Fast Algorithms for Large-Scale Phylogenetic Reconstruction

by

Jakub Truszkowski

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2013

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

One of the most fundamental computational problems in biology is that of inferring evolutionary histories of groups of species from sequence data. Such evolutionary histories, known as *phylogenies* are usually represented as binary trees where leaves represent extant species, whereas internal nodes represent their shared ancestors. As the amount of sequence data available to biologists increases, very fast phylogenetic reconstruction algorithms are becoming necessary. Currently, large sequence alignments can contain up to hundreds of thousands of sequences, making traditional methods, such as Neighbor Joining, computationally prohibitive. To address this problem, we have developed three novel fast phylogenetic algorithms.

The first algorithm, QTree, is a quartet-based heuristic that runs in $O(n \log n)$ time. It is based on a theoretical algorithm that reconstructs the correct tree, with high probability, assuming every quartet is inferred correctly with constant probability. The core of our algorithm is a balanced search tree structure that enables us to locate an edge in the tree in $O(\log n)$ time. Our algorithm is several times faster than all the current methods, while its accuracy approaches that of Neighbour Joining.

The second algorithm, LSHTree, is the first sub-quadratic time algorithm with theoretical performance guarantees under a Markov model of sequence evolution. Our new algorithm runs in $O(n^{1+\gamma(g)} \log^2 n)$ time, where γ is an increasing function of an upper bound on the mutation rate along any branch in the phylogeny, and $\gamma(g) < 1$ for all g. For phylogenies with very short branches, the running time of our algorithm is close to linear. In experiments, our prototype implementation was more accurate than the current fast algorithms, while being comparably fast.

In the final part of this thesis, we apply the algorithmic framework behind LSHTree to the problem of placing large numbers of short sequence reads onto a fixed phylogenetic tree. Our initial results in this area are promising, but there are still many challenges to be resolved.

Acknowledgements

First, I would like to thank my advisor, Dan Brown, for all his guidance, encouragement, and patience throughout my PhD. I want to thank my committee members: David Sankoff, Ming Li, Shai Ben-David and Jonathan Witt for taking the time to read this thesis and for their helpful comments. I would also like to thank the many people with whom I had the opportunity to discuss various ideas related to this research: David Bryant, Miklos Csűrös, Yanqi Hao, Ming Li, Andre Masella, Erick Matsen, Peter Meinicke, Josh Neufeld, Sebastien Roch, and others. In particular, I would like to thank Josh Neufeld and Andre Masella for sharing some of their data sets, and for useful discussions about metagenomics. I would also like to thank Yanqi Hao for his help with some of the experiments, and Peter Meinicke for introducing me to the problem of phylogenetic placement. In addition, I want to thank Ming Li and Pascal Poupart for interesting discussions and useful advice they have given me on many occasions.

Finally, I would like to thank all of my family and friends for their kindness and support during the past five years. I am particularly grateful to my parents, Tadeusz and Ewa, and my aunt Alicja, as well as my friends Elena Andronic, Rahul Vats, and Qi Zhang for all of their support.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

One of the fundamental questions in biology is determining the evolutionary origin of living organisms. It is believed that all life on earth originated in a single common ancestor. That universal ancestor subsequently split into several distinct lineages, in a phenomenon known as speciation. Further speciation and extinction events followed, gradually increasing the total number of species. Meanwhile, mutations and diverse selective pressures in different environments caused organisms to differentiate from each other, gradually giving rise to the diversity of life as we know it today. After over 3 billion years of evolution, the Earth is home to 1.2 million documented species [140] and it is estimated that many more are yet to be discovered. For example, a recent study by Mora *et al.* puts the estimated total number of species at 8 million [124]. Investigating the common evolutionary history of various groups of organisms is a mammoth task for biologists.

Phylogenetics is the study of evolutionary histories. Such histories can be conveniently represented as graph-theoretic trees, or *phylogenies*, where leaves correspond to present-day species, whereas internal nodes correspond to past speciation events - see Figure 1.1. The root of the tree represents the most recent common ancestor of all species in the tree. A typical phylogenetic analysis starts with a set of sequences from different species. A phylogenetic algorithm compares these sequences to reconstruct the tree that best explains the similarities between different species. Similar sequences are assumed to come from closely related species.

The idea of reconstructing evolution by examining similarities between different organisms has a long history. Even before evolutionary theory, Linnaeus created a hierarchical classification of over 11000 species of plants and animals [105], though he made no presumption in his classification that nearby species shared common ancestry. Darwin was the first

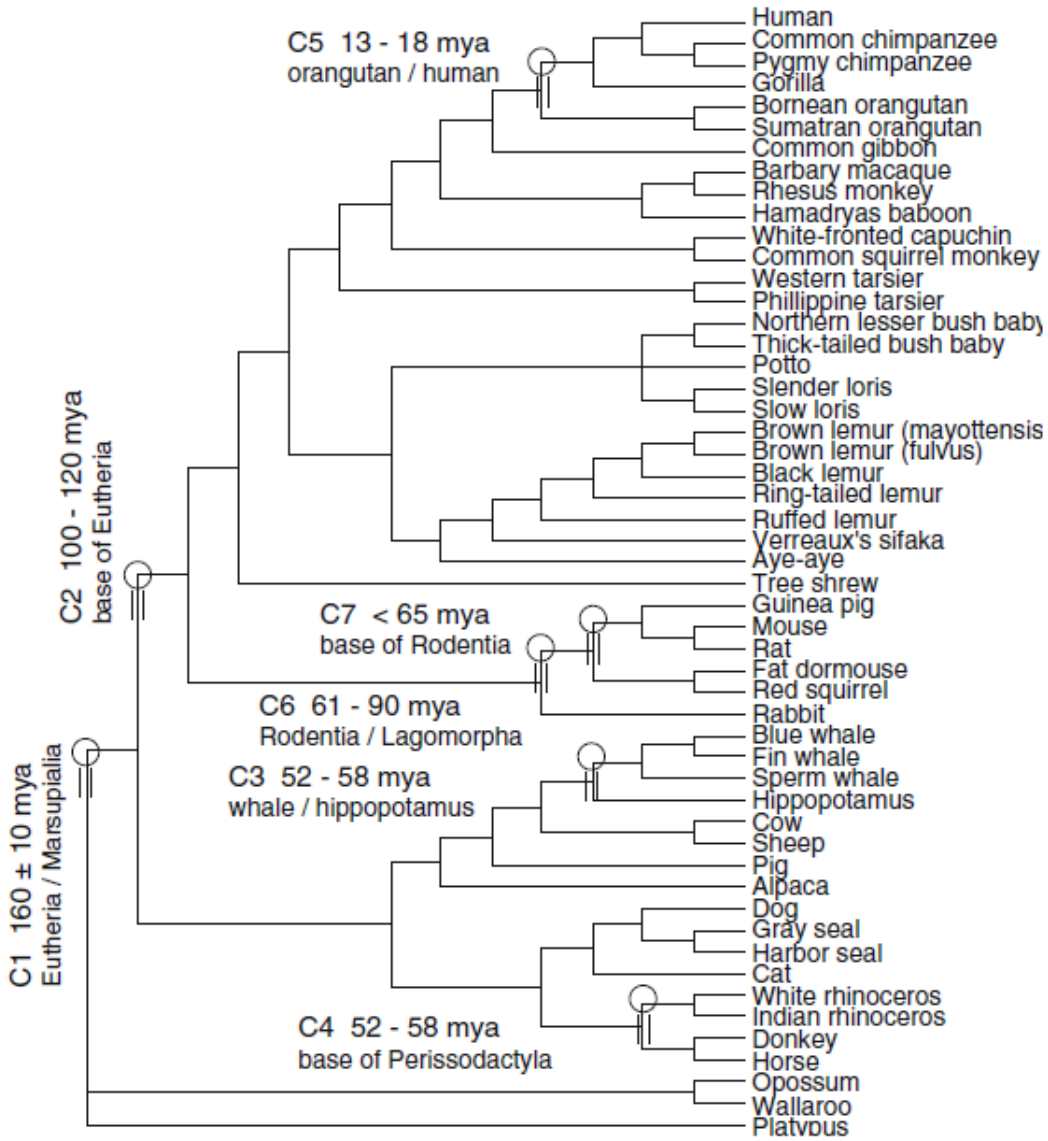

Figure 1.1: A phylogeny of mammals, taken from Matsui and Hasegawa [117], page 257.

to observe the shared evolutionary origin behind the similarities between organisms [41], and formally introduced the concept of phylogenies.

Up until the 1970s, phylogenies were reconstructed from morphological data [88]. For example, researchers would compare the number of toes in different mammals, and postulate that species with the same number of toes form a single taxonomic group [73]. This often required considerable human effort to identify which physical features of species are most informative. Since the introduction of sequencing in the 1970s, systematists have adopted sequences (DNA, RNA and protein sequences) as the main source of data for phylogenetic analyses [68]. The use of sequence data has enabled the development of principled numerical methods that require less human effort and allow for rigorous statistical interpretation of the data. Researchers continue to use morphological data in limited ways, most notably in the analysis of fossils [79].

The increased amount of data and new computational tools broadened the range of possible research questions. Researchers now use phylogenetic methods to investigate evolution within a single species: for example, to investigate the evolution of different strains of the influenza virus [171]. or even the history of cells within a single organism [72]. Thus, phylogenetic analysis has become an essential technique in many areas of biology and medical science. To reflect this broad range of applications of phylogenetics, researchers commonly refer to the basic units of analysis as *taxa*, rather than species.

In the past 15 years, improvements in sequencing technology have led to a spectacular increase in the amount of sequence data available to researchers. Researchers have compiled large data sets containing hundreds of thousands of aligned sequences. For example, the GreenGenes database [48] contains alignments of ribosomal RNA sequences from hundreds of thousands of microbial strains. Other databases exist for monitoring the evolution of pathogens such as HIV [2] or HCV [102]. Several large-scale projects are under way with the ultimate goal of reconstructing the complete tree of life, for all known species [1, 3].

Analyzing the evolutionary history of such large data sets requires highly scalable algorithms, able to infer accurate phylogenies from hundreds of thousands of sequences. The goal of this thesis is to propose new algorithms in this domain, and to motivate the use of these algorithms, both theoretically and experimentally.

1.1 Preliminaries

1.1.1 Phylogenies

Let S be a set of taxa. A phylogeny is a binary tree (rooted or unrooted) whose leaves are labelled with elements of S. Internal nodes represent past speciation events. If the most recent common ancestor of S is known, it is denoted as the root of the tree.

Branches can be labelled with positive weights to represent evolutionary time. We assume that mutations occur according to a random process along the branches of the tree. Consequently, we will measure evolutionary time in units of the number of expected mutations per site of a common multiple sequence alignment [68]. Evolutionary time cannot be easily translated to calendar time, as mutation rates differ across lineages. For example, the nucleotide substitution rate in rodent genomes appears to be around two times higher than in the human genome [167].

For a set S of n taxa, the number of possible tree topologies is very large. The following lemma is widely known [68, 153].

Proposition 1. *The number of unrooted binary tree topologies on n taxa is*

$$\alpha(n) = 1 \cdot 3 \cdot 5 \cdot \ldots \cdot (2n-5) = \frac{(2n-5)!}{2^{n-3}(n-3)!}$$

Algorithms that enumerate all possible topologies in search of an optimum are thus impractical, except for very small numbers of taxa.

1.1.2 Sequences, mutations and alignments

Over time, sequences evolve by accumulating mutations. For the purpose of reconstructing phylogenies, the most important types of mutations are substitutions, insertions and deletions. Substitutions change one character in the sequence to another. Insertions and deletions add or remove several characters from a sequence. Since past insertions and deletions are usually indistinguishable when comparing two sequences, we refer to them collectively as *indels*. To reliably estimate the evolutionary distance between two sequences, we need to know which pairs of sites have arisen from a common ancestral site. Such sites are called *homologous*. Because indels are common, identifying homologous positions in two sequences is not straightforward.

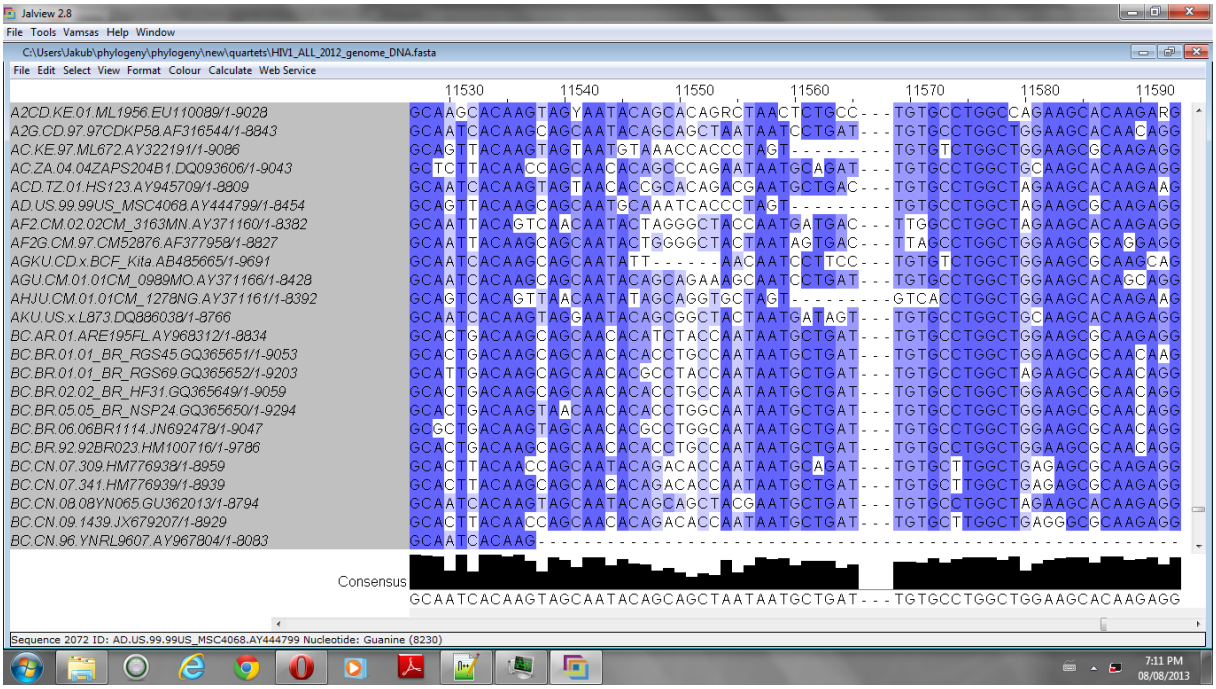

Figure 1.2: A fragment of a large HIV sequence alignment from the Los Alamos HIV Sequence database [2]. Colours indicate percentage identity within columns. We used Jalview [165] to display the alignment.

For n DNA sequences, a sequence alignment A is an $n \times m$ matrix with entries from $\{A, C, G, T, -\}$. Each row of A contains the bases of the i-th sequence, possibly separated by *gaps* which denote insertion or deletion events. Bases in the same column are hypothesized to have descended from the same ancestral site. Figure 1.2 shows a typical sequence alignment.

Building large and accurate sequence alignments is a hard computational problem. Many tools exist for building multiple sequence alignments; popular ones include MUSCLE [58], MAFFT [98] and ProbCons [53]. Other tools align a new sequence to an already constructed multiple sequence alignment. The two most popular tools in this category are HMMer [69] and Infernal [129]. For a survey of sequence alignment tools, see Notredame [132].

In this thesis, we are particularly interested in reconstructing phylogenies for very large alignments, with tens to hundreds of thousands of sequences. Many standard alignment programs are prohibitively slow on such large data sets. For these reasons, many of the

alignments that our methods are intended for are built by first building a "seed" alignment for a smaller set of sequences, and then adding the rest of the sequences using HMMer or Infernal. This strategy produces large alignments in running times that are close to linear in the number of sequences, possibly at a cost of accuracy.

We assume for simplicity that our sequence alignments do not contain errors. This is a standard assumption taken in virtually all phylogenetic methods used to date. Real alignments often contain errors, which may impact the accuracy of the phylogenetic method. Several strategies are used to limit the effect of errors on the accuracy of phylogenetic methods. One strategy is to manually curate alignments, using expert knowledge to correct some of the alignment errors. Another approach is to automatically discard unreliable columns from further analysis, using tools such as the one by Cline *et al.* [38].

1.2 Summary of the results

In this thesis, we describe three novel algorithms aimed at analyzing the evolutionary history of large numbers of DNA and RNA sequences. We describe them briefly below.

Our first algorithm, QTree, reconstructs a phylogeny by first building a *guide tree* on a small subset of the taxa, and then sequentially adding the other taxa into the guide tree to obtain a phylogeny for the whole data set. The core of the algorithm is a balanced search tree structure that locates an edge in a number of steps that is logarithmic in the number of taxa. Each step consists of reconstructing a *quartet* - a small tree on just four taxa. The data structure is an analog of binary search trees for phylogenetic trees. We show that when the taxa are inserted in random order, the resulting search tree is balanced with high probability. To accommodate for possible errors in quartet estimates, we the algorithm runs a random walk on the search tree that is guaranteed to reach the correct node in the search tree with high probability, under a simple error model where all quartet queries err independently with known probability. Under this model, the trees produced by our algorithm are guaranteed to be correct with probability that tends to 1 for large numbers of taxa. The running time of our algorithm is $O(n \log n)$, which is optimal for any phylogenetic algorithm that can produce all possible topologies.

Our assumed error model does not reflect the characteristics of errors encountered in the data. This leads to a rather poor performance of the basic algorithm. We introduce a number of simple improvements to the algorithm with the goal of improving the accuracy. Most of the improvements seek to reduce the effect of unreliable quartet inferences on the reconstructed tree. In experiments, the improved version of our algorithm produces

trees whose accuracies reach the level of the widely-used Neighbour Joining algorithm. The running time of our algorithm is orders of magnitude lower than that of Neighbour Joining, and several times lower than the fastest current phylogenetic algorithm, FastTree. While these results are encouraging, theoretical and experimental evidence implies that there are fundamental obstacles to further increasing QTree's accuracy. Our results on QTree appear in a number of conference and journal papers [21, 23, 22, 162].

Our second algorithm, LSHTree, is a principled fast algorithm that explicitly models evolution as a Markov process. Like many phylogenetic algorithms, LSHTree starts with a forest where each taxon is its own tree, and then successively merges forests until a single tree is built. Unlike other methods, our algorithm avoids evaluating every possible merge of trees by using *locality-sensitive hashing* to focus on a small number of sequence pairs from different trees in the forest that are likely to be close to each other in the true tree. This hashing step is the key idea behind the efficiency of our algorithm. We also reconstruct ancestral sequences at internal nodes to ensure that for each pair of subtrees that should be merged, there exists a pair of close sequences that can be detected by the hash tables. The main technical challenge in designing the algorithm is efficiently using the information in ancestral sequences. Under Markov models of evolution, our algorithm is guaranteed to correctly reconstruct the phylogeny provided that the sequences have length $\Omega(\log n)$. This guarantee matches the best theoretical algorithms, and is optimal up to a constant. The running time of our algorithm is $O(n^{1+\gamma(g)} \log^2 n)$, where the exponent $1 + \gamma(g)$ depends on the branch lengths in the true tree and is always less than 2, provided that the branch lengths in the true tree are not too long. This runtime is considerably lower than all previous algorithms with theoretical guarantees for Markov models of evolution [62, 40, 43, 101]. Earlier results suggested that no accurate distance-based phylogenetic algorithm can be faster than $O(n^2/\log\log n)$, for most trees [101]. Our algorithm circumvents that lower bound by using information in the sequences, in addition to distance estimates. This shows that accurate phylogenetic reconstruction is possible in sub-quadratic runtimes, which could have been doubted in the light of previous results. In experiments, the accuracy of LSHTree usually surpasses that of QTree and FastTree, while its running time is only slightly slower than that of QTree. Thus, LSHTree shows that fast and accurate phylogenetic reconstruction is possible, both in theory and in practice. This algorithm appears in a recent conference publication [19].

In the final technical chapter of this thesis, we present our efforts to adapt the algorithmic framework behind LSHTree to the related problem of placing short sequence reads onto an available phylogenetic tree of a set of reference taxa. Abstractly, this problem can be thought of as a special case of the phylogeny reconstruction problem. In practice, however, the parameters of the data sets lead to a number of unique challenges not encoun-

tered in standard phylogenetic analyses. The reads are typically very short, which means the amount of information in the data is much lower than in other data sets. Many reads originate from taxa that are very distant from the reference tree, further complicating the problem of finding the optimal placement. To ameliorate these problems, we introduce several improvements to our locality-sensitive hashing scheme aimed at choosing the most informative columns for hashing. We also employ local likelihood search to improve the accuracy. The resulting algorithm is 8-25 times faster than the leading algorithm for the problem, while its accuracy is only moderately lower. Despite the progress we have made in the past months, several challenges remain, and we outline future research directions aimed at addressing them. An early version of the algorithm appeared in our conference publication [20].

Chapter 2

Background and related work

The problem of reconstructing phylogenetic trees has a long history. Starting in the late nineteenth century, researchers used morphological features (e.g. shape of feathers, number of toes, etc.) to reconstruct evolutionary histories [68]. As sequencing technology became cheaper and more available in the 1970's and 1980's, DNA and protein sequences became the primary data source for phylogenetic analysis of extant species. One of the main reasons behind the move from morphological to molecular characters was the availability of standardized computational methods for reconstructing and analyzing molecular data.

Over the past 40 years, a variety of computational methods have been developed for reconstructing phylogenies from aligned sequences. These approaches can be divided into two broad categories. Probabilistic approaches model sequence evolution as a stochastic process, and then try to find an evolutionary history that best fits the data under the assumed model. Combinatorial approaches seek to optimize a certain objective function of the data and the tree over the space of all phylogenetic trees. In this chapter, we review the main concepts and algorithms developed in probabilistic phylogeny reconstruction. These concepts provide the theoretical foundation for most of the work in this thesis, which is also probabilitistic in nature. We also discuss an important paradigm in combinatorial phylogenetics, known as *quartet methods*.

We do not give a full overview of phylogenetic reconstruction methods; for an extensive treatment of all major paradigms in phylogenetics, the reader is referred to the excellent book by Felsenstein [68]. For a more mathematical perspective, see books by Semple and Steel [145], or Gascuel [76].

