An Approximation Algorithm for Character Compatibility and Fast Quartet-based Phylogenetic Tree Comparison

by

John Tsang

A thesis

presented to the University of Waterloo

in fulfilment of the

thesis requirement for the degree of

Master of Mathematics

in

Computer Science

Waterloo, Ontario, Canada, 2000

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Phylogenetic analysis, or the inference of evolutionary history is done routinely by biologists and is one of the most important problems in systematic biology. In this thesis, we study two computational problems in the area. First, we study the evolutionary tree reconstruction problem under the character compatibility (CC) paradigm and give a polynomial time approximation scheme (PTAS) for a variation of the formulation called fractional character compatibility (FCC), which has been proven to be NP-hard. We also present a very simple algorithm called the Ordinal Split Method (OSM) to generate bipartitions given sequence data, which can be served as a front-end to the PTAS. The performance of the OSM and the validity of the FCC formulation are studied through simulation experiments.

The second part of this thesis presents an efficient algorithm to compare evolutionary trees using the quartet metric. Different evolutionary hypothesis arises when different data sets are used or when different tree inference methods are applied to the same data set. Tree comparisons are routinely done by biologists to evaluate the quality of their tree inference experiments. The quartet metric has many desirable properties but its use has been hindered by its relatively heavy computational requirements. We address this problem by giving the first $O(n^2)$ time algorithm to compute the quartet distance between two evolutionary trees.

Acknowledgments

It is indeed true that a piece of scientific research—regardless of its size and significance—is never a single person's effort. Obviously my supervisors Paul Kearney and Ming Li has played a significant role in my research work for which I am very grateful. They introduced me to bioinformatics back in 1998 and has continue to inspire and aspire me to explore and investigate various areas in the field. Perhaps an itemized list of their help and advice during the course of my graduate studies is too long to include here. But what I will remember the most is the many exciting late night discussions with Ming, as well as Paul's many thorough advice on research and his careful reading of various manuscripts.

I would also like to thank Professor Ian Munro and Jonathan Badger for taking the time to read my thesis. Their valuable feedback has improved my thesis a great deal. I am also grateful for the many advice given by Professor Chrysanne DiMarco. She introduced me to research and has helped me to decide what I should pursue in graduate school. Additionally, I would like to thank the following people for helpful discussions: David Bryant, Mike Hu, Naomi Nishimura, and En-hui Yang.

I am lucky to have many great colleagues and friends, who have helped and supported me throughout my studies. Jonathan Badger has taught me biology, history and even some linguistics. My officemate Haoyong Zhang has kept me company during many late nights of work and I would also like to thank him for many helpful discussions

on research and life in general. The friends at Minota Hagey residence: Kai, Jennifer, Hermia, Paul, Richard, and Raymond has provided me with lots of fine food, relaxing hours of entertainment, and very interesting intellectual discussions for which I miss very much and would like to thank them all. Many people in the CS department have given me help in one form or another, but it would be too numerous to thank each individually, so I thank them collectively.

I am grateful for the financial support I received throughout my studies. This thesis is completed under the financial assistance of a NSERC PGS Scholarship, a University of Waterloo Graduate Scholarship, a Mathematics Faculty Scholarship, a CITO Fellowship, and several Teaching and Research Assistantships from the Department of Computer Science.

Finally, I would like to thank my family: my parents, my grandmother, and my sister. They have never restricted my freedom to study and explore and have always been supportive in whatever I am doing throughout these years. This thesis is dedicated to them for their love and support.

Contents

List of Tables

List of Figures

Prologue

Once upon a time, a self-replicating molecule emerged on earth. Due to the self-replicating capability of this molecule, it had the ability to make many copies of itself, and those copies in turn could replicate and hence produce more copies. This process quickly increased the abundance of this molecule on earth. But as the replication process continued, different forms of this molecule emerged due to the imperfect nature of the replication process. This was the first sign of the forces of evolution at work. Billions of years have past and it is indeed amazing to realize today that the descendents of that molecule includes complex beings such as ourselves! If, billions of years ago, there existed a recording device which somehow recorded the entire process of evolution from the emergence of that first self-replicating molecule to today, it would be for sure a crowd-drawing show. Of course such a device did not exist. What we need today is a telescope that allows us to look back through time and discover how human beings and other species on earth are related by evolution. Fortunately, evolution has been kind to us and has left many tell-tale signs. And indeed, recent advances in fields such as computer science and mathematics has made the construction of such a time telescope more than just fantasy.

Chapter 1

Introduction

The emerging field of bioinformatics has received an enormous amount of attention in recent years as more and more biological data are produced in laboratories around the world. These include genomic sequence data, gene expression profiles, protein structure information, as well as other clinical-related information. For example, a rough draft of the human genome is already available, which contains three billion nucleotides and many other genome sequencing projects are also completed or underway. The amount of data is rising at an exponential rate and has far out-paced our ability to analyze them. This problem induces the need to combine techniques from computer science and the physical sciences, including mathematics, and physics, with the biological sciences to explore and mine this information.

One of the most widely studied problems in bioinformatics is the inference of phylogenetic trees. A phylogenetic tree (see Figure 1.1) depicts the evolutionary relationship among a set of species and other important evolutionary information such as the occurrences of speciation events and the length of lineages. Besides the intrinsic scientific importance of studying the evolutionary history of organisms on earth, it has a wealth of other applications. For instance, it helps biologists to better align protein and DNA

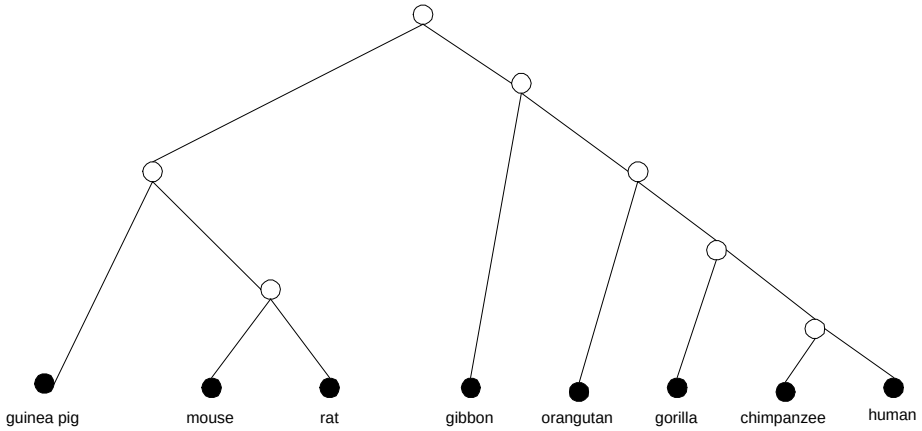

Figure 1.1: A phylogeny of eight species.

sequences [35]. It also helps in functional studies—knowing the function of a particular gene in an organism says a lot about a similar gene in a closely related organism. It is, of course, of great scientific interest to study the process of evolution itself. A phylogenetic tree tells us a great deal about the mechanisms of evolution as well as various evolutionary events and their causes. Please see [2] for sample applications of phylogenetic trees.

A phylogenetic tree T is characterized by its topology, the weight (or length) of its edges, the root vertex, and the label of the leaves. More formally, an *unrooted phylogenetic tree* is a acyclic connected graph (V, E) where degree-1 vertices are labeled bijectively by a set S. T is also said to be *labeled* by S. A *fully resolved* phylogenetic tree only has degree-3 vertices and leaves, otherwise the tree is *unresolved*. Furthermore, there is a function $f : E \to \mathcal{R}$ that weights the edges. A *rooted phylogenetic* tree is an unrooted tree with a single degree-2 vertex that is designated as the root. In this thesis, our focus will be on the inference of the topology of a phylogenetic tree. The problem of determining the weight and root of a phylogenetic tree is well-studied and is relatively easy compare to the inference of the topology.

To infer a phylogenetic tree, a blue-print of each organism is needed. Several types of such blue-prints can be used. In the advent of molecular sequence data, a majority of biologists use DNA and protein sequences as the signature of individual organisms as opposed to the older approach, where a set of *morphological* features is used. The use of molecular data, which was first proposed by Zuckerkandl and Pauling [76], has many advantages. The obvious merit being that physical traits or phenotypic measurable quantities are in fact just manifestation of the genome of an organism, or its *genotype*. Using sequence information in theory covers all measurable morphological features of an organism and hence gives more information to infer the evolutionary history. Perhaps more importantly, gene and protein sequences are the direct product of evolution and hence gives us clues about the evolutionary processes that produced them [47]. In addition, proteins, the functional building block of life, is the direct product of genes. The evolutionary history of genes provides information on how functions are evolved [47]. Lastly, it is possible to objectively evaluate an experiment based on the data it uses. Whereas when morphological data are used, it is often very subjective regarding whether the right traits have been used and whether the morphology set picked is large enough to answer the questions asked. In experiments that involve micro-organisms such as bacteria, physical traits are hard to measure and quantify. The use of sequence data greatly remedies this situation. For instance, a novel branch of archaebacteria was identified using sequence information [67].

In practice, a phylogenetic tree not only can model evolution on the organismal scale, but it is also frequently used for individual genes or particular segments in the genome. Since the evolutionary force acts directly on gene sequences, the evolutionary history of a set of genes is of great importance. For many studies, the input to the phylogenetic inference method is a set of protein or gene sequences from each organism. The resulting tree is called a *gene tree*, whereas the tree in Figure 1.1 is called a *species tree*. Due to various

biological reasons, a gene tree in general does not accurately reflect the evolutionary history at the species level (one obvious reason being that the trees obtained from different genes are not necessarily consistent.) [10]. As more genomes are sequenced, methods that utilize the entire genome are in active research (for example, see [41, 63, 49, 36]).

Clearly, the inference of phylogenetic trees is not exclusively biological, but also involves mathematical modeling and algorithm design. For example, given two genes, how do we determine their similarity? One way is to use a model of evolution, where mathematical modeling is required. Given a model of evolution, efficient algorithms are needed to calculate the pairwise evolutionary distances. Various algorithms that are based on different assumptions were developed to infer the phylogeny given pairwise distance information. For instance, a method called Neighbour-Joining [60] is use widely by biologists. Alternatively, we can totally abandon the use of pairwise distance information, and infer the tree directly from sequence data, which often require fast algorithms to search through the tree topology space for the optimal tree(s) with respect to certain criterion.

The success of a phylogenetic method or paradigm is greatly determined by its computational merit and the underlying model of evolution. In fact, we currently have relatively simple models of evolution, but many formulations based on them almost all lead to intractable computational problems. Methods that utilizes unrealistic assumptions often give unsatisfactory results. Therefore, it is vital for computer scientists to design efficient, robust, and accurate phylogenetic inference methods. This thesis takes one step in that direction.

Due to the fact that the main audience of this thesis will be computer scientists and mathematicians, we will begin with a brief introduction to the biological background and concepts needed to understand this thesis. There are two main components in this thesis. Chapter two first briefly surveys the computational aspects of phylogenetic inference and then presents an approximation algorithm that solves a variation of the well-known

character compatibility (CC) formulation of the phylogenetic inference problem, which is NP–hard. The algorithm is a so-called polynomial time approximation scheme (PTAS), which can find a solution with arbitrary accuracy in polynomial time. Simulation studies were performed to evaluate the biological relevance of this particular formulation of the problem. In addition, an algorithm called Ordinal Split Method is also presented, which is very fast and serves as a front-end to the PTAS. Simulation studies were also performed to evaluate the method.

The second part of this thesis presents the first $O(n^2)$ algorithm to compare evolutionary trees based on the quartet metric. Systematic comparison of evolutionary trees is very important since inconsistencies (i.e. different trees) often arise when different data sets are used, also when different inference algorithms are used. A sensitive and accurate method to compare trees can help the biologists to evaluate the quality of the result as well as the evolutionary divergence of different genes. The quartet distance metric has some very nice properties, but has been hindered by its relatively heavy computational requirements. Our algorithm is an order of magnitude faster than the best known method and hence facilitates further the use of the quartet metric for large phylogenetic trees.

1.1 Some Biology

1.1.1 DNA/RNA and Protein Sequences

Deoxyribonucleic acid (or DNA for short) molecule is the genetic material of all living organisms except for some viruses. [1] The molecule consists of a phosphate back-bone chain with molecules called *nucleotides* or *bases* attached. There are four types of nucleotides, they are adenine (A), cytosine (C), guanine (G), and thymine (T). Furthermore, due to

[1] What is presented here is a simplified version of "real" molecular biology, please consult [51] for a detailed treatment.

the difference in their molecular base structure, A and G are classified as *purines*, while C and T are *pyrimidines*. Figure 1.2 depicts the basic molecular structure of DNA. It is a double helical structure with nucleotides attached at the inside of the phosphate chain. Note that there are two phosphate chains running opposite of each other (i.e. it is a *double-stranded* molecule) with nucleotides attached. Hydrogen bonds are formed between adenine and thymine, as well as cytosine and guanine. The molecular structure of the two ends of a phosphate back-bone chain is different, one is called the 5' end and the other is the 3' end. The sequence of a DNA molecule is typically obtained by reading the nucleotide content from the 5' end to the 3' end. Note that a DNA sequence is just a string with the alphabet $\{A, C, G, T\}$. Every single cell of an organism consists of a set of DNA molecules and their sequence content is the *genome* of an organism. The genome encodes via the DNA alphabet all essential information for the functioning of the organism.

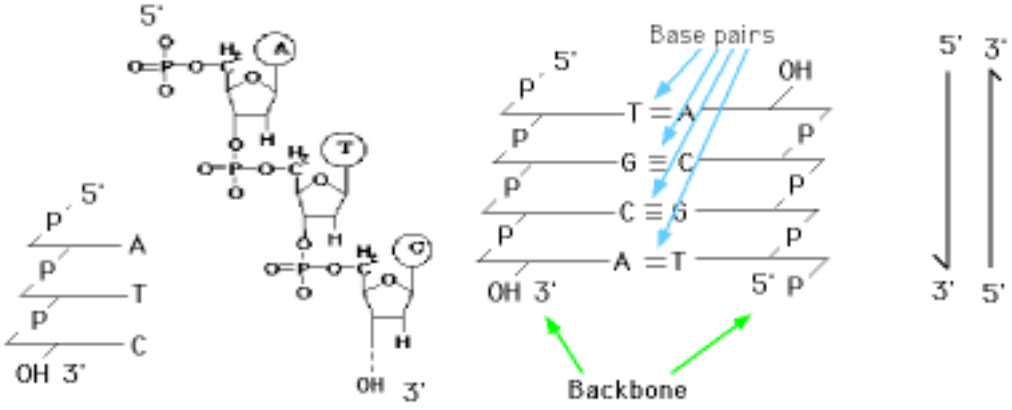

Figure 1.2: The basic structure of DNA. Courtesy of Bruce Walsh [70].

While DNA stores genetic information, the building block of life is in fact protein molecules. A protein consists of a *polypeptide* chain where different *amino acid* molecules are attached. There are twenty different common kinds of amino acids and they are listed

in Table 1.1. Again, a protein sequence can be represented by a string as in DNA except the alphabet size is twenty instead of four.

The genome is an fixed entity of an organism, whereas proteins are dynamic. That is, different proteins are produced under different circumstances. For instance, some proteins are essential for the functioning of muscle cells, but not blood cells. Hence, proteins are produced on-the-fly and their production is generally believed to be a two stage process. To understand this process, it is important to know that proteins are in fact coded by DNA sequences. Certain segments of an organism's genome, which are called *genes* or *protein coding regions*, is responsible for the coding of proteins. The coding scheme is very simple: every three nucleotides, or a *codon* codes for a single amino acid [2]. Hence, one can read off the target protein sequence given a protein coding DNA sequence. Table 1.1 outlines this code. For example, the protein sequence

```
Ile Cys Lys Ala Val Leu Ile
```

can be coded by RNA sequence

```
AUA UGU AAG GCA GUC UUA AUA.
```

The first stage of protein production is called *transcription*, where double-stranded DNA are opened up and the coding region is read by a biochemical molecule and the corresponding single-stranded molecule called ribonucleic acid (RNA) are produced, which has the same sequence content as the original DNA sequence but with T replaced by U. In the second stage, called *translation*, the RNA sequence is then transported to the ribosome of the cell where proteins are produced based on the coding rules in Table 1.1.

[2]The first codon is typically a *start codon*, to signal the start of a protein coding region, while the last codon is a *stop codon*, to signal the end of the coding region

Amino acid	Abbreviation	Possible Codons
Alanine	Ala or A	GCU, GCC, GCA, GCG
Leucine	Leu or L	UUA, UUG, CUU, CUC, CUA, CUG
Isoleucine	Ile or I	AUU, AUC, AUA
Valine	Val or V	GUU, GUC, GUA, GUG
Proline	Pro or P	CCU, CCC, CCA, CCG
Phenylalanine	Phe or F	UUU, UUC
Tryptophan	Trp or W	UGG
Methionine	Met or M	AUG
Glycine	Gly or G	GGU, GGC, GGA, GGG
Serine	Ser or S	UCU, UCC, UCA, UCG, AGU, AGC
Threonine	Thr or T	ACU, ACC, ACA, ACG
Tyrosine	Tyr or Y	UAU, UAC
Cysteine	Cys or C	UGU, UGC
Asparagine	Asn or N	AAU, AAC
Glutamine	Gln or Q	CAA, CAG
Aspartic acid	Asp or D	GAU, GAC
Glutamic acid	Glu or E	GAA, GAG
Lysine	Lys or K	AAA, AAG
Arginine	Arg or R	CGU, CGC, CGA, CGG, AGA, AGG
Histidine	His or H	CAU, CAC

Table 1.1: The 20 possible amino acids and their possible coding condon. Courtesy of Haoyong Zhang [75].

