion of "uniform distribution" is different (and stronger) than that in a conventional probabilistic algorithm [49]. In a Possion process, only the *expected* number of regular points is the same for all regions of equal area.

Chapter 8

Summary

In the thesis, several important problems arising in computational biology have been investigated. The problems concern multiple sequence alignment, comparison of various types of trees, and Steiner trees. The major contributions of this thesis are summarized below.

1. Multiple sequence alignment. Three problems, *multiple sequence alignment with SP-score*, *tree alignment* and *generalized tree alignment* are investigated. It is shown that multiple sequence alignment with SP-score and tree alignment are NP-complete, and that generalized tree alignment is MAX SNP-hard. These results appear in [79]. An efficient approximation algorithm with performance ratio 2 is designed for tree alignment when the given evolutionary tree is of bounded degree. The algorithm is then extended to a polynomial-time approximation scheme, which is believed to be the first polynomial-time approximation scheme in computational biology. These results appear in [80].

2. Comparison of labeled trees. The *alignment of trees* is proposed as an alternative to tree edit. Both *ordered* and *unordered* trees are considered. An efficient algorithm

is designed for ordered trees. The time complexity of this algorithm is $O(|T_1| \cdot |T_2| \cdot (\deg(T_1) + \deg(T_2))^2)$, where $|T_i|$ is the number of nodes in T_i and $\deg(T_i)$ is the degree of T_i, $i = 1, 2$. Our algorithm is faster than the best known algorithm for tree edit. For unordered trees, it is shown that the alignment of trees can be solved in polynomial time if both trees are of bounded degree. And it becomes NP-hard if one of the trees is allowed to have an arbitrary degree. These results are included in [46].

3. Comparison of evolutionary trees. Several problems arising in the comparison of evolutionary trees are investigated. It is shown that the maximum agreement subtree (MAST) problem for three trees with unbounded degree cannot be approximated within any constant ratio unless $P = NP$ and that this problem cannot be approximated within ratio n^{ϵ}, for some $0 < \epsilon \leq 1$, unless $NP \subseteq DTIME(2^{polylogn})$. Moreover, MAST with edge contractions for two binary trees is shown to be NP-hard. This answers two open questions posed in [2]. For the maximum refinement subtree (MRST) problem involving two trees, it is shown that it is polynomial-time solvable when both trees have bounded degree and that it is NP-hard when one of the trees can have an arbitrary degree. Finally, the problem of optimally transforming a tree into another by transferring subtrees around is considered. The NP-hardness of the subtree-transfer problem is proved and an approximation algorithm with performance ratio 3 is given. These results appear in [36].

4. Steiner trees. The polynomial-time approximation scheme designed for tree alignment works for the Steiner tree problem under a given topology in any metric space. A polynomial time approximation scheme is also designed for the planar Steiner tree problem when the given set of regular points is *c-local*. The algorithm works for both Euclidean and rectilinear metrics. The results appear in [81].

Bibliography

[1] S. Altschul and D. Lipman, "Trees, stars, and multiple sequence alignment", *SIAM Journal on Applied Math.* 49, pp. 197-209, 1989.

[2] A. Amir and D. Keselman, "Maximum agreement subtree in a set of evolutionary trees - metrics and efficient algorithms", *Proc. IEEE FOCS'94*, 1994.

[3] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy, "Proof verification and hardness of approximation problems", *Proc. 33rd IEEE Symp. Found. of Comp. Sci.*, pp. 14-23, 1992.

[4] D. Baconn and W. Anderson, "Multiple sequence alignment", *Journal of Moleccular Biology* 191, pp. 153-161, 1986.

[5] V. Bafna, E. Lawler, and P. Pevzner, "Approximate methods for multiple sequence alignment", *Proc. CPM'94*, pp. 43-53, 1994.

[6] P. Berman and V. Ramaiyer, "Improved approximation for the Steiner tree problem", *Manuscript*, 1993.

[7] M.W. Bern, "Two probabilistic results on rectilinear Steiner trees", *Algorithmica* **3**, pp. 191-204, 1988.

[8] M. Bern and P. Plassmann, "The Steiner problem with edge lengths 1 and 2", *Information Processing Letters* 32, pp. 171-176, 1989.

[9] H. Carrillo and D. Lipman, "The multiple sequence alignment problem in biology", *SIAM Journal on Applied Math.* 48, pp. 1073-1082, 1988.

[10] S.C. Chan, A.K.C. Wong and D.K.T. Chiu, "A survey of multiple sequence comparison methods", *Bulletin of Mathematical Biology* 54(4), pp. 563-598, 1992.