2.1 Probabilistic models of evolution

We assume sequences evolve according to a continuous-time Markov process on a tree. Each column in the sequence alignment evolves independently of others, according to the same process. At each site, mutations appear randomly in time according to a Poisson process with rate equal to 1. The frequency of various kinds of mutations is specified by the *rate matrix* M, where each off-diagonal entry M_{ij} represents the expected number of times each i mutates into j per unit time. Diagonal entries are chosen so that each row of M sums to 0. The matrix is normalized so that the sum of off-diagonal elements in each row equals 1, which means that time is measured in expected substitutions per site. For a time t, we can derive a transition matrix $P(t)$ where $P(t)_{(ij)}$ is the probability of character i being replaced by j after time t, summing over all possible sequences of mutations in time t.

$$\mathbb{P}(t) = e^{tM} = \mathbb{I} + \frac{tM}{1!} + \frac{(tM)^2}{2!} + \frac{(tM)^3}{3!} + \dots$$

The simplest evolutionary model is the Cavender-Farris-Neyman (CFN) model [33, 62], on just two characters. Its rate matrix is:

$$\mathbb{M}_{\mathbb{CFN}} = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}.$$

In this model, the probability of a state differing at two ends of an edge is $p(e) = \frac{1}{2}[1 - \exp(-2\ell(e))]$ where $\ell(e)$ is the length of the edge (again, measured in the expected number of mutations per site); or, we can write $\ell(e) = -\frac{1}{2}\log(1-2p(e))$. The maximum likelihood estimator of the distance between two properly aligned binary sequences is

$$\hat{d}(a,b) = -\frac{1}{2}\log(1 - 2\hat{p}(a,b)),$$

where $\hat{p}(a,b)$ is the proportion of differing sites between sequences at a and b. Note that if $\hat{p}(a,b) > 0.5$, $\hat{d}(a,b) = \infty$. This estimate is statistically consistent, and gives a concentration inequality on its accuracy, as a function of sequence length. For more background on probabilistic models of evolution, see Felsenstein [68].

For DNA sequences with a 4-letter alphabet, the Jukes-Cantor model is the simplest, and has rate matrix:

$$\mathbb{M}_{\mathbb{JC}} = \begin{pmatrix} -1 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & -1 & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & -1 & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & -1 \end{pmatrix}$$

The corresponding maximum-likelihood distance estimator is

$$\hat{d}_{JC} = -\frac{3}{4}\log(1 - \frac{4}{3}\hat{p})$$

More complicated models have been developed to account for the biochemical similarities between different nucleotides, or the redundancy of the genetic code. For simplicity, we use the CFN model in our theoretical studies, as derivations for other models are usually similar [139, 62]. For more background on probabilistic models of evolution, see [68].

2.2 Probabilistic paradigms for inferring phylogenies

2.2.1 Maximum Likelihood

Given a probabilistic model of evolution and a sequence alignment, perhaps the most natural approach is to choose the tree topology T, along with associated edge lengths ℓ, that makes the sequence alignment A most likely. In other words, we want to maximize the following:

$$\arg\max_{T,\ell} \Pr[A|T,\ell]$$

This framework of inference is known as Maximum Likelihood, and has enjoyed widespread popularity in phylogenetics and other domains in bioinformatics. Its first use in phylogenetics dates back to Felsenstein [67]. Since then it has become the most widely-adopted paradigm of phylogenetic inference, and is the principle behind several popular programs such as RAxML [151], PhyML [85], and FastTree [134].

Let us take a closer look at the phylogenetic likelihood. Since we assume that each column of the alignment evolves independently of the other columns, we can write this probability as the product of column likelihoods:

$$\Pr[A|T,\ell] = \prod_{j=1}^{m} \Pr[A_{*j}|T,\ell]$$

where A_{*j} is the j-th column of the alignment. The likelihood of each alignment column is the total probability of generating that column from the Markov model of evolution on tree T, summed over all possible states at internal nodes. For DNA alphabets, we have:

$$\Pr[A_{*j}|T,\ell] = \sum_{(s_1,s_2,...,s_{n-2})\in\{A,C,G,T\}^{n-2}} \Pr[A_{*j}, \sigma_1 = s_1, \sigma_2 = s_2, ..., \sigma_{n-2} = s_{n-2}|T,\ell]$$

where σ_i's represent the characters at internal nodes of the tree. This sum has 4^{n-2} components. This is prohibitively large even for moderate-size trees. Fortunately, there is a faster way of computing the likelihood, as we explain below.

Computing the likelihood

Felsenstein [67] introduced an elegant dynamic programming algorithm for computing the likelihood of an alignment under a phylogenetic tree. The algorithm maintains a table D with partial likelihoods for each rooted subtree T_i of T, for all choices of ancestral charater at the root r_i of T_i. Formally:

$$D_{ik} = \Pr[A_{descij}|T_i, r_i = k]$$

for $k \in \{A, C, G, T\}$. Computing D_{ik} can be done efficiently in a bottom-up fashion. Let l and r be the two children of i. We have [68]

$$D_{ik} = (\sum_x \Pr[x|k, t_l]D_{lx})(\sum_y \Pr[y|k, t_r]D_{ry})$$

where t_l and t_r are the lengths of the branches leading to l and r from i. The total likelihood of the alignment column is thus $\sum_x \pi_x D_{ix}$. Multiplying the values for each column gives the total likelihood for the alignment.

For most real alignments, the likelihood is very small due to the large number of columns. This can lead to numerical underflows. For this reason, phylogenetic programs usually store the logarithm of the likelihood. For a more elaborate discussion on avoiding numerical underflows in probabilistic models, see the book by Durbin *et al.* [57].

Felsenstein's algorithm can be used to infer ancestral states at the internal nodes of the phylogeny. This application of the algorithm will be of interest in Section 4, where we describe a fast algorithm that makes extensive use of ancestral state reconstruction. The problem of reconstructing ancestral proteins has recently gathered much attention from biologists [104, 15]. One interesting example is a study by Chang *et al.* [34] reconstructing ancestral protein responsible for the visual pigment of the common ancestor of archosaurs in the Triassic period.

Finding the best tree

The computational complexity of finding the Maximum Likelihood phylogeny remained an open problem for over 20 years. In 2005, Chor and Tuller [37] showed that finding the

maximum-likelihood tree for a given set of sequences and a given evolutionary model is NP-hard in general. Roch [138] subsequently provided a simpler proof.

In practice, finding reasonable approximations of the maximum likelihood tree is not quite as hard as the NP-hardness result suggests. Over the years, a number of efficient heuristics have been developed for the problem. These heuristics generally start by constructing an initial tree using a faster distance-based method (see Section 2.2.2), then use local hill-climbing strategies to improve the likelihood of the tree. Hill-climbing is done using *edit operations*. The three most commonly used edit operations are:

1. Nearest Neighbour Interchange (NNI) - this operation takes two adjacent subtrees and swaps them to create a new topology. The number of possible NNI moves is proportional to the number of edges in the phylogeny.

2. Subtree Prune and Regraft (SPR) - As the name suggests, an SPR operation takes a subtree and reattaches it onto a different edge in the tree. Since there are $O(n)$ choices of subtrees and $O(n)$ possible regraft locations, the overall number of possible SPR moves is $O(n^2)$.

3. Tree Bisection and Reconnection (TBR) - the most general edit operation of the three, it involves breaking an edge and reconnecting edges of the resultant subtrees in an arbitrary way. There are $O(n^3)$ possible TBR moves from any topology. The application of TBR to maximum likelihood search has been hindered by the high computational cost of choosing the best TBR move.

FastTree and RAxML use a combination of NNI and SPR moves to optimize the topology. Exhaustive search for the best SPR move might be computationally expensive. Because of this, both programs restrict their search space to SPR moves that can be obtained as a sequence of NNI moves such that none of the NNI moves decrease the likelihood substantially.

2.2.2 Distance Methods

Distance methods are a faster but generally less accurate alternative to maximum likelihood inference. The input to a distance method is usually a distance matrix D specifying all pairwise distances between the taxa. The methods then try to find the tree that "fits" the distances between taxa. A natural criterion for fitting trees to distance matrices is the least-squares criterion (e.g. [30]). Optimizing the least-squares objective is known

to be NP-hard [45], so we have to resort to heuristic algorithms. Most methods start by considering each taxon as its own tree, and then iteratively merge taxa into larger subtrees until only one tree remains. Distance methods are often used to produce the starting tree for a maximum likelihood local search procedure. This is done in both RAxML and FastTree.

UPGMA

One of the earliest algorithms used for reconstructing phylogenies is UPGMA, which stands for Unweighted Pair Group Method with Arithmetic Means. It was popularized by Sokal and Sneath [149] in 1963. UPGMA assumes that the true evolutionary history is *ultrametric*, meaning that the evolutionary distance from the root of the tree to any of the leaves is constant. This is also known as the *molecular clock* assumption.

The idea behind UPGMA is to repeatedly find the closest pair of taxa and merge them as siblings in a new tree. The algorithm then calculates the distance from the root of the new tree to all the other taxa. The process is repeated until all the taxa are joined.

The following pseudocode follows the presentation in [68].

Algorithm 1 UPGMA(D)

Start with a forest with each node in its own tree.
while the forest has more than one tree **do**
  Find i and j that minimize D_{ij}
  Create a new group (ij). Set $n_{(ij)} = n_i + n_j$
  Create edges from i and j to a new node (ij). Set both edge lengths to $D_{ij}/2$.
  Compute the distance from (ij) to all the other nodes (except for i and j) from the formula:

$$D_{(ij),k} = \frac{n_i}{n_i + n_j} D_{ik} + \frac{n_j}{n_i + n_j} D_{jk}$$

  Remove from D the rows and columns corresponding to i and j. Add a row and a column corresponding to (ij).
end while

The running time of this procedure is dominated by finding the closest pair of nodes to join. In the naive implementation, this takes $\Theta(n^2)$ time per loop iteration, which results in a total runtime of $\Theta(n^3)$. This can be easily improved to $\Theta(n^2 \log n)$ by using a heap to quickly find closest pairs [83]. A more careful use of data structures makes it possible to

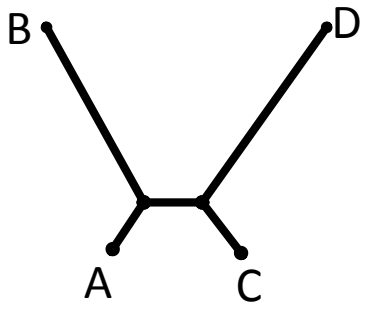

Figure 2.1: Running UPGMA on a distance matrix corresponding to the above tree will yield an incorrect tree with A and C as siblings.

shrink the runtime to $\Theta(n^2)$. This was done by Gronau and Moran [83], and independently by Eppstein [60].

The molecular clock assumption severely restricts the application of UPGMA to real phylogenetic data sets, as rates of evolution vary across lineages. For example, it is estimated that the evolutionary rate in rodents is roughly two times higher than in primates [167]. For data sets consisting of closely-related organisms, evolutionary rates are more likely to be very similar across lineages. For this reason, UPGMA is mainly used for the analysis of within-species evolution, for example in viruses [161].

We note that UPGMA is not strictly a phylogenetic algorithm, but rather the application of a general algorithm for hierarchical clustering to the phylogenetic domain. In the clustering literature, it is commonly known as *average-linkage clustering* [128].

Neighbour Joining (NJ)

Neighbour Joining is an adaptation of the hierarchical clustering paradigm of UPGMA to non-ultrametric trees. When the data do not satisfy the molecular clock assumption, choosing to merge the closest pair of nodes may result in an incorrect topology. For example, for a distance matrix corresponding to the tree in Figure 2.1, UPGMA would produce an incorrect tree where taxa A and C are siblings. In order to adapt UPGMA to non-clock trees, we need a different criterion for choosing the pair of nodes to agglomerate. We also need a new way of estimating the distances between nodes.

The NJ algorithm was first proposed by Saitou and Nei in 1987 [142]. A minor error in

the original algorithm was pointed out a year later by Studier and Keppler [159]. Neighbour Joining remains one of the most popular bioinformatics algorithms of all time, with over 30,000 citations.

The merging criterion in Neighbour Joining is somewhat more involved than that of UPGMA. Define

$$u_i = \sum_{j \neq i} D_{ij}/(n-2)$$

Note that the $n-2$ term is not equal to the number of elements in the sum. Neighbour Joining joins the pair of nodes that minimizes

$$C_{ij} = D_{ij} - u_i - u_j$$

While this criterion might seem somewhat counterintuitive, it can be shown that if the distances in the matrix correspond exactly to distances on the tree, NJ is guaranteed to reconstruct the correct tree. Perhaps the most elegant proof of that fact can be found in a paper by Bryant [26].

We also need a way of estimating branch lengths and distances to newly-created nodes. The lengths of the new branches incident to i and j are estimated as

$$v_i = \frac{1}{2}D_{ij} + \frac{1}{2}(u_i - u_j) \tag{2.1}$$

$$v_j = \frac{1}{2}D_{ij} + \frac{1}{2}(u_j - u_i) \tag{2.2}$$

The distance from the new node to any node k is computed as

$$D_{(ij),k} = (D_{ik} + D_{jk} - D_{ij})/2$$

The above equations are just simple least-squares estimates of branch lengths and distances.

We can now state the algorithm as Algorithm 2.

The running time of Neighbour Joining is $O(n^3)$. Unlike with UPGMA, there is no straightforward way to improve this runtime. After each join, the u_i terms have to be recomputed and every C_{ij} term has to be re-evaluated. It is a long-standing open question whether this can be done in $o(n^3)$ time. In practice, a considerable speedup can be achieved by clever use of data structures, though the worst-case running time is still $O(n^3)$. Several fast implementations exist, such as NINJA [166], Rapid Neighbour Joining [146], the method of Zaslavsky and Tatusova [173], or the method of Mailund *et al.* [114].

Algorithm 2 NeighbourJoining(D)

Start with a forest with each node in its own tree.
while the forest has more than two trees **do**
Find i and j that minimize $D_{ij} - u_i - u_j$
Create edges from i and j to a new node (ij). Set edge lengths according to Equations 2.1 and 2.2.
Compute the distance from (ij) to all the other nodes (except for i and j) from the formula:

$$D_{(ij),k} = (D_{ik} + D_{jk} - D_{ij})/2$$

Remove from D the rows and columns corresponding to i and j. Add a row and a column corresponding to (ij).
end while
Connect the two remaining nodes a and b with an edge of length D_{ab}

A distance matrix is *additive* if all distances correspond exactly to distances in some phylogenetic tree. Most distance methods are guaranteed to recover the correct tree from an additive distance matrix. This includes all the other algorithms described in this chapter, except UPGMA. The fastest algorithm for additive distance matrices is an $O(n \log n)$ triplet algorithm by Kannan, Lawler and Warnow [95]. Our $O(n \log n)$ algorithm from Chapter 3 is also guaranteed to reconstruct the correct tree from additive distance matrices. In real data sets, distance estimates are usually not perfectly additive due to statistical noise. Atteson [7] showed that NJ reconstructs the correct tree as long as every distance estimate is within $f/2$ of its true value, where f is the length of the shortest edge in the phylogeny. This is actually a fairly stringent requirement, as estimates for distant pairs of taxa tend to be very inaccurate; see discussion in Section 2.4. Mihaescu *et al.* [121] provided a slightly weaker criterion for correctness. However, their analysis is somewhat hard to interpret.

Improvements to Neighbour Joining

The cubic runtime of Neighbour Joining is a major obstacle in applying it to larger data sets. Starting in the mid-2000's, a number of methods were developed with the goal of building trees similar to those of Neighbour Joining, in less runtime.

Fast Neighbour Joining [59] attempts to approximate the NJ objective in $O(n^2)$ time, by limiting the set of candidate pairs considered for merging at each step. The algorithm maintains the so-called *visible set* of candidate pairs. The visible set is initialized as the

set of pairs that are optimal with in their respective rows of the distance matrix. At each merge, the visible set is updated by removing the pairs involving merged nodes and replacing the optimal pair with a pair involving a newly created node, if its score is higher. Note that this does not, in general, produce the same result as NJ, as a pair might cease to be optimal as a result of a merge elsewhere in the tree. In experiments, the authors show that FNJ produces similar trees to NJ.

A similar search strategy is employed in Relaxed Neighbour Joining [63]. Rather than finding a join that minimizes the NJ criterion, RNJ tries to find an entry in the matrix that is optimal for both its row and column. The algorithm's claimed runtime is $O(n^2 \log n)$ on balanced trees (assuming additivity of the distance matrix), though the actual runtime in this case appears to be $O(n^2)$. On unbalanced trees, the runtime is $O(n^3)$ in the worst case.

A class of theoretical NJ-like algorithms was also devised by Gronau and Moran [82]. For each pair of taxa, their algorithm first estimates the distance from the root of the tree to their lowest common ancestor in the tree. The algorithms have similar properties to Neighbour Joining and UPGMA and work in $O(n^2)$ time.

In another line of work, researchers have tried to improve the accuracy of NJ by exploiting the variance and covariance structure of the distance matrix. Gascuel [75] designed a new merging criterion for NJ that assumes the variance of distance estimates to be linear with the true evolutionary distance. This leads to a new merging criterion where the distances are weighted to minimize the probability of an incorrect merge. The BIONJ criterion also accounts for covariances between distance estimates. Weighted Neighbour Joining [24] takes a similar approach, using an approximate variance estimator derived from a Markov model of evolution. The criterion used in WNJ is a sum of two terms that are meant to guard against deviations from additivity and ensure that all edges have positive length. While the latter term would ordinarily lead to a runtime of $O(n^4)$, the authors use heuristics to ensure cubic runtime.

FastTree

All the distance methods we have mentioned so far start by estimating tree distances between all pairs of sequences. This initial step is a natural bottleneck in terms of both runtime and memory usage. When the number of sequences exceeds 30,000, storing all pairwise distances in main memory is no longer possible on a standard desktop with 4GB RAM. Even if more memory is available, computing the full distance matrix can take hours or days.

In 2009, Price *et al.* introduced FastTree, an NJ-like algorithm that reconstructs phylogenies from incomplete distance matrices [134]. FastTree consists of two stages: in the "Neighbour Joining" phase of FastTree, an NJ-like algorithm reconstructs the initial topology. In the local search phase, that topology is subsequently refined by a series of edit operations that seek to optimize the score. In the current version of FastTree [135], the local search phase consists of a series of operations optimizing the minimum evolution criterion, followed by a series of operations that seek to maximize the likelihood. In this section, we will describe the "Neighbour Joining" phase of FastTree, as this is the main algorithmic improvement behind the program. The local search phase involves algorithmic techniques that are similar to those used in other local search procedures for phylogenetic reconstruction, such as RAxML or PhyML.

The main idea behind FastTree is to estimate only those entries in the distance matrix that are most likely to be informative. It is generally known that distance estimates between distant sequences tend to be unreliable (see Section 2.4 for a more elaborate discussion). For each taxon x, FastTree heuristically finds the $\sqrt{n}$ taxa that are closest to it. This is done by computing all the distances from the first taxon x_1, then retaining a set $S_{x_1,2\sqrt{n}}$ of $2\sqrt{n}$ closest taxa to x_1, with 2 being a safety factor. For a taxon x in $S_{x_1,\sqrt{n}}$, the algorithm searches for the closest $\sqrt{n}$ taxa to x among $S_{x_1,2\sqrt{n}}$. The process is repeated for the remaining set of taxa (excluding the taxa in $S_{x_1,\sqrt{n}}$) until each taxon has at least $\sqrt{n}$ distances estimated. Thus, after $O(n\sqrt{n})$ steps, we obtain a partial distance matrix with at least $\sqrt{n}$ entries in each row.

The second idea is to use profiles to approximate the join criterion used in Neighbour Joining. Instead of computing u_i values from Equation 2.2.2 by averaging corrected distances, FastTree computes the average Hamming distance between two groups of sequences by comparing two profiles. For each position in the profile, we can compute the expected disagreement between sequences sampled from profiles a and b using the formula

$$E[a_i \neq b_i] = 1 - \sum_{x \in \{A,C,G,T\}} \Pr[a_i = x] \Pr[b_i = x]$$

The average Hamming distance between sequences in group a and b is $\sum_i E[a_i \neq b_i]$. FastTree then applies the correction formula to this distance. For closely related sequences evolving under the Jukes-Cantor model, this formula is a good approximation of the average corrected distance computed by Neighbour Joining. By using profiles instead of averages, FastTree computes approximate u_i values in constant rather than linear time.

For most data sets, the NJ phase of FastTree produces trees almost identical to Neighbour Joining, as shown in experiments in Chapter 3. Together with the local search phase, trees produced by FastTree have similar accuracy to those produced by RAxML [108].

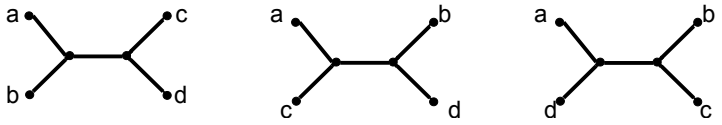

Figure 2.2: Three quartet topologies for taxa a, b, c, d.

2.3 Quartet Methods

One possible strategy for inferring large phylogenies is to estimate a large number of "small" trees, then amalgamate them into a complete tree for the whole data set. Quartet methods take the same approach by building the tree from a large number of *quartets* - trees on four taxa.

For four taxa, there are only three possible topologies (see Figure 2.2). For taxa a, b, c, d, we denote by $ab|cd$ the topology where a and b are separated from c and d by the middle edge.

The choice of quartets as the building block for larger trees is motivated by simplicity. Below, we will review the theoretical and practical algorithms for constructing trees from quartets.

2.3.1 Inferring a quartet

The simplest algorithm for inferring a four-taxa phylogeny is known as the *four-point method*. If distances d are additive, then for a quartet topology $ab|cd$ we should have

$$d_{ab} + d_{cd} < d_{ac} + d_{bd}, d_{ad} + d_{bc}$$

In other words, the sum of the distances corresponding to the two pairs of siblings in the correct topology is smaller than the other two sums. The difference is equal to twice the length of the middle edge. It follows that the topology of a quartet can be estimated by picking the minimum of the three sums:

$$d_{ab} + d_{cd}, d_{ac} + d_{bd}, d_{ad} + d_{bc}$$

We use this simple and fast method in our algorithms presented in the following chapters.