1.1.2 Sequence Evolution and Phylogenetic Trees

While the effects of evolution are clearly evident in the macroscopic level such as morphological traits as discovered by Darwin, it can also be detected in the molecular level. For example, closely related species typically have slightly different forms of the same gene, where the slight variations are mainly due to the force of evolution. An important function of genetic information carriers such as DNA is to pass heredity information from one generation to the next. Even though the mechanism of copying genetic information is highly precise, inevitably there must be some imperfections (or *mutation*) in the process. Consequently, these mutations plus other genetic effects (such as cross-over events) give

rise to new types of blue-print of life, or new *genotypes*. It is very unlikely that these mutations would give the mutated organism an evolutionary advantage. However, if they do, these mutations are likely to be kept and after a long period of time, a *speciation event* can happened when the group of mutated individuals can no longer exchange genetic material with groups not sharing the mutations. A speciation event gives rise to two distinct species and as time progresses, many such speciation events can happened and hence diverse kinds of organisms proliferate on earth today.

The evolutionary history of a set of molecular sequences can be modeled by a phylogenetic tree as illustrated in Figure 1.1. A binary tree suffices in most situations since biologists believe that in nature, simultaneous speciation events seldom happen within a single group of species [54]. Only the leaves of the phylogenetic tree are labeled, which are present-day sequences. All internal nodes represent intermediate sequences that have been present in the evolutionary history. If the tree is rooted, then the root sequence is the common ancestor of all leave sequences. The tree can also be unrooted when the direction of evolution is not of importance. But of course, such a tree only denotes the relationship among the leave sequences and the exact evolutionary history are missing. The edges of the tree can be weighted and can be used to denote evolutionary time, mutation rate, or the total amount of mutation along that line of evolutionary change. Furthermore, if a phylogenetic tree is believed to denote the true evolutionary history of the sequences under study, it is called the *phylogeny* of the sequences. Two genes are said to be *homologous* if their ancestral sequence are the same in the phylogeny. That is, the two sequences were evolved from the a single ancestor sequence along the phylogeny.

A phylogenetic tree is not only a very nice model for the evolutionary history of molecular sequences, it can also be used to model the evolutionary history of a variety of objects. For example, the evolutionary history of human languages is of great interest and a phylogenetic tree serves well as a modeling tool. Of course, the evolutionary history

of a set of species themselves (i.e. not one of their genes or proteins) can be modeled by a phylogenetic tree as well. Traditionally, biologists used morphological data to perform phylogenetic analysis, individual species are the main objects under study and hence the results are *species trees*. With the availability of molecular sequence data, most studies are performed using single genes or proteins, which gives *gene trees*. However, many biologists make assertions about species relationships from gene trees. Some researchers have proposed the use of whole genomes to infer more accurate species trees. But some argued that due to the different rate of evolution among individual sequences within the genome, it is not very meaningful to use the whole genome. While others disagree and are searching for better whole genome analysis techniques.

1.1.3 Mutations

As discussed above, the ancestral sequences evolved along the edges of the phylogeny and mutations occurred during the process. Mutations can be classified into two main categories: *gene mutations* and *genomic mutations*. Gene mutations occurred within a small locus of the genome and have only local effects such as alternating part of the structure of the protein that the gene encodes. While genomic mutations typically affect a large portion of the genome. It can change the order of the genes as well as inserting new sequences into the genome. While genomic mutations are very important for whole genome analysis, variations among homologous genes (and hence the proteins they encoded) are mainly due to gene mutations.

There are four types of gene mutations that are of interests in phylogenetic studies: substitutions, deletions, insertions, and inversions. A substitution occurs when a nucleotide is replaced by another. For example, the DNA sequence ATATGTACA becomes ATACGTGCA after two substitutions. An insertion event occurs when a new nucleotide is being added to a sequence. While a deletion event refers to the removal of a nucleotide

from the sequence. For example, ATACA changing to ATATGTACA is due to an insertion and ATATGTACA changing to ATACA is due to a deletion. Lastly, an inversion event reverses the ordering of a sub-sequence. For example, the sequence ATAGGAACCA becomes ATAAAGGCCA after an inversion event on the sub-sequence GGAA.

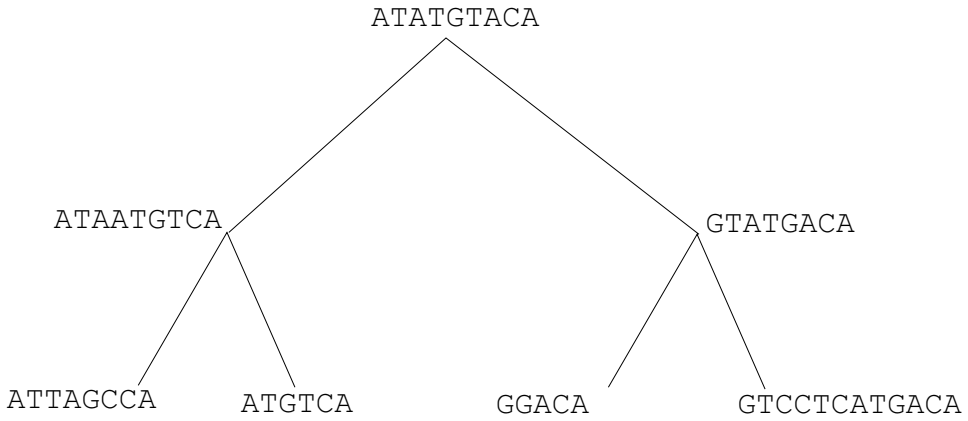

Figure 1.3: A phylogeny of four sequences. Courtesy of Haoyong Zhang [75].

For example, Figure 1.3 is a phylogeny of four sequences, and Figure 1.4 shows a possible history of mutations.

For more details on molecular sequence evolution, please consult *Molecular Evolution* [54].

1.2 Phylogenetic Analysis

The present-day molecular sequences we observed has undergone the evolutionary process for millions of years. The main goal of phylogenetic analysis is to infer the history of their evolution through the reconstruction of their phylogeny. More generally, phylogenetic analysis infers the evolutionary history of a set of entities. The entities can be a set of genes, the proteins they encode, a set of species, or even just some specific region on the

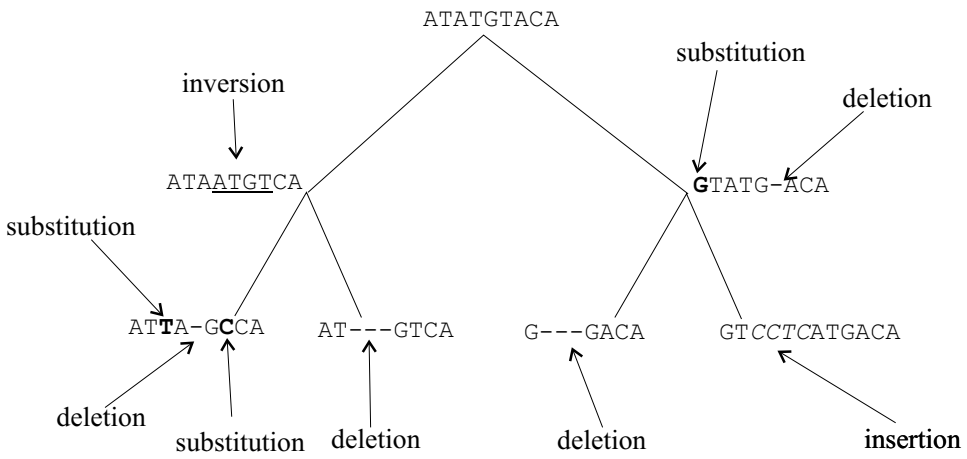

Figure 1.4: A possible mutation history of the phylogeny in Figure 1.3. Courtesy of Haoyong Zhang [75].

genome. More formally, let S be the set of entities, where $|S| = n$, the problem is to infer a phylogenetic tree T labeled by S. Obviously more information about each of entities have to be provided in order to infer a tree. Usually a $n \times m$ matrix M is provided where each row denotes one entity from S. Each row vector encodes information about each entity. Two types of such information are typical. Recently, molecular sequence data are used in most studies due to their abundance and its advantage over morphological data [69]. Each column in M then denotes a particular base position of the input sequences and m would be the length of the sequences. In cases where the input sequences are of different length, they need to be *aligned* where gaps are inserted to make their lengths equal. For details on sequence alignment and related algorithmic issues, please consult [35]. The input data could also be morphological data. In this case, each column of M denotes a morphological feature. For example, a morphological feature could be the shape of the wings of a species or the number of fingers they have. The value in $M(i, j)$ denotes the value of morphology j of entity i.

To accurately infer the phylogeny, there must be some guideline to evaluate what

the true tree should be like. Usually an optimization criterion is formulated so that a solution can be evaluated. For example, one of the most popular method is the Maximum Likelihood formulation [28] where the resulting phylogeny is the most likely among all possible phylogenetic trees for S under a particular model of evolution. This method is model dependent, models such as Jukes Cantor [43] and Kimura-2-parameter [50] are often used. Chapter 2 contains further details and more example on different formulations of the phylogeny reconstruction problem.

The reconstruction of a phylogeny usually involves the inference of three components: the topology, the edge lengths, and the root. Finding the root and edge lengths are well-solved. For instance, a common strategy to determine the root is to mixed S with one or a set of so-called *outgroup* entities and then perform the analysis. The outgroup entities are chosen such that it is known *a priori* that they are evolutionarily far apart from the entities in S. Hence, the resulting tree can be rooted at the point where the outgroup subtree is attached. The rest of this thesis will focus on the inference of the topology of the phylogeny.

In the following sections, detailed definitions that are of relevance for the rest of the thesis will be given.

1.2.1 Bipartitions, Splits and Edges

Consider the unrooted Figure 1.5. If we remove edge e_4 from the tree, we end up with two trees where one is labeled by $\{f, g, h\}$ and the other is labeled by $S - \{f, g, h\}$. More formally, let S be the set of leaves of a tree T. A **bipartition** is a pair of non-empty sets (X, Y) such that $X \cup Y = S$ and $X \cap Y = \emptyset$. A bipartition (X, Y) can uniquely represent an edge e in T if the removal of e from T (i.e. $T - e$) results in two trees where one is labeled by X and the other Y. (X, Y) is often referred to as an *edge*, a *split*, or a *bipartition* of T and we write $e = (X, Y)$.

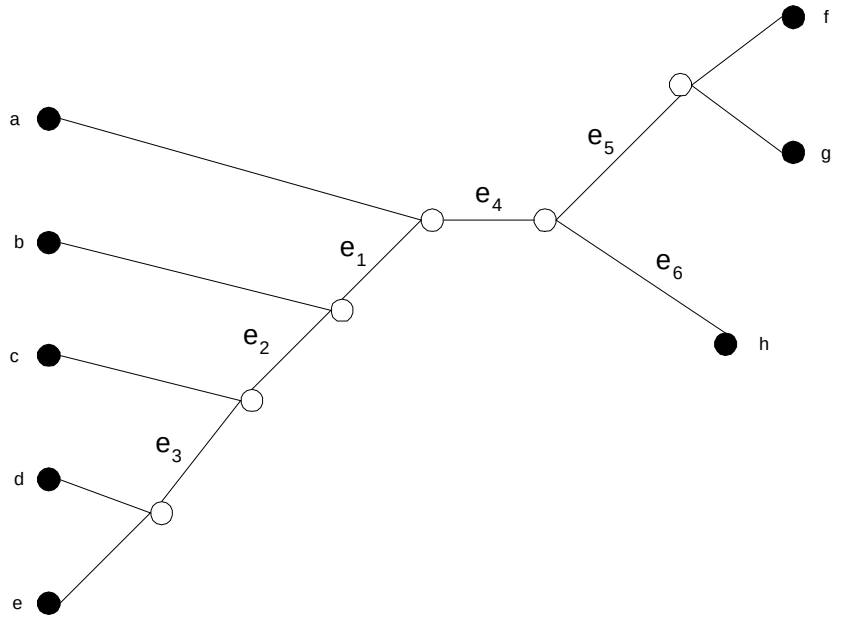

Figure 1.5: An unrooted, unweighted tree topology.

For instance, in Figure 1.5, we have,

- $e_1 = (\{a, f, g, h\}, \{b, c, d, e\})$;
- $e_2 = (\{c, d, e\}, \{a, b, f, g, h\})$;
- $e_6 = (\{h\}, \{a, b, c, d, e, f, g\})$.

Observe that e_6 is attached to a leave and we called such an edge *trivial*. More formally, an edge (X, Y) is called a trivial edge if $|X| = 1$ or $|Y| = 1$. All internal edges, that is, edges with no leaves attached are always non-trivial, while edges with leaves attached are always trivial.

An unrooted tree T with n leaves is uniquely characterized by its $n-3$ internal edges. We denote this set of non-trivial splits of T as $splits(T)$. Given S, the topology of T can be obtained if we can accurately infer $splits(T)$ as it was shown that T can be constructed

from $splits(T)$ in linear time [8]. Furthermore, given a set of splits α, we call it *tree-like* if there exist a tree T such that $splits(T) = \alpha$. The above concepts are very important as it reduces the inference of the topology of a tree to inferring its set of splits. This is the underlying basis for formulations such as character compatibility and the perfect phylogeny problem.

The term *character* is often used interchangeably with edges or bipartitions in some literatures. The origin of this practice dates back to when phylogenies were mainly constructed using morphological data. Consider if the characters are binary (i.e. there is only two possible value for each character) and under certain conditions (see Chapter 2), each edge (X, Y) would denote two sets of species, one having a particular character and the other not.

The concept of *compatibility* often arises in character-based phylogenetic analysis. There are two important concepts of compatibility: compatibility among splits and compatibility among a tree and a set of splits. A set of splits B is *compatible* if there exists a tree T such that for every $\alpha \in B$, α is an edge of T. Finally, a set of splits B is compatible with a tree T if $B \cup splits(T)$ is compatible. Hence, given a phylogenetic tree T, $splits(T)$ is always compatible.

1.2.2 Quartets

Given a tree T, any subset of S *induces* a tree topology. For example, for the tree in Figure 1.5, the set $\{a, b, f, g, h\}$ induces the tree topology in Figure 1.6. Any size four subset of S is called a *quartet* and the tree topology it induces is a *quartet topology*. Figure 1.7 illustrates the four possible quartet topologies. If labels a, b is separated from c, d, we write it as $ab|cd$. Given a tree with n leaves, there are a total of $\binom{n}{4}$ quartet topologies and we denote the complete set of quartet topologies of T as $Q(T)$ or just Q when the context is clear. It is well-known that Q is unique to T and given Q, T can

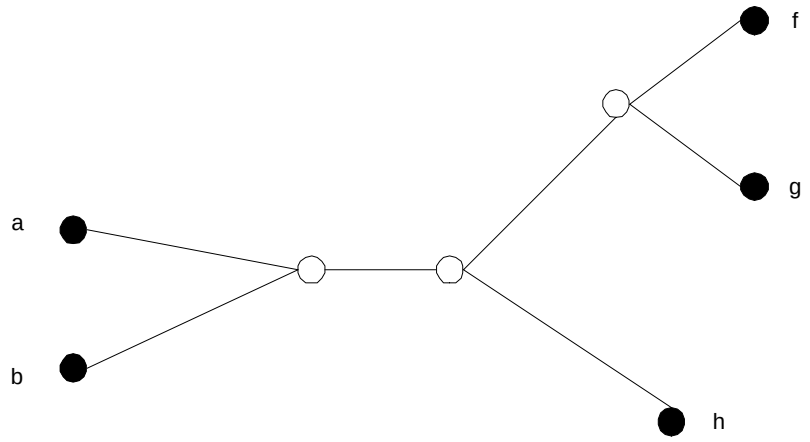

Figure 1.6: The tree induced by $\{a, b, f, g, h\}$ from Figure 1.5.

be determined in polynomial time [8]. This entails another paradigm for inferring the topology of T: first infer the topologies of the quartets and then combine the quartet topologies to form the final tree topology. Inferring the topology of quartets is much less computational intensive and hence very accurate methods can be employed. The research focus of the quartet paradigm is mainly on the combining stage since rarely the quartet topologies inferred are compatible with each other. Please consult [47] for more details on the computational aspects of the quartet method.

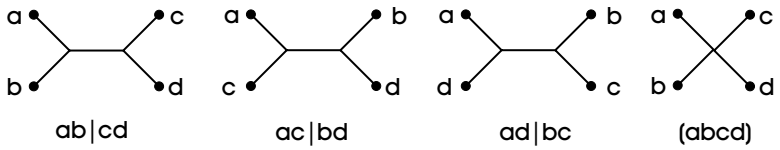

Figure 1.7: The four possible quartet topologies.

Given a split $e = (A, B)$, a quartet topology can be formed by picking two labels from A and two labels from B. A total of $\binom{A}{2} \times \binom{B}{2}$ quartet topologies can be formed this way and we denote the set as $Q_e = \{ab|cd \mid a, b \in A, c, d \in B\}$. Furthermore, Q_e is said to be

induced by e.

The quartet concept is not only central to the quartet paradigm of reconstructing trees, but it is also a very useful tree comparison metric. We will investigate the problem of comparing trees using quartets in Chapter 3.