[11] X. Chen, "An interactive system for sequence analysis", *Master of Science Thesis, Dept. of Computer Science and Systems, McMaster University*, 1994.

[12] M. O. Dayhoff, R. V. Eck and C. M. Park, "A model of evolutionary change in proteins", *Atlas of Protein Sequence and structure*, Vol. 5, pp. 89-99, 1972.

[13] C. DeLisi, "Computers in molecular biology: Current applications and and emerging trends", *Science*, 24 (Apr. 1,), pp. 47-52, 1988.

[14] P. Doyle and J.L. Snell, *Random Graphs and Electric Networks*, The Carus Mathematical Monographs, The Mathematical Association of America, 1984.

[15] D.Z. Du and F.K. Hwang, "An approach for proving lower bounds: solution of Gilbert-Pollak's conjecture on Steiner ratio," *Proc. 31st IEEE FOCS*, pp. 76-85, 1990.

[16] D.Z. Du, Y. Zhang and Q. Feng, "On better heuristic for Euclidean Steiner minimum trees", *Proc. 32nd IEEE Symp. Found. of Comp. Sci.*, pp. 431-439, 1991.

[17] G. Estabrook, C. Johnson and F. McMorris, "A mathematical foundation for the analysis of cladistic character compatibility", *Math. Biosci.* 29, pp. 181-187, 1976.

[18] R. Fagin, "Generalized first-order spectra and polynomial-time recognizable sets", *Complexity of Computations* (R. Karp ed.), Amer. Math. Soc., 1974.

[19] M. Farach, S. Kannan and T. Warnow, "A robust model for finding optimal evolutionary trees", *Proc. STOC'93*, pp. 137-145.

[20] M. Farach and M. Thorup, "Fast comparison of evolutionary trees", in *Proc. 5ith Annual ACM-SIAM Symposium on Discrete Algorithms*, 1994.

[21] M. Farach and M. Thorup, "Optimal evolutionary tree comparison by sparse dynamic programming", *IEEE FOCS'94*, 1994.

[22] J.S. Farris, Methods for computing Wagner trees, *Systematic Zoology* 19, pp. 83-92, 1970.

[23] C. Finden and A. Gordon, "Obtaining common pruned trees", *Journal of Classification* 2, pp. 255-276, 1985.

[24] L.R. Foulds and R.L. Graham, "The Steiner problem in phylogeny is NP-complete", *Advances in Applied Mathematics* 3, pp. 43-49, 1982.

[25] K. A. Frenkel, "The human genome project and informatics", *Communications of the ACM* 34(11), pp. 41-51, 1991.

[26] M.R. Garey, R.L. Graham and D.S. Johnson, "The complexity of computing Steiner minimal trees," *SIAM J. APPL. MATH* **32**, 1977.

[27] M.R. Garey and D.S. Johnson, "The rectilinear Steiner tree problem is NP-complete," *SIAM J. APPL. MATH* **32**, pp. 826-834, 1977.

[28] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, 1979.

[29] R.L. Graham, D.E. Knuth, and O. Patashnik, *Concrete Mathematics - A Foundation for Computer Science*, Addison-wesley publishing company, 1989.

[30] D. Gusfield, "Efficient algorithms for inferring evolutionary trees", *Networks* 21, pp. 19-28, 1991.

[31] D. Gusfield, "Efficient methods for multiple sequence alignment with guaranteed error bounds", *Bulletin of Mathematical Biology* 55, pp. 141-154, 1993.

[32] J. Hein, "A tree reconstruction method that is economical in the number of pairwise comparisons used", *Mol. Biol. Evol.* 6(6), pp. 669-684, 1989.

[33] J. Hein, "A new method that simultaneously aligns and reconstructs ancestral sequences for any number of homologous sequences when the phylogeny is given", *Mol. Biol. Evol.* 6(6), pp. 649-668, 1989.

[34] J. Hein, "Reconstructing evolution of sequences subject to recombination using parsimony", *Math. Biosci.* 98, pp. 185-200, 1990

[35] J. Hein, "A heuristic method to reconstruct the history of sequences subject to recombination", *Journal Molecular Evolution* 36, pp. 396-405, 1993.

[36] J. Hein, T. Jiang, L. Wang and K. Zhang, "On the complexity of comparing evolutionary trees", to be presented at the *Sixth Symposium on Combinatorial Pattern Matching (CPM'95)*, 1995.

[37] D.S. Hochbaum and W. Maass, "Approximation schemes for covering and packing problems in image processing and VLSI", *J. ACM* **32**, pp. 130-136, 1985.

[38] H.B. Hunt III, M.V. Marathe, V. Radhakrishnan, S.S. Ravi, D.J. Rosenkrantz, and R.E. Stearns, "Efficient approximations and approximation schemes for geometric graphs", *Manuscript*, 1993.