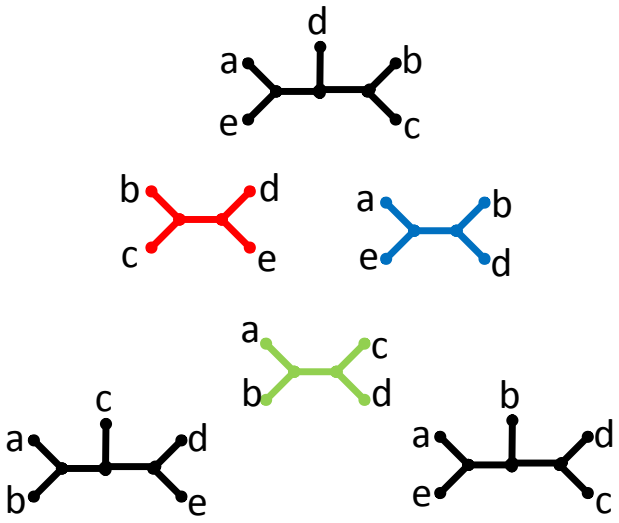

Figure 2.3: An example of three quartet topologies on five taxa that are not consistent with any five-taxa topology. For each pair of quartets from the set $\{ab|cd, bc|de, ae|bd\}$, there exists a single topology on five taxa consistent with that pair of quartets (shown in black).

In most programs, quartets are inferred by maximum likelihood. This is usually done numerically, by repeatedly optimizing branch lengths for each of the three topologies. The program then chooses the the topology with the highest likelihood. This strategy results in higher accuracy compared to the four-point method, at the cost of increased runtime. Chor *et al.* [36] derived an analytical formula for maximizing the likelihood of ultrametric quartets.

2.3.2 Maximum Quartet Compatibility

It is not always possible to find a tree that agrees with every quartet in a given set of quartets. For example, there is no tree that is consistent with the quartets in Figure 2.3. Such *inconsistent* sets of quartets will arise frequently in practice as a result of errors in individual quartet inferences. Determining if a set of quartets is consistent is an NP-complete problem [153].

A natural approach when dealing with inconsistent sets of quartets is to find a tree that is consistent with the maximum number of quartets. This is known as the Maximum

Quartet Compatibility (MQC) problem. Finding an MQC tree is NP-hard, as proven by Berry *et al.* [14]. Berry and Gascuel [13] proposed an exponential-time algorithm for the problem. A series of more scalable exponential-time algorithms was proposed by Lin *et al.* [170, 169]. Jiang *et al.* [93] derived an polynomial-time approximation scheme for the problem.

Quartet Puzzling (QP) [158] is a widely used quartet method inspired by earlier MQC approaches. After inferring all $\binom{n}{4}$ quartets, it starts from a star tree of three taxa, and adds taxa sequentially until an n-taxon phylogeny is produced. At each insertion, the new taxon is attached to the existing tree so as to maximize the number of quartet topologies consistent with the location of the new taxon. The iterative insertion algorithm is repeated a large number of times for different orderings of taxa, and the algorithm outputs the consensus tree of all the trees produced during the puzzling step. While this approach does not offer any theoretical guarantees on accuracy, it gives reasonable results in practice and has found a large number of users, despite its quartic runtime.

A major flaw of the MQC approach is that it treats all quartet trees as equally likely to be correct. This is not true, as we will see in Chapter 3. For example, quartets consisting of closely related taxa are more likely to be inferred correctly since short distance estimates have lower variance. To account for this, several methods weight quartets by their likelihood score or a posteriori probability. If the three possible quartet topologies for a set of taxa have likelihoods ℓ_1, ℓ_2, ℓ_3, we can approximate the posterior probability of the first topology by writing

$$p_1 = \frac{\ell_1}{\ell_1 + \ell_2 + \ell_3}$$

This weighting scheme was first used in the second version of Quartet Puzzling [157], which led to considerable improvement in the quality of inferred trees. Ranwez and Gascuel [137] proposed an improved algorithm using a similar weighing scheme. In their experiments, they showed that their Weight Optimization algorithm outperformed QP, but the accuracy of both algorithms remained markedly lower than that of maximum likelihood.

St. John *et al.* [94] performed a large-scale experimental study comparing the accuracy of unweighted quartet methods to Neighbour Joining. They found that in most cases, unweighted quartet methods were much less accurate than Neighbour Joining.

2.4 Mathematical guarantees on the accuracy of phylogenetic reconstruction

Evaluating the accuracy of a phylogenetic method is challenging for a number of reasons. For most real data sets, the true evolutionary history is not known with certainty, so there is no easy way to independently verify the topology proposed by the method we are trying to evaluate. For this reason, empirical studies of algorithm accuracy usually take one of two approaches. The first approach is to compare the tree produced by the algorithm to the consensus view of the evolutionary history of the set of taxa. The second approach is to simulate the evolutionary process on a known tree topology, and then compare the output of the algorithm to the tree used in the simulation. Both of these experimental design approaches have serious issues.

The consensus among biologists about the evolutionary tree of a given group of species might be erroneous. This is especially true for large sets of species, as they will contain less-studied species whose precise evolutionary origin is more uncertain. Even if we know the true evolutionary history for a set of species, the tree for a specific gene might differ from the species tree due to diverse phenomena such as incomplete lineage sorting, horizontal gene transfer or ancient gene duplications [113]. Despite these problems, researchers have tried using reference trees, such as the NCBI taxonomy [65] to evaluate the relative accuracy of various alignment and phylogeny algorithms. A recent, relatively principled approach to such evaluations can be found in a paper by Dessimoz and Gil [50].

Compared to the consensus approach, simulating an alignment on a known phylogenetic tree offers the advantage of being able to directly compare the result of the algorithm with the ground truth. On the other hand, assessing the performance of an algorithm on simulated alignments might give an overly optimistic picture of its accuracy. It may be hard to choose a realistic model of sequence evolution. In fact, it has been observed that standard substitution models used in many studies poorly fit observed data [131], indicating that these models do not accurately reflect biological reality. Even if the model of sequence evolution is realistic, it may still be challenging to design a representative set of simulations so as to explore the space of likely evolutionary scenarios.

These problems have motivated the theoretical study of algorithm accuracy. Such analyses have the advantage that they hold for large classes of phylogenetic trees, thereby covering many scenarios that may be omitted in empirical comparisons. They can also give an idea of the asymptotic behaviour of algorithms, illuminating what would happen for very large trees. They can also be used to investigate the limits of accuracy in phylogenetic reconstruction in general, without restricting to a particular algorithm, or a particular im-

plementation. While many of the algorithms analyzed in this section are purely theoretical and have not been widely used in practice, the results of such analyses can inform the design of new practical algorithms.

The results presented in this chapter assume that the alignment is generated by a certain kind of Markov process evolving on the evolutionary tree. Given this assumption, we are interested in two questions:

1. How accurate can a phylogenetic algorithm be given an alignment with m columns? More specifically, how many alignment columns are needed to reconstruct the tree correctly, with a specified probability?

2. Given enough columns to guarantee correct reconstruction, how fast can it be done?

Ideally, we would like to find an algorithm that is guaranteed to correctly reconstruct phylogenies from short sequences, in fast runtimes.

The sequence length bounds obtained from the theoretical analyses mentioned in this section are usually not very tight. While the bounds are tight up to a constant as the number of taxa tends to infinity, the constants are usually enormous. Nevertheless, this kind of analysis is still a valuable tool when assessing which algorithm is likely to be more accurate for large numbers of taxa, and is somewhat analogous to asymptotic runtime analysis in complexity theory. An algorithm with an asymptotically lower sequence length requirement is likely to be more accurate for large trees much in the way an asymptotically faster algorithm is likely to be faster for large data sets; as data sets grow in size, slowly-growing functions will result in best bounds, even if accompanied by large leading constants.

In our investigations, we will assume that the sequences are generated from a Cavender-Farris-Neyman model of evolution [33]. This is motivated by the simplicity of the underlying mathematics, but all of the results presented in this section should translate to more complicated models, though the corresponding proofs might be more involved. We will also assume that the lengths of all edges are bounded from above and below by constants f and g, respectively.

Assuming bounded edge lengths is meant to exclude types of phylogenies that are intrinsically challenging to reconstruct. If an edge has length very close to 0, there is a considerable risk that no mutations will happen on that edge, unless sequences are extremely long. Consequently, the alignment will not contain any information that would enable us to reconstruct the edge. On the other hand, if an edge is very long, it is likely to give rise to infinite distance estimates, which carry no phylogenetic signal. The precise

values of f and g depend on the number of columns, as we will see below. As the number of columns tends to infinity, even phylogenies with very short or very long edges will be reconstructible.

The choice of CFN as the model of evolution is motivated by its mathematical simplicity. Most of the results in this section generalize to more complicated models, but the derivations might be substantially more involved.

2.4.1 Concentration inequalities for distance estimates

As we have seen in Section 2.1, distance estimators are *statistically consistent*, meaning that they will estimate distances exactly for sequences of infinite length. For finite-length sequences, we are interested in estimating the distribution of error in distance estimates. In particular, we are interested in bounding the probability that a distance estimate is more than some ϵ away from its true value.

We will need the well-known Hoeffding inequality (see e.g. [55, 127]).

Theorem 1 (Hoeffding bound)**.** *Let $X_1, X_2, \ldots, X_m$ be independent random variables taking values in $\{0, 1\}$ and let $X = \sum_{i=1}^{m} X_i$. Then*

$$\Pr[X - E[X] > t] < \exp(-2t^2/m)$$

$$\Pr[X - E[X] < -t] < \exp(-2t^2/m)$$

We shall use this result to bound the probability that a distance estimate $\hat{d}$ deviates from the true distance d by more than some value δ. For the CFN model, the distance estimator is

$$\hat{d} = -\frac{1}{2}\log(1 - 2\hat{p})$$

where $\hat{p}$ is the fraction of sites that have changed. We first bound the probability that

$$\hat{d} - d > \delta$$

which is equivalent to

$$-\frac{1}{2}\log(1 - 2\hat{p}) + \frac{1}{2}\log(1 - 2p) = -\frac{1}{2}\log\frac{1 - 2\hat{p}}{1 - 2p} > \delta$$

Through a series of transformations, we get

$$\frac{1 - 2\hat{p}}{1 - 2p} < e^{-2\delta}$$

$$1 - 2\hat{p} < e^{-2\delta}(1 - 2p)$$
$$\hat{p} > \frac{1 - e^{-2\delta}(1 - 2p)}{2}$$
$$\hat{p} > \frac{1 - e^{-2\delta} + 2pe^{-2\delta})}{2}$$

Hence

$$\hat{p} - p > \frac{(1 - e^{-2\delta})(1 - 2p)}{2}$$

Hence by the Hoeffding bound, we have:

$$\Pr[\hat{d} - d > \delta] = \Pr[\hat{p} - p > \frac{(1 - e^{-2\delta})(1 - 2p)}{2}] < \exp(-(1 - e^{-2\delta})^2(1 - 2p)^2 m/2)$$

Equivalently, we can write the above formula in terms of distance d:

$$\Pr[\hat{d} - d > \delta] < \exp(-(1 - e^{-2\delta})^2 e^{-4d} m/2) \tag{2.3}$$

By analogous reasoning, we can bound the probability of *underestimating* d

$$\Pr[\hat{d} - d < -\delta] < \exp(-(1 - e^{2\delta})^2 e^{-4d} m/2) \tag{2.4}$$

Formula 2.3 has some important implications. Most importantly, notice that as the distance d increases, the bound gets more loose (the e^{-4d} term gets smaller). This means that estimates between distant taxa are likely to be less reliable. Naturally, longer alignments will give better distance estimates.

Equations 2.3 and 2.4 also gives an estimate of the number of alignment columns needed to estimate distances up to d with accuracy δ, with certain probability.

2.4.2 The simplest case: inferring a quartet

With this concentration inequality for distance estimates, we can start estimating the probability of reconstructing phylogenies correctly. The simplest case is reconstructing a quartet tree by the four-point method. The probability of correctly reconstructing the quartet tree will depend on the lengths of the edges and the number of columns in the alignment.

Suppose we have a quartet $ab|cd$ where all the edge lengths are between f and g. The four-point method will pick the correct topology if and only if

$$d_{ab} + d_{cd} < d_{ac} + d_{bd}$$

and

$$d_{ab} + d_{cd} < d_{ad} + d_{bc}$$

The length of the middle edge is at least f. Therefore, the difference between $D_{ab} + D_{cd}$ and any of the other two sums will be at least $2f$. It is easy to see that the four point method will give the correct topology if the following conditions hold:

$$\begin{aligned}
\hat{d}_{ab} &< d_{ab} + f/2 && (2.5)\\
\hat{d}_{cd} &< d_{cd} + f/2 && (2.6)\\
\hat{d}_{ac} &> d_{ac} - f/2 && (2.7)\\
\hat{d}_{ad} &> d_{ad} - f/2 && (2.8)\\
\hat{d}_{bc} &> d_{bc} - f/2 && (2.9)\\
\hat{d}_{bd} &> d_{bd} - f/2 && (2.10)\\
& && (2.11)
\end{aligned}$$

From equations Equation 2.3 and 2.4, we can bound the probability of each of these conditions being not satisfied by putting $\delta = f/2$:

$$\begin{aligned}
\Pr[\hat{d}_{ab} &> d_{ab} + f/2] < \exp(-(1-e^{-f})^2 e^{-4d_{ab}} m/2) && (2.12)\\
\Pr[\hat{d}_{cd} &> d_{cd} + f/2] < \exp(-(1-e^{-f})^2 e^{-4d_{cd}} m/2) && (2.13)\\
\Pr[\hat{d}_{ac} &< d_{ac} + f/2] < \exp(-(1-e^{f})^2 e^{-4d_{ac}} m/2) && (2.14)\\
\Pr[\hat{d}_{ad} &< d_{ad} + f/2] < \exp(-(1-e^{f})^2 e^{-4d_{ad}} m/2) && (2.15)\\
\Pr[\hat{d}_{bc} &< d_{bc} + f/2] < \exp(-(1-e^{f})^2 e^{-4d_{bc}} m/2) && (2.16)\\
\Pr[\hat{d}_{bd} &< d_{bd} + f/2] < \exp(-(1-e^{f})^2 e^{-4d_{bd}} m/2) && (2.17)\\
& && (2.18)
\end{aligned}$$

D_{ab} and D_{cd} are bounded by $2g$, whereas all the other distances are bounded by $2g + f$. From the union bound, we get the following worst-case bound on the probability of incorrect reconstruction:

$$\begin{aligned}
\Pr[\text{incorrect quartet}] &< 4\exp(-(1-e^{f})^2 e^{-4(2g+f)} m/2) + 2\exp(-(1-e^{-f})^2 e^{-4(2g)} m/2)\\
&< 6\exp(-(1-e^{f})^2 e^{-4(2g+f)} m/2)
\end{aligned}$$

$$\Pr[\text{incorrect quartet}] < 6\exp(-(1-e^{f})^2 e^{-4(2g+f)} m/2) \quad (2.19)$$

2.4.3 A naive bound

A naive approach to estimating the number of columns needed to reconstruct the phylogeny is to estimate the number of columns needed to ensure that all pairwise distances are within $f/2$ of their true values. By Atteson's result [7], NJ will then reconstruct the correct tree. Unfortunately, this approach leads to a very pessimistic estimate, as we shall see below.

From Equations 2.3 and 2.4, we have

$$\begin{aligned}\Pr[|\hat{D_{ij}} - D_{ij}| > f/2] &< \exp(-(1-e^{-f})^2 e^{-4D_{ij}} m/2) + \exp(-(1-e^{f})^2 e^{-4D_{ij}} m/2) \\ &< 2\exp(-(1-e^{f})^2 e^{-4D_{ij}} m/2), \text{ for each } i,j\end{aligned}$$

By the union bound, we get

$$\Pr[\text{phylogeny wrong}] < n(n-1)\exp(-(1-e^{f})^2 e^{-4diam(T)} m/2) < n^2 \exp(-(1-e^{f})^2 e^{-4diam(T)} m/2)$$

where $diam(T)$ is the diameter of the tree, measured in total path length.

Now we can derive the number of alignment columns needed so that NJ reconstructs the correct phylogeny with probability at least $1-\epsilon$:

$$\begin{aligned}\epsilon &= n^2 \exp(-(1-e^{f})^2 e^{-4diam(T)} m/2) \\ \epsilon/n^2 &= \exp(-(1-e^{f})^2 e^{-4diam(T)} m/2) \\ \ln(\epsilon/n^2) &= -(1-e^{f})^2 e^{-4diam(T)} m/2 \\ 2\ln(n^2/\epsilon) e^{4diam(T)}/(1-e^{f})^2 &= m \qquad (2.20)\end{aligned}$$

For "caterpillar" trees where each internal node is adjacent to a leaf, the diameter can be as large as $(n-1)g$, leading to an exponential number of alignment columns:

$$m = 2\ln(n^2/\epsilon) e^{4(n-1)g}/(1-e^{f})^2 < C e^{4gn} \ln n$$

for some constant C depending on f. This would imply that accurate phylogenetic reconstruction is possible only for very small numbers of taxa, due to the finite length of genetic sequences. On the other hand, for fully balanced trees, the diameter is bounded by $2\log_2 n$, and Equation 2.20 becomes

$$m = 2\ln(n^2/\epsilon) e^{8g\log_2 n}/(1-e^{f})^2 < C n^{8g/\ln 2} \ln n$$

This is a much more optimistic scenario, as it implies that only polynomial-length sequences are required.

Most tree topologies lie between those two extreme cases. How many columns do we need to reconstruct *most* phylogenies? We can prove the following result [62].

Theorem 2. *The average diameter of a tree topology sampled uniformly at random from the set of all n-leaf tree topologies is* $\Theta(\sqrt{n})$ *branches.*

This is potentially very bad news as it suggests an exponential number of columns required for accurate reconstruction for most trees, if we must closely approximate *all* distances of the tree. In the next two sections, we will see that one can have algorithms that are guaranteed to reconstruct the phylogeny from much shorter alignments. The key idea behind those algorithms is to rely on distance estimates for pairs of taxa that are close to each other. Fewer columns are required to ensure accuracy of these estimates, which leads to better bounds.

Given the widespread use of Neighbour Joining in practice, one might wonder whether these bounds are tight. While it is hard to give a theoretical analysis for all realistic scenarios, Lacey *et al.* [103] recently showed that NJ does in fact require exponential-length sequences for some tree topologies, under similar assumptions to ours.

2.4.4 Short quartet methods

As we have seen in Section 2.4.2, the reliability of quartet inference depends on the lengths of the external branches. The longer the external branches, the more columns are needed to guarantee correct reconstruction with high probability. The idea behind short quartet methods is to reconstruct the phylogeny solely based on quartets whose external edges are not too long. This idea first appeared in a paper by Erdős *et al.* [61], and was subsequently developed by several authors in the late 1990's and early 2000's.

The main difficulty in designing a short quartet algorithm is finding a set of quartets that a) consists of quartets that are each correct with high probability and b) uniquely determines the phylogeny. Erdős *et al.* solve this problem by defining the set of *representative quartet splits*. For each edge e in T, a representative quartet contains a taxon from each of the four subtrees adjacent to e. In each subtree, the closest available taxon is chosen. Erdős *et al.* show that the set of representative quartets chosen this way uniquely determines the phylogeny. They then devise an algorithm that finds a consistent superset of this set of quartets, without knowledge of the tree. The algorithm runs in $O(n^5 \log n)$ time.

To analyze the length of sequences required for accurate reconstruction, we will need a few definitions. An *edge-deletion-induced subtree* (*edi-subtree*) is any subtree t created by deleting an edge e in T. The *edge depth* of t is the number of edges between e and its

nearest leaf in t. The *depth* of the whole tree T is the maximum edge depth of any of its edi-subtrees.

It follows that the length of any external edge in the quartets used by Erdős *et al.* is bounded by $(depth(T)+1)g$. Equation 2.19 gives us

$$\Pr[\text{quartet wrong}] < 6\exp(-(1-e^f)^2 e^{-4(2depth(T)g+2g+f)}m/2)$$

This holds for all $n-3$ representative quartets. Hence

$$\Pr[\text{phylogeny wrong}] < 6n\exp(-(1-e^f)^2 e^{-4(2depth(T)g+2g+f)}m/2)$$

Suppose we want the error probability to be below some constant ϵ. This yields

$$\epsilon = 6n\exp(-(1-e^f)^2 e^{-4(2depth(T)g+2g+f)}m/2)$$

$$\ln(\epsilon/6n) = -(1-e^f)^2 e^{-4(2depth(T)g+2g+f)}m/2$$

$$-2\ln(\epsilon/6n)e^{4(2depth(T)g+2g+f)}/(1-e^f)^2 = m$$

The above is a simplified analysis. The full analysis by Erdős *et al.* is somewhat more complicated as the algorithm has to infer a *superset* of the representative set in order to identify the topology. The full analysis gives the same sequence length requirement up to a constant factor.

The depth of an n-taxon tree can range from 1 for caterpillar trees to $\log n$ for fully balanced trees. Erdős *et al.* showed that for most trees, the depth is $O(\log\log n)$. This also holds with high probability when the trees are sampled from the Yule-Harding process. Thus, for most trees, the number of columns required for correct reconstruction with probability $1-o(1)$ is $O(\text{poly}\log n)$. For fully balanced trees, the short quartet methods requires sequences of polynomial length, just like Neighbour Joining. On the other hand, for caterpillar trees, $O(\log n)$ columns are sufficient, as all quartets used have constant external edge length.

In a subsequent paper [62], Erdős *et al.* give a faster algorithm with the same guarantees for accurate reconstruction whose running time depends on the depth of the tree. The running time of the algorithm is $O(n^2\log n)$ for most trees, and $O(n^4\log n)$ in the worst case. Csűrös [40] gave an $O(n^2)$ algorithm with similar reconstruction guarantees. King *et al.* presented an algorithm with a runtime of $O(n^2\log\log\log n/\log\log n)$ for most trees, with similar reconstruction guarantees. Prior to the work presented in this thesis, this was the only sub-quadratic algorithm with theoretical guarantees on reconstruction. All of the

above algorithms rely on the same concepts introduced by the seminal paper by Erdős *et al.* [61].

We stress once again that most short quartet algorithms are not meant to be practically useful, but merely to demonstrate the possibilities of reconstructing trees from short sequences, in reasonable runtimes. To our knowledge, the only exceptions are Short Quartet Puzzling [148], and the aforementioned algorithm by Csűrös.