Chapter 2

Inferring Phylogenetic Trees—An Approximation Algorithm for Character Compatibility

2.1 Introduction

We have introduced the goal and some basic concepts of phylogenetic analysis in the previous chapter. In this chapter we will explore deeper computational issues in phylogenetic analysis. We will first briefly survey several popular paradigms for the reconstruction of phylogenetic trees. This is follow by our new result: an approximation algorithm that solves a variation of the Character Compatibility (CC) formulation. We called this variation Fractional Character Compatibility (FCC) and have performed simulation studies to evaluate its biological relevance by comparing it to the CC formulation. The results of the simulation study is reported. The approximation algorithm presented is a so-called polynomial time approximation scheme (PTAS) where the optimal solution can be ap-

	Characters			
Taxon	1	2	3	4
A	0	0	1	0
B	0	0	1	1
C	1	1	0	0
D	0	0	1	1
E	0	1	0	0

Table 2.1: A character state matrix that emits a perfect phylogeny.

proximated with arbitrary accuracy in polynomial time. To effectively utilize this PTAS, we need an algorithm to generate a set of bipartitions based on the input sequences. We have designed a method called the Ordinal Split Method (OSM) that serves as a front-end to the PTAS. We also performed simulation studies to evaluate the OSM.

2.2 A Survey of Phylogenetic Reconstruction Methods

Before the advent of computers, phylogenetic analyses were mainly done by hand through intuition and experience [29]. In addition, the old school of phylogenetic analysis favours the use morphological data since molecular sequence data were scarce and unreliable [69]. Biologists typically pick certain important morphological features of the species under study and derive their phylogeny by analyzing the differences in these specific traits. Recall that in a character state matrix M, each column can denote a particular morphology under study and each row is a species. Hence, the j^{th} trait of species i is $M(i, j)$. The morphological values can be encoded in any reasonable way. For example, it can be binary to denote the presence or absence of the trait.

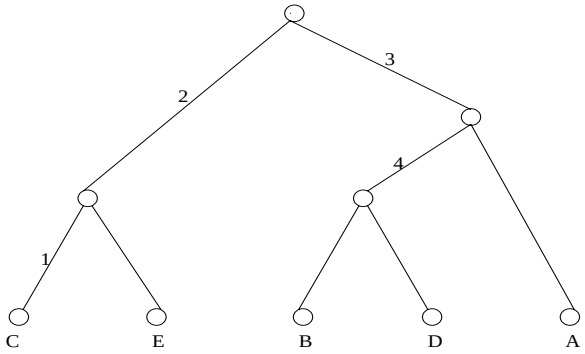

Figure 2.1: A perfect phylogeny from the character state matrix in Table 2.1. Edge labels indicate which character is under transition.

2.2.1 The Perfect Phylogeny Problem

The phylogeny reconstruction problem was not formulated in a formal fashion until the 50's, when Hennig [37] proposed a systematic way of finding a phylogeny for a set of species given a character state matrix. His method was to process one character at a time and group those species that have the same character state to form a so-called *monophyletic group*. This process will be repeated for each character and a tree will be constructed based on these groups. Consider the character state matrix in Table 2.1. Character 1 suggests that C should be in a group of its own; character 2 suggests $(\{C, E\}, \{A, B, D\})$ and so on. The tree in Figure 2.1 realizes this matrix. Note that for each character, the transition from "0" to "1" occur in only one edge of the tree and there is no edge that corresponds to a transition from "1" to "0".

Definition 1 *Given a label set S and a character state matrix M. A k-state character c_i induces the partition $P_i = (s_1, s_2, ..., s_k)$ of S such that each s_i where $1 \leq i \leq k$ is a set of species sharing the same state in c_i. Furthermore, $S = \bigcup_{1 \leq j \leq k} s_j$ and $s_i \cap s_j = \emptyset$ for all i and j such that $i \neq j$.*

Hennig's method first forms all P_i for each character i and then searches for a tree that realizes all P_i. Intuitively, a tree realizes a partition $P = (s_1, s_2, \ldots, s_k)$ if each s_i belongs to a different subtree. Clearly, there are times this approach would not produce a solution. Consider the instance $S = \{a, b, c, d, e\}$. If a binary character c_1 induces the split $(\{a, b, c\}, \{d, e\})$, while another character c_2 induces $(\{a, d\}, \{b, c, e\})$, a tree that realizes both partitions are not possible. Or by the concept introduced in 1.2.1, these two splits are not compatible.

Hennig's method was based on the following assumptions:

1. No *convergence* (or *parallel evolution*) events occurred in nature. For example, the eyes of humans and octopus are extremely similar in structure and function but they are not genetically close. Hence their eyes were evolved independently of each other. Most biologists believe that such events are rare for morphological features. However, convergent DNA sites are very common [54].

2. Evolution is irreversible. That is, a character can not change back to one of its ancestral states. Again, this is very common for sequences.

Given the above assumptions, the resulting phylogeny will have the following properties: both internal nodes and leaves that share the same state s with respect to a character c must form a subtree. This implies that a transition from some state x to another state s in c can only occur in one edge of the phylogeny. No other edge should correspond to a transition to state s in c. A phylogeny having these properties is called a *perfect phylogeny*. This leads to the **Perfect Phylogeny** problem:

Perfect Phylogeny (PP)

Instance: A label set S and a $n \times m$ character state matrix M, $n = |S|$ where the characters have r states

Problem: Find a perfect phylogeny labeled by S

The **PP** problem is in general NP–hard [64]. However, polynomial time algorithms exist for several restricted versions of the problem. For instance, if an ordering exists for the character states, the problem can be solved efficiently in polynomial time [56]. In the special case where the characters are binary, it can be solved in $O(nm)$ time [34] for both ordered and unordered characters. The algorithm is based on the following observations (assume that "0" is the ancestral state):

Let s_{i1} be the set of species having state "1" with respect to character c_i. Similarly, let s_{j1} be the set of species having state "1" with respect to character c_j. A binary character state matrix emits a perfect phylogeny if and only if each pairs of induced sets $P_i = (s_{i0}, s_{i1})$,$P_j = (s_{j0}, s_{j1})$ for $i \neq j$ and $1 \leq i, j \leq m$ satisfy one of the following:

- $s_{i1} \subseteq s_{j1}$
- $s_{j1} \subseteq s_{i1}$
- $s_{i1} \cap s_{j1} = \emptyset$;

$O(nm)$ is a lower bound for the problem since every entry in the matrix needs to be visited at least once. Hence the above algorithm is optimal.

The **PP** formulation does not have much practical value since real data rarely emit perfect phylogenies. Although there are research interests in efficient algorithms for the **PP** problem, they are mainly of theoretical importance. However, the study of **PP** leads to other more practical formulations of the phylogeny reconstruction problem, which we will discuss in more detail.

2.2.2 Maximum Parsimony

One possible relaxation of **PP** is to find a tree that minimizes the number of parallel and reverse evolution events. This leads to the maximum parsimony (MP) formulation, which was first proposed by Edwards and Cavalli-Sforza [17].

As mentioned in the previous section, some biologists believe that parallel and reverse evolution rarely occur in nature. Thus, every such event in the resulting phylogeny corresponds to a hypothesis made in account of the event. The MP approach tries to minimize the number of such hypothesis. More philosophically, it follows the principle of Occam's Razor [53]: the simpler the explanation, the better. Felsenstein [27] among others [24] has also tried to use statistical arguments to investigate the plausibility of the MP formulation.

There are several variations of the MP formulation. Each of them has different biological assumptions that affects the definition of the criterion used to evaluate the optimality of a tree. One of the most well-studied formulation tries to minimize the total number of state changes in the tree. State changes are measured by a metric. For example, Hamming distance is an instance of such a metric. The number of state changes between the sequences `ACGT` and `ACTT` is 1 as measured by the Hamming distance since there is only one nucleotide difference between the two sequences. In general, such a metric can be defined as a function d that maps two state vectors (e.g. two rows in a character state matrix) to a positive real number.

Given a tree $T = (V, E)$, if we compute the value of d for each edge in T, we obtain the parsimony score $S_p(T) = \sum_{e \in E} d(e)$.

In general, the optimal tree (or the most parsimonious) T_{opt} is the one with the minimum S_p. This corresponds to a tree that contains the least number of parallel and reverse evolution events. If this is not true, we can find another tree T such that it has

less parallel and reverse evolution events. But each parallel or reverse evolution event adds more state changes to the tree (assuming the distance metric is defined properly), hence T is more parsimonious than T_{opt}. This is a contradiction as T_{opt} has the smallest S_p among all possible phylogenies.

Computational Issues

The Maximum Parsimony formulation can be formalized as an optimization problem:

Maximum Parsimony (MP)

Instance: A label set S with a $n \times m$ character state matrix, a parsimony metric S_p

Problem: Find a tree T labeled by S with the minimum $S_p(T)$

MP is known to be NP–hard. There is quite a number of similarities between **MP** and the well-studied Steiner tree problem [72], which is also NP–hard. The undirected Steiner tree problem is the following:

Steiner Tree Problem (ST)

Instance: A weighted undirected graph $G = (V, E)$ with $|V| = n$, $|E| = m$, edge cost function $c : E \rightarrow \mathcal{R}^+$, $Z \subseteq V$ where $|Z| = p$, and a positive integer B

Question: Does a tree $T_z = (V_z, E_z)$, where $V_z = Z$, $E_z \subseteq E$ and $\sum_{e \in E_z} c(e) \leq B$ exist?

The NP–Completeness proof for MP is also related to the Steiner tree problem, which can be found in [33]. The problem formulation in that paper is slightly different than what we are studying here: the most important one is that labels in S are not required to be the leaves of the resulting phylogeny.

A number of approximation algorithms exist for ST. However, none can be easily

adopted for the MP problem. Let T_{opt} be an exact solution to an instance of **MP**. Let I be the set of internal nodes of T_{opt}, where $|I| = |S| - 2$ (there are $n-2$ internal nodes for a binary tree with n leaves). Let's assume that I and an instance of **MP** are given, An instance of **ST** can be constructed as follows:

1. Construct G: $V = I \bigcup S$, form E by connect every node in V to all nodes in V except itself.

2. Let Z=S and $c = D_p$

A solution to the **ST** instance above would return a tree T_s that minimizes the total edge cost while all labels in S are connected. This seems to satisfy the maximum parsimony criterion. However, this is still not a solution to the corresponding **MP** instance since all members of S are leaves of T_s. This imposes more constraint on the solution of **ST**. In addition, the set of internal nodes I were assumed to be given, which is not realistic since the internal vertex space is exponential in size and it is non-trivial to find the optimal set other than search through all combinations. This is the reason why approximation algorithms for **ST** can not be trivially applied to approximate **MP**.

It turns out that even restricted versions of **MP** are NP–Complete. For example, **MP** with two characters each of which has states on an integer scale (also known as the Wagner Network problem), is known to be NP–Complete [29]. The proof is based on the observation that the Rectilinear Steiner Problem (RSP), which is known to be NP–Complete [31], can be reduced to this problem. In addition, **MP** with an arbitrary number of binary characters has also been proven to be NP–Complete [29].

The inherent intractability of the maximum parsimony formulation ceases our hope for exact efficient algorithms unless $P = NP$. To the best of our knowledge, there is no known approximation algorithm with performance guarantees. Biologists are resolved to

use either heuristics, which do not have performance guarantees and thus can lead to erroneous results. An alternative way would be to use use exhaustive search techniques, which are only practical for a small number of sequences [69]. The use of branch-and-bound techniques imposed constraints on the optimization criterion. In practical situations, it can handle data sets with 20 or more labels depending on the speed of the machine as well as the efficiency of the implementation [69].

2.2.3 Character Compatibility

The character compatibility formulation is another approach to cope with the restrictions of the perfect phylogeny formulation. Instead of accepting parallel and reverse evolution events, the largest possible subset of the characters are picked such that a perfect phylogeny can be inferred. This corresponds to only using certain columns of the character state matrix to reconstruct the phylogeny.

This can also be viewed as an optimization problem: pick the tree that is compatible with the largest number of characters. A character c is compatible with a tree if and only if for every state s of c, all nodes in the phylogeny having s with respect to c form a subtree and the ordering constraints on the character states are preserved. Hennig's method is in fact an instance of character compatibility.

Definition 2 *Given a set S of labels and a character state matrix M with character set C, two characters c_1,$c_2 \in C$ are said to be compatible if and only if a perfect phylogeny labeled by S can be formed in accordance with any ordering constraints on the states of these two characters.*

Recall that if a character is binary, it induces an edge $e = (A, B)$ on the phylogeny where labels in A are in state 0 and B are in state 1. The compatibility concepts we

introduced for splits and trees in Section 1.2.1 are precisely an instance of the above definition.

Definition 3 *A set C of characters are mutually compatible if and only if a perfect phylogeny labeled by S can be formed with all characters in C, such that all orderings constraints on the states of these characters are preserved.*

Again, the compatibility definition for bipartitions is an instance of the above definition for binary characters.

To use character compatibility for molecular sequence data, it is undesirable and is often impossible to eliminate certain base positions in the sequence data so that the remaining bases are compatible with each other. Typically, a method is used to infer a set of bipartitions R and the task is to search for a tree T that shares the largest number of internal edges with R. In fact, each bipartition in R is simply a binary character in the **CC** formulation. Note that $n-3$ bipartitions are needed to form a fully-resolved tree.

Given a set of characters, an interesting question is whether pairwise compatibility implies mutual compatibility. As the Pairwise Compatibility Theorem (PCT) [29, 19, 20] states, it is true for binary characters with or without a known ancestral state. Hence, this also holds for a set of bipartitions. As for ordered multi-state characters, a coding algorithm can be used to convert a multi-state character to a collection of binary characters, which are called *binary factors* of the multi-state character [21]. Hence, the binary PCT can be extended to ordered multi-state characters. In general, two multi-state *ordered* characters are compatible as long as every binary factor of one is compatible with every binary factor of the other. This was proved by Estabrook et al. [21].

Given ordered characters and the PCT, the test for mutual compatibility for a set of characters reduces to pairwise compatibility testing. Theoretically speaking, testing the binary factors is sufficient. A condition for two binary characters to be compatible is the

	Characters		
Taxon	1	2	3
A	0	0	1
B	0	0	1
C	1	1	1
D	0	0	1
E	0	1	0

Table 2.2: Characters 1 and 2 are compatible, while 2 and 3 are not.

following: Let (a, b) denote a row in the $n \times 2$ matrix M_c extracted from M such that it has the two character columns of concern. Two characters are compatible if and only if $(0,1),(1,1),(1,0)$ are *not* simultaneously present in M_c. Consider the character state matrix in Table 2.2, characters 2 and 3 are incompatible since the triple $(0,1)$,$(1,1)$,$(1,0)$ are present while characters 1 and 2 are compatible. Again, this also applies to two bipartitions.

Computational Issues

Since every multi-state character has a corresponding set of binary factors, the CC problem can be formulated in terms of binary characters. More importantly, binary characters correspond to a set of bipartitions which are inferred from sequence data.

Binary Character Compatibility (BCC)

Instance: A binary $n \times m$ character state matrix M with n objects and a set C of binary characters, where $|C| = m$.

Problem: Find the largest subset of C such that a perfect phylogeny can be inferred from L.

BCC is NP–hard [15]. The proof is based on a reduction from the **Clique** problem.

Clique

Instance: A graph $G = (V, E)$, $|V| = n$ $|E| = m$, and a positive integer $B \leq n$.

Problem: Find the largest subset L of V such that every member of L is connected by an edge in E.

It is relatively easy to transform **BCC** to **Clique**: create a vertex for every character and connect two vertices by an edge if and only if they are compatible. By the Pairwise Compatibility Theorem, the solution to **Clique** corresponds to a set of compatible characters. However, this is not very useful if one thinks that algorithms for **Clique** can be used to solve **BCC**. It is well-known that **Clique** is NP–hard, and it is also NP–hard to achieve an approximation ratio of n^{ϵ} for some $\epsilon > 0$ [4].

To reduce **Clique** to **BCC**, we form the character state matrix M as follows:

1. Create a matrix M of $(3 \times n)$ rows by n columns. Initialize all entries of M to 0. Note that there are 3 objects per character to work with.

2. For each column i, enter 0 at row $(3i - 2)$, 1 at row $(3i - 1)$, and 1 at row $3i$.

3. For all pairs of vertices v_i and v_j in G, if there is *no* edge connecting the two, at column j, enter 1 at row $(3i - 2)$, 1 at row $3i - 1$, and 0 at row $3i$.

The intuition behind the construction of M is that each vertex would transform into a character and the goal is to make pairs of characters incompatible if there is no edge connecting their corresponding vertices in the graph. Hence, a set of compatible characters correspond to a clique in the graph. The formal proof can be found in [15].

BCC can be easily shown to be transformable to other variations of the CC formulation [15], thus making all of them NP–Complete. These include unordered binary

character compatibility, as well as ordered and unordered multi-state character compatibility.

2.2.4 Distance-Based Methods

While parsimony, character compatibility and many other methods use the character state matrix directly to reconstruct the phylogeny, distance based methods utilize a distance function to estimate how far apart is each entity from one another and then the tree is inferred from the distance data.

Typically, a distance metric d is defined between two entities under study. In the case of DNA sequences for example, d is defined as $d : \{\{A, C, G, T, \zeta\}^n\}^2 \rightarrow \mathcal{R}$ where ζ denotes a gap. Gaps are needed to align two sequences of different length. One of the most important factors in the success of a distance-based method is the distance metric used. It should reflect accurately the amount of evolutionary divergence (i.e. amount of mutation) among two entities. Many useful and interesting metrics have been devised along with evolutionary model-dependent correction schemes. Please consult [54, 16] for details.