[39] L. Hunter, "Molecular Biology for Computer Scientists", in *Artificial Intelligence and Molecular Biology*, ed. L. Hunter, AAI Press / MIT Press, 1993.

[40] F.K. Hwang, "On Steiner minimal trees with rectilinear distance", *SIAM. J. APPL.MATH,* **30** pp. 104-114, 1976.

[41] F.K. Hwang and Y.C. Yao, "Comments on Bern's probabilistic results on rectilinear Steiner trees", *Algorithmica* **5**, pp. 591-598, 1990.

[42] F.K. Hwang and D.S. Richards, "Steiner tree problems", *Networks* 22, pp. 55-89, 1992.

[43] F.K. Hwang and J.F. Weng, "The shortest network under a given topology", *Journal of Algorithms* 13, pp. 468-488, 1992.

[44] T. Jiang and M. Li, "Optimization problems in molecular biology", in *Advances in Optimization and Approximation*, D.Z. Du and J. Sun (eds.), pp. 195-216, 1994.

[45] T. Jiang and M. Li, "On the approximation of shortest common supersequences and longest common subsequences", to appear in *SIAM J. Comput.*; also presented at *ICALP'94*.

[46] T. Jiang, L. Wang and K. Zhang, "Alignment of trees - an alternative to tree edit", *Theoretical Computer Science*, to appear; also presented at *CPM'94*, pp.75-86.

[47] T.H. Jukes and C.R. Cantor, "Evolution of protein molecules", in H.N. Munro, ed., *Mammalian Protein Metabolism*, Academic Press, pp. 21-132, 1969.

[48] V. Kann, "Maximum bounded 3-dimensional matching is MAX SNP-complete", *Information Processing Letters* 37, pp. 27-35, 1991.

[49] R.M. Karp, "Probabilistic analysis of partitioning algorithms for the traveling salesman problem in the plane", *Math. Operations Research* 2, pp. 209-224, 1977.

[50] R.M. Karp, "Mapping the genome: some combinatorial problems arising in molecular biology", *Proc. ACM Symp. Theory of Computing*, pp. 278-285, 1993.

[51] P. Kilpelainen and H. Mannila, "Ordered and unordered tree inclusion", Report A-1991-4, Dept. of Comp. Science, University of Helsinki, August 1991; to appear in *SIAM J. on Computing.*

[52] J. Komlos and M.T. Shing, "Probabilistic partitioning algorithms for the rectilinear Steiner problem," *Networks* **15**, pp. 413-423, 1985.

[53] E.S. Lander, R. Langridge and D.M. Saccocio, "Mapping and interpreting biological information", *Communications of the ACM* 34(11), pp. 33-39, 1991.

[54] S.-Y. Le, J. Owens, R. Nussinov, J.-H. Chen B. Shapiro and J. V. Maizel, "RNA secondary structures: comparison and determination of frequently recurring substructures by consensus", *Comp. Appl. Biosci.* 5, pp. 205-210, 1989.

[55] S.-Y. Le, R. Nussinov, and J.V. Maizel, "Tree graphs of RNA secondary structures and their comparisons", *Computers and Biomedical Research*, 22, pp. 461-473, 1989.

[56] R. J. Lipton, T. G. Marr, and J. D. Welsh, "Computational approaches to discovering semantics in molecular biology", *Proceedings of the IEEE*, vol. 77, 7, pp. 1056-1060, 1989.

[57] S.Y. Lu, "A tree-tree distance and its application to cluster analysis", *IEEE Trans. Pattern Anal. Mach. Intelligence* 1, pp. 219-224, 1979.

[58] M. Middendorf, "More on the complexity of common superstring and supersequence problems", to appear in *Theoret. Comp. Sci.*

[59] W.Miller, S. Schwartz, and R. C. Hardison, "A point of contact between computer science and molecular biology", *IEEE Computational Science and Engineering*, Spring 1994, pp. 69-78.

[60] C.H. Papadimitriou and M. Yannakakis, "Optimization, Approximation, and complexity classes", *Journal of Computer and System Sciences* 43, pp. 425-440, 1991.

[61] D. Penny, "Criteria for optimising phylogenetic trees and the problem of determining the root of a tree", *J. Mol. Evol.* 8, pp. 95-116, 1976.

[62] P. Pevzner, "Multiple alignment, communication cost, and graph matching", *SIAM J. Applied Math.* 56(6), pp. 1763-1779, 1992.

[63] D. Sankoff, "Minimal mutation trees of sequences", *SIAM J. APPL. Math.* 28(1), pp. 35-42, 1975.