More recent papers have extended this framework to cases where parts of the tree cannot be reconstructed due to insufficient phylogenetic signal. Gronau and Moran [84] presented an algorithm which reconstructs a tree with polytomies (internal vertices of degree above 3) to indicate parts of the tree that could not be resolved due to very short edges. Daskalakis, Mossel and Roch [42] reconstruct a forest to indicate very long edges that could not be resolved.

2.4.5 The impossibility result

It is a natural question to ask whether an algorithm with similar reconstruction guarantees could be made to run much faster than $O(n^2)$. King, Zhang and Zhou [101] proved a combinatorial result that suggests an $O(n^2/\log\log n)$ lower bound for most trees. In their setting, the algorithm is allowed to ask distance queries that are correct within ϵ, as long as the true distance is below some parameter τ. If the true distance is above τ, the oracle can return an arbitrary value that is also above τ. This oracle model mimics the behaviour of distance estimators from DNA sequences, which tend to be inaccurate for distant sequences. Under these assumptions, the authors show that any algorithm requires $\Omega(n^2/\tau)$ queries to reconstruct the tree. Since τ has to be at least equal to the depth of the tree, this suggests an $O(n^2/\log\log n)$ lower bound by the result of Erdős *et al.*, for poly-logarithmic alignment lengths, for most trees.

Note that this result does not imply that it is *impossible* to reconstruct phylogenies in faster runtimes. It merely implies that this is the case when distance queries are the only source of information about the phylogeny. In Chapter 4, we will use information in sequences themselves to circumvent this lower bound.

2.4.6 Ancestral state reconstruction

The short quartet methods we have seen so far reconstruct the tree based solely on distances between leaf sequences. This makes it notably challenging to reconstruct parts of the

tree that lie far away from any leaf. In order to reconstruct deep regions of the tree, any distance-based algorithm must have access to accurate estimates of distance between distantly-related taxa. This requires very long alignments.

A radically different approach is to reconstruct ancestral sequences at internal nodes of the tree and use them as "virtual" taxa. If we can reconstruct ancestral sequences with high accuracy, distance estimates between reconstructed sequences can give us valuable phylogenetic signal. The advantage of using reconstructed sequences over leaf sequences is that we can always find reconstructed sequences that are a constant number of edges from the part of the tree that we are trying to reconstruct. If we can guarantee that the error rate in reconstructed sequences is low, we can thus obviate the need to compare distant sequences.

The idea of using reconstructed sequences in phylogenetic inference was pioneered by Mossel in 2004 [125], and further developed in algorithms by Daskalakis *et al.* [43] and Mihaescu *et al.* [120]. The approach rests on a well-known fact about Markov processes on trees: for known tree topologies with mutation probabilities below a certain threshold, we can correctly reconstruct the ancestral state based on the leaf characters with probability bounded away from $\frac{1}{2}$, for arbitrarily deep internal nodes in the tree. The following formulation mirrors that of Mossel [125]:

Theorem 3. *Let T be a binary tree with root ρ and edges e all satisfying $p(e) < \frac{1}{2} - \sqrt{\frac{1}{8}}$. Let δT denote the leaves of T. Let σ be a Cavender-Farris character on T. There exists an algorithm $A(.)$ that infers the correct ancestral state with probability satisfying*

$$\Pr[A(\sigma_{\delta T}) = \sigma_\rho | \sigma_\rho = 1] = \Pr[A(\sigma_{\delta T}) = \sigma_\rho | \sigma_\rho = -1] > \frac{1}{2} + \beta.$$

for some constant β independent of T.

This was first proved by Bleher, Ruiz and Zagrebnov for balanced trees [16]. Similar bounds were proved for arbitrary trees by Evans *et al.* [64]. These results used the *recursive majority* [125] algorithm as it is easier to analyze than Felsenstein's maximum likelihood algorithm. When branch lengths are known, maximum likelihood provides optimal probability of correct reconstruction, so Theorem 3 applies to it as well.

The exact value of β is not important for the algorithms discussed in this section, but knowing bounds on β will be important for our algorithm in Chapter 4. We defer the discussion of the known bounds to that section.

A similar phase transition exists for the Jukes-Cantor model. The precise threshold is still unknown, though fairly accurate upper and lower bounds have been derived [43]. Thus,

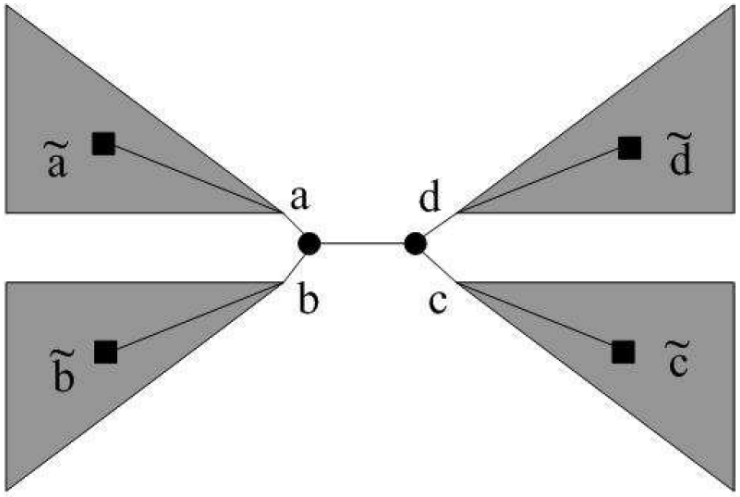

Figure 2.4: Applying the four-point method to a set of four error-independent reconstructed sequences ã,b̃,c̃,d̃ will yield a quartet whose external branches are longer due to the reconstruction error. Given enough columns, the inferred topology will still be correct, with high probability. This figure is taken from Mihaescu *et al.* [120].

the ideas behind these algorithms will translate naturally to more complicated models of evolution.

Suppose that we reconstruct ancestral sequences for subtrees T_1, T_2 of T, rooted at ρ_1, ρ_2, respectively. Moreover, suppose that the path connecting T_1 and T_2 has ends ρ_1, ρ_2 (see Figure 4.1). By the Markov property, reconstructing σ_{ρ_1} correctly is independent of reconstructing σ_{ρ_2} correctly. We call such two sequences *error-independent*. The distance estimate $\hat{d}(\sigma_{\rho_1}, \sigma_{\rho_2})$ will converge to $d(\sigma_{\rho_1}, \sigma_{\rho_2}) + g_1 + g_2$, where g_1 and g_2 are the edge lengths corresponding to the probabilities of incorrectly reconstructing states in the two sequences. When comparing independently reconstructed sequences, we can effectively treat these errors in the reconstructed sequences as "extra edges" [43], whose length can be bounded using Theorem 3. This means that the algorithm can reconstruct the phylogeny based on quartets whose external branch lengths (including the effect of ancestral reconstruction errors) are of constant length; see Figure 2.4. The number of such quartets is polynomial in n, so by Equation 2.19, we only need $O(\log n)$ alignment columns to ensure that all of them are inferred correctly with high probability. This observation has been used extensively in many theoretical algorithms [43, 125, 120].

Mossel [125] introduced the ancestral sequence reconstruction technique, and devised an $O(n^2)$ algorithm for reconstructing balanced trees. Daskalakis, Mossel, and Roch [43]

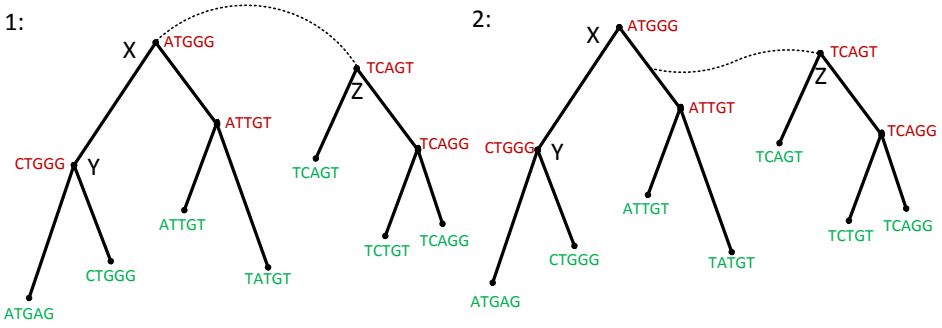

Figure 2.5: Independence relationships between reconstructed sequences: in the left tree, the sequences at nodes X and Z are error-independent. In the right tree, X and Z are not error-independent, but Y and Z are error-independent. The algorithm was run on subtrees drawn in solid lines. The dashed line represents an edge of the true tree that have not yet been reconstructed.

extended the technique to arbitrary topologies, giving an $O(n^5)$ time algorithm. Mihaescu, Hill and Rao [120] devised an improved algorithm, with a running time of $O(n^3)$. Roch [139] showed that the algorithmic framework behind the two algorithms can be rephrased as a purely distance-based method, without the need to explicitly reconstruct ancestral sequences.

The major challenge in algorithms using ancestral state reconstruction is making sure only error-independent quartets are used. This is challenging when the true topology is unknown. All the algorithms introduced so far detect non-independence by reconstructing additional quartets and monitoring for disagreement. Occasionally, non-independence of a quartet is detected long after it was used to join two subtrees, meaning that large parts of the reconstructed tree have to be erased and re-estimated. This greatly increases the worst-case running time.

2.5 Conclusion

While much has been written on improving the running time of phylogenetic algorithms, few algorithms are presently able to tackle data sets consisting of tens to hundreds of thousands of sequences on standard desktop computers. Such data sets generally require sub-quadratic runtime and memory usage. In practice, FastTree is currently the only

algorithm capable of processing data sets with hundreds of thousands of sequences within a few days. Unfortunately, FastTree lacks theoretical guarantees on accuracy, so it is uncertain whether it can reconstruct reasonably accurate phylogenies in all settings. In fact, recent results on Neighbour Joining by Lacey *et al.* [103] suggest that FastTree may not be the optimal choice for some phylogenies.

The only sub-quadratic algorithm with accuracy guarantees is the method by King *et al.* [101]. Its $O(n^2 \frac{\log\log\log n}{\log\log n})$ runtime bound is only marginally better than $O(n^2)$. Moreover, the algorithm is very sensitive to quartet errors, and consequently unlikely to be useful in practice.

A natural question is whether we can circumvent the need for fast algorithms by reconstructing the phylogeny on a smaller representative subset of the taxa. Such a heuristic approach would likely encounter several problems. First, the phylogeny of such a representative set would have some long branches not present in the phylogeny of the full data set. This could lead to lower accuracy, as suggested by the theoretical results in this chapter. This intuition is supported by empirical studies [175, 133], as well as our own experimental results in Chapter 5. Moreover, the choice of representatives may itself be a computationally non-trivial task, especially if the tree is highly unbalanced. Even if such heuristic methods could produce useful results in many practical settings, investigating the limits of phylogenetic reconstruction algorithms is still interesting in its own right.

We conclude that there is a need for principled algorithms that would be able to reconstruct large phylogenies with provably high accuracy. In the following chapters, we will describe algorithms that improve on the state of the art in both speed and accuracy, both in theory and in practice.

Chapter 3

QTree: a fast practical algorithm

Our first contribution is a fast quartet algorithm that builds a phylogeny by incrementally adding new taxa onto an existing tree. Several established quartet algorithms, such as Quartet Puzzling [158], employ a similar strategy, though their computational cost is substantially higher.

Incremental phylogenetic reconstruction algorithms add new taxa to a topology until all n taxa have been added. They optimize a greedy objective at all insertions, much as agglomerative algorithms (like neighbour joining or UPGMA) optimize an objective at all $n-1$ agglomerations. Such algorithms can be quite efficient. If each addition requires $O(f(n))$ time, the overall runtime is $O(nf(n))$.

We present an incremental algorithm where each insertion requires $O(\log n)$ runtime with high probability. Our algorithm first reconstructs a "guide tree": a phylogeny for a small subset of the taxa, and then completes it by incrementally inserting the remaining taxa in random order. If the guide tree is reconstructed correctly, other taxa can be added quickly to produce a tree for the full data set. Insertions are aided by a data structure which enables us to efficiently locate insertion sites in the phylogeny.

The probability that the algorithm makes any mistakes is $o(1)$ in a simple error model where every quartet used by the algorithm errs with known probability, independently of others. Thus, our randomized algorithm has runtime $O(n \log n)$ and returns the true topology, both with high probability (regardless of the true topology), in this model. We believe it is the first $O(n\text{poly}\log n)$-runtime algorithm with this sort of guarantees. An $o(n \log n)$-runtime phylogeny algorithm cannot return all topologies, so our algorithm is asymptotically optimal.

The assumption that quartet errors are independent does not hold in practice. For example, two quartet queries that include the same taxon are definitely not independent. The basic version of our algorithm makes heavy use of this unrealistic indepdendence assumption, and consequently performs poorly in experiments. We introduce a number of extensions to our method that greatly improve the accuracy of the resulting trees, while increasing the running time only by a modest constant factor. Our improved algorithm produces trees whose accuracy approaches that of Neighbour Joining. When combined with local search, our algorithm can produce trees whose accuracy is almost identical to that of FastTree, while being substantially faster.

We first give basic definitions and review several previous results needed in this chapter. We present a review of related work, give basic definitions, and then give the algorithm in the case of error-free data. Then, we extend the algorithm to the case of data containing noise and present some extensions to the basic data structure. We then describe the practical extensions to improve performance. Finally, we give some experimental results.

This work has previously been published in several publications. The theoretical algorithm was first published in a conference paper by Brown and Truszkowski [21] and its journal version [23]. The practical improvements and most experiments were published in another paper by Brown and Truszkowski [22] and its journal version by Truszkowski, Hao, and Brown [162]. The presentation in this chapter is largely based on the content of the two journal articles.

Our algorithm reconstructs large trees faster than any other algorithm currently in use, while offering reasonably good accuracy in practice.

3.0.1 Related work

Our theoretical analysis of QTree assumes a simple model of phylogenetic signal first introduced by Wu *et al.* [168]. In this model, the algorithm is allowed to ask quartet queries, some of which might be erroneous. Each quartet query gives the correct topology with probability at least $1 - p$. Otherwise, any of the two incorrect topologies can be returned. Wu *et al.* designed an $O(n^4 \log n)$ algorithm that reconstructs the correct phylogeny with probability at least $1 - p$ under this model. The algorithm is a greedy incremental algorithm, similar in spirit to Quartet Puzzling. This model has also been used for evaluating algorithms for solving the maximum quartet consistency problem [170].

We improve on Wu *et al.* in runtime and accuracy with an algorithm that runs in $O(n \log n)$ time (with high probability) and reconstructs the correct tree with probability

$1 - o(1)$ for large numbers of taxa. To our knowledge, it is the first provably error-tolerant, substantially sub-quadratic time algorithm for phylogenetic reconstruction.

Fast algorithms have been proposed for models where queries are error-free. Kannan *et al.* [95] use error-free *rooted triples* in an $O(n \log n)$ algorithm. Rooted triples reduce to quartets if we pick one taxon as an outgroup and always ask quartet queries for sets with that taxon, so that the algorithm works on error-free quartets.

Our algorithm uses ideas from work on noisy binary search in which comparisons have fixed error probability, by Feige *et al.* [66] and by Karp and Kleinberg [97]. Feige *et al.* consider the problem of locating an element in a sorted list in a scenario where each comparison may err with constant probability. They design a random walk that moves between intervals of the list, depending on the outcomes of certain comparisons. The random walk gradually converges to the correct location of the element in the list. We adapt this technique to searching in trees. Karp and Kleinberg apply a similar strategy to the problem of ranking a set of biased coins according to their probability of turning heads up.

The data structure used here appears similar to one used by Brodal *et al.* [18] for computing quartet distance between two phylogenies. While both structures represent a partition of the tree into hierarchically nested sets, the goal of their data structure is to aid enumerating certain sets of quartets for a given tree. Our structure is dynamic and supports different queries. Some ideas used in the proofs in this paper are similar to the results of Kao *et al.* [96] on randomized tree splitting, though they were developed independently.

3.1 Definitions

We begin with definitions about the two trees we will focus on: the phylogeny we are reconstructing and the search tree data structure that allows us to do efficient insertions.

Let T be a phylogeny and let v be an internal node of T. Removing v, and its incident edges, from T yields three subtrees, $t_i(T, v)$ for $i = 1, 2, 3$. The tree $t_i(T, v)$ joined with its edge to v is the *child subtree* $c_i(T, v)$. Phylogeny T' is *consistent* with T if its taxa are a subset of those of T, and T' is the union of all paths in T between taxa in T', with internal nodes of degree 2 removed. A *border node* of subtree T' in T is any internal node of T that is a leaf in T'.

A *quartet query* $q(a, b, c, d)$, returns one of three possible quartet topologies: $ab|cd$, $ac|bd$ and $ad|bc$. We assume a quartet query can be done in $O(1)$ time. In practice, this is done

using the four-point method, whose running time depends only on sequence length, but not on the total number of taxa. In Section 3.3 our error model considers how often quartet queries for four taxa of T are inconsistent with T. A *node query* $N(T, v, x)$ for internal node v of phylogeny T and new taxon x is a quartet query $q(x, a_1, a_2, a_3)$, where a_i is a leaf of T in $t_i(T, v)$. Such a query identifies the subtree $c_i(T, v)$ where taxon x belongs, if it is consistent with the true topology; for example, $xa_1|a_2a_3$ indicates x belongs in $t_1(T, v)$.

3.1.1 Search tree

A natural algorithm to add taxon x to phylogeny T begins at an internal node v and uses node query $N(T, v, x)$ to identify the $t_i(T, v)$ where taxon x belongs. We move to the neighbour of v in that subtree, and repeat the process until the subtree into which x is to be placed is only one edge e. We then break e into two edges and hang x onto the newly created internal node between them; see Figure 3.1. If we follow the path from v to an endpoint of e, the number of node queries equals this path length plus one. For a balanced tree with diameter $\Theta(\log n)$, this gives a $\Theta(n \log n)$ incremental phylogeny algorithm. For trees with $\Theta(n)$ diameter, this algorithm requires $\Omega(n^2)$ queries. We give a search tree

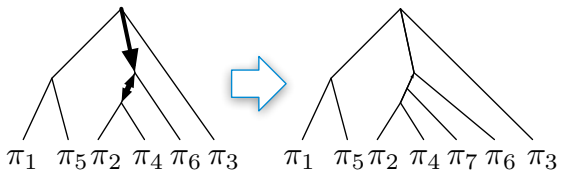

Figure 3.1: Natural incremental algorithm: start at root and search to find place for new taxon π_7 by asking queries down the path. Break an edge to insert the new taxon.

structure to manage the expected number of queries on the search path, regardless of the underlying tree topology. The goal of the structure is to provide the insertion algorithm with a sequence of node queries that always results in locating the right edge in $O(\log n)$ queries.

Definition 1. *A search tree $Y(T)$ for a phylogeny T is a rooted ternary tree satisfying the following conditions:*

1. *Each node y in $Y(T)$ is associated with a distinct subtree $r(y)$ of T.*
2. *The root of $Y(T)$ is associated with the full tree T.*

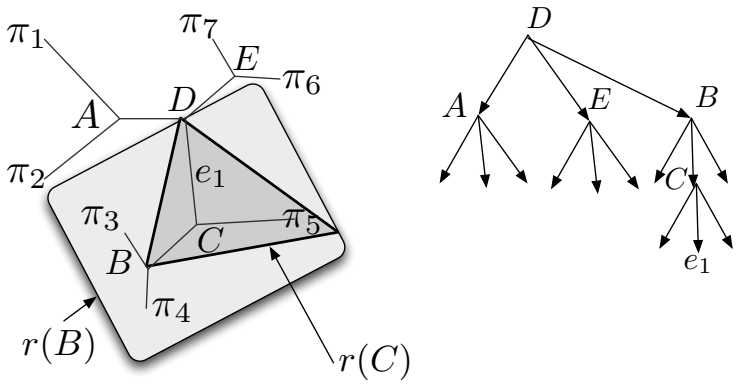

Figure 3.2: A phylogeny and a corresponding search tree. Internal nodes of the search tree correspond to subphylogenies: two, for $r(B)$ and $r(C)$, are indicated. Leaves of the search tree correspond to edges of the phylogeny.

3. *For each internal node y in $Y(T)$, there exists an internal node $s(y)$ in T such that the three subtrees associated with the children of y are the child subtrees $c_i(r(y), s(y))$. There are also three nonempty lists $\ell_i(y)$ stored at each internal node y; each element of $\ell_i(y)$ is a terminal taxon in $t_i(T, y)$.*

4. *For each node y in $Y(T)$, $r(y)$ has at most two border nodes in T*

$Y(T)$ is *complete* if each leaf in $Y(T)$ is associated with a single edge of T, and each edge of T has a corresponding leaf in $Y(T)$ (see Figure 3.2 for an example). For a given node y in the search tree, its associated node $s(y)$ in T may be picked so the three child subtrees are balanced. One of our main results, Lemma 2, shows that the tree *is* balanced with high probability if the taxa are inserted in random order.

3.2 An algorithm for error-free data

Using our search tree structure gives a straightforward incremental phylogeny algorithm if quartets are all consistent with T, the true topology.

We pick a random permutation π of the taxa, and start with the unique topology T_3 for $\{\pi_1, \pi_2, \pi_3\}$, and a search tree $Y(T_3)$ with four nodes: a root w with $r(w) = T_3$ and $s(w)$ as the internal node of T_3, and with one leaf for each edge of T_3. We also store $\ell_i(w) = \{\pi_i\}$;

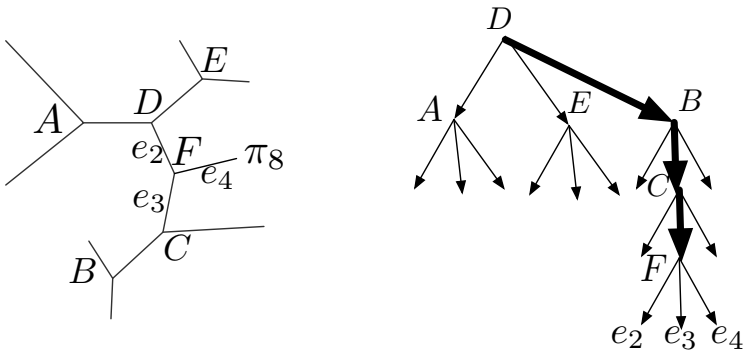

Figure 3.3: Inserting into a phylogeny. To insert π_8 into the phylogeny from Figure 3.2, we follow the path through the search tree indicated with bold lines. We find the correct edge to break to add π_8 to the tree, and modify the search tree locally to accommodate the change.

we also use $\ell_i(w)$ to represent the unique member of this set. This fits our requirements for a complete search tree of T_3.