Given a label set S, we can use d to compute the pairwise distances between all pairs in S and they can be stored in a $n \times n$ symmetric square matrix D, called the *distance matrix*, where $D(a, b)$ is the distance between a and b. D is said to be *additive* if there exists a tree T such that $D(a, b)$ is the sum of the edge lengths in the unique path connecting a and b in T. Conversely, if D is additive, then there exists a unique T that realizes the distances in D. A widely used distance-based method, called Neighbour-Joining (NJ) [60], reconstructs T if D is additive in $O(n^3)$ time. NJ is basically a clustering procedure: it groups entities that are close together into groups. More formally, let V be a set of tree nodes labeled by S. NJ first searches for two nodes $x, y \in V$ that are the closest (i.e. $D(x, y)$ is the minimum) and form a subtree from them. It then removes x and y

from V and adds a pseudo-node p_{xy} to V (i.e. $V = V - \{x, y\}$), which can be considered as the parent of x and y. Finally, it computes the distance between p_{xy} and every other node $m \in V$ by the following formula: $D(p_{xy}, m) = \frac{1}{2}(D(x, m) + D(y, m) - D(x, y))$ and updates D correspondingly. The above procedure is perform iteratively until only one node remains in V, which is the root node. The distance updating formula makes intuitive sense when D is additive. However, due to noise and stochastic errors in real data, D is rarely additive and Neighbour-Joining can perform rather poorly in practice [38, 39, 40].

The main advantage of distance-based methods is speed. Unlike parsimony or character compatibility, several popular distance-based methods all have relatively light computational requirements. For example, another method called UPGMA [62], like NJ, also runs very fast. However, almost all distance-based methods have hidden assumptions that are unrealistic. The key to make them more robust might be to first, devise better distance metrics, or second, make more realistic assumptions. The recent works of Kearney [46, 45], which utilize the assumption of ordinality, has improved significantly the robustness of distance-based methods. We will apply orindality later in this chapter to devise an algorithm to generate bipartitions from sequence data.

2.3 Fractional Character Compatibility

The NP–hardness of the character compatibility formulation leads to the search for approximate solutions. In this section, we change the **BCC** formulation to a version called **Fractional Character Compatibility**, which is also NP–hard. However, we will demonstrate that a polynomial time approximation scheme (PTAS) exists under this formulation by using smooth-integer programming techniques.

We assume the input will be a set of bipartitions R obtained by another methods

from a set of sequences labeled by S. We will give a bipartition inference method later.

When the input is a set R of bipartitions, **CC** can be re-stated as follows:

Character Compatibility (CC)

Instance: A set of bipartitions R labeled by S.

Problem: Find a tree T labeled by S such that $|splits(T) \cap R|$ is maximized.

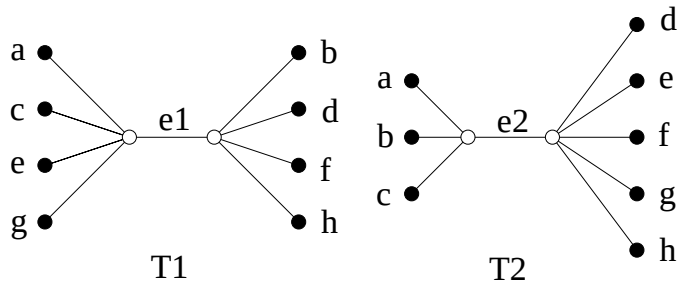

Figure 2.2: T_1 and T_2: T_2 fits $(\{a,b,c,d\},\{e,f,g,h\})$ better.

Given a tree T, we call $|splits(T) \cap R|$ the *CC score* of T. One criticism of the CC score is its inability to use information provided by bipartitions that are not fully compatible with T. For instance, consider the trees in Figure 2.2 and the bipartition $(A,B) = (\{a,b,c,d\},\{e,f,g,h\})$. Neither trees matched the bipartition perfectly, but clearly T_2 is a better approximation of (A,B) as it needs only one edge relocation to match the bipartition, where T_1 needs four. Based on this example, a new scoring metric can be defined to measure how close a bipartition matches a split in a tree.

Definition 4 *Given two bipartitions $(A,B),(C,D)$, the similarity score between the two partitions is $s((A,B),(C,D)) = max\{|A \bigcup C| + |B \bigcup D|, |A \bigcup D| + |B \bigcup C|\}$.*

Definition 5 *The similarity score between a tree T and a bipartition (A,B) is $s(T,(A,B)) = max_{e \in splits(T)}\{s(e,(A,B))\}$.*

Definition 6 *The similarity score between a set of bipartitions R and a tree T is* $s(T,R) = \sum_{(A,B)\in R} s(T,(A,B))$.

We are now ready to modify the **CC** formulation by using this new evaluation criterion.

Fractional Character Compatibility (FCC)

Instance: Set R of bipartitions of label set S, $|R| = m$, $|S| = n$.

Problem: Find a tree T labeled by S where $s(T,R)$ is maximized.

2.3.1 How Good is FCC?

Although the FCC optimization criterion makes intuitive sense, its biological significance as well as performance in real situations is unknown. Our goal in this section is to demonstrate through simulation experiments that FCC performs well and it is worthwhile to design efficient algorithm to solve it.

One of the intents of formulating FCC is to have a variation of CC that can be approximated in an efficient manner (i.e. our PTAS). Therefore, our method is most appealing to those whose intend is to use the character compatibility paradigm in their phylogenetic studies. Hence, given an instance of the tree reconstruction problem, if the a solution of FCC is at least as good as the solution of CC, then the formulation of FCC is at least as good as CC and the people who intends to infer trees using character compatibility would be at least as happy.

It is unclear how and could be very difficult to perform a theoretical analysis comparing CC and FCC. Therefore, we have decided it would be more interesting and practical to design a simulation study to perform the comparison. The main idea is to solve both

CC and FCC on a given set of bipartitions and then compare the resulting trees to the known "real tree" to see which one is closer. In case their performance is similar, we can compare the CC and FCC trees to see how close they are. The closer FCC is to CC, the better.

In order to control the tree parameter (i.e. making it known), we can either evolve sequences on a tree topology and infer bipartitions from them, or we can use a set of sequences that has a consensus phylogeny, which is typically generated by a well-tested method like exact Maximum Likelihood or Parsimony. The advantage of the first approach is that we will have absolute control over the real tree topology whereas in the latter approach, the trees are just created by another method. However, evolving sequences on a tree requires a realistic tree topology—it is very difficult to evaluate the performance and often lead to unrealistic results if the topologies are randomly generated. This entails the use of some known "real" tree topologies just like the second approach. However, this is a chicken-and-egg problem: how do we know what topologies are real? Hence, axiomatically we must agree that certain trees are "real" and our best bet is to use the consensus result of some well-tested methods. In addition, the first approach uses somewhat artificial sequences that depends too much on the model of evolution used while evolving the sequences whereas the second method uses real sequence data. Given the above observations, we believe the second method would serve us better. Of course, there is the possibility that the "real tree" is wrong and if the FCC tree is closer to it than the CC tree, we would make the wrong claim that FCC is better. But again, if the first method is used, this implies that the "real tree" that we used to evolve sequences are wrong too and the result cannot be very meaningful.

In summary, the simulation experiment is as follows:

1. Randomly pick a set S of n sequence from the RDP database.

2. Reconstruct phylogenies from S using both CC and FCC.

3. Determine the number of shared edges between the RDP tree and the CC tree.

4. Determine the number of shared edges between the RDP tree and FCC tree.

The above procedure is ran k times to obtain an average figure.

Methods and Tools

The RDP database [55] contains a wealth of ribosomal RNA sequences from different species. The evolutionary history of various sequence families were constructed by a version of Maximum Likelihood.

To perform the experiment, we need to solve CC and FCC. Since there is no known efficient method, we will do this exhaustively. That is, search through the entire tree space for the optimal tree. In addition, we will also need a means to generate bipartitions from sequence data. We will use a method called Hypercleaning [5], which has been proven to be robust and accurate in several experiments. Given S, Hypercleaning uses a quartet inference method to generate a set of quartets Q and then search for a set of bipartitions R that are within certain error range from Q. A parameter m is used to control the amount of error allowed. The greater the m, the bipartitions in R are less accurate with respect to Q. However, more bipartitions will also be generated given a large m. In addition, the time complexity of Hypercleaning also depends on m: the higher the m value, the slower. Each bipartitions generated by Hypercleaning has an associated error value, which can be used to rank the bipartitions in R. A tree can be constructed by picking the bipartitions with lower errors first and continue with bipartitions that are compatible with the ones that have already been picked. It was demonstrated in [5] that given a high enough m value, the above scheme tend to produce very accurate trees.

Since both CC and FCC do not take bipartition weighting into account, we will ignore the error value of the bipartitions generated by Hypercleaning.

Enumerating Evolutionary Trees

To solve CC and FCC optimally, the easiest way is to search the entire tree topology space. Hence, we need a way to enumerate all tree topologies. Since both CC and FCC work with bipartitions, the problem is equivalent to enumerating all sets of splits that are tree-like. This turns out to be an interesting problem. A simple way is to start with a star tree (see Figure 2.3) and split it into two partitions (i.e. a bipartition). Now each set of the bipartitions correspond to a smaller star tree. Hence we can perform the splitting recursively until a tree-like bipartition set is obtained.

The order that the partitions (i.e. star trees) are splited is very important. If we were to split a single partition until it is fully-resolved, it would be impossible to obtain a complete set of tree-like bipartitions. Instead, a breadth-first split (i.e. split the partitions in the order that they are created) is required. A queue is needed to maintain the order that the partitions should be splitted.

There is one caveat in the above algorithm, namely that it will visit a tree topology more than once in the tree space since there are more than one set of split paths from the star tree to an unrooted tree. This problem is illustrated in Figure 2.3. This would be highly inefficient and makes even branch-and-bound implementations impractical. The problem is caused by the fact that the tree is unrooted. Observe that if the tree is rooted at a leave, then there would be only one single split path from the star tree. That is, we can only split from top to bottom from the root node. Hence, we need to root the initial star tree at a leave node r and then perform the splitting with the remaining leaves. Figure 2.4 illustrates this idea.

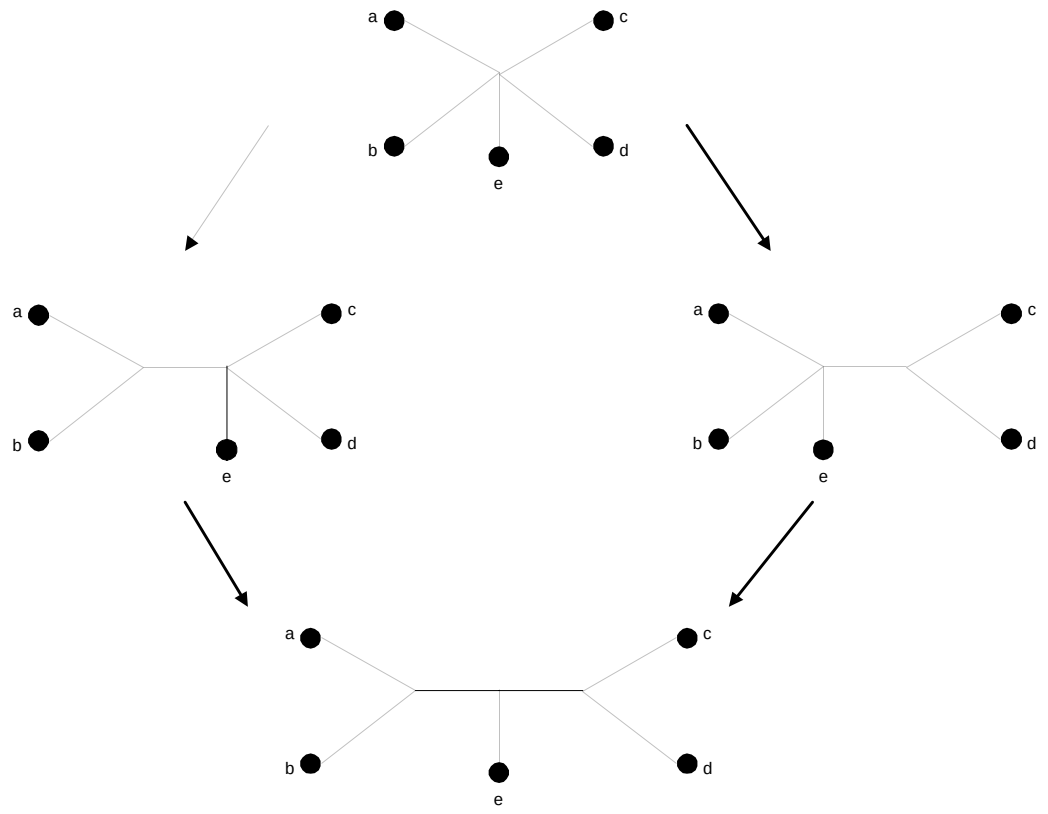

Figure 2.3: The top tree is a star tree (without any internal edge). There are two possible ways to split it to obtain the tree at the bottom.

An Interesting Recurrence

The way we enumerate unrooted evolutionary trees entails a recurrence for the total number of evolutionary trees, which has been studied by Felsenstein in the past [25]. Let $p(n)$ be the number of rooted fully-resolved evolutionary trees with n leaves. According to our enumeration method:

$$p(n) = \sum_{i=1}^{\lceil \frac{n}{2} \rceil} \binom{n}{n-i} p(n-i)p(i) \textbf{ if n is odd} \tag{2.1}$$

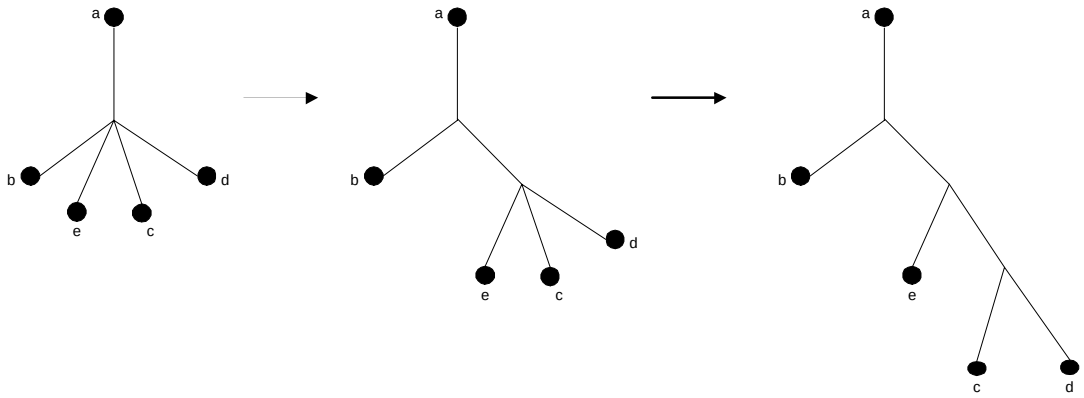

Figure 2.4: If we root at a, there is only one way to split the star trees to obtain the final tree on the right.

$$p(n) = \sum_{i=1}^{\lceil \frac{n}{2} \rceil} \binom{n}{n-i} p(n-i)p(i) + \frac{\binom{n}{n/2}}{2} p^2(\frac{n}{2}) \textbf{ if n is even} \tag{2.2}$$

In [25], Felsenstein uses a different approach to determine that $p(n) = \frac{(2n-3)!}{2^{n-2}(n-2)!}$, which should be the same as the solution of the above recurrence given that our enumeration procedure is correct. We will omit the proof of the correctness of our enumeration procedure.

For unrooted trees, since our method roots the tree at a leave, $p(n-1)$ would be the total number of unrooted trees with n leaves.

Simulation Result

Due to the heavy computational requirement of the exhaustive algorithm, we could only perform our experiment on trees up to ten leaves ($n = 10$). However, since the tree topologies are randomly chosen from the RDP and we run the experiment over many iterations, we expect the result to be a fair assessment. Table 2.3 summarizes the result for an experiment with $k = 50$. As noted before, m is the parameter that Hypercleaning

uses to control the amount of error allowed in the bipartitions generated. When m is high, some bipartitions generated could be less reliable. However, higher m values typically gives more bipartitions and according to [5], a higher value is often required to generate all bipartitions needed to resolve the tree.

m value	fcc&rdp(%)	cc&rdp(%)	cc&fcc(%)	greedy&rdp(%)
1	41	41	100	37
2	54	54	95	50
3	43	38	85	51
4	43	38	85	51

Table 2.3: Result of the simulation study.

Clearly FCC performs as well as CC for all m values. With m equals to 1 or 2, the bipartition sets generated are small in size and they typically do not contain all internal edges needed to reconstruct a fully-resolved tree. But they also tend to be highly accurate, which is why when m is low, the performance of CC and FCC are almost identical, whereas the greedy method is sub-optimal. As m gets higher, FCC performs better than CC since the bipartitions have more error and by the nature of the similarity score used in FCC, it can take advantage of bipartitions that are slightly off. The greedy method is better with higher m value since it also uses the weight associated with each bipartition whereas CC and FCC do not use weights.

Discussions

After the above experiments, we can quite firmly conclude that FCC performs as well as CC and it is worthwhile to investigate efficient algorithm to solve it.