[64] D. Sankoff and P. Rousseau, "Locating the vertices of a Steiner tree in an arbitrary metric space", *Mathematical Programming* 9, pp. 240-246, 1975.

[65] D. Sankoff, R. Cedergren and G. Lapalme, "Frequency of insertion-deletion, transversion, and transition in the evolution of 5S ribosomal RNA", *J. Mol. Evol.* 7, pp. 133-149, 1976.

[66] D. Sankoff and J. Kruskal (Eds), *Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison,* Addison Wesley, Reading Mass., 1983.

[67] D. Sankoff and R. Cedergren, "Simultaneous comparisons of three or more sequences related by a tree", in D. Sankoff and J. Kruskal (eds), *Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison*, pp. 253-264, Addison Wesley, Reading Mass., 1983.

[68] N. Saitou and M. Nei, "The neighbor-joining method: a new method for reconstructing phylogenetic trees", *Mol. Biol. Evol.* 4-4, pp. 406-425, 1987.

[69] G.D. Schuler, S.F. Altschul, and D.J. Lipman. "A workbench for multiple alignment construction and analysis", in *Proteins: Structure, function and Genetics*, in press.

[70] R.Schwarz and M. Dayhoff, "Matrices for detecting distant relationships", in M. Dayhoff, ed., *Atlas of protein sequences*, National Biomedical Research Foundation, 1979, pp. 353-358.

[71] B. Shapiro, "An algorithm for comparing multiple RNA secondary structures", *Comput. Appl. Biosci.* pp. 387-393, 1988.

[72] B. Shapiro and K. Zhang, "Comparing multiple RNA secondary structures using tree comparisons", *Comput. Appl. Biosci.* vol. 6, no. 4, pp. 309-318, 1990.

[73] F.Y. Shih and O.R. Mitchell, "Threshold decomposition of grayscale morphology into binary morphology", *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-11, pp. 31-42, 1989.

[74] F.Y. Shih, "Object representation and recognition using mathematical morphology model", *J. System Integration*, vol. 1, pp. 235-256, 1991.

[75] M. Steel and T. Warnow, "Kaikoura tree theorems: computing the aaximum agreement subtree", *Information Processing Letters* 48, pp. 77-82, 1993.

[76] K.C. Tai, "The tree-to-tree correction problem", *J. ACM*, 26, pp. 422-433, 1979.

[77] Y. Takahashi, Y. Satoh, H. Suzuki and S. Sasaki, "Recognition of largest common structural fragment among a variety of chemical structures", *Analytical Science*, vol. 3, pp. 23-28, 1987.

[78] S.A. Vavasis, "Automatic domain partitioning in three dimensions", *SIAM J. Sci. Stat. Comput.* **12-4**, pp. 950-970, 1991.

[79] L. Wang and T. Jiang, "On the complexity of multiple sequence alignment", *Journal of Computational Biology*, vol. 1, pp. 337-348, 1994.

[80] L. Wang, T. Jiang, and E.L. Lawler, "Aligning sequences via an evolutionary tree: complexity and approximation", *Algorithmica*, to appear; also presented at the *26th ACM Symp. on Theory of Computing*, 1994.

[81] L. Wang and T. Jiang, "An approximation scheme for some Steiner tree problems in the plane," submitted to *Networks*; also presented at the *5th Annual International Symposium on Algorithms and Computation*, pp. 414-422, Beijing, 1994.

[82] T. Warnow, "Tree compatibility and inferring evolutionary history", *J. of Algorithms* 16, pp. 388-407, 1994.

[83] T. Warnow, Private communication, 1994.

[84] M.S. Waterman and M.D. Perlwitz, Line geometries for sequence comparisons", *Bull. Math. Biol.* 46, pp. 567-577, 1984.

[85] M.S. Waterman, "Sequence alignments", in *Mathematical Methods for DNA Sequences*, M.S. Waterman (ed.), CRC, Boca Raton, FL, pp. 53-92, 1989.

[86] K. Zhang and D. Shasha, "Simple fast algorithms for the editing distance between trees and related problems", *SIAM J. Comput.* 18, pp. 1245-1262, 1989.

[87] K. Zhang, R. Statman, and D. Shasha, "On the editing distance between unordered labeled trees", *Information Processing Letters*, 42, pp. 133-139, 1992.

[88] K. Zhang and T. Jiang, "Some MAX SNP-hard results concerning unordered labeled trees", *Information Processing Letters* 49, pp. 249-254, 1994.

[89] A.Z. Zelikovsky, "The 11/6 approximation algorithm for the Steiner problem on networks", *Algorithmica* 9, pp. 463-470, 1993.