Now, assuming T_i is consistent with T, and $Y(T_i)$ is a valid search tree for T_i, we add π_{i+1}, to produce T_{i+1} and $Y(T_{i+1})$ in the following way. We start at the root w of $Y(T_i)$ and ask the node query $N(T_i, s(w), \pi_{i+1})$, using the quartet $q(\pi_{i+1}, \ell_1(w), \ell_2(w), \ell_3(w))$; this tells us, in constant time, which child of w we should move to next. We continue until we reach a leaf y of $Y(T_i)$; this corresponds to the edge e of T_i where the new taxon π_{i+1} belongs. We break edge e into two parts, creating a new node u and a new edge from u to the new leaf π_{i+1}. The new tree is T_{i+1}.

To update $Y(T_i)$, we create three edges from y to a new node for each of the three newly created edges and let $\ell_1(y)$ be $\{\pi_{i+1}\}$, and set $\ell_2(y)$ and $\ell_3(y)$ to contain the taxon closest to π_{i+1} in the final quartet query and one of the two taxa that was not closest to π_{i+1} in that query. Since node y was a leaf in $Y(T_i)$, these nodes are in proper configuration with respect to y in T_{i+1}. See Figure 3.3. Assuming the quartet queries all are consistent with the true topology T, we discover in this way the proper place in the tree to insert each new taxon and maintain the invariants required for a complete search tree. In particular, the only subtrees whose border nodes need to be considered are those created by the new node addition, and as they are all either single edges or derived from a single edge in $Y(T_i)$, they continue to have at most two border nodes.

Theorem 4. *If all quartet queries made by this algorithm are consistent with T, then this algorithm returns T. Its runtime is $O(n \log n)$ with probability $1 - o(1)$.*

Proof. We have seen that the algorithm returns T. In the next subsection, we show that

inserting taxon π_i requires $O(\log n)$ queries with high probability. Each query requires $O(1)$ time, and the work to create a new edge requires constant time. The overall runtime is $O(n \log n)$ with high probability. □

3.2.1 The height of the search tree

To prove Theorem 4, we need to know the height of the search tree $Y(T)$. We now show that this tree is almost surely balanced if the permutation of taxa is uniformly chosen, regardless of the topology of the phylogeny.

Lemma 1. *For any phylogeny T, with n taxa, there exist two disjoint child subtrees A and B of the form $c_i(T, v)$ and $c_j(T, u)$ with at least $n/6$ and at most $n/3$ taxa each.*

Proof. We first show there exists a node u where all $t_i(T, u)$ have at most $n/2$ taxa. Pick an internal node u in T; if all $t_i(T, u)$ have at most $n/2$ taxa, we are done. Otherwise, move to the its neighbour in the $t_i(T, u)$ with the most taxa. This process terminates at a node u satisfying the property. Let $n_1 \geq n_2 \geq n_3$ be the numbers of taxa in the trees $t_i(T, u)$ at some step. Call the largest of these trees T_1. If $n_1 > \frac{n}{2}$, we move to the neighbour u^* of u in T_1; trees $t_i(T, u^*)$ have n_{11}, n_{12} and $n_2 + n_3$ taxa respectively where n_{11}, n_{12} are the numbers of taxa in the subtrees T_{11}, T_{12} of T_1 created by removing u^*. Since $n_2 + n_3 < \frac{n}{2}$, the only components that can have size over $\frac{n}{2}$ must be either T_{11} or T_{12}, which have fewer taxa than T_1 since they are its subtrees. Thus the number of taxa in the largest component decreases at each step of the process, which proves the claim.

Now, consider the node u we have found by this process, and let t_1 and t_2 be the two largest $t_i(T, u)$ subtrees, both of which have between $n/4$ and $n/2$ taxa. If t_1 has more than $n/3$ taxa, consider the three child subtrees in t_1 of the neighbour of u in t_1; the subtree that contains u has zero taxa, so the largest of the subtrees three must have at least $n/6$ taxa. If this tree has at most $n/3$ taxa, we have found our subtree A; if not, we move one step more away from u in the direction of the subtree having the most taxa until we find a subtree small enough. We analogously find B as a subtree of t_2. □

Lemma 2. *The number of node queries asked by the phylogeny algorithm to assign taxon π_{i+1} to its place in the tree is at most $37(\log_{6/5} i) \approx 203 \ln i$, with probability $1 - o(1/i^4)$, and at most $37(\log_{6/5} n)$ with probability $1 - o(1/n^4)$.*

Proof. Consider the process of adding π_{i+1} to the tree. We consider a sequence $y_1 \dots y_k$ of nodes in the search tree Y, each corresponding to a subtree $r(y_j)$ of the existing phylogeny.

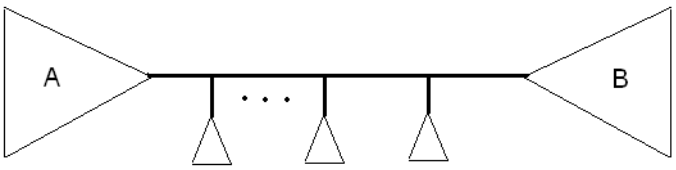

Figure 3.4: Illustration of the proof of Lemma 2. A phase lasts until we have asked queries involving at least one taxon from subtrees A and B, and at least one taxon outside A and B. Subtrees A and B are defined as in Lemma 1.

We divide the y_j into phases: phase t corresponds to the period in which $r(y_j)$ contains between $\frac{5}{6}^t i$ and $\frac{5}{6}^{t-1} i$ leaves; after $\log_{6/5} i$ phases, the algorithm has found where to put π_i. We show that the distribution of the length of each phase is bounded above by the sum of three geometrically-distributed random variables.

Each phase corresponds to taking a subtree and shrinking it by a factor of 5/6. This happens either if the largest of the three subtrees of the phylogeny descendant from the current search tree node y_j has at most 5/6 of the number of taxa we had at the beginning of the current phase, or earlier if π_i belongs in a tree with fewer than that many taxa. We concern ourselves only with the first of these ways of ending a phase, so we upper bound the length of a phase.

Let A and B be the subtrees of $r(y_j)$ satisfying Lemma 1. The queries asked include taxa found in $r(y_j)$, in the order that they occur in permutation π. In particular, we will ask a node query including a node of A with probability at least 1/6 at step, independently, until we finally do ask a query of a node from A. (Since our queries always include at least 5/6 of the taxa, and we have not queried any members of A, we always have all members of A available.) After querying a member of A, for the phase to continue, we must choose the subtree containing all of B. Now, we ask queries corresponding to the current subtree, until we see a taxon from B, which will happen with probability 1/6 or greater at each step. Now, we arrive in a state where the current subtree of the phylogeny includes border nodes inside A and B, since we must have cut off parts of A and of B, but cannot have cut off all of either without ending the phase. Now, we ask queries until we see a node from neither A nor B; this happens with probability at least 1/3 at each step. Then, the current search tree node y_j must correspond to a node on the path from A to B in the phylogeny, since otherwise one of its subtrees would have three border nodes.

Thus, the length of a phase is at most the sum of three geometric random variables, with expectations 6, 6 and 3; we then move to a subtree with at most $\frac{5}{6}n$ taxa. However, that new tree may have up to two border nodes as well; we label each of these border nodes

with a taxon from their neighbouring subtrees, thereby adding two taxa to the current subtree. After another node query, we will have descended to a smaller child subtree, having removed at least two taxa. This gives a new subtree with at most $\frac{5}{6}n$ taxa in which we can perform the next phase.

Thus, if $G(i)$ are independent geometric random variables with mean i, then the length of one phase is bounded above by $G(6) + G(6) + G(3) + 1$, and the expected total number of queries is at most $19(\log_{6/5} i) + 1$, where for simplicity, we let the $G(i)$ all have mean 6. Moreover, this variable is rarely above $37 \log_{6/5} i$. In particular, let $Q(t, r)$ be the negative binomial random variable that is the sum of t geometrically distributed variables with mean r. Then $\Pr[Q(t, r) > ktr] = \Pr[B(ktr, 1/r) < t]$, where $B(n, p)$ is a binomial random variable that results from the sum of n independent Bernoulli trials, each with mean p. By the multiplicative Chernoff bound ([55], p. 6), this probability is bounded above by $\exp(\frac{-kt(1-1/k)^2}{2})$. Here, we have $t = 3 \log_{6/5} i$, since there are 3 geometric random variables per phase, and $r = 6$. Putting $k = 2$, we get

$$\Pr[Q(3 \log_{6/5} i, 6) > 36 \log_{6/5} i] \leq \exp(-\frac{2 \cdot 3 \log_{6/5} i (1 - \frac{1}{2})^2}{2})$$

$$\Pr[Q(3 \log_{6/5} i, 6) > 36 \log_{6/5} i] \leq \exp(-\frac{3}{4} \log_{6/5} i)$$

$$\Pr[Q(3 \log_{6/5} i, 6) > 36 \log_{6/5} i] \leq i^{-\frac{3}{4 \ln 6/5}}) \approx i^{-4.11}$$

meaning that the probability we use more than $37 \log_{6/5} i$ queries for taxon π_{i+1} is $o(1/i^4)$; similarly, the probability that we use more than $37 \log_{6/5} n$ queries for taxon π_{i+1} is $o(1/n^4)$. □

We emphasize: $Y(T)$ is almost surely balanced regardless of the topology of T. Even if the diameter of T is $\Theta(n)$, its search tree almost surely has height $O(\log n)$. We expect that the actual values of the constants are much smaller than in the above lemma; certainly in our experiments, typical heights for the search tree have been on the order of $3 \ln n$, not $203 \ln n$.

3.3 Accounting for errors

Our search tree algorithm adapts to the case of error-prone quartets where each quartet query independently errs with probability $p > 0$, and that when it errs, it chooses each incorrect topology with probability 1/2. We assume that $(1 - p)^3 > 0.5 + \epsilon$ for some $\epsilon > 0$; we relax this assumption at the end of the section.

3.3.1 Random walk in the search tree

Let $Y(T')$ be a complete search tree for T' and let x be a taxon not in T'. We will perform a random walk on $Y(T')$ to place x into its proper location in T', where each step of the random walk is determined by at most 3 quartet queries.

Let y_i be the location of the random walk after i steps, with y_0 the root of $Y(T')$. If $y(i)$ is not a leaf node, query the border nodes of $r(y_i)$. If any border node queries gives answer $x \notin r(y_i)$, go to the parent node of y_i. If all border nodes give answers consistent with $x \in r(y_i)$, query the node y_i and descend to the child of y_i indicated.

If y_i is a leaf, corresponding to an edge of T', let it have counter variable c initially set to 0. Query its border nodes as before; if each is consistent with $x \in r(y)$, increment c. Otherwise, decrement it if it is greater than 0; if $c = 0$, move to the parent node of y. The pseudocode for one step of the random walk is shown as Algorithm 3. After a number of queries we will soon compute, we are at a node in $Y(T')$: if it is a leaf, add x to that node of the search tree as for the insertion algorithm with error-free data. If not, signal failure. Table 3.1 shows the first few steps of a run of this algorithm.

Algorithm 3 RandomWalkStep$(y, x, T, Y(T))$

```
Ask node queries on border nodes of r(y)
if One of them gives an answer consistent with x ∉ r(y) then
  if y is not a leaf OR c = 0 then
    return parent(y)
  else
    c ← c − 1
  end if
else
  if y is a leaf then
    c ← c + 1
  end if
  Ask a node query on y
  Let y′ be the child of y consistent with the answer
  if y′ is a leaf then
    c ← 0
  end if
  return y′
end if
```

Table 3.1: An example of execution of the random walk insertion algorithm to insert taxon π_8 into a tree shown in Figure 3.2, where the correct location for the new taxon is on the edge e_1 joining nodes D and C. For simplicity, all the queries in this execution give correct answers. If an incorrect quartet were returned in step 3, we might, for example, be returned to the node B as the active node.

Step	Current node	Border Nodes	Quartets queried	Decision
1	D		$\pi_1\pi_7\|\pi_4\pi_8$	B
2	B		$\pi_3\pi_4\|\pi_6\pi_8$	C
		D	$\pi_1\pi_6\|\pi_5\pi_8$	
3	C		$\pi_4\pi_5\|\pi_7\pi_8$	e_1, set c to 0
		D	$\pi_2\pi_7\|\pi_4\pi_8$	
		B	$\pi_3\pi_4\|\pi_5\pi_8$	
4	e_1			e_1, set c to $c+1$
		D	$\pi_2\pi_6\|\pi_4\pi_8$	
		C	$\pi_3\pi_5\|\pi_1\pi_8$	

The algorithm finds the proper place in the tree with high probability. Let y_x be the leaf in the search tree where we should insert taxon x. After i steps in the random walk, let the random variable d_i be the distance in the search tree between y_i and y_x. Let the random variable g_i have value $-c$ if $y_x = y_i$, $d_i + c$ if $y_x \neq y_i$ and y_i is a leaf of $Y(T')$, and d_i if y_i is not a leaf. If $g_i \leq 0$, then the current node of the random walk is the correct place to put x. The following simple observation is essential to proving the correctness of our algorithm.

Lemma 3. *Consider the random variables g_i defined above.*

1. $E[g_i] \leq d_0 + (1 - 2(1-p)^3)i$.
2. *If* $i > \frac{-d_0}{1-2(1-p)^3}$, *then* $\Pr[y_i \neq y_x] < \exp(\frac{-(d_0+i(1-2(1-p)^3)^2}{2i})$

Proof. At each step of the random walk, there are at most two border nodes, so at most three queries. If each gives a correct answer, g_i decreases by 1; if any incorrect queries occur g_i increases by at most one, though it might still decrease by 1. In the worst case, the probability that g_i decreases is at least $(1-p)^3$, so $E[g(i+1) - g(i)] \leq -(1-p)^3 + (1 - (1-p)^3) = 1 - 2(1-p)^3$. The result follows from linearity of expectation, since $g_0 = d_0$. The second claim follows from the Chernoff bound, as the queries are independent. □

Now, we have a straightforward taxon insertion algorithm. For each taxon π_{i+1}, we run the random walk long enough to handle the case that $g_0 = 203 \ln i$, consistent with our estimate of the depth of the tree. To make the error probability at most $(1/i^2)$, we require that the random walk have j steps, where $\exp(\frac{-(203 \ln i + j(1-2(1-p)^3))^2}{2j}) \leq \frac{1}{i^2}$. The minimum value of j to make this guarantee is $j \geq k \ln n$, for $k = \frac{-203(1-2(1-p)^3)+2+2\sqrt{1-203((1-2(1-p)^3)}}{(1-2(1-p)^3)^2}$.

We can now state the taxon insertion procedure in detail, as Algorithm 4.

Algorithm 4 InsertTaxon$(x, T, Y(T))$

```
Initialize y_0 as the root of Y(T).
for i = 1 to k log n do
  y_i = RandomWalkStep(y_{i-1}, x, T, Y(T))
end for
Let y_{k log n} be the current node of the random walk.
if y_{k log n} is a leaf then
  Attach x to r(y_{k log n}) in T and update Y(T).
else
  return Failure.
end if
```

Assuming that the tree T_{i-1} is correct, then, this algorithm adds a new taxon in $O(\log n)$ queries, with error or failure probability $O(1/i^2)$.

3.3.2 Finding quartets to ask

We must ensure that we can always find a quartet that has not been queried before in $O(1)$ time. This requires two separate conditions to hold: first, that enough such quartets exist, and second, that we can find them in $O(1)$ time per query.

The first of these is easy if we start with a constant-sized guide tree T_S on a set S of at least M taxa, where M is the smallest number such that $k \log M < M - 2$, with k equal to the multiple of $\log i$ found using the formula in the previous section. In each insertion, we use at most $k \log i$ quartets at any node of the search tree; the extreme case is where the three child subtrees of the current tree T have 1, 1, and $i - 2$ taxa in them.

The latter is more complicated. Assume that for each node y in Y, $\ell_j(y)$ is the list of all taxa in the child subtree $t_j(r(y), s(y))$ (for $j = 1, 2, 3$). To find the next quartet in $O(1)$ time, we must fetch the next taxon in $t_j(T, s(y))$ in $O(1)$ time. We first enumerate

taxa in $\ell_j(y)$. Once all taxa in $\ell_j(y)$ have been used, we pick the border node $b_j(y)$ of y in $t_j(T, s(y))$ (if it exists). The node $b_j(y)$ is associated with some ancestor y_1 of y and we have $r(y) \subseteq t_i(r(y_1), b_j(y))$ for some i. Taxa in $\ell_{(i+1) \bmod 3}(y_1) \cup \ell_{(i+2) \bmod 3}(y_1)$ are also in $t_j(T, s(y))$ so we enumerate them. Once they have been used, we find border nodes of $r(y_1)$ such that two of their taxa lists contain taxa in $t_j(T, s(y))$ that have not been used so far. Once all taxa from node y_i have been used, we look at border nodes of $r(y_i)$. This process can be thought of as breadth first search on a directed graph where an arc denotes the relationship of being a border node, and each query takes constant time.

Now, we give the complete algorithm. First, pick a random set $S_0 \subset \mathcal{S}$ of M taxa and find the phylogeny for S_0 consistent with the most quartets. Then iteratively add taxa to the tree using the procedure InsertTaxon described above. The running time of this

Algorithm 5 Reconstruct($\mathcal{S}$,M)

Pick a random subset $S_0 \subset \mathcal{S}$ with M taxa
Find phylogeny T on S_0 consistent with the most quartets by exhaustive search over all topologies on the taxa of S_0.
Build a search tree $Y(T)$ for T.
for all $s \in \mathcal{S} \backslash S_0$ **do**
  insertTaxon(s,T,Y(T))
end for

algorithm is $O(n \log n)$ with high probability. The error probability can be bounded by $\mu(M) + \sum_{i=M}^{n} \frac{1}{i^2}$, where $\mu(M)$ is the probability that the maximum quartet compatibility tree on a random set of M taxa is not consistent with T. This quantity is constant for constant M, since $\sum_{i=0}^{\infty} \frac{1}{i^2}$ is a constant; in the next section we show $\mu(M)$ is $o(1)$ for very small guide trees made using MQC.

The case where $(1-p)^3 \leq \frac{1}{2}$ can be solved by redefining node queries. Each node query is now implemented by asking c_p queries and returning the majority direction, with constants c_p and M chosen appropriately.

3.4 Shrinking the error probability to $o(1)$

The algorithm presented in the previous section errs with constant probability, since it starts with a constant-sized tree that may have errors, and since the additions to this tree also have constant probability of containing any error.

If we start with a non-constant-sized guide tree, we can reduce the error probability.

Theorem 5. *The algorithm Reconstruct*$(\mathcal{S}, \max(\lceil \log\log n \rceil, M))$ *both returns the correct tree and runs in* $O(n \log n)$ *time with probability* $1 - o(1)$.

Proof. The exhaustive search step requires enumerating all $O((\log\log n)^4)$ quartets, on all $O((\log\log n)! \log n)$ topologies on $\log\log n$ taxa; the product of these is $O((\log\log n)^{4+\log\log n} \log n)$, which is sublinear in n. We have already shown that the rest of the algorithm requires $O(n \log n)$ time with high probability.

We will show below that $\mu(\log\log n)$, the failure probability of the guide tree algorithm, is $o(1)$. The failure probability of the insertion procedure is at most $\sum_{i=\log\log n}^{n} \frac{1}{i^2}$, which is $O(\frac{1}{\log\log n})$, and so $o(1)$. As such, the overall failure probability is $o(1)$. □

We note that the guide tree could have more or fewer than $\log\log n$ taxa; we merely require that the brute-force guide tree construction requires $O(n \log n)$ time and has $o(1)$ error probability.

3.4.1 Maximum quartet consistency is consistent in the simple error model

Here, we show that the *maximum quartet consistency* approach is consistent for our error model. This result (which may be of independent interest, as our error model has been studied before [170]), shows that $\mu(n) \to 0$ as n grows.

Theorem 6. *Let* T_{mqc} *be the phylogeny compatible with the most quartet queries for a set of* n *taxa and let* T^* *be the true phylogeny. If each quartet query errs independently with probability* p*, then* $\mu(n) = \Pr[T_{mqc} \neq T^*] = o(1)$ *as a function of* n.

To prove this theorem, we first show a few properties of quartets.

Definition 2. *The quartet distance* $d_Q(T, T')$ *of phylogenies* T *and* T' *on the same set of taxa is the number of quartets on which* T *and* T' *differ.*

This distance was studied in [18, 28] among others.

Lemma 4. *The quartet distance between distinct phylogenies is at least* $n - 3$.

Proof. Let T and T' be distinct phylogenies. Let (S_1, S_2) be a split in T not present in T'. Let (S'_1, S'_2) be a split in T' not present in T where none of the sets $A = S_1 \cap S'_1, B = S_1 \cap S'_2, C = S_2 \cap S'_1, D = S_2 \cap S'_2$ is empty; such a split exists since T and T' are distinct [31]. Choose taxa a, b, c, d from sets A, B, C, D, respectively. The quartet induced by T is $ab|cd$, whereas in T' it is $ac|bd$. This pair of splits gives $\phi = |A||B||C||D|$ conflicting quartets; ϕ is at least $n - 3$ since $|A| + |B| + |C| + |D| = n$, and the product is minimized when $|A| = n - 3$ and $|B| = |C| = |D| = 1$. □

We now show that the number of trees with small quartet distance from a fixed tree T is small.

Definition 3. *A taxon reinsertion (TR) operation consists of deleting a taxon from a phylogeny and attaching it to a remaining edge, creating three new edges.*

Lemma 5. *Let T and T' be phylogenies such that $d_Q(T, T') < n \log^2 n$. The number of TR operations required to transform T into T' is at most $c \log^4 n$ for some constant c.*

Proof. Let (S_1, S_2) be a split of T not present in T'. Let (S'_1, S'_2) be some split in T' that is not present in T that minimizes $\phi = |A||B||C||D|$ as defined earlier. Without loss of generality, assume that A is the largest of the sets. Observe that, for n large enough, each of the sets B, C, D must have at most $\log^2 n$ taxa: otherwise $\phi > n \log^2 n$, so $d_Q(T, T') > n \log^2 n$. We delete all taxa in B and C from both T and T' to create trees $T^{(1)}$ and $T'^{(1)}$. By Lemma 4, this erases at least $n - 3$ conflicting quartets. We pick splits (S_1, S_2) and (S'_1, S'_2) in $T^{(1)}$ and $T'^{(1)}$ as we previously did for the original trees and repeat the process to obtain trees $T^{(2)}$ and $T'^{(2)}$, this time removing at least $n - 2 \log^2 n - 3$ discordant quartets.