The experiment is, however, still flawed in several aspects. Most importantly, the above result only applies to small trees, we do not know what the situations will be until further experimentation is performed on larger trees. However, we suspect the result

would be similar since with larger trees, the number of erroneous bipartitions would scale up and FCC is expected to pick up useful information from these bipartitions just as it did for smaller trees. Secondly, the RDP trees are huge and by randomly taking a small number of small topologies from it does not necessarily reflect a good sample in the tree topology space. Hence, strictly speaking the above result may not be valid for certain tree topologies. However, this situation is hard to remedy until CC and FCC can be efficiently solved and a large sample size can be used in the experiment. In addition, the result of any numerical simulation is bound to statistical errors.

Both CC and FCC are not formulated with weights on the bipartitions. However, the performance of the greedy method with higher m values clearly demonstrates the importance of incorporating weight information. The sole reason why greedy performs better when noisy data are present is due to the fact that it takes highly reliable bipartitions first before considering the ones that are ranked lower. If CC and FCC were weighted, it would of course outperform the greedy method by definition. However, the focus of this chapter is still to approximate FCC as formulated, since our result is mainly of theoretical importance. In practice, weighted FCC can be employed to select optimal bipartition sets produced by methods such as Hypercleaning. Although further experimentation is needed to support the claim, we would still like to suggest that a branch-bound version of FCC can probably be practically used to do bipartition selection—especially when the dataset is "good" and the greedy tree is close to optimal. We leave that as future research.

2.3.2 A PTAS for Split Recombination

FCC is known to be NP–Complete [48]. Unless $P = NP$, the best one can do is to design a polynomial time approximation scheme (PTAS) for the problem. That is, for any ϵ, the approximation algorithm would return a tree T_{APP} such that $s(T_{APP}, R) \geq (1-\epsilon)s(T_{OPT}, R)$, where T_{OPT} is the optimal tree. A PTAS is desirable as it allows us

to achieve arbitrary degree of accuracy by varying the length of the computation.

The first important idea in approximating T_{OPT} is to approximate an approximation of T_{OPT}, which we called a *contraction* of T_{OPT}. The search space would be too large if we approximate T_{OPT} directly.

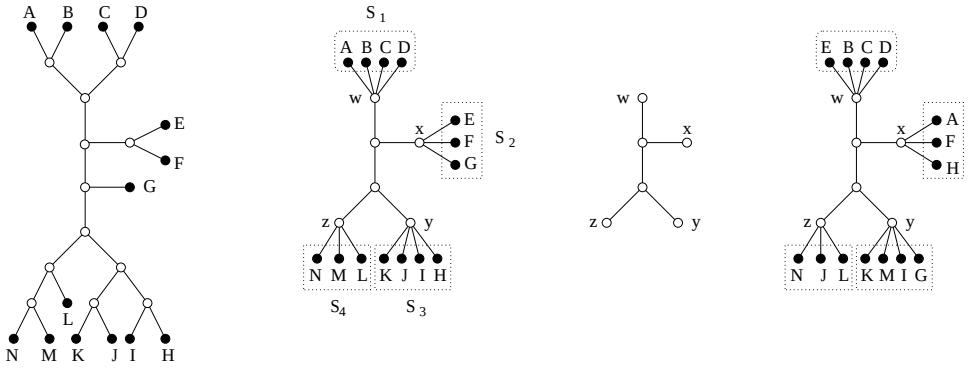

Figure 2.5: From left to right: A tree T; a 4–bin contraction T_4 of T with bin roots w, x, y and z; the kernel K of T_4; and a completion of K [48].

Definition 7 ([42]) T_k is a *k–bin contraction* of T_{OPT} if there is a partition of S into *bins* S_1, S_2, ... S_k such that

- For each S_i, $|S_i| \leq 6n/k$. Furthermore, there is a vertex v_i of degree $|S_i| + 1$, called the *bin root*, that is adjacent to each vertex in S_i.
- For each edge e of T_{OPT} there is an edge e' of T_k such that $s(e, e') \geq n - 6n/k$.

k–bin contraction is being defined this way so that T_k is a good approximation of T_{OPT} and the specific details are derived from the requirements of the desired solution. Suppose edge e of T_{OPT} is the edge with the highest similarity score for a bipartition r in R, then directly from the second property above, there must exist an edge e' of T_k such that $s(r, e') \geq s(r, e) - 6n/k$. Hence, if T_k exist for any given T_{OPT} and k, then $s(T_k, R) \geq s(T_{OPT}, R) - (6/k)mn$, since there are m bipartitions in R.

The existence of T_k can be proved by an algorithm that outputs a k–bin contraction of an input tree T given k. This algorithm is given below, without loss of generality, assume T is binary.

Algorithm k–Bin Contraction(T) [48]

1. Root T at an arbitrary internal vertex. Let $T(v)$ denote the subtree of T rooted at v.

2. Traverse T, beginning at the root, such that for each vertex v visited:

   If $|T(v)| \leq 6n/k$ then

   - contract all internal edges of $T(v)$,
   - label v as a bin root and
   - continue traversal at v's parent.

   Otherwise, continue traversal at an unvisited child of v.

3. For each bin root v with parent v' and sibling u':

   If $|T(v)| \leq 3n/k$ and u' has a child u with $|T(u)| \leq 3n/k$ then

   - transfer the leaves in $T(u)$ to the bin of v,
   - contract $\{u, u'\}$ and
   - contract $\{u', v'\}$.

4. For each leaf u of T not assigned to a bin, bisect the edge between u and its parent with a new vertex v, and mark v as a bin root.

Step 3 tries to group smaller bins into one larger bin and step 4 is needed since a leaf cannot be a bin root.

Label each bin root with the size of its bin. Furthermore, remove all leaf vertices that are attached and contract the edges and call the resulting tree K a *kernel* of T_k. Call a bin root (now a leave in K) *small* if its label is less than $3n/k$, otherwise, the bin root is *large* with label in $[3n/k, 6n/k]$. Let s be the number of small bin roots and l be the number of large bin roots.

Lemma 1 ([42]) *$s < 2l$ holds for T_k.*

Proof:

Note that a bin root must be a leaf in K. By induction on the height of T_k: if u is an internal vertex (i.e. not a bin root) of height h, then the lemma holds for $T(u)$.

Base case (height = 1): let p and q be u's children and both must be bin roots. If both p and q are small, then they would have been contracted by Step 1 of the algorithm and u would be a bin root. Hence, one of them must be large. $s < 2l$ holds.

For any vertex u of height $h+1$, let p and q be its children. If both are bin roots, they cannot be both small, otherwise the algorithm would have contracted their internal edges and made u a bin root as in the base case. If neither p nor q are bin roots, then the induction hypothesis applies and hence $s < 2l$. Finally, without loss of generality, p could be a bin root but q is not. Let q_1 and q_2 be the children of q. If p is large, then the lemma holds for $T(q)$ by the induction hypothesis, hence the lemma also holds for $T(u)$. If p is small, then neither q_1 nor q_2 can be small, otherwise Step 3 of Bin-Contraction would merge them with p. Let the number of small bins in q_1 and q_2 be $s(q_1)$ and $s(q_2)$ respectively and the number of large bins in q_1 and q_2 be $l(q_1)$ and $l(q_2)$ respectively. By the induction hypothesis, $s(q_1) < 2l(q_1)$ and $s(q_2) < 2l(q_2)$, hence $s(q_1) + s(q_2) + 2 \leq 2l(q_1) + 2l(q_2)$, and $s(q_1) + s(q_2) + 1 < 2l(q_1) + 2l(q_2)$ for vertex u. Hence, the lemma also holds for this case.

□

Since each large bin has size at least $3n/k$, thus $l < k/3$, otherwise we would end up with more than n labels. So the total number of bins $l+s < l+2l = 3l < 3\times k/3 = k$. This gives an upper bound for the number of bins obtained after the contraction procedure.

Lemma 2 *There is a k-bin contraction of T_{OPT}.*

Proof Sketch:

We need to show that T_k satisfies both properties of Definition 7. First of all, the number of bins is upper bounded by k. In case the number of bins is less than k, one could choose not to merge certain pairs and obtain exactly k bins. By the design of the Bin-Contraction procedure, each bin is guaranteed to be of size bounded by $6n/k$.

For the second property of the definition, edges that were not contracted are completely preserved in T_k, hence the similarity score is n. For an edge e that was contracted, it must belong to a bin with bin root u in T_k. To approximate e in T_k, the edge e' that connects u to its parent can be used. Due to the bounded size of bins(i.e. $< 6n/k$), this gives a lower bound on $s(e,e')$ of $n - 6n/k$. Consult [42] for more details.

□

For a given k and T_{OPT}, we can obtain T_k with the above algorithm. Recall that removing all leaves of T_k gives the kernel K with k leaves. Assume that T_k is unknown, and we would like to form T_k from K given a set of bipartitions R inferred from data about T_k. T_k can be approximated by assigning the labels in S as children of the leaves (or the bin roots) of K to form T' such that $|s(T',K)|$ is maximized. We called T' a *completion* of K. This gives a way to approximate T_k from K and an optimization problem can be formulated as follows:

Label–to–Bin Assignment (LBA)

Instance: Set R of bipartitions of S and a binary kernel K with k leaves.

Problem: Find a completion T' of K that maximizes $s(T', R)$.

This give us a framework to approximate T_{OPT}:

1. Form a set of bipartitions R from the character state matrix.
2. Fixed k, form all possible tree topologies with k leaves.
3. For each topology formed, solve the **LBA** problem to form T' and record $s(T', R)$.
4. Return the tree with the highest similarity score.

Note that it only takes constant time for the second step since the number of trees with k leaves does not grow as n grows.

The only problem that remains to be solved is the **LBA** problem, which can be formulated as an integer program.

Define a set of variables $x = (x_{sb})$ such that $x_{sb} = 1$ if label s is assigned to bin b. Otherwise, $x_{sb} = 0$. For each label s, the following constraint is needed to ensure that the label is only assigned to one bin:

$$\sum_{b} x_{sb} = 1$$

And in order to ensure the k–bin contraction property, add the following for each bin b:

$$\sum_{s \in S} x_{sb} \leq 6n/k$$

Let T' be a completion of K. In order to evaluate $s(T', R)$, each bipartition $r \in R$ needs to be assigned to an edge in T'. Define a set of variables $y = (y_{ie})$ such that $y_{ie} = 1$ if bipartition (A_i, B_i) is assigned to edge e of T'. Otherwise, $y_{ie} = 0$. For each bipartition

r, we also need to make sure that it is only assigned to one edge:

$$\sum_{e} y_{ie} = 1$$

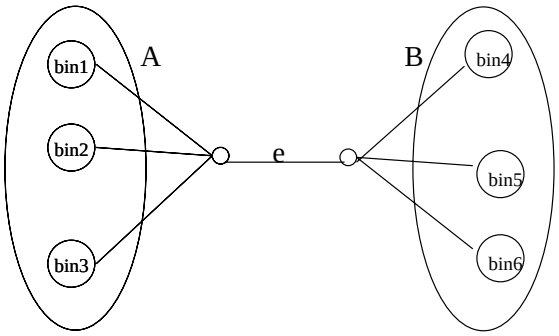

Figure 2.6: edge e induces bipartition (A, B). A and B can also be considered as two sets of bins.

For each bipartition $(A_i, B_i) \in R$, by definition, the similarity score $s((A_i, B_i), T')$ is an assignment to the variable y_{ie} such that the following is maximize:

$$p_i(x, y) = \sum_{e=(A,B)\in K} y_{ie}\left(\sum_{s\in A_i}\sum_{a\in A} x_{sa} + \sum_{s\in B_i}\sum_{b\in B} x_{sb}\right)$$

The first summation above can be interpreted as the process of trying all the edges of T' such that the similarity score is maximize. Also note that the edge e above induces the bipartition (A, B), where A and B can also be considered as sets of bins. This is illustrated in Figure 2.6

Hence, for all bipartitions in R, the similarity score $s(T', R)$ is:

$$p(x, y) = \sum_{1\le i\le m} p_i(x, y) \tag{2.3}$$

$p(x, y)$ precisely expresses the optimization criterion in x and y. The goal is to find a

0-1 assignment to x and y such that $p(x, y)$ is maximize. In summary, the integer program is:

$$p(x, y) \text{ is maximized}$$

$$\sum_b x_{sb} = 1, \text{ for each label } s$$

$$\sum_e y_{ie} = 1, \text{ for each bipartition } (A_i, B_i)$$

$$\sum_s x_{sb} \leq 6n/k, \text{ for each bin } b$$

By the definition in [3], $p(x, y)$ is called a *smooth polynomial* and an integer program with a smooth polynomial as the objective function is called a *smooth-integer program.* Arora et al. [3] gave a PTAS that solves smooth-integer programs.

Definition 8 ([3]) *An n-variate, degree-d polynomial has smoothness c if the coefficient of each degree i monomial(term) is at most cn^{d-i}.*

Definition 9 ([3]) *A c-smooth degree-d polynomial integer program (PIP) is a PIP in which the objective function is a c-smooth polynomial with degree at most d.*

More specifically then, $p(x, y)$ is a 1-smooth degree-2 polynomial with $kn+m(k-3) = O(m+n)$ number of variables (an unrooted tree with k leaves has $(k-3)$ internal edges).

Arora et al. [3] proved that for each fixed $\epsilon > 0$, a PTAS runs in time $O(n^\alpha)$ exist that produces a 0-1 assignment z for a c-smooth integer program, such that $\alpha = 2c^2d^3/\epsilon^2$ and $p(z) \geq p(z^*) - \epsilon n^d$ where $p(z^*)$ is the value of optimal assignment and n is the number of variables in z. Hence, applying this PTAS to the **LBA** integer program would give a 0-1 assignment for x and y that corresponds to a completion T' of K such that $s(T', R) \geq s(T_k, R) - \epsilon(m+n)^2$ for any $\epsilon > 0$.

T' is an approximation of T_k, which is an approximation of T_{OPT}. We can relate the performance of T' to T_{OPT} through T_k:

$$s(R, T') \geq s(R, T_{OPT}) - 6mn/k - \epsilon(m+n)^2 \tag{2.4}$$

For any two bipartition $(A, B), (C, D)$ of a label set S where $|S| = n$, $s((A, B), (C, D)) \geq n/2$ since $|A \cup C| + |A \cup D| + |B \cup C| + |B \cup D| = n$, thus $|A \cup C| + |B \cup D| = n - (|A \cup D| + |B \cup C|)$, so if $|A \cup C| + |B \cup D| < n/2$, then $|A \cup D| + |B \cup C| \geq n/2$ and vice versa. Hence, $s(R, T_{OPT}) \geq mn/2$ since there are m bipartitions in R. If $m = \Theta(n)$ then (1) becomes:

$$s(R, T') \geq s(R, T_{OPT}) - c_1 s(R, T_{OPT}) - c_2 s(R, T_{OPT}), \text{ for some } c_1, c_2 > 0 \tag{2.5}$$

Hence the approximation algorithm satisfies the requirement of a PTAS such that for each $\epsilon' > 0$, the following is true:

$$s(R, T') \geq (1 - \epsilon') s(R, T_{OPT}) \tag{2.6}$$

Theorem 1 *There is a PTAS for FCC.*

The PTAS requires that $m = \Theta(n)$ in order to deliver the performance guaranteed. This is in fact a reasonable assumption since only $n - 3$ edges are needed to reconstruct a fully-resolved tree.

A More Practical Implementation Is Needed

The PTAS presented is the first known polynomial time approximation algorithm under the character compatibility formulation. However, it is mainly of theoretical importance

and it would not be practical to use the algorithm for large inputs. To illustrate, consider the time complexity of the PTAS for the **LBA** problem: $O(n^{2c^2d^3/\epsilon^2})$. If we would like the solution to be at least 90% of the optimal, the running time is $O(n^{1600})$! Clearly, this corresponds to a very slow algorithm. In addition, if we would like 90% accuracy for the final tree, this would impose a higher accuracy constraint on the solution of **LBA**, thus making it even slower. This is the main reason why this algorithm was not implemented to evaluate its performance with real data.

We suspect some new ideas are needed to make the algorithm practical, especially during the implementation of the algorithm. One possible solution is to reduce the number of variables in the smooth polynomial. For instance, it might be possible to extract $O(logn)$ labels from S and obtain a local optima by searching through all possible assignments. Some of the unassigned labels can be put into a bin if it is clear that such an assignment would maximize the increase in similarity score. After this process, a smooth polynomial can be formulated for labels that have not been assigned.

2.3.3 Generating Bipartitions–The Ordinal Split Method

The PTAS described assumes that the input is a set of bipartitions. Furthermore, there can only be $\Theta(n)$ bipartitions in the input. To generate such binary characters from sequence data, we present here a distance-based method called the Ordinal Split Method (OSM) that generates bipartitions based on ordinal assertions [45], which uses relative proximity information rather than absolute distance measure to generate a set of bipartitions. The redundancy criterion is used to select the best $\Theta(n)$ bipartitions. Simulation studies revealed that for trees of relatively long edges, over 90% of the bipartitions of a given tree are recovered using this method.

Alternatively, the Hypercleaning method used in the FCC simulation can be used. In fact, Hypercleaning generally performs better than the OSM for difficult data sets and

it has the capability to use any quartet topology inference method. However, OSM was devised before Hypercleaning and it runs extremely fast compare to Hypercleaning.

The bipartition generation problem based on distance data can be defined as follows:

Bipartition Generation (BG)

Instance: A symmetric square distance matrix M where $M(i,j)$ denotes the observed dissimilarity between sequence i and j.

Problem: Produce $\Theta(n)$ splits.

To generate M, a variety of techniques have been used to compute the observe dissimilarity between a pair of sequences. For example, a simple way is to compute the hamming distance between two sequences. More sophisticated techniques rely on models of sequence evolution and use correction schemes to obtain more accurate measures of the evolutionary distance between two sequences [54, 69].

Recall that given a tree with weighted edges, its edge lengths are said to be *additive* if the distance between a pair of leaves is the sum of the edge lengths on the path connecting them. If a distance matrix M is computed from such a tree (for each pair of leaves), then M is additive and the tree can easily be reconstructed from M alone by using the Neighbour-Joining Method [60]. However, M is rarely additive when computed from real sequence data. This makes distance-based methods such as Neighbour-Joining perform rather poorly in many situations. Instead of assuming M to be additive, a more robust assumption called *ordinality* [45, 44] has proven to be quite effective when used with sequence data [46]. An ordinal assertion is the statement of the form "$M(x,y) < M(x,z)$", which gives relative proximity information (i.e. x is closer to y than to z) rather than absolute distance assertions as in additivity.

To apply ordinal assertions to generate bipartitions from M, let $d(x,y)$ be the real

evolutionary distance between x and y. By the assumption of ordinality, if $M(x,y) < M(x,z)$ then $d(x,y) < d(x,z)$. This entails a way to cluster the sequences into two groups. Define $S_{xy} = \{t \in S \mid M(x,t) < M(y,t)\}$ and $S_{yx} = \{t \in S \mid M(x,t) > M(y,t)\}$. S_{xy} contains the set of sequences that are closer to x than to y and S_{yx} contains the set of sequences that are closer to y than to x. It follows that (S_{xy}, S_{yx}) is a valid bipartition and we called it an *ordinal split*. Note that this bipartition corresponds to the edge on the tree that contains the midpoint between x and y (see Figure 2.7). If this is done for every pair of sequences (x,y) in S, we can generate $O(n^2)$ bipartitions.

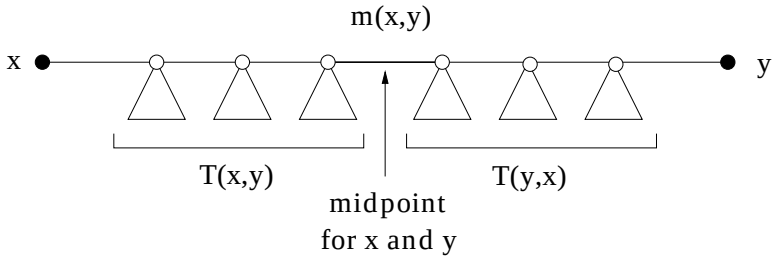

Figure 2.7: The ordinal split corresponds to the edge where the midpoint between x and y lies.

However, we only need $\Theta(n)$ bipartitions in order for the PTAS to work. To achieve this result, we rank the $O(n^2)$ bipartitions by redundancy and pick the first n bipartitions for the PTAS. We expect some of the bipartitions to be redundant since it is very likely that multiple midpoints would fall on a long internal edge of T. After all, there are only $n-3$ internal edges. Hence, more redundant splits are more reliable and they represent well-supported edges in the true tree. By picking the splits through redundancy, we expect relatively long edges to be well-covered. We will consider shorter edges in Section 2.3.3.

A Simulation Study to Evaluate the Ordinal Split Method

We have performed a simulation study to evaluate the effectiveness of the OSM method for selecting good edges and its coverage of internal edges. Our goal is to have a known evolutionary tree T and see how many real internal edges can the OSM generate from a set of sequences artificially evolved from T. Also, we would also like to see the percentage of internal edge coverage when the first $\Theta(n)$ bipartitions as ranked by redundancy are used.

In the simulation, a randomly generated binary tree T with n leaves and mean branching length of b were generated. The simulation procedure is as follows:

1. Evolve a set of n sequences of length l on T according to the Kimura-2-Parameter (K2P) [50] model of sequence evolution.

2. Compute a distance matrix M based on the n sequences and apply the Kimura-2-Parameter distance correction scheme [54].

3. Generate a set of bipartitions R by the Ordinal Split Method and rank them by redundancy.

4. Determine the edge coverage statistics when $\Theta(n)$ splits are picked.

5. Also determine whether more redundant bipartitions have a higher chance of being a real internal edge. Given a bipartition r, this can be done via a search through the internal edges of T to see if there is a match for r. In the spirit of the FCC formulation, partial matches are scored by the similarity metric defined in Definition 4

Step 1 needs a bit of clarification. When we say "to evolve" a set of sequences, we mean to simulate the evolutionary process on T. Given an initial sequence that serves

as the root sequence, we can simulate the process of evolution by mutating bases with substitutions (insertions and deletions are in general hard to model and most popular models do not incorporate them). Since the model of evolution is stochastic, we will run the above procedure multiple times to obtain reliable results.

Several packages were used to help implement the simulation. We used the DNADIST program in the PHYLIP [30] package to compute distance matrices from sequence data. The SEQ-GEN program [57] was used to generate sequences along a tree based on the K2P model. Finally, the LISTTREE program from [74] was used to generate random tree topologies.

Trees With Long Edges

To first test whether the OSM is at all functional, we ran the simulation on a randomly generated tree with relatively long internal edges ($b = 0.05$). The following parameters—due to their resemblance to typical phylogenetic parameters—are used: $n = 50, l = 500$ and the transition to transversion ratio is 2. Figures 2.8 and 2.9 illustrates the result. In fact, the result stays the same after l and the transition/transversion ratio are varied within valid ranges.

The redundancy plot confirms our prediction that the more redundant the split, the higher the quality. The highly rank splits all have near perfect similarity scores. The coverage plot is also encouraging in the sense that $O(n)$ splits gives pretty good coverage of the edges.

Tree Topologies From RDP

To better evaluate the OSM, we again extract four real tree topologies with relatively short edges (with edges as short as 1×10^{-6}) from the RDP and evolve sequences on them to perform the above experiment. We suspect the OSM might perform more poorly on

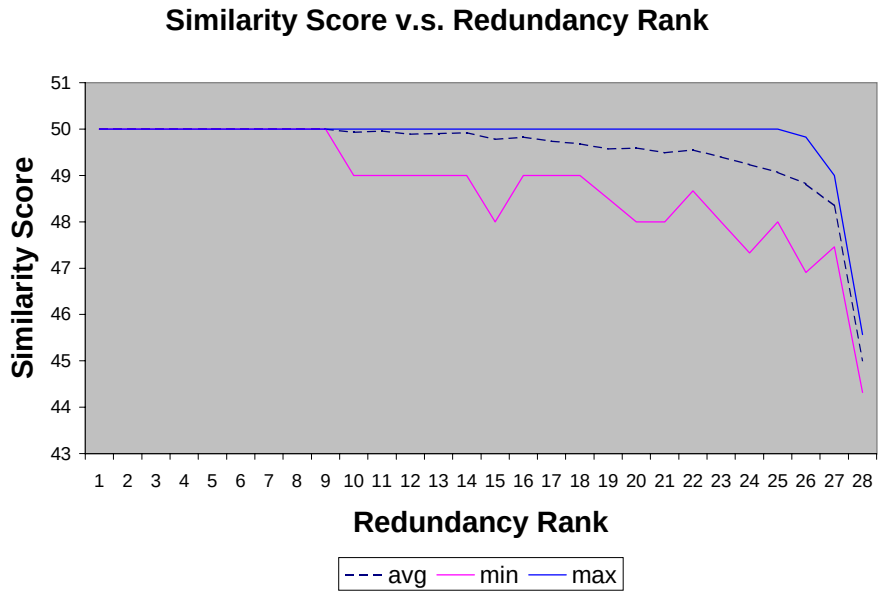

Figure 2.8: The similarity score of the splits inferred ranked by redundancy. The trend is that the more redundant the split, the higher the similarity score.

	Edge Coverage		
Tree	min(%)	avg(%)	max(%)
1	0	21.3	42.9
2	0	25.6	57.1
3	28.6	57.9	71.4
4	42.9	84.4	100

Table 2.4: Experiment result with RDP short edge trees.

trees with short edges since these edges might be too short to have a midpoint falling on them. The simulation were ran 100 times and the average figures are reported in Table 2.4. Tree 1 is the one with the most number of short edges and in general, OSM perform

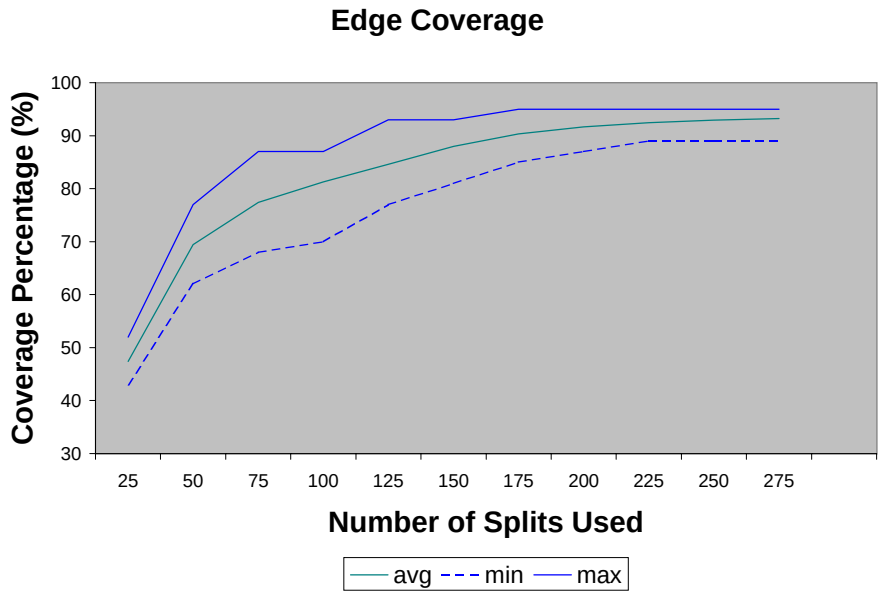

Figure 2.9: Internal edge coverage plot against the number of ordinal splits used. A linear factor of the input size gives $> 90\%$ coverage.

rather poorly as expected.

Discussions

While our experiment confirms that the OSM works well on trees with relatively long edges, it fails to recover shorter ones. Hence, this would not be the most desirable generic front-end for the PTAS. By the results in [5], we believe Hypercleaning would be a better front-end for the PTAS. However, the OSM is fast ($O(n^2)$) and can well-served as a first step tool to generate "easy" bipartitions before more time-consuming algorithms such as Hypercleaning are used to generate the shorter edges. For example, if the OSM can resolve

certain long edges of the tree perfectly (i.e. produce bipartitions that are compatible), then we just run Hypercleaning on the partitions (i.e. star subtrees) that are unresolved. This would probably save substantial computational time. Further experimentation is needed and we left them for further research.

Chapter 3

Computing the Quartet Distance Between Evolutionary Trees

3.1 Introduction

The comparison of evolutionary trees is another fundamental problem in evolutionary biology. Different evolutionary hypotheses (or conflicting phylogenies) arise mainly in two situations. As we have discussed in Chapter 1, the tree inferred from a set of homologous genes only represents the relationship among the genes under study, but not necessarily the species themselves. It is often the case that different genes lead to different trees. However, if the biologist would like to have a species tree, it is unlikely that trees inferred based on different genes would agree with each other. For instance, in a recent biology paper [10] Cao et. al studied the phylogenetic relationship among primates (e.g. human), ferungulates (e.g. horse), and rodents (e.g. mouse) using the genes on the H strand of mitochondria DNA and found that two different trees were constructed depending on the gene used in the analysis. Different evolutionary hypotheses can also arise when different phylogenetic reconstruction methods are applied to the same data set. The previous

chapter showcases only a few popular phylogenetic reconstruction methods. However, a diverse variations of various popular paradigms are used by different schools of systematics. In many cases the trees inferred by different phylogenetic methods would be different. In another recent study by Cammarano et. al. [9], the tree reconstructed using Maximum Parsimony is distinct from the one reconstructed using Maximum Likelihood when the EF-G(2) sequences are used to study the relationship between Archaea and Bacteria.

There are currently two main approaches to resolve the issue of conflicting phylogenies. In one direction, researchers are trying to utilize the information provided by the entire genome to infer a species tree. Many different notion of measuring genome similarity has been proposed. This include genome rearrangements [49, 36], information-theoretic metrics [73], and gene content [63]. An older and perhaps more mature approach is to select a consensus tree (or trees) that best represents the information provided by each conflicting tree [7]. A notable formulation is the Maximum Agreement Subtree method [68, 7, 23, 22, 11, 66] for finding a consensus tree among two or more different trees. A substantial amount of effort has been devoted to efficient algorithms for finding the MAST of two or many evolutionary trees, see [68] for a summary of results. While resolving conflicting trees, biologists are also interested in knowing the degree of dissimilarity among different evolutionary trees so that the consistency of the analysis can be assessed. More importantly, it is desirable to have an objective metric to compare trees so that the statistical significance of their difference can be evaluated numerically [65]. In addition, the stability of the analysis results can also be evaluated. The distributions of various tree similarity metrics are well-studied [65] and are very useful in testing statistical hypotheses.

Several distance metrics between evolutionary trees are currently in use [7]. In this chapter, we study the quartet metric, which is based on common subtrees induced by four leaves. This metric has several attractive properties, though its use has been limited by the time required to compute the distance [65]. In this chapter, we address this

problem by describing an $O(n^2)$ algorithm that computes the quartet distance between two evolutionary trees.

3.2 Some Tree Distance Metrics

An evolutionary tree represents the direction of evolution by the location of its root, the rate of evolutionary by its edge lengths and the history of speciation events by its branching pattern or topology. Biologists are often interested in the distance between two evolutionary trees independent of the direction and rate of evolution, which gives an indication of how similar two trees are in terms of the relationships among leaves. Various metrics have been proposed to measure the similarity based on the undirected tree topology and we survey some of these below.

3.2.1 The Partition Distance

The partition distance (PM) [6, 61, 58] measures the number of splits (i.e. internal edges) that are induced by one tree but not the other. Recall that an unrooted tree T can be represented as a set of splits denoted as $splits(T)$ (i.e. non-trivial edges of the tree). Given two trees T_1 and T_2, the partition distance between them is:

$$d_p(T_1, T_2) = splits(T_1) + splits(T_2) - 2|splits(T_1) \cap splits(T_2)|$$

Which is basically the symmetric difference between the two sets of splits. This metric is very easy to calculate and a linear time algorithm was given by Day [13]. Since there are at most $n - 3$ internal edges for an unrooted tree, the range of this metric is $2n - 6$, which is very narrow and hence is not very sensitive in detecting finer similarities. The statistical properties of this metric is well-studied and the details can be found in [65].

3.2.2 The Nearest-Neighbour Interchange Distance

The nearest-neighbour interchange (NNI) metric was introduced in [71]. It is based on a tree transformation operation, called *nearest-neighbour interchange*. Figure 3.1 illustrates the NNI operations. The *NNI distance* between two evolutionary trees is defined as the minimum number of NNI moves needed to transform one to another.

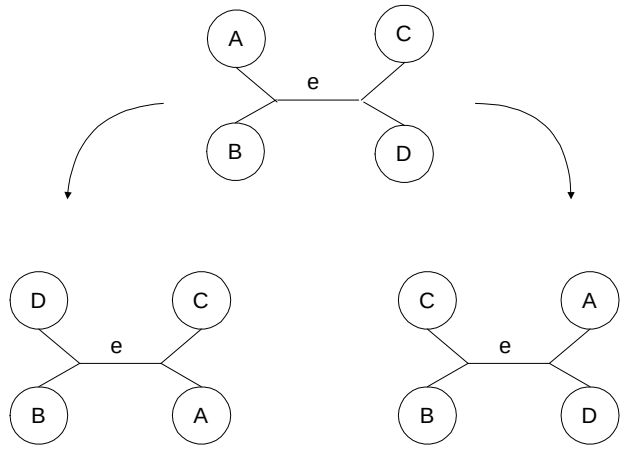

Figure 3.1: The nearest-neighbour interchange operations. There are two possible nni moves on an internal edge e: one transforms the top tree to the lower left one, the other transforms the top tree to the lower right tree.

While it is known that a series of NNI operations can always transform one tree to another, its biological significance is unknown. In addition, it is NP-hard to compute the NNI distance [12]. Hence in practice, it can only be used on small trees. A wealth of theoretical work has been done on the distance, see [52] for further details.

3.2.3 The Robinson and Foulds Distance

The Robinson and Foulds metric (RF) [59] is also based on transformation operations. It defines two operations: edge contraction and node expansion. And again the distance between two trees is the minimum number of such operations to transform one to the

other. For the ease of discussion, let internal nodes be labeled. Consider any edge e connecting two nodes u, v labeled by the sets $label(u)$ and $label(v)$ respectively. A contraction operation on e removes e from the tree and creates a new node w labeled as $\{label(u), label(v)\}$. An expansion operation is the opposite: given a node u with label $\{l_1, l_2, ..., l_n\}$ where $n > 1$, an expansion on u removes u from the tree and adds two new nodes u_1 and u_2 and an edge e that connects u_1 and u_2. Also, $label(u_1) \cup label(u_2) = label(u)$ and $label(u_1) \cap label(u_2) = \emptyset$. Figure 3.2 illustrates this distance.