We iterate the process until $T^{(i)} = T'^{(i)}$ for some i, which is $O(\log^2 n)$ since the total number of conflicting quartets is at most $n \log^2 n$, and each iteration erases $\Omega(n)$. The sets B and C have at most $\log^2 n$ taxa at each step of the algorithm. Therefore, at most $O(\log^4 n)$ taxa are deleted from both trees.

Let R be the taxa removed. The restrictions of both T and T' to $\mathcal{S} - R$ are the same. To transform T to T', we move all nodes in R to a new side of the tree T, and then move each to the proper place in T' in $O(\log^4 n)$ TR operations. □

Corollary 1. *For any phylogeny T, the number of phylogenies T' such that $d_Q(T, T') < n \log^2 n$ is at most $n^{b \log^4 n}$ for a large enough constant b.*

Proof. Each T' with distance from T at most $n \log^2 n$ can be obtained from T by $c \log^4 n$ TR operations. For any tree, the number of ways to perform a TR operation is less than $2n^2$ since we can choose any of the n taxa and reinsert it at any of the $2n - 5$ edges other than the one at which it was before the operation. This gives fewer than $(2n^2)^{c \log^4 n}$ phylogenies that can be created by repeating the operation $c \log^4 n$ times. Taking $b = 4c$ finishes the proof. □

Now we can prove the maximum quartet compatibility consistency theorem.

Proof. Suppose some tree T' is consistent with more quartets than T^*, and $d_Q(T', T^*) = q$. At least half of the q quartets where T^* and T' differ must be erroneous; since they are independent errors, this has probability at most $\exp(-q(\frac{(1-2p)^2}{2}))$ by the Chernoff bound.

Let $\mathcal{T}_0$ be the set of all incorrect phylogenies with quartet distance from T^* less than $n \log^2 n$. Then $|\mathcal{T}^0| \leq n^{b \log^4 n}$, and for trees in $\mathcal{T}_0$, Lemma 4 gives that $q \geq n - 3$. The probability that any tree in T_0 is consistent with more queries than T^* is bounded by $n^{b \log^4 n} \exp(-(n-3)(\frac{(1-2p)^2}{2}))$, which is $o(1)$ as n grows.

Now, consider the incorrect phylogenies $\mathcal{T}_1$ that are not in $\mathcal{T}_0$. There are fewer than $2^n n! < 2^{n(1+\log n)}$ such topologies, and for each, $d_q(T, T^*) \geq n \log^2 n$. The probability that any tree in $\mathcal{T}^1$ is consistent with more quartets than T^* is bounded above by $2^{n(1+\log n)} \exp(-n \log^2 n(\frac{(1-2p)^2}{2}))$ by the union bound. This is $o(1)$ as n grows.

So the probability that any incorrect tree is consistent with more quartets than T^* converges to 0 as n grows. □

The following corollary, which may be of independent interest, is a consequence of Theorem 5 and Theorem 6.

Corollary 2. *For the error model assumed above, the algorithm Reconstruct*$(\mathcal{S}, \max(f(n), m))$ *reconstructs the maximum quartet consistency tree with probability* $1 - o(1)$*, for any monotonic, divergent function* f*.*

Finding the maximum quartet consistency tree is *NP*-hard in general [14], though polynomial-time approximation schemes exist [93]. Wu *et al.* [170] used essentially the same model to test their algorithm for finding MQC trees. The above result suggests that most MQC instances generated this way are solvable in polynomial time, using first the guide tree and then our algorithm, so the benchmarks provided by the model are unlikely to reflect the difficulty of finding MQC trees in general.

3.5 A dynamic data structure

The depth of our search tree depends on the order of taxon insertions. In particular, if taxa are inserted in a non-random order, the depth of the search tree may be $\Omega(n)$. In this section, we present a modification of the search tree data structure that achieves (with high probability) logarithmic tree depth regardless of the order of insertions, and also enables deletions from the tree.

Our modification is an analogue of *random treaps* [127]. A random treap is a randomized binary search tree that maintains logarithmic depth with high probability. Each node in a random treap has a priority value as well as a key value. The priorities satisfy the max-heap property where each node has priority higher than its children, whereas the keys form a binary search tree. The structure of a treap is the same as that of as a binary search tree created by inserting keys in decreasing order of priorities.

To insert a key into a treap, we first insert the key in the existing binary search tree structure. Then, we sample a priority from the unit interval. We then do a series of rotation operations on the nodes of the search tree to preserve the heap property on the priorities. The resultant treap corresponds to a binary search tree created by inserting the set of keys in random order, which is well-known to be balanced with high probability (see *e.g.* [51]).

We follow a similar logic with our data structure. First, we define rotation operations and show how to maintain the heap property with them. Then, we give algorithms for insertion and deletion from the search tree. Analogously with the treap case, the resultant search tree is chosen from the distribution of search trees that result from a uniform choice of order permutations, which we have shown to be balanced with high probability. This result holds regardless of the insertion order.

First, let us define the new data structure.

Definition 4. *A random priority search tree $Y'(T)$ for a phylogeny T is a search tree for T with the additional requirements:*

1. *For each taxon x in T, we have a random priority $\pi(x) \in [0, 1]$, chosen uniformly and independently of all taxa.*
2. *The topology of the search tree is consistent with inserting the taxa in decreasing order of priorities using the algorithm of Section 4.*
3. *For each non-leaf node y in $Y'(T)$, its priority $\pi(y)$ is defined as the priority of the taxon whose insertion made it a non-leaf.*

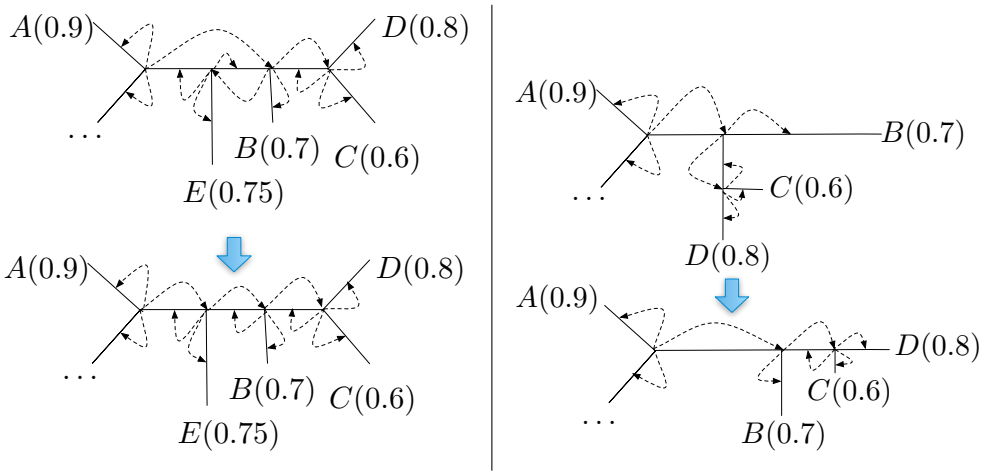

Figure 3.5: Here, we see the effect of one search tree rotation in the standard case (left) and the strongly peripheral case (right). The phylogeny is shown with solid lines, while the directed search tree is shown with dotted arrows. In the left case, the taxon E, with priority 0.75, has been inserted into the phylogeny, but the node corresponding to its insertion has priority below that of its parent, which was created when B (priority 0.7) was inserted, so we must rotate those two nodes. In the right case, the taxon D, with priority 0.8, has been inserted into the phylogeny, and the corresponding new node has priority below that of its parent. However, the result of the rotation is no change to the structure of the search tree, as the corresponding search tree would be the same; only the labels of the internal nodes will change.

Priorities satisfy the max-heap property, meaning that a node always has higher priority than any of its descendants. Since all permutations are equally likely, a random priority search tree is balanced with high probability, by Theorem 2. We now describe the rotation operations needed to maintain the heap property.

Definition 5. *A node y is peripheral if $r(y)$ contains the taxon whose insertion created y. It is strongly peripheral if its parent is also a peripheral node.*

Let y_c be a non-leaf node in the search tree and let y_p be its parent. The algorithm Rotate(y_p,y_c) is illustrated in Figure 4. In most cases, it swaps y_p and y_c and rearranges their children to preserve the search tree properties. If y_c is strongly peripheral, no rearrangement of the search tree topology is needed.

The procedure InsertTaxonDynamic first inserts the taxon into the search tree and then performs rotations moving up the search tree until the heap property is satisfied by the

Algorithm 6 Rotate(y_p,y_c)

if y_c is not strongly peripheral **then**
  Identify the unique child y_{mid} of y_c whose borders are $s(y_p)$ and $s(y_c)$
  Detach y_c from y_p and y_{mid} from y_c
  Make y_{mid} a child of y_p
  Make y_p a child of y_c
  Make y_c a child of the former parent of y_p
else
  Swap the nodes y_p and y_c
end if

Algorithm 7 InsertTaxonDynamic(x,T,$Y'(T)$)

InsertTaxon(x,T,$Y'(T)$)
Sample $\pi(x)$ uniformly at random from $[0, 1]$
Let y be the node at which x was inserted
while $\pi(y) > \pi(parent(y))$ **do**
  Rotate($parent(y)$,y)
end while

priorities in the search tree. The correctness of the insertion procedure is proved by the following lemma.

Lemma 6. *At the end of each iteration of the main loop of InsertTaxonDynamic, the subtree rooted at y satisfies the heap property.*

Proof. Induction on the number of rotations performed so far. At the beginning of the execution of the main loop, y has only new leaves as children, so the property holds trivially. Suppose the property holds before a rotation is performed. It is easy to see that the only pair of nodes that could possibly violate the heap property after a rotation is (y_p, y_{mid}), since y_{mid} was not a child of y_p before the rotation. But y_{mid} was a descendant of y_p before adding x to the search tree, so we must have $\pi(y_{mid}) < \pi(y_p)$, which concludes the proof. □

After the execution of *InsertTaxonDynamic*, the topology of the search tree is the same as if taxa had been inserted in decreasing order of priorities. Since the priorities are drawn uniformly at random, each permutation of taxa is equally likely. By Theorem 2, it follows that the tree is balanced with high probability.

It is also possible to delete taxa from the search tree using a similar strategy. The algorithm descends down the search tree, performing rotations until it reaches a topology that is consistent with x being the final inserted taxon. Once that is achieved, x is removed from the tree, and the resulting search tree is consistent with inserting all taxa other than x in random order.

Algorithm 8 DeleteTaxonDynamic(x,T,$Y'(T)$)

$\pi(x) \leftarrow 0$
Let y be the node at which x was inserted
while y is not a leaf **do**
  Pick the child y_c of y with highest priority
  Rotate(y,y_c)
end while
Remove y and its siblings

In the above analysis, we have ignored the problem of maintaining the lists $\ell_i(y)$ for each search tree node. This can be handled by using linked lists so that merging operations take $O(1)$ time.

3.6 Extensions to improve performance

So far, we have assumed an error model where quartets queries err independently of each other, with known error probability. This model is seriously oversimplified; in practice, quartet errors are positively correlated and occur with unknown probability, and the guide tree is often error prone. As a result, our basic algorithm suffers from a number of issues that greatly limit its practical performance. The accuracy of quartet inference is often very low, particularly for quartets asked at the top nodes of the search tree; when running the basic algorithm on the COG840 data set of 1250 taxa [134], we found that only 66% of quartets were inferred correctly, with only 50% of correct inferences at the root of the search tree a random choice would be correct 1/3 of the time. In practice, quartet inference errors are also not independent, which may cause the random walk to "drift away" from the optimal placement of the taxon either to an incorrect place or to drift in the higher nodes of the search tree, leading to reduced coverage. As a result of these problems, the accuracy of our basic algorithm is often much lower than that of Neighbour Joining (see Table 3.2). In this section, we discuss a number of improvements to the algorithm to address these problems.

Quartet weights

The basic algorithm considers all quartets to be equally reliable. In practice, some quartets are more likely to be correct than others. Assigning equal weight to all quartets can lead to serious errors due to many erroneous quartets. Various quartet phylogeny algorithms estimate the reliability of inferred quartet topologies based on their likelihood scores [158, 157, 137] or distances between taxa (*e.g.* [148]).

We assign quartet weights based on the inferred middle edge length normalized by the the sum of all the edge lengths of the quartet. For a quartet with external edge lengths a, b, c, d and an internal edge length e, this becomes $w = e/(a+b+c+d+e)$. The intuition behind this idea is that if the inferred middle edge is long compared to the distances between taxa, then the inferred quartet topology is less likely to have arisen from errors in the distance estimates. The edge lengths are inferred from the six pairwise distances using the four-point method, which was dictated by the speed of this approach.

We note that this weighing scheme is somewhat different than other commonly used schemes. Most quartet algorithms use likelihood weights as opposed to distances. Snir *et al.* [148] use the topological diameter of the quartet in a preliminary tree constructed by Neighbor Joining.

To incorporate the reliability information into the algorithm, we ask k quartet queries at each node query and vote according to the weights. We have tested two voting schemes. In the weighted-majority scheme, the weights of each quartet pointing in the same direction are added and the direction with the highest vote total is chosen. In the winner-takes-all scheme, the direction is chosen according to the quartet query with the highest weight from the k used. We found that the weighted-majority voting scheme results in better accuracy according to the quartet measure, while the winner-takes-all approach yields higher accuracy according to the Robinson-Foulds measure (see Section 3.8).

Biased choice of quartets

Many authors have suggested that large distances between quartet taxa lower the accuracy [148, 62]. While our algorithm is forced to ask long quartets at the top of the search tree, biasing the choice towards shorter quartets at the lower nodes of the search tree might reduce the number of unreliable quartets. When asking a node query at node y of the search tree, we require that the representatives for each quartet query are contained in a subtree $r(y^*)$ associated with a node y^* that is at most d levels above y in the search tree. Recall that proximity in the search tree need not imply proximity in the phylogeny.

The value d is chosen for each y so that the number of possible representatives in each direction is at least 20. Note that this does not change the behaviour of the algorithm in the upper parts of the search tree, since the subtrees associated with the upper nodes are typically much larger. This heuristic is used in all experiments.

Multiple insertion rounds

Due to the ambiguity in inferred quartets, the random walk will often terminate at an internal node in the search tree, resulting in the taxon not being inserted into the phylogeny. After the algorithm has attempted to insert every taxon in the search tree, we try reinserting the taxa that did not make it to a leaf in the first round. We do two such rounds of reinsertions.

Confidence threshold

Incorrectly inserting a taxon can prevent subsequent taxa from being inserted in the correct place. To mitigate this, we only add a new taxon to the phylogeny if it has spent more than the $\ell = 30$ last steps of the random walk at the current leaf.

Repeating the random walk

To improve our confidence in the placement of new taxa, we ran the random walk two times for each inserted taxon. The taxon was then inserted only if both walks terminated at the same leaf node.

3.7 Unsuccessful attempts to improve accuracy

Several natural methods for improving our algorithm's performance have not yet been successful. We note these extensions here to document the challenging process of improving a theoretical prototype into a useful method for phylogenetic inference.

Increasing the number of quartet queries without weighting them improved the fraction of taxa inserted, but decreased the accuracy. This emphasizes the need for weighing quartets.

We have tried various other methods of estimating the reliability of quartets. Earlier work on short quartet methods [148, 62] suggested using the diameter of the quartet for

estimating its reliability. Contrary to our expectations, this approach did not provide satisfactory results. This may be because the diameter of the inferred quartets varies widely between different levels of the search tree. Quartets inferred near the root of the search tree tend to have large distances between taxa, whereas quartets inferred in the deeper parts of the search tree are shorter since they only span a small fraction of the overall tree. Using least squares fit of the distances to the quartet topology to weigh the quartets also did not seem to improve the accuracy of node queries. Using weighted least squares did not improve the accuracy of quartet inferences.

We have tried to find a good starting point for the random walk by using profile search. For each of the top $\log n$ nodes of the search tree, we constructed a profile of sequences in the subtree associated with that search tree node. Before starting the random walk for the new taxon, we aligned its sequence to each profile and started the random walk from the subtree whose profile gave the highest alignment score. This did not have a significant effect on the results, perhaps because of the large diversity of sequences within the top subtrees.

3.8 Experiments

We have evaluated our algorithm QTree on several simulated data sets. The simulated data sets are taken from Price *et al.* [134] who used them to evaluate their heuristic program FastTree, which has $O(n^{1.5} \log n)$ runtime on typical instances. These data sets were generated by taking real protein alignments and their maximum likelihood trees, and then simulating evolution of sites along that phylogenetic tree, treating it as ground truth. The evolutionary rates varied across sites.

We used two different accuracy metrics: the Robinson-Foulds accuracy and the quartet accuracy. Given a ground-truth topology for a set of taxa and a second topology on the same taxa, the Robinson-Foulds accuracy is the fraction of the non-trivial splits (internal edges) of one topology found in the other. The quartet accuracy (see, *e.g.*, [28, 18]) is the fraction of the $\binom{n}{4}$ quartets that have the same result in both tree topologies. The Robinson-Foulds measure is fragile: a single misplaced taxon can ruin all of the splits of the tree. By contrast, the quartet distance is robust to individual errors: even a randomly-placed taxon will probably have the correct topology in one-third of its quartets, as there are only three choices for the topology. Thus, a topology with a single misplaced taxon will likely have a high quartet accuracy, provided that relationships between other taxa are inferred correctly.

Some of the phylogenies produced by QTree will not include all of the taxa in the alignment. We refer to the percentage of the taxa inserted as the *coverage*.

In most experiments, the initial guide tree was created using Neighbor Joining on a randomly chosen subset of 200 taxa. For protein sequences, our algorithm uses ScoreDist [150] to estimate distances. FastTree uses a similar, but distinct, distance estimator [134]. For nucleotide sequences, both algorithms use the standard log-corrected Jukes-Cantor distance described in Chapter 2. For a fair comparison of accuracies and running times, we only compare against the initial Neighbor Joining phase of FastTree. FastTree then performs nearest neighbour interchanges to improve the quality of the resultant tree, which takes up much of its runtime; we have not incorporated these into our algorithm, either, so we compare against the initial phase of FastTree for consistency of comparison.

In the first experiment, we assessed how various improvements discussed in the previous section changed the performance of the algorithm. We ran each version of the algorithm 100 times on the COG840 data set. The results are given in Table 3.2.

Overall, the improvements to the algorithm boosted the RF accuracy from 46% to around 60%, and increased the proportion of inserted taxa from 56% to 82-95%, depending on the settings. The biggest gains were achieved by introducing quartet weights and a confidence threshold. Multiple rounds of insertions increased the coverage of the taxa set as expected. Increasing the number of quartets and running the random walk twice increased the accuracy, but at the cost of increased running time. It should be noted, however, that the increase in running time was not directly proportional to the number of quartets per query: one run of the algorithm for 5 quartets per query takes around 11 seconds, compared to just under 20 seconds for 20 quartets per query. This is because the running time is dominated by computing the distances, which we save and reuse in the subsequent quartet queries that include the same pair of taxa.

In general, WTA voting tended to produce trees with higher RF accuracy than weighted-majority voting. In contrast, weighted-majority voting resulted in higher quartet accuracy, as we shall see in the next experiment.

We see that there is a trade-off between the proportion of the inserted taxa and the RF accuracy. This is not surprising since incorrectly inserting a hard-to-place taxon into the phylogeny can cause many splits to be incorrect.

The accuracy of guide trees was considerably lower than that of the NJ tree for the full data set. To investigate the impact of errors in the guide tree on the accuracy of the algorithm, we ran the algorithm starting from a guide tree consistent with the true phylogeny. Contrary to our expectations, errors in the guide tree have little impact on the

Table 3.2: Shown are results for the COG840 protein alignment with 1250 taxa and 391 alignment columns. We show our algorithm's performance in various settings, and compare it to Neighbor Joining. We report accuracies using the Robinson-Foulds measure. Our algorithm places approximately $80\% - 90\%$ of taxa with accuracy around 60%. We ran each version of the algorithm 100 times. In all cases, the guide tree is on 200 taxa; except in the second line of the table, this was generated with Neighbor Joining, and had RF accuracy of $50\% \pm 3\%$. Three voting schemes were used in the experiments: unweighted majority (UM), weighted majority (WM), and winner-takes-all (WTA). In some experiments, we also added 2 additional rounds of insertions (2E), and a confidence threshold for insertion (CT).

method	% taxa inserted	RF accuracy
basic RW+NJ guide tree	56.3 ± 2.4	46.3 ± 4.6
basic RW+true guide tree (not feasible in practice)	57.3 ± 2.1	49.4 ± 5.0
5 quartets per node query, UM	76.4 ± 2.0	41.0 ± 3.8
5 quartets, WM	85.6 ± 1.6	48.6 ± 3.7
5 quartets, WM, 2E	95.4 ± 1.0	45.5 ± 3.4
5 quartets, WM, CT, 2E	84.1 ± 1.8	57.4 ± 3.5
5 quartets, WM, re-running the RW, 2E	78.8 ± 2.4	59.5 ± 3.7
5 quartets, WTA, CT, 2E	80.2 ± 2.1	62.3 ± 2.9
20 quartets, WTA, CT, 2E	92.1 ± 1.4	60.8 ± 2.9
NJ	n/a	62.6

Table 3.3: Performance of the random walk algorithm on synthetic protein alignments.The size of the guide tree was 200 except for the 250 taxon data set, where it was set to 100. The average RF accuracy of the guide trees was $65, 50$, and 46% for the $250, 1250$, and 5000-taxon data sets, respectively. The average quartet accuracies for the guide trees were $73, 83$ and 55%. When all taxa are forced into the random walk tree (see text), the RF accuracy decreases by $7 - 14\%$, depending on the data set. All random walk runs use the confidence threshold heuristic and two additional rounds of insertions.

	data set								
	COG1011 250 sequences 228 columns			COG840 1250 sequences 391 columns			COG1028 5000 sequences 251 columns		
method	%taxa	RF	QA	%taxa	RF	QA	%taxa	RF	QA
weighted majority,5 quartets	88.5	66.2	72.8	84.1	57.4	85.8	74.4	51.4	59.5
WTA-vote,5 quartets	86.0	69.4	70.8	80.2	62.3	85.4	70.1	57.0	59.6
weighted majority,20 quartets	95.6	60.4	69.9	96.4	50.6	83.9	94.3	41.1	56.5
WTA-vote,20 quartets	94.0	69.4	73.4	92.1	60.8	83.1	89.7	57.6	55.3
NJ	100	73.6	70.0	100	62.6	88.0	100	73.0	66.3
FastTree(NJ phase only)	100	69.7	85.9	100	61.0	86.6	100	73.6	66.4
weighted maj.,5 quartets,force all	100	59.0	69.7	100	48.7	80.8	100	37.3	52.4

accuracy; substituting the NJ guide tree with the true one improved the accuracy by 2 to 3 percent, depending on the version of the algorithm.