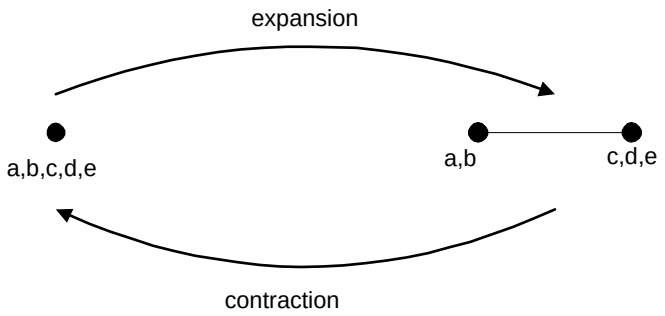

Figure 3.2: The Robinson and Foulds operations. An expansion introduces an edge whereas a contraction reduces an edge.

3.3 The Quartet Distance

For the duration of this chapter, we assume all evolutionary trees are unrooted. Given two trees T_1 and T_2 with quartet topology sets Q_1 and Q_2 respectively, the *quartet distance* is defined as the symmetric difference between the respective set of quartet topologies:

$$d_Q(T_1, T_2) = |Q_1| + |Q_2| - 2|Q_1 \cap Q_2| \tag{3.1}$$

For example, the quartet distance between the two trees in Figure 3.3 is 4.

The quartet metric does not suffer from drawbacks of other distance metrics. For

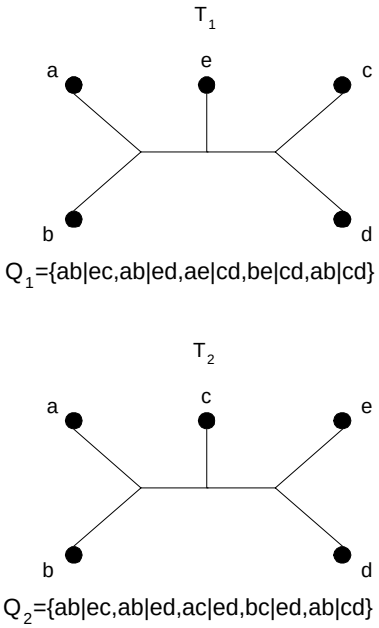

Figure 3.3: An example of the quartet distance. The quartet topologies in bold are common to both trees. The quartet distance is the symmetric difference between the two respective quartet topology sets.

instance, metrics that are based on transformation operations, such as NNI, ST and RF, do not distinguish between rearrangements that affect the relationships between many leaves and rearrangements that affect only a few. Moreover, the quartet metric can be used to handle unresolved trees [18, 14] by counting the quartet topologies that are unresolved separately. However, the NNI distance is undefined for unresolved trees and it is known that the RF distance handles unresolved trees rather poorly. Figure 3.4 illustrates this problem. Basically, RF can make two fully resolved trees farther than they are from an unresolved tree. This can be confusing and does not make sense in certain situations. For example, one could argue that since an unresolved tree provides

less information than a fully-resolved one, then why shouldn't it be farther away from a resolved tree than another resolved tree?

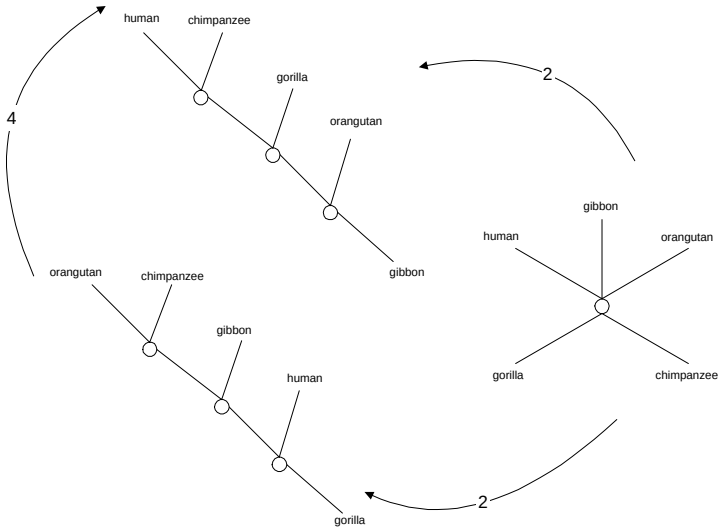

Figure 3.4: According to the RF metric, an unresolved tree is closer to a fully resolved tree than two fully resolved trees.

In addition, metrics that are based on the number of split differences (e.g. PM) are unstable with respect to the placement of a few leaves. That is, they can make two highly similar trees very distant. Figure 3.5 illustrates this potential problem. But the quartet metric is more stable especially when n is large [65]. Furthermore, the quartet metric has a far greater range than PM, and hence greater sensitivity [65] (see [18] for a more detailed discussion on the biological advantages of this metric). More importantly, the result of this chapter enables fast computation of the quartet metric and hence makes this nice metric a more practical one to use for large trees.

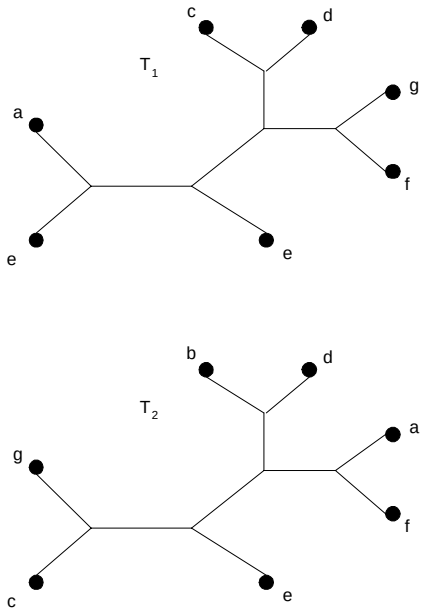

Figure 3.5: There is no common internal edge shared between T_1 and T_2. But in fact there are only two label swaps: a with g and b with c. Note that there are still a number of quartet topologies shared between them, so the trees are not as far away by the quartet metric, which is more accurate and reflects the real situation.

3.4 Computing the Quartet Distance

To compute (3.1), we need to determine $|Q_1 \cap Q_2|$ efficiently. The naive approach would be to compare the quartets one by one. This takes $O(n^4)$ time as there are $\binom{n}{4}$ quartets. To our knowledge, the best existing result is an unpublished algorithm that runs in $O(n^3)$ time [65]. Our contribution is a simple algorithm that runs in $O(n^2)$ time. The algorithm can also return implicitly the set of quartet topologies shared by two trees.

For simplicity, let the input to the algorithm be two fully-resolved unrooted evolutionary trees T_1 and T_2 labeled by S. The algorithm can be easily extended to handle partially-resolved trees. We will first give an overview of our algorithm and then follow up with the details and analysis in subsequent sections.

3.4.1 Algorithm Overview

The algorithm was motivated by the following observation. An internal edge e of the tree partitions the leaf labels into two disjoint sets $A, B \subset S$ such that $S = A \cup B$. For any two labels a_i,a_j from A and b_i,b_j from B, we have the quartet topology $a_ia_j|b_ib_j$ and we say the quartet topology is *induced* by e. This association of quartet topologies to internal edges gives us a simple framework to count common quartets. We only need to consider the $O(n^2)$ internal edge pairings between T_1 and T_2. However, a quartet topology can be induced by more than one edge. To avoid double counting, we perform pre-processing on the input trees. In the pre-processing stage, each internal edge claims as many induced quartet topologies as possible as long as the quartets it claimed have not been claimed by any neighbouring edges. The quartet topologies claimed by each edge can be encoded by a constant number of sets. Hence, we can determine the common quartet topologies claimed by two edges by computing the size of certain set intersections. The set intersection operation can be done in constant time if we pre-compute all possible set intersections. This can be done in $O(n^2)$ time as follows. Given an evolutionary tree T, let x, y, z be the neighbours of an internal node u. Three distinct binary trees rooted at u can be formed by removing one of the subtrees rooted at x,y, and z (see Figure 3.8). We called such rooted trees *rooted subtrees* of T. There are $O(n)$ such rooted subtrees for each input tree. The set intersection problem reduces to computing the common leaves for each of the $O(n^2)$ rooted subtree pairings (one from each input tree). We can process each pairing in constant time since we can first compute the pairings that involve their children. It follows that the sizes of all set intersections (also the intersections themselves) can be found in $O(n^2)$ time. Summing up the number of common quartet topologies between each pair of internal edges, one from each tree, gives the total number of agreed quartet topologies. This runs in $O(n^2)$ time since there are $O(n^2)$ internal edge

pairings.

3.4.2 Claiming Quartet Topologies

In this section we describe the procedure that pre-processes each of the input trees such that $O(n^4)$ quartet topologies of each tree are distributed into $O(n)$ quartet topology sets (or *qt-sets*), which is the key in achieving the $O(n^2)$ bound even though we are counting $O(n^4)$ objects.

For the following discussions, if A,B,C,D are disjoint and proper subsets of S, we use the following notations to denote qt-sets. $A\|B = \{pq|rs \mid p,q \in A, p \neq q, r,s \in C, r \neq s\}$, $A\|BC = \{pq|rs \mid p,q \in A, p \neq q, r \in B, s \in C\}$, and $AB\|CD = \{pq|rs \mid p \in A, q \in B, r \in C, s \in D\}$.

Consider an internal edge e of an evolutionary tree. It partitions the leaf labels into two disjoint sets $A,B \subset S$ such that $S = A \cup B$. We denote e as $e = (A,B)$. For any two distinct labels a_i,a_j from A and b_i,b_j from B, we have the quartet topology $a_ia_j|b_ib_j$. We denote the qt-set induced by e as $Q_{e=(A,B)} = \{a_ia_j|b_ib_j|a_i,a_j \in A, b_i,b_j \in B\}$. This association of quartets to internal edges gives us a simple framework to count quartets. A simple but *incorrect* idea would be to count common quartet topologies across all possible pairing of internal edges, one from T_1 and one from T_2. Given an edge $e = (A,B)$ from T_1 and $e' = (A',B')$ from T_2, the common quartet topologies are $((A \cap A')\|(B \cap B')) \bigcup ((A \cap B')\|(A \cap B'))$ This immediately gives us an $O(n^3)$ algorithm since there are $O(n^2)$ internal edge pairings and computing the set intersections takes $O(n)$ time using the trivial approach.

The problem with the algorithm just described is that a quartet topology can be induced by more than one edge. For instance, given the tree in Figure 3.6, the quartet $ab|cd$ is induced by both e_1 and e_2. This leads to incorrect counting.

In order to eliminate these duplications, we perform pre-processing on the input trees.

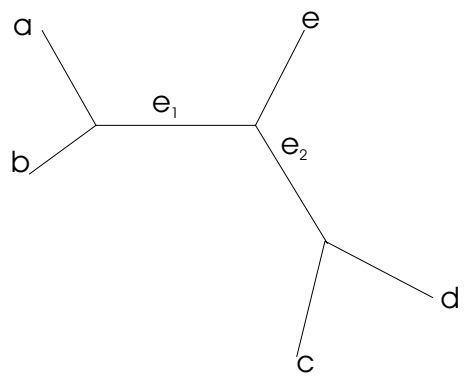

Figure 3.6: $ab|cd$ is induced by both e_1 and e_2

In the pre-processing stage, each internal edge claims as many induced quartets as possible as long as the quartets it claimed have not been claimed by its neighbouring edges. The nice thing is that each edge can only claim a constant number of qt-sets as in the naive (but wrong) algorithm. Hence, computing the number of agreed quartet topologies reduces to computing the size of certain set intersections.

Let us consider a general configuration of an internal edge e as illustrated in Figure 3.7. We will show how e claims its set of quartet topologies by examining its neighbouring edges e_1,e_2,e_3, and e_4. Being greedy, if none of its neighbouring edges has claimed anything, e claims as much as possible, that is, $(A \cup B)\|(C \cup D)$. We denote the set of quartets claimed by e as $claimed(e)$. In the case where e_1 has already claimed its set of quartets, e cannot claim any of the quartets in $A\|(B \cup C \cup D)$. Hence it can only claim $B\|(C \cup D)$ and $AB\|(C \cup D)$, which does not intersect $A\|(B \cup C \cup D)$. Note that each edge can at least claim $AB\|CD$ since they are only induced by e. Following this line of reasoning, e can claim quartet topologies according to the rules in Table 3.1. We use binary values to indicate whether the corresponding edge has claimed its set of quartet topologies according to the same rules.

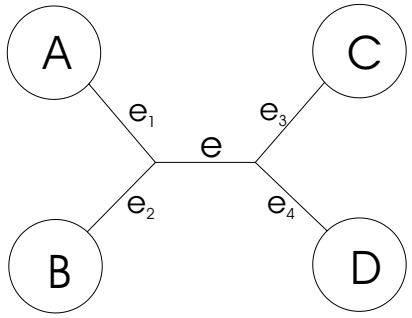

Figure 3.7: An internal edge e with its four neighbouring edges. A,B,C, and D are partitions of S.

e_1	e_2	e_3	e_4	quartets claimed by e
0	0	0	0	$(A \cup B)\|(C \cup D)$
0	0	0	1	$(A \cup B)\|C$,$(A \cup B)\|CD$
0	0	1	0	$(A \cup B)\|D$,$(A \cup B)\|CD$
0	0	1	1	$(A \cup B)\|CD$
0	1	0	0	$A\|(C \cup D)$,$AB\|(C \cup D)$
0	1	0	1	$A\|C$,$A\|CD$,$AB\|C$,$AB\|CD$
0	1	1	0	$A\|D$,$A\|CD$,$AB\|D$,$AB\|CD$
0	1	1	1	$A\|CD$,$AB\|CD$
1	0	0	0	$B\|(C \cup D)$,$AB\|(C \cup D)$
1	0	0	1	$B\|C$,$B\|CD$,$AB\|C$,$AB\|CD$
1	0	1	0	$B\|D$,$B\|CD$,$AB\|D$,$AB\|CD$
1	0	1	1	$B\|CD$,$AB\|CD$
1	1	0	0	$AB\|(C \cup D)$
1	1	0	1	$AB\|C$,$AB\|CD$
1	1	1	0	$AB\|D$,$AB\|CD$
1	1	1	1	$AB\|CD$

Table 3.1: Quartet claiming rules.

Lemma 3 *Given two neighbouring edges e_i, e_j, the claiming rules guarantee that $claimed(e_i) \cap claimed(e_j) = \emptyset$.*

Proof: From the way rules are constructed. □

As Lemma 3 states, the claiming rules only guarantee that two neighbouring edges

would not claim the same quartet topology. To ensure global exclusiveness, that is, no two internal edges would claim the same quartet topology, we need to avoid processing two edges in sequence that are *not* neighbours of each other. This can easily be done by ordering the internal edges by a depth-first traversal and process the edges in this order. Given an evolutionary tree T, let Q be the set of quartet topologies of T, E_{int} be the set of internal edges, and $order(e)$ be the order of edge e given by the depth-first traversal.

Lemma 4 *Processing E_{int} in a depth-first traversal ordering guarantees the following:*

1. $claimed(e_i) \cap claimed(e_j) = \emptyset$ *for all pair of* $e_i, e_j \in E_{int}$ *such that* $e_i \neq e_j$.
2. $\bigcup_{e \in E_{int}} claimed(e) = Q$

Proof: If e_i and e_j are neighbours, then (1) follows from Lemma 3. Otherwise, assume there exist a quartet topology $pq|rs$ that were claimed by two edges $e_i = (a_i, b_i)$ and $e_j = (a_j, b_j)$ which is *not* a neighbour of each other. This implies that $pq|rs$ is in both Q_{e_i} and Q_{e_j}. Let P be a path from a_i to b_j, clearly each edge in P induces $pq|rs$. Let e_i' and e_j' be edges in P which are neighbouring edges of e_i and e_j respectively. By Lemma 3, e_i claims $pq|rs$ if and only if $order(e_i) < order(e_i')$. Similarly e_j claims $pq|rs$ if and only if $order(e_j) < order(e_j')$. However, these conditions can not be both true given that the edge ordering is obtained by a depth-first traversal. If $order(e_i) < order(e_i')$, then one must reach e_j' before reaching e_j since P is the only path to e_j. Similarly, if $order(e_j) < order(e_j')$, then e_i' is reached before e_i.

To prove (2), Assume there exist a quartet topology $pq|rs$ that is *not* claimed by any edge. By (1) and given that each internal edge claims all quartet topologies it induces that have not yet been claimed by its neighbours, it follows that $pq|rs$ is not induced by any internal edge. But each quartet topology must be induced by at least one internal edge.

□

3.4.3 Computing the Set Intersections

After the above pre-processing, we need to compute the number of common quartet topologies in each qt-set pairing among the two input trees. For each qt-set pairing, this requires computing certain set intersections. For instance, $A\|B \cap C\|D = (A \cap C\| B \cap D) \cup (A \cap D\|B \cap C)$.

In order to achieve the overall $O(n^2)$ time bound, we first compute all set intersections that are needed. Clearly, we only need to consider a constant number of sets for each internal edge e. Namely, there are six sets (refer to Figure 3.7): A,B,C,D,$A \cup B$, $C \cup D$. When computing the set of common quartet topologies between any two edges, the intersections between any two such sets are needed. We pre-compute all such set intersections by the following approach.

Given a tree T, we denote the set of labels of T as $label(T)$. Consider an internal node u in the tree as illustrated in Figure 3.8. Recall that three distinct subtrees T_x,T_y,T_z rooted at u can be formed by removing one of the subtrees rooted at x,y, and z. There are $O(n)$ such rooted subtrees for T and we called such a collection as the *kernel forest* of T. Furthermore, $label(T_x) \cup label(T_y) \cup label(T_z) = S$.

Let Γ_1 and Γ_2 be the kernel forest of T_1 and T_2 respectively. For the following discussions, let $t_i \in \Gamma_1$ and $t_j \in \Gamma_2$. Define $t_i \cap t_j = label(t_i) \cap label(t_j)$. Clearly, computing $t_i \cap t_j$ for all $(t_i, t_j) \in \Gamma_1 \times \Gamma_2$ gives the required set intersections.

Let $height(T)$ be the height of a rooted tree T. Define $h(t_i, t_j) = \max\{height(t_i), height(t_j)\}$.

Lemma 5 *All set intersections can be found in $O(n^2)$ time.*

Proof: Order the elements of $\Gamma_1 \times \Gamma_2$ by h in ascending order and process them in this order. Clearly, trees with one node (the leaves) are processed first and their intersections

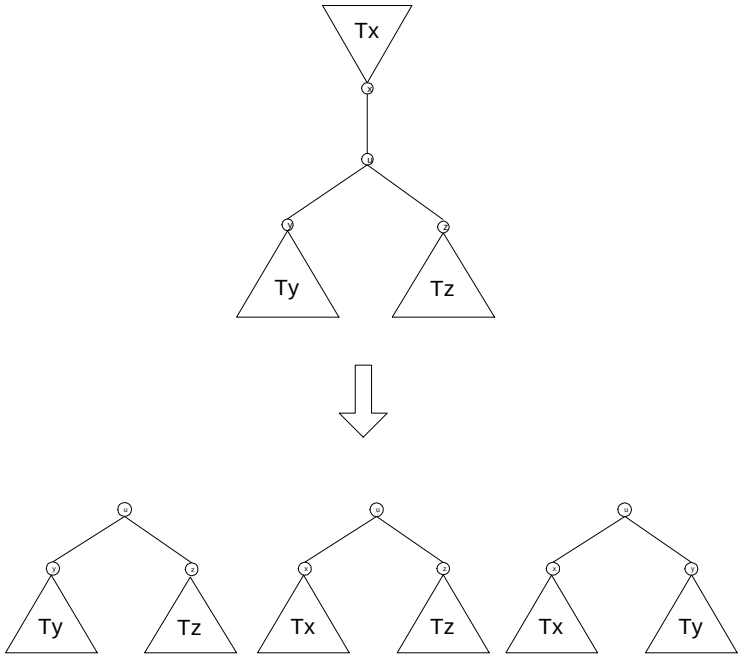

Figure 3.8: An internal node induces three rooted subtrees, T_x, T_y, and T_z.

can be found in $O(1)$ time. To determine any pair (t_i, t_j), let (t_{il}, t_{ir}) and (t_{jl}, t_{jr}) be the children of t_i and t_j respectively. Hence, $t_i \cap t_j = (t_{il} \cap t_{jl}) \bigcup (t_{ir} \cap t_{jr}) \bigcup (t_{ir} \cap t_{jl}) \bigcup (t_{il} \cap t_{jr})$ and each of $(t_{il} \cap t_{jl})$, $(t_{ir} \cap t_{jr})$, $(t_{ir} \cap t_{jl})$ and $(t_{il} \cap t_{jr})$ are known since their height is smaller. The lemma follows since $|\Gamma_1 \times \Gamma_2| = O(n^2)$.

□

Theorem 2 *Given two unrooted evolutionary trees T_1 and T_2, the number of quartet topologies shared by T_1 and T_2 can be determined in $O(n^2)$ time.*

Proof: Once T_1 and T_2 are pre-processed (which takes $O(n)$ time), the quartet topologies of each tree are represented as $O(n)$ qt-sets. The complete set of agreed quartet topologies is the union of the common quartet topologies between all possible internal edge pairings of T_1 and T_2. By Lemma 4, the above counting is guaranteed to be correct. Since the

quartet topologies claimed by each edge are represented as a constant number of qt-sets, we only need to consider all possible pairings of these qt-sets. From Table 3.1, possible types of qt-set include $X\|Y$, $PQ\|RS$, $M\|NO$. Hence there are six cases.

1. $X\|Y \bigcap X'\|Y'$
2. $X\|Y \bigcap PQ\|RS$
3. $X\|Y \bigcap M\|NO$
4. $PQ\|RS \bigcap P'Q'\|R'S'$
5. $PQ\|RS \bigcap MN\|O$
6. $MN\|O \bigcap M'N'\|O'$

Given that all set intersections are available after the $O(n^2)$ pre-processing procedure, each of the above can be computed in $O(1)$ time. Hence, computing the entire set of common quartet topologies takes $O(n^2)$ time since there are $O(n^2)$ edge pairings between T_1 and T_2.

□

REMARK 1 *Instead of just determining the size of set intersections, we in fact keep track of what is in the set intersections so that the shared quartet topologies is also returned implicitly by the algorithm. By implicit, we mean that the result is encoded as qt-sets rather than individual quartets.*

3.5 Discussions

We have presented an algorithm that computes all agreed quartet topologies between two evolutionary trees in $O(n^2)$ time. Sometimes biologists are interested in comparing

partially resolved trees. Our algorithm can easily be extended to handle such trees without increasing the time bound.

The bottleneck of this algorithm lies in the fact that all $O(n^2)$ internal edge pairings are considered. It would definitely be more efficient if some of these pairings can be eliminated. Note that our algorithm does not naturally lead to an efficient solution for finding common quartet topologies for k trees. For $k = 3$, it would take $O(n^3)$ time and for $k \geq 4$, straight counting would perform at least as good. It would be very interesting to see if ideas in this chapter can lead to a better than $O(kn^4)$ algorithm for $k \geq 4$. Finally, several problems are still open: is our algorithm optimal? Can a non-trivial lower bound for computing the quartet distance be proved?

Bibliography

[1] Adachi, J. 1995. (email: adachi@ism.ac.jp) *Modeling of molecular evolution and maximum likelihood inference of molecular phylogeny* Ph.D. dissertation, The Graduate University for Advanced Studies, Japan.

[2] Agodi, A. and Campanile, F. and Basile, G. and Viglianisi, F. and Stefani, S. 1999 *Phylogenetic analysis of macrorestriction fragments as a measure of genetic relatedness in staphylococcus aureus: the epidemiological impact of methicillin resistance* Eur. J. Epidemiol. 15(7):637-42. 1999.

[3] Arora, S. and Karger, D. and Karpinski, M. 1995. *Polynomial time approximation schemes for dense instances of NP–hard problems* Proceedings of the 27th ACM STOC, pages 284-293.

[4] Arora, S. and Lund, C. and Motwani, R. and Sudan, M. and Szegedy, M. 1992. *Proof verification and intractability of approximation problems* Proceedings of the 33rd ACM STOC, 13-22.

[5] Berry, V. and Bryant, D. and Kearney, P. and Li, M. and Jiang, T. Wareham, T. and Zhang, H. 2000. *A practical algorithm for recovering the best supported edges in an evolutionary tree* Proceedings of the 11th ACM-SIAM SODA, 2000.

[6] Bourque, M. 1978. *Arbres de steiner et reseaux dont varie l'emplagement de certains sommets* Univ. Montréal, Canada.

[7] Bryant, D. 1997. *Building trees, hunting for trees, and comparing trees* Ph.D. thesis, University of Canterbury, New Zealand.

[8] Buneman, P. 1971. *The recovery of trees from measures of dissimilarity* In Hodson, Kendall, and Tautu, editors, Mathematics in the Archaeological and Historical Sciences. Edinburgh University Press, Edinburgh.

[9] Cammarano, P. and Creti, R. and Sanangelantoni, A. and Palm, P. 1998. *The archaea monophyly issue: a phylogeny of translational elongation factor G(2) sequences inferred from an optimized selection of alignment positions* Molecular Evolution 49:524-537.

[10] Cao, Y. and Janke, A. and Waddell, P. and Westerman, M. and Takenake, O. and Murata, S. and Okada, N. and Pääbo, S. and Hasegawa, M. 1998. *Conflict among individual mitochondrial proteins in resolving the phylogeny of eutherian orders* Molecular Evolution 47:307-322.

[11] Cole,R. and Hariharan,R. 1996. *An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees* Proceedings of the 7th Annual ACM-SIAM SODA.

[12] DasGupta, B. and He, X. and Jiang, T. and Li, M. and Tromp, J. and Zhang, L. 1997. *On distances between phylogenetic trees* Proceedings of the 8th Annual ACM-SIAM SODA.

[13] Day, W. 1985. *Optimal algorithms for comparing trees with labeled leaves* J. Classif. 2:7-28. 1985.

[14] Day, W. 1986. *Analysis of quartet dissimilarity measures between undirected phylogenetic trees* Syst. Zool. 35(3):325-333.

[15] Day, W. and Sankoff, D. 1986. *Computational complexity of inferring phylogenies by compatibility* Syst. Zool., 35(2):224-229.

[16] Durbin, R. and Eddy, S. and Krogh, A. and Mitchison, G. 1998. *Biological sequence analysis* Cambridge University Press, Cambridge, England.

[17] Edwards, A. and Cavalli-Sforza, L. 1963. *The reconstruction of evolution* Ann. Hum. Genet., 27(105).

[18] Estabrook, G. and McMorris, F. and Meacham, C. 1985. *Comparison of undirected phylogenetic trees based on subtrees of four evolutionary units* Syst. Zool. 34(2):193-200

[19] Estabrook, G.F. 1972. *Cladistic methodology: a discussion of the theoretical basis for the induction of evolutionary history* Annu. Rev. Ecol. Syst., 3:427-456.

[20] Estabrook, G.F. and Johnson, C.S. and McMorris, F.R. 1975. *An idealized concept of the true cladistic character* Math. Biosci., 23:263-272

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

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

[23] Farach,M. and Przytycka, T. and Thorup, M. 1995. *Computing the Agreement of Trees with Bounded Degrees* European Symposium on Algorithms

[24] Farris, J.S. 1973. *A probability model for inferring evolutionary trees* Syst. Zool., 22:250-56.

[25] Felsenstein, J. 1977. *The number of evolutionary trees* Syst. Zoo.

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

[27] Felsenstein, J. 1973. *Maximum likelihood and minimum-steps methods for estimating evolutionary trees from data on discrete characters* Syst. Zool., 22:27-33.

[28] Felsenstein, J. 1981. *Evolutionary trees from DNA sequences: A maximum likelihood approach* Journal of Molecular Evolution 17:368-376

[29] Felsenstein, J. 1982. *Numerical methods for inferring evolutionary trees* The Quarterly Review of Biology, 57(4):379-404.

[30] Felsenstein, J. 1993. *Phylogenetic Inference Package (PHYLIP), Version 3.5* University of Washington, Seattle.

[31] Garey, M. and Johnson, D. 1977. *The rectilinear steiner tree problem is NP–Complete* SIAM J. Appl. Math. 32:826-834.

[32] Ge, S. and Lu, T. and Hong, D. 1999. *Phylogeny of rice genomes with emphasis on origins of allotetraploid species* Proc. Natl. Acad. Sci. USA. 96:14400-14405.

[33] Graham, R. and Foulds, L. 1982. *Unlikelihood that minimal phylogenies for a realistic biological study can be constructed in reasonable computational time* Math. Biosci. 60:133-142.

[34] Gusfield, D. 1991. *Efficient algorithms for inferring evolutionary history* Networks, 21:19-28.

[35] Gusfield, D. 1997. *Algorithms on strings, trees, and sequences* Cambridge University Press, Cambridge, England.

[36] Hannenhalli, S. and Pevzner, P. 1995. *Transforming mice into men: polynomial algorithm for genomic distance problem* Proceedings of the 36th IEEE FOCS, 581-92.

[37] Hennig, W. 1950. *Grundzüge einer Theorie der Phylogenetischen Systematik* Deutscher Zentral-verlag, Berlin.

[38] Hillis, D. and Huelsenbeck, J. and Swofford, D. 1994. *Hobgoblin of phylogenetics?* Nature, 369:363-364.

[39] Huelsenbeck, J. 1997. *Is the Felsenstein zone a fly trap?* Systematic Biology, 42(3):247-264.

[40] Huelsenbeck, J. and Hillis, D. 1994. *Success of phylogenetic methods in four-taxon cases* Systematic Biology, 42(3):247-264.

[41] Jeffrey, B. and Wesley, B. 1998. *Big trees from little genomes: mitochondrial gene order as a phylogenetic tool* Current Opinion in Genetics and Development 8:668-674.

[42] Jiang, T. and Kearney, P. and Li, M. 1998. *Orchestrating quartets: approximation and data correction* Proceedings of the 39th IEEE FOCS, pages 416-425.

[43] Jukes, T.H. and Cantor, C.R. 1969. *Evolution of protein molecules* In: Mammalian protein metabolism, H.N. Munro, ed., pages 21-123. New York: Academic Press

[44] Kannan, S. and Warnow, T. 1995. *Tree reconstruction from partial orders* SIAM J. Computing 24(3):511-519.

[45] Kearney, P. 1997. *The relationship between a phylogeny and its ordinal assertions* Ph.D. thesis, Department of Computer Science, University of Toronto

[46] Kearney, P. 1998. *The ordinal quartet method* Proceedings of the Second Annual International Conference on Computational Molecular Biology, pages 125-134.

[47] Kearney, P. 2000. *The development of phylogenetic methods* Manuscript in preparation.

[48] Kearney, P. and Li, M. and Tsang, J. and Jiang, T. 1999. *Recovering branches on the tree of life: an approximation algorithm* Proceedings of the 10th Annual ACM-SIAM SODA, pages 537-546, 1999.

[49] Kececioglu, J. and Ravi, R. 1995. *Of mice and men: algorithms for evolutionary distances between genomes with translocation* Proceedings of the 6th ACM-SIAM SODA, 307-25.

[50] Kimura, M. 1980. *A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences* Journal of Molecular Evolution 16:111-120.

[51] Klug, W. and Cummings, M. 2000. *Concepts of Genetics, 6th Edition* Prentice Hall, Upper Saddle River, New Jersey

[52] Li, M. and Tromp, J. and Zhang, L. 1996. *Some notes on the nearest neighbour interchange distance* Journal of Theoretical Biology 182:463-467

[53] Li, M. and Vitanyi, P. 1997. *Kolmogorov complexity and its applications* Springer-Verlag, New York.

[54] Li, W.H. 1997. *Molecular evolution* Sinauer Associates, Inc.

[55] Maidak, B. and Cole, J. and Parker Jr, C. and Garrity, G. and Larsen, N. and Li, B. and Lilburn, T. and McCaughey, M. and Olsen, G. and Overbeek, R. and Pramanik, S. and Schmidt, T. and Tiedje, J. and Woese, C. 1999. *A new version of the RDP (Ribosomal Database Project)* Nucleic Acids Res. 27:171-173 (1999).

[56] Meidanis, J. and Setubal, J. 1997. *Introduction to computational molecular biology* PWS Publishing, Boston, Mass.

[57] Rambaut, A. and Grassly, N.C. 1996. *Seq-Gen: an application for the monte carlo simulation of dna sequence evolution along phylogenetic trees*

[58] Robinson, D. and Foulds, L. 1979. *Comparison of weighted labelled trees* Lecture notes in mathematics vol. 271 119-126, Springer-Verlag, Germany.

[59] Robinson, D. and Foulds, L. 1981. *Comparison of phylogenetic trees* Math. Biosci. 53:131-147.

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

[61] Sokal, R. and Rohlf, F. 1962 *The comparison of dendrograms by objective methods* Taxon 11:33-40.

[62] Sokal, R. and Sneath, P. 1963. *Principals of numerical taxonomy* W.H. Freeman, San Francisco.

[63] Sorel, F. and Christopher, H. 1999. *Whole genome-based phylogenetic analysis of free-living microorganisms* Nucleic Acids Research 27(21):4218-4222.

[64] Steel M. 1992. *The complexity of reconstructing trees from qualitative characters and subtrees* J. of Classification, 9:91-116.

[65] Steel, M. and Penny, D. 1993. *Distribution of tree comparison metrics–some new results* Syst. Biol. 42(2):126-141.

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

[67] Stetter, K. and Lauerer, G. and Thomm, M. and Neuner, A. 1987. *Isolation of extremely thermophilic sulfate reducers: Evidence for a novel branch of archaebacteria* Science 236: 822-824.

[68] Sung, W. 1998. *Fast labeled tree comparison via better matching algorithms* Ph.D. thesis, University of Hong Kong, Hong Kong.

[69] Swofford, D.L. and Olsen, G.J. and Waddell, P.J. and Hillis, D. 1996 *Phylogenetic Inference* in Molecular Systematics, 2nd Edition, Sinauer Associates, Mass.

[70] Walsh, B. *EEB105 biology lecture notes* University of Arizona, Arizona.

[71] Waterman, M. and Smith, T. 1978 *On the similarity of dendrograms* J. Theoret. Biol 73:789-800.

[72] Winter, P. 1987. *Steiner problem in networks: a survey* Networks 17:129-167.

[73] Xin, C. and Kwong, S. and Li, M. 1999. *A compression algorithm for DNA sequences and its applications in genome comparison* Genome Informatics Workshop, 1999, Dec. Japan

[74] Yang, Z. 1996. *Phylogenetic analysis by maximum likelihood(PAML), version 1.2*

[75] Zhang, H. 2000. *Design, implementation, analysis of a novel quartet-based phylogenetic reconstruction method* M.Math thesis, University of Waterloo.

[76] Zuckerkandl, E. and L. Pauling. 1965. *Molecules as documents of evolutionary history* Journal of Theoretical Biology Volume 8, 357-366. 1965.