In the second experiment, we evaluated the algorithm on three unrelated data sets having 250, 1250, and 5000 taxa, respectively. For each data set, we ran the algorithm in four different configurations, using two voting schemes and varying the number of quartet queries per node query. The size of the guide tree was 200 except for the 250 taxon data set, where it was set to 100. The results are shown in Table 5.2.

We see that the weighted-majority voting scheme tends to produce trees with higher quartet accuracy, whereas the winner-takes-all voting scheme yields higher RF accuracy. In general, we see that the two measures are very different: an algorithm can have relatively good performance according to one while being considerably worse in the other, though the difference between voting schemes tends to be more pronounced for RF accuracy. In most cases, the accuracy of the random walk algorithm is lower than the accuracy of Neighbor Joining.

In the next experiment, we measured the running time of our algorithm on large data sets. For this experiment, we took a large simulated nucleotide alignment (78,132 taxa and 1287 columns) based on a real 16S alignment and compared the runtimes of the initial phase of FastTree and our algorithm. We ran the algorithms on the full alignment and on

Table 3.4: Running times of the random walk algorithm compared to FastTree. We used the huge.1 alignment from the original FastTree paper [134]. Smaller data sets were created by choosing a random subset of sequences from the large alignment. Our algorithm runs 2.1 to 3.4 times faster than FastTree on these very large data sets.

	# of sequences		
	20,000	40,000	78,132
weighted majority, 5 quartets	6m 41s	15m 52s	34m
FastTree (NJ phase only)	13m 52s	41m 15s	116m

two smaller alignments created by randomly sampling 20000 and 40000 sequences from the large alignment. All running times were measured on a standard desktop computer with an AMD 7750 Dual Core 2712MHz processor and 4 GB RAM. In all cases, our algorithm runs faster, with the relative speedup increasing with the size of the data set; Table 3.4 shows the running times. Table 3.5 shows the accuracies obtained in these runs. In contrast to the previous experiment, our algorithm achieves higher accuracy than FastTree.

On the data set with 20000 sequences, the QuickTree implementation of Neighbour Joining [89] took over 360 minutes to complete and produced a tree almost identical to the one produced by FastTree, with over 99% of splits agreeing between the two trees. We were not able to run Neighbour Joining on larger data sets due to QuickTree's memory consumption.

To investigate the differing relative performance of our algorithm and FastTree on different data sets, we generated more data sets under varying conditions. When designing this series of experiments, we wanted to test two hypotheses: first, that our algorithm is more accurate than FastTree when sequences of sufficient length are available; second, that our algorithm is more accurate when the branch lengths are short. Both of these are consistent with the differences between the COG data sets used in the first experiment and the 16S data sets used in the second experiment.

We simulated 10 trees on 20000 taxa from the pure-birth process. We then multiplied the length of each branch by a factor chosen uniformly at random from interval $[0.5, 2]$ to deviate the trees from ultrametricity. We then scaled the branch lengths by several constant factors. For each choice of tree and scaling factor, we generated nucleotide alignments whose length varied between 250 and 4000 positions. The sequences were simulated from the Jukes-Cantor model, with variable evolutionary rates across sites drawn from the exponential distribution. No indels were introduced in the simulation. We used *rose* [156] to generate the sequences and *r8s* [143] to generate the trees. This experimental methodology follows the work of Liu *et al.* [107].

We see that the accuracy of both algorithms improves as more sequence data is available. However, the accuracy of the random walk algorithm improves faster than that of FastTree as sequences get longer. For trees with shorter branches, the advantage of the random walk algorithm is visible for shorter sequence lengths. Even for long sequences, the advantage of the random walk algorithm over FastTree was not as large as on the simulated 16S data set. Figure 3.6 shows the performance of the two algorithms in various conditions. The average taxon coverage for random walk trees was 97.3%, with only three experimental settings yielding coverage below 95%. Unsurprisingly, coverage was lower for short sequences and long branches.

The impact of local search

FastTree uses local search to improve the accuracy of the Neighbour Joining tree. The improvement can be quite dramatic: for the huge.1 alignment of 20,000 sequences, local search improved the RF accuracy from 62% to 96%.

We ran FastTree's local search procedure starting from trees obtained from our algorithm and the Neighbour Joining phase of FastTree for the tree sets used in the previous experiment. We inserted the remaining taxa into the random walk trees by running the random walk followed by a simple descent down the search tree. We then ran local search on the resultant trees. The results are shown in Figure 2. Despite huge differences in the accuracy of starting trees for some data sets, the trees after improvement by local search have very similar accuracies. This suggests that our algorithm can be combined with the local search phase of FastTree to produce trees that are very similar to FastTree, in less runtime. We also obtained similar results for the huge.1 data set (see Table 3.5).

The quality of the starting tree did not have much impact on the running time of the local search procedure. For all datasets we investigated, the differences between the two runs of local search were less than 15% of the overall running time. Table 3.6 shows the running times for the huge.1 data set.

Trees produced by running local search on incomplete random walk trees (without forcing) had very similar accuracies compared to those obtained from the full taxa set; for all data sets, the difference was at most 3%. In most cases, the differences in accuracy were less than 1%. Before local search, the loss of accuracy was more substantial, with the RF accuracy dropping by up to 10% when the remaining taxa were inserted into the random walk tree. In most cases, this negative effect is much more minor, averaging less than 2%.

Figure 3.6: The performance of the random walk algorithm and FastTree as a function of the length of the sequences. The four graphs represent the performance on 10 tree topologies with branch lengths scaled by constant factors $25, 50, 100$, and 200. In all cases, the random walk algorithm compares increasingly favourably with FastTree as the sequence length increases. After applying local search, the differences between the average accuracies of the two methods are less than 1% for all the settings except the shortest sequences in the data set scaled by 200, where trees obtained from FastTree are 3.8% more accurate. The average taxon coverage for random walk trees was 97.3%, with only three experimental settings yielding coverage below 95%. Missing taxa were inserted into random walk trees before applying local search.

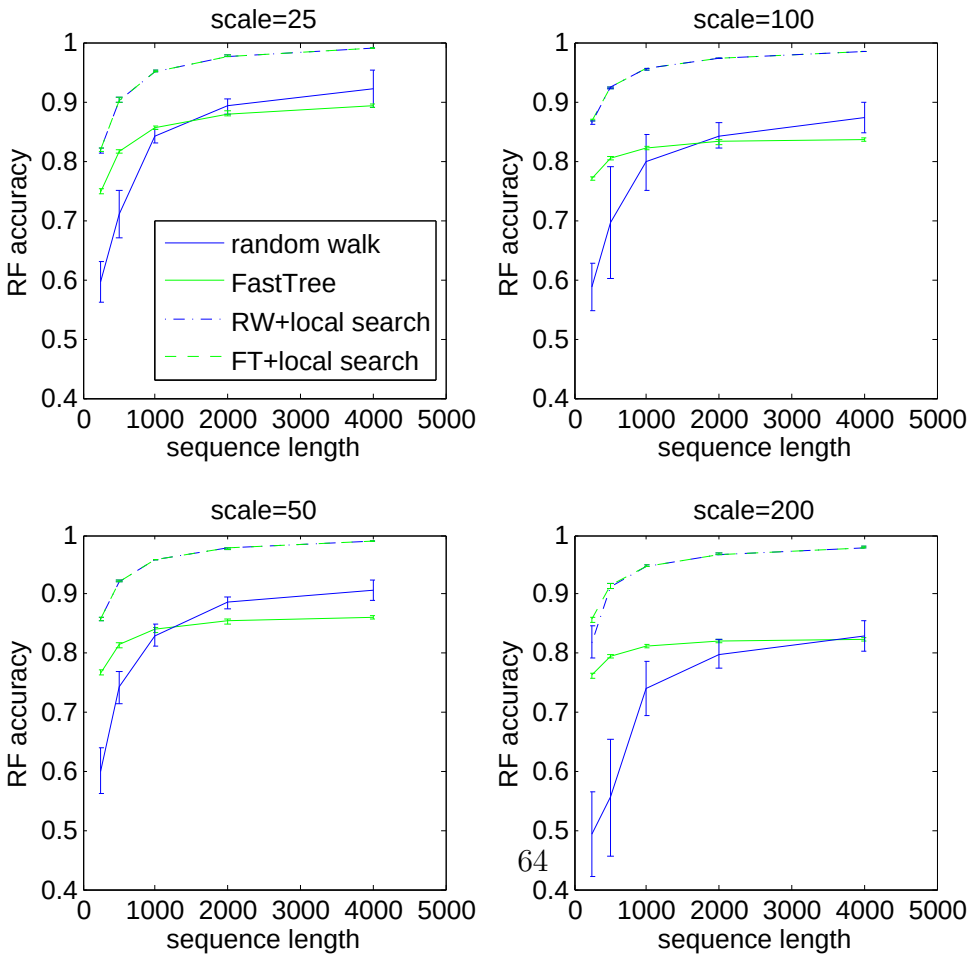

Table 3.5: Robinson-Foulds accuracies of FastTree and the random walk algorithm for the huge.1 data set of 1287 columns [134]. The figures for the random walk algorithm represent the average accuracy over 10 runs of the algorithm, together with empirical standard deviations. We used a confidence threshold, with two additional rounds of insertions. The average taxon coverage for weighted majority was 98.6, 98.6, and 98.0 per cent for the 20,000, 40,000, and 78,132 taxa alignments, respectively. After applying local search, the variance between the runs of the random walk algorithm is negligible.

	# of sequences		
	20,000	40,000	78,132
weighted majority, 5 quartets	80.8 ±1.1	78.9 ±1.4	80.8 ±1.1
weighted majority, 5 quartets, all taxa forced	79.9 ±1.1	77.8 ±1.4	75.3 ±1.6
weighted majority, 5 quartets+local search	96.1	94.4	92.8
weighted maj., 5 quartets, all forced+local search	95.8	93.8	92.0
FastTree (NJ phase only)	62.9	58.1	52.2
FastTree + local search	95.8	93.8	92.0

Table 3.6: Running times of the local search procedure of FastTree applied to trees produced by our algorithm and the Neighbour Joining phase of FastTree on the huge.1 nucleotide data set of 1287 columns. Total runtimes, including the time required to produce the initial tree, are shown in brackets

	# of sequences		
	20,000	40,000	78,132
weighted maj.,5 quartets+local search	21m (27m)	41m (57m)	79m (113m)
FastTree + local search	21m (35m)	42m (84m)	80m (196m)

Table 3.7: The performance of the random walk algorithm as a supertree method. We generated 5 input trees by running the random walk five times independently on the COG840 alignment. We then ran the random walk algorithm with quartet queries evaluated by taking the induced quartet in each tree, and choosing the most common one. The guide tree was chosen as the subtree induced by 200 randomly chosen taxa on one of the 5 input trees.

	% taxa inserted	R-F Accuracy	Quartet Accuracy
5 input trees (average)	83	62	85
output tree	94	48	85
output tree with forcing	100	39	83

3.9 Aggregating information from many trees

Our random walk algorithm is not limited to quartets inferred from aligned sequences. It can also be used with other types of quartet queries, *e.g.* evaluating how many times a given quartet topology appears in a collection of phylogenetic trees. We investigated how our algorithm can be used to aggregate conflicting information from many phylogenetic analyses to create a single, more accurate tree. Such algorithms are known as *supertree methods* or *consensus methods*. Bryant [25] gives a slightly outdated survey of consensus methods.

In the first experiment, we ran the random walk algorithm five times on the COG840 data set, generating five distinct trees. We then ran the algorithm for the sixth time, this time answering quartet queries based on the most common quartet topology found in the five trees constructed in the previous phase. The quartets were weighted by the square root of the proportion of the input trees that contained the majority quartet.

The results are shown in Table 3.7. Somewhat surprisingly, aggregating several inferred trees did not improve the accuracy, but the coverage was increased from 83% to 94%. The quartet accuracy remained roughly the same, while the Robinson-Foulds accuracy dropped. The quartet accuracy remained the same when the remaining taxa were forced into the tree by running the random walk and inserting the taxon at the first leaf that was reached. However, the Robinson-Foulds accuracy dropped further as a result of inserting those remaining taxa.

One problem with using our algorithm for amalgamating trees on different taxa sets is that many quartets needed by our algorithm are not contained in all the input trees. For example, if all the input trees have coverage of 83%, the probability that an input tree contains a random quartet is roughly $0.83^4 \approx 0.5$, meaning that most quartets will be

decided based on just two or three votes our of the five trees being summarized, and some of them might not appear in any of the input trees, which means that another quartet query has to be asked. This problem will be more challenging for datasets where input trees have even lower coverage.

In the second experiment, we generated 100 bootstrap samples from the COG840 alignment and ran Neighbour Joining 100 times. We then ran the random walk algorithm with the majority vote quartet oracle, as in the previous experiment. The resulting tree contained over 98% of the taxa and had quartet accuracy of 94%, compared to the average accuracy of 88% for the input trees. The Robinson Foulds acccuracy dropped from 62% to 50%.

These experiments show that our random walk algorithm may be potentially useful in aggregating information from multiple trees. However, the trees obtained from the algorithm tend to sacrifice Robinson-Foulds accuracy to increase quartet accuracy or coverage, which is a substantial drawback. If the input trees have low coverage, many quartets asked by the algorithm do not appear in any of the trees, which may harm speed and accuracy. We leave the solution to these two problems for future work.

3.10 Conclusions

We have presented a fast algorithm that is guaranteed to reconstruct the correct phylogeny with high probability under an error model where each quartet query errs with a fixed probability, independently of others. The algorithm runs in $O(n \log n)$ time, which is the lower bound for any phylogeny reconstruction algorithm. Our algorithm is based on a data structure that stores the information of the tree in a rapidly searchable structure that is balanced with high probability for a random insertion order of the taxa, regardless of the phylogeny; we also show how to implement the structure in the case where the insertion order is fixed, using random priorities to ensure balancedness.

We have also presented our efforts to move our algorithm from being a theoretical result to a practical algorithm for inferring trees. A variety of sensible heuristics, including weighting quartets by our confidence in them, using multiple quartet queries per insertion, and using multiple rounds of insertion, can increase both coverage and accuracy substantially over our original implementation. If we force all taxa into the tree and use local search, accuracy is comparable to existing programs. As a result, our algorithm is close to being competitive with Neighbour Joining, while being much faster. For very large data sets, it may be suggested as one of few heuristics for the task.

Despite the progress, it appears that some of the problems we have encountered cannot be entirely eliminated by the heuristics. This is in line with the result by King *et al.* [101], which suggests that no purely distance-based algorithm can be accurate in $o(n^2/\log\log n)$ runtime. The low accuracy of quartets at the top of the search tree is one such problem. While we can evaluate the reliability of the quartets by looking at edge length estimates, the number of *r*eliable quartets remains low, thus leaving the algorithm with little phylogenetic signal. In order to overcome this problem, the choice of the representatives at each node query would have to be biased to yield reliable quartets. This would require the algorithm to know, in advance, which taxa are likely to be closely related to the taxon being inserted. In the next chapter, we describe an algorithm that uses hashing to accomplish essentially this goal. While the algorithm operates on a vastly different principle, in particular including sequence information rather than just distance estimates, it uses quartet queries that are likely to be much more reliable than the ones used by QTree.

Chapter 4

LSHTree: a principled fast algorithm

4.1 Introduction

So far, we have seen two sub-quadratic algorithms for phylogeny reconstruction: we discussed FastTree, by Price *et al.* [134, 135], in Section 2.2.2, and presented our algorithm QTree in the previous chapter. While both algorithms produce fairly accurate phylogenies for a wide range of outputs, their performance cannot be guaranteed in general. One worrying aspect of both algorithms is that they will be forced to rely on distance estimates between distant taxa. Such estimates are known to be unreliable, as we have seen in Section 2.4. For a caterpillar tree whose branch lengths are between f and g, FastTree will have to compare sequences that are $\Omega(\sqrt{n})$ edges apart, leading it to require exponential-length sequences in the setting of Erdős *et al.* [62]. At the beginning of the insertion process for each taxon, QTree will compare sequences separated by $\Omega(n)$ edges, which again suggests that the algorithm requires exponential-length sequences for quartet queries to be sufficiently accurate (see Section 2.4).

While this does not necessarily mean that these algorithms will be inaccurate in practice, it suggests that in some scenarios the quality of trees produced by FastTree and QTree will be poor even when phylogenetic 'signal' is present in the data. Moreover, it is conceivable that an algorithm that takes into account the Markovian nature of the evolutionary process could produce more accurate trees than either QTree or FastTree.

Here, we give an algorithm to correctly reconstruct, with high probability, phylogenetic trees that come from a Markov model of evolution, in sub-quadratic time using sequences of $O(\log n)$ length. Recall that King *et al.* [101] proved an $\Omega(n^2 \frac{\log\log n}{\log n})$ lower bound on

the running time of all distance-based phylogeny reconstruction algorithms using such short sequences. Their result suggests that any accurate distance-based algorithm cannot be substantially faster than $\Theta(n^2)$. Our result circumvents this bound by directly using information in the sequences, as well as distance estimates. This leads to a runtime of $O(n^{1+\gamma(g)} \log^2 n)$, where the exponent of the polynomial is always less than 2, provided that the edges in the generating tree are not too long.

The algorithm is based on three ideas. First, we use locality-sensitive hashing [91] to find sequences that are near-neighbours in the tree, in sublinear time. This hashing is a first step in choosing which two nodes should be joined in the tree we incrementally build. Second, we use ancestral sequence reconstruction [64, 125] to reliably approximate the sequences found at internal tree nodes. Finally, we use reliable estimates of distance, to identify exactly the correct join at each step; this step involves some hoary computation, as we must ensure that inferred sequences are independent estimates. We start with a forest with each taxon in its own tree, and perform this joining step until only one tree remains in the forest. The overall runtime is sub-quadratic, since each join step time is sublinear. Specifically, under Cavender-Farris models, if g is an upper bound on the length of any edge in T, and $g < \frac{1}{4} \ln 2$, then we show that we can do the locality-sensitive hashing, which is the runtime-determining step, at each step in $O(n^{\gamma(g)} \log^2 n)$ time, where $\gamma(g)$ is always less than 1; the overall runtime is thus $O(n^{1+\gamma(g)} \log^2 n)$.

In experiments, our algorithm usually outperforms both FastTree and QTree in terms of accuracy. An important exception to that is when branch lengths are very long, which makes it hard to estimate internal node sequences. The lower accuracy for trees with long branch lengths is consistent with our theoretical results. The algorithm is faster than FastTree, while being only slightly slower than QTree. We believe that both speed and accuracy could be improved, and we outline the avenues for future research in the final section of this chapter.

The key results in this chapter have been published in a conference publication [19]. Here, we present the results in more detail, including several minor corrections to some of the proofs.

4.2 Preliminaries

Our algorithm builds on two lines of research: a family of algorithms known as short quartet methods, first introduced by Erdős *et al.*, and the more recent line of algorithms based on ancestral state reconstruction [125, 43, 120]. Following Erdős *et al.*, we assume

the Cavender-Farris model of binary character states, evolving according to a continuous-time Markov process. Each edge e is labelled with length $\ell(e)$, and the probability that the ends of e have different states is $p(e) = \frac{1}{2}[1 - \exp(-2\ell(e))]$. If two sequences differ in $\hat{p}$ fraction of the sites, the maximum likelihood estimator of the distance between them is $\hat{d} = -\frac{1}{2}\log(1 - 2\hat{p})$.

Following Daskalakis *et al.* [43], we assume there exist constants f and g such that for each edge e in the phylogeny, $f < \ell(e) < g < \frac{1}{4}\ln 2$. This gives a minimum length for each edge, so that nodes k edges apart have $\Omega(k)$ evolutionary distance, and also gives each edge state change probability less than $1/2 - \sqrt{1/8}$, which guarantees a bounded probability of error when reconstructing ancestral sequences [43, 64], as discussed in Section 2.4. We also assume that all edge lengths are multiples of some constant Δ, an assumption known as the Δ-*branch model* in previous work [43, 120]. With this assumption in place, a surprising fact arises: with sequences of length $O(\log n)$, we can exactly identify the tree distance between close nodes in the phylogeny [43, 120].

The following theorem follows immediately from Equations 2.3 and 2.4 and the Δ-branch assumption.

Theorem 7. *If d_{ab} is a multiple of Δ, we have*

$$\Pr[|d_{ab} - \hat{d}_{ab}| > \frac{\Delta}{2}] \leq 2\exp(-(1 - e^{\Delta})^2 e^{-4d_{ab}} m/2)$$

where m is the sequence length and d_{ab} is the true evolutionary distance.

In particular, if $m > (6\ln n + 2\ln 2)e^{4d_{ab}}/(1 - e^{\Delta})^2$, we can identify the correct distance with probability at least $1 - n^{-3}$.

This means we can exactly infer any distance below a constant times g, the upper bound on a single edge length, from sequences of length $\Omega(\log n)$. Consequently, all quartets whose diameter is bounded by a constant times g can be inferred correctly, as we will see in Section 4.2.2.

4.2.1 Locality-sensitive hashing

Our algorithm requires finding pairs of sequences within a specified small distance from each other, without having to compute all pairwise distances. Indyk and Motwani [91] solved this problem using a collection of randomized hash tables: enough hash tables are chosen so that close sequences likely collide in one of the tables, while keys are long enough

that distant sequences do not. This idea, known as *locality-sensitive hashing*, has been applied to many problems in bioinformatics, such as motif finding [29].

Specifically, Indyk and Motwani solve a related problem, the (r_1, r_2)-approximate Point Location in Equal Balls ((r_1, r_2)-PLEB):

Input: A set of sequences P in $\{0,1\}^k$, a query sequence q, and radii $r_1 < r_2$

Output: If there exists a sequence $p \in P$ within normalized Hamming distance r_1 from q, output "yes" and a sequence within r_2 of q. If there is no sequence in P within normalized Hamming distance r_2 from q, output "no". Otherwise, output either "yes", with a sequence within r_2 of q, or "no".

Indyk and Motwani's solution constructs n^{r_1/r_2} hash tables, each keyed on $O(\log n)$ randomly chosen sequence positions. Given q, a point within distance r_1 of it has a constant probability of colliding with q in at least one hash table, while points further than r_2 from q have $O(1/n)$ probability of colliding. After inspecting a constant number of collisions with q, we can find, with constant probability, a point whose distance from q is at most r_2. This is because the expected number of collisions between q and points further than r_2 is constant. If we boost the algorithm by running $O(\alpha \log n)$ times independently, the success probability is at least $1 - n^{-\alpha}$, for any choice of α. For more details, see Indyk and Motwani [91]. Overall, finding an (r_1, r_2)-approximate near neighbour for a query point q with high probability takes $O(n^{r_1/r_2} \log n)$ hash table lookups, each on a key of length $O(\log n)$ bits. The overall runtime is thus $O(n^{r_1/r_2} \log^2 n)$.

Solving the exact version of the PLEB problem, where $r_1 = r_2$, by hashing is challenging for instances where a large number of points are at distance $r_1 + \epsilon$ from q, for very small ϵ. Such points will generate collisions with q roughly as often as points within r_1 of q, making it hard to find the closest point to the query among many points with similar distances. This was the original motivation for the approximate PLEB problem [91]. In our setting, this is not a concern since the lower bound f on edge lengths implies that each sequence will have only a small number of highly similar sequences in the tree. Consequently, we can find all sequences within distance exactly r from a given query: we choose r_2 to be small enough that there are at most $O(\log n)$ within r_2 distance from q, so we can examine all of them and still have fast runtimes. We choose r_2 to be $1/2 - 1/2(\exp(-2cf \log\log n))$ the relative Hamming distance corresponding to all sequences within evolutionary distance $cf \log\log n$, for a constant c that takes into account the errors arising from reconstructing internal sequences of the tree (see Section 4.2.3). In $\log\log n$ edges, we can reach $O(\log n)$ nodes. The distance r_2 converges to $1/2$ as n grows, so in the limit, the number of hash tables grows to n^{2r_1}.

Finding all neighbours within r_1 normalized Hamming distance thus takes $O(n^{2r_1+\epsilon} \log^2 n)$

time with high probability, where $\epsilon \to 0$ as n increases: we use $O(n^{2r_1+\epsilon} \log n)$ hash tables, each of which requires $O(\log n)$ time to examine, and we take $O(\log^2 n)$ time examining all hash table hits.

We use the hashing algorithm to find sequences within evolutionary distance d, implicitly relying on the simple correspondence between Hamming and evolutionary distances under the Cavender-Farris model: our procedure $FindAllClose(q, d)$ finds all sequences within evolutionary distance d of q with probability $1 - o(1/n^2)$, and so with high probability makes no errors during the course of running our entire algorithm, as it needs to be run $O(n)$ times.

4.2.2 Four-point method

To identify the correct place to join two trees, we will use the four-point method, as described in Section 2.4.2. We have seen that the four-point method is guaranteed to infer the correct topology when the sequences are long enough. Under the Δ-branch model, we can ensure that the four-point method also infers correct branch lengths, since all distances are inferred correctly. The following theorem is a stronger version of the result in Equation 2.19.

Theorem 8. *Let f and g be the upper and lower bounds on the edge length in a quartet tree. Then there exists a constant $c_2(g, \Delta)$ such that we can reconstruct the lengths of the edges of the quartet exactly using sequences of length $c_2(g, \Delta) \log n$ with probability at least $1 - \frac{6}{n^3}$.*

Proof. The claim follows from Theorem 7, and the observation that all the distances are bounded by $3g$. □

4.2.3 Ancestral states

When all edge lengths are below the phase transition threshold, we can correctly infer the ancestral state of a character at any internal node of the tree with probability greater than $\frac{1}{2} + \beta$ for some constant β [64, 125, 43], as we have seen in Section 2.4.6. Our algorithm starts with a forest where each taxon is its own tree. Then, it repeatedly identifies nodes which should be near-neighbours in the tree, joins them together, and infers new ancestral node sequences, until we have only a single tree.

We use Felsenstein's maximum likelihood algorithm [68] to infer ancestral sequences. The algorithm computes the posterior distribution of the character at each position of the sequence at an internal node based on previously computed posterior distributions at its children; see Section 2.2.1. For constant-sized alphabets, it takes constant time per internal node per position. We pick the symbol of largest posterior probability. If the exact edge lengths are known, this algorithm has optimal probability of correctly reconstructing the ancestral state, among all possible algorithms.

Earlier algorithms [43, 120] did not require the knowledge of β. For our purposes, we need to be able to bound the error probability in ancestral sequence reconstruction to know how many hash tables are required for near neighbour search. The following result by Evans *et al.* [64] provides a bound on β.

Theorem 9. *Let T be a phylogenetic tree with edge mutation probabilities p bounded by $p_g < 1/2 - \sqrt{1/8}$. Assign to each edge e a resistance $(1-2p)^{-2|e|}$, where $|e|$ is the number of edges on the path from root to e, including e itself. The probability p_{err} of incorrectly reconstructing the root state is bounded by $p_{err} < 1/2 - 1/(1+\mathcal{R}_{\mathit{eff}})$, where $\mathcal{R}_{\mathit{eff}}$ is the effective resistance between the root and the leaves of T.*

This bound is quite loose. For this reason, we will use a better bound, originally developed for the simpler maximum parsimony algorithm by Fitch [71]. For more on Fitch parsimony, see Felsenstein [68]. The following bound is sharper for $p_q < 0.118$.

Theorem 10. *Let T be a phylogenetic tree where all mutation probabilities across edges are equal to $p_g < 1/8$. The probability p_{err} of incorrectly reconstructing the root state using Fitch parsimony is bounded by*

$$p_{err} < \frac{1}{2} - \frac{\sqrt{(1-4p_g)(1-8p_g)}}{2(1-2p_g)^2} < 1 - 4p_g \tag{4.1}$$

The original result, proved by Steel [154], stated that the bound was also true for variable edge lengths. Recently, Zhang *et al.* [174] noticed that this is not true. Below, we will show that the bound 4.1 applies to the maximum likelihood algorithm, even for variable edge lengths, as long as they are bounded by p_g.

For constant length edges, the result follows from the optimality of maximum likelihood. If we shrink an edge in tree T, creating a tree T', the mutual information between the leaves and the root of T' is greater than the mutual information between the leaves and the root of T. Hence the probability of reconstructing the root of T' can only increase. Applying

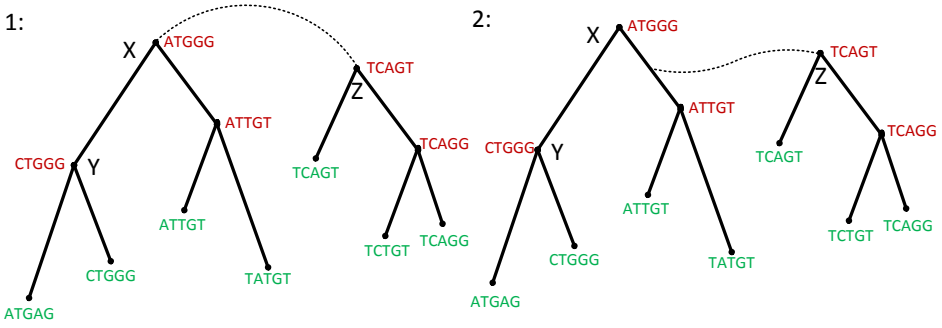

Figure 4.1: Independence relationships between reconstructed sequences: in the left tree, sequences X and Z are error-independent. In the right tree, X and Z are not error-independent, but Y and Z are error-independent. The algorithm was run on subtrees drawn in solid lines. The dashed line corresponds to an edge of the true tree that has not yet been reconstructed. This picture also appears in Chapter 2.

this argument to all edges except the longest, we obtain that the bound holds for trees of variable edge lengths when the maximum likelihood algorithm is used, as long as the edge lengths are known exactly.

The bound on p_{err} is important in our algorithm because we will use locality-sensitive hashing on these sequences: it is one factor determining the number of hash tables required. Tighter bounds on p_{err} will give faster algorithms with the same accuracy guarantee. Let g_{err} be the distance corresponding to a mutation probability of p_{err}.

Suppose that we reconstruct ancestral sequences for all internal nodes of subtrees T_1, T_2 of T, rooted at ρ_1, ρ_2, respectively. Moreover, suppose that the path connecting T_1 and T_2 has ends ρ_1, ρ_2 (see Figure 4.1). By the Markov property, reconstructing a character of σ_{ρ_1} correctly is independent of the true character at σ_{ρ_2}. We call such two sequences *error-independent*. The distance estimate $\hat{d}(\sigma_{\rho_1}, \sigma_{\rho_2})$ will converge to $g + g_1 + g_2$, where g_1 and g_2 are edge lengths corresponding to the probabilities p_1 and p_2 of incorrectly reconstructing states in the two sequences, and we have $g_i = \frac{1}{2}\log(1-2p_i)$. When comparing independently reconstructed sequences, we can effectively treat these errors in the reconstructed sequences as "extra edges" [43], whose length can be bounded using Theorems 9 and 10. This observation has been used extensively in many theoretical algorithms [43, 125, 120].

Theorem 9 combined with Theorem 8 and the Markov property enable us to estimate internal branch lengths from reconstructed ancestral sequences, and also correct quartets.

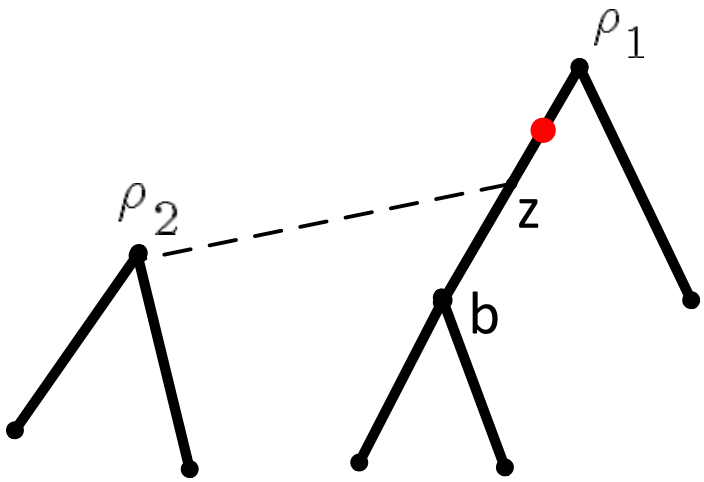

Figure 4.2: The sequences at ρ_1 and ρ_2 are not error-independent. If a mutation occurs between ρ_1 and z, it increases the both **a)** the likelihood that the character at ρ_1 is reconstructed incorrectly, and **b)** the likelihood that the true character at ρ_1 is different from the character at ρ_2. Consequently, the reconstruction error at ρ_1 is not independent of mutations between ρ_1 and ρ_2.

Theorem 11. *Let i, j, k, l be ancestral sequences reconstructed by maximum likelihood from four disjoint subtrees, and such that no path between any two of them in the true phylogeny shares an edge with any of the subtrees. Let f' and g' be the upper and lower bounds on the edge lengths in the quartet $ijkl$. Then there exists a constant $c(f', g', \Delta)$ such that given reconstructed ancestral sequences of length $c \log n$, we can estimate the correct topology and the length of the middle edge exactly, with probability at least $1 - 6n^{-3}$.*

On the other hand, if the path between T_1 and T_2 connects to T_1 below ρ_1, reconstructing a character at σ_{ρ_1} is not independent of the true character at ρ_2. The occurrence of a mutation below ρ_1 increases the probability of incorrect reconstruction at σ_{ρ_1}. If the mutation occurs on the path between ρ_1 and ρ_2, it also increases the probability that the true character at ρ_2 is different from the character at ρ_2. Thus, reconstruction error cannot be modelled as additional mutations, since they are not independent of the mutations between true ancestral sequences - see Figure 4.2. To ensure the correctness of our algorithm, we need a way to detect lack of error-independence without knowing the complete tree.

4.3 The algorithm

The algorithm starts with a forest F of n trees, each with one taxon. It progressively merges trees into larger subtrees of the true tree, finding trees that are quite close using locality-sensitive hashing, identifying where they should be joined, and inferring ancestral sequences. This basic structure is complicated by the error-independence requirement in Theorem 11.

Before we explain the principle behind the algorithm, we need one more definition.

Definition 6. *A core region T_{cr} is any maximal subtree of any tree T' in F satisfying the following conditions:*

1. *Every leaf in T_{cr} is a leaf in T'*
2. *After suppressing all vertices of degree 2 in T_{cr}, every edge in T_{cr} has length at most g*

A *non-core region* is any maximal subtree of any tree T' in F that does not overlap any core region; see Figure 4.3.

Our algorithm maintains four invariants while successively merging trees:

1. Every tree in F is a subtree of the true tree T.
2. No two trees in F overlap as subtrees of T.
3. For each non-core region T' in F, all of its edges except at most one have length at most g. The remaining edge has length at most $2g$. We call it the *long edge* of T', if it exists.
4. The length of every edge in every tree in F is reconstructed correctly.

Invariants 1 and 2 are the same as in the work of Mihaescu *et al.* [120]. Invariant 3 is a modified version of invariant $I2$ of Mihaescu *et al.*. The fourth invariant is maintained using Theorem 11. The invariants, together with routine $CheckErrorIndependence$, ensure the theorem's preconditions are satisfied.

Algorithm 9 presents a simplified version of our algorithm. At each iteration, our algorithm first finds a pair of sequences x, y that are close to each other, but belong to

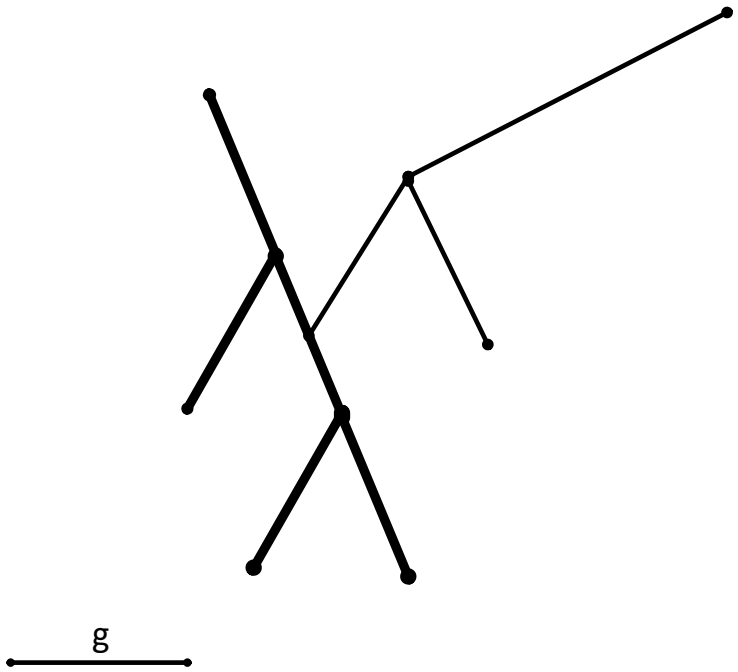

Figure 4.3: A tree consisting of a core region (thick lines) and a non-core region (thin lines). The non-core region contains an edge of length $2g$. All other edges have length g.

different trees $T(x)$ and $T(y)$ in F. The algorithm then searches the neighbourhood of x and y, and finds edges (i, j) and (k, l) that should be connected to merge $T(x)$ and $T(y)$. It then checks whether connecting (i, j) and (k, l) will violate any of the invariants. If the answer is yes, the pair (x, y) is either discarded, or stored on the waiting list if there is a possibility that $T(x)$ and $T(y)$ might be joined later.

There are three scenarios when $T(x)$ and $T(y)$ cannot be merged, despite being close enough to each other in the true tree. We discuss them briefly below.

- *Lack of error-independent sequences* - in this scenario, $T(x)$ and $T(y)$ do not contain error-independent sequences that are needed to identify the correct join, and to estimate the length of the new edge that would connect $T(x)$ and $T(y)$. In this case, the hit (x, y) is put on the waiting list. After more ancestral sequences are reconstructed, it may be possible to reevaluate the neighbourhoods of x and y. This can happen when x or y become part of a core region (see Section 4.3.3).

- *Overlap* - merging $T(x)$ and $T(y)$ would create an edge that would overlap some other tree T' in F. In this case, x and y are discarded, as $T(x)$ and $T(y)$ should not be merged directly. Instead, the algorithm may try to merge $T(x)$ or $T(y)$ with T' so that no overlaps are created.

- *Multiple long edges* - merging $T(x)$ and $T(y)$ would create a tree with multiple long edges in the same non-core region, violating Invariant 3. In this case, the hit (x, y) is put on the waiting list, with the hope that it might be used after the long edges are broken into shorter edges.

In what follows, we will expand the details of this algorithm, focusing on ensuring invariants 3 and 4. We will assume the existence of three procedures: $FindAllClose(q, d)$ uses locality-sensitive hashing to identify all sequences within relative Hamming distance d of q. $Quartet(a, b, c, d)$ uses the four-point method to identify the correct topology of the quartet $abcd$, given the sequences at nodes a, b, c, and d. And $MiddleEdge(ab|cd)$ computes the length of the middle edge in the quartet $ab|cd$ by using the four-point method and then rounding to the nearest multiple of Δ. Assuming the preconditions to Theorem 9, these procedures, when called by the algorithm, work with total failure probability at most $o(1)$, across all calls to them.

Most of the subroutines are presented for the case where all their arguments are internal nodes. The cases where some nodes are leaves are analogous; we omit them for simplicity.

We will often treat trees in F with long edges as rooted, with the root located in the middle of the long edge.

Algorithm 9 SimplifiedReconstruct($\{\sigma\}, f, g$)

Start with a forest with each node in its own tree.
Use locality-sensitive hashing to find all pairs of sequences whose distance is less than $3g + 2g_{err}$. Put them in a queue, *DistQueue*.
while the forest has more than one tree **do**
  Find two sufficiently close nodes x and y in *DistQueue* that are not currently in the same tree of F.
  Identify the nearby edge (i, j) to x and (k, l) to y that should be joined to connect the trees containing x and y in the forest
  Check whether connecting (i, j) and (k, l) does not violate invariants 1 or 3. If it does, put (x, y) on the waiting list and skip to the next loop iteration.
  Create two new nodes, a and b in the middle of (i, j) and (k, l), and join a and b together with a new edge.
  If the (a, b) violates Invariant 2, delete a, b and the edge between them, and skip to the next loop iteration.
  Estimate the lengths of all five edges in the quartet $ijkl$.
  Reconstruct the ancestral sequences at a and at b.
  Find all sequences within $3g + 2g_{err}$ of the newly inferred sequences, and add these distances to *DistQueue*.
  For each pair (x, y) on the waiting list within distance $3g + 2g_{err}$ from (a, b), put (x, y) in *DistQueue*.
end while

4.3.1 Independent inferences

If reconstructed sequences in quartet queries are not error-independent, the quartet middle edge length estimates and the inferred topology might be incorrect, as we have explained in Section 4.2.3. This could lead the algorithm to join the wrong pair of edges. To prevent this, we introduce the routine *CheckErrorIndependence* to detect situations where lack of error-independence may lead to an incorrect join.

The order in which ancestral sequences are reconstructed defines a partial order of the nodes in F. We call it the *reconstruction order*. Suppose we are trying to estimate the middle edge of quartet $ab|xy$. If a is on the path from b to x in T, then a cannot be error-independent from both b and x. However, at least one of the children of a is not on the path from b to x. Let a_2 be such a child of b. Then a_2 is error-independent from both b and x; see Figure 4.4. *CheckErrorIndependence* uses this observation to detect lack of error-independence between the reconstructed sequences in the quartet. The core idea is that if the four sequences are error-independent, substituting some of them with any of their children should give the same edge length estimate.

Algorithm 10 CheckErrorIndependence(x, y, a, b)

Require: a and b are error-independent, x and y are error-independent, $T(a)$ and $T(x)$ do not overlap
  for all $z \in \{a, b, x, y\}$ **do**
    Let z_1, z_2 be the children of z in reconstruction order(if they exist)
  end for
  $d \leftarrow MiddleEdge(xy|ab)$
  for $i, j \in \{1, 2\}$ **do**
    $d_{a_i b_j} \leftarrow MiddleEdge(xy|a_i b_j)$
    $d_{x_i y_j} \leftarrow MiddleEdge(x_i y_j|ab)$
  end for
  if any of the $d_{a_i b_j}$ or $d_{x_i y_j}$ differs from d **then**
    return false
  end if
  return true

Let $ME(xy|ab)$ denote the true length of the middle edge in the quartet $xy|ab$ in T. Before we prove the correctness of *CheckErrorIndependence*, we need two auxiliary lemmas. The following lemma is a direct consequence of Lemma 5.4 by Mihaescu *et al.* [120].

Lemma 7. *Let a and b be not error-independent and let g_a and g_b be distances corresponding to the probabilities p_a, p_b of incorrectly reconstructing characters at a and b. Then $E[\hat{d}(a,b)] < d(a,b) + g_a + g_b$. In particular, $\hat{d}(a,b) < d(a,b) + g_a + g_b + \Delta/2$ with high probability.*

This means that lack of error-independence between reconstructed sequences will be biased towards *underestimating* the distance. Below, we show that $MiddleEdge$ will not overestimate middle edge lengths when using sequences that are not error-independent.

Lemma 8. *Let x be error-independent of y and let a be error-independent of b. If either x or y is not error-independent of a or b, then $MiddleEdge(ab|xy) \leq ME(ab|xy)$.*

Proof. The middle edge length estimate is computed as

$$MiddleEdge(xy|ab) = \left[(\hat{d})(x,a) + \hat{d}(y,b) - \hat{d}(x,y) - \hat{d}(a,b))/2\right]_\Delta, \tag{4.2}$$

where $[.]_\Delta$ denotes rounding to the nearest multiple of Δ. Since x and y are error-independent, we have

$$\hat{d}(x,y) > d(x,y) + g_x + g_y - \Delta/2,$$

with high probability. Analogously, we have

$$\hat{d}(a,b) > d(a,b) + g_a + g_b - \Delta/2$$

By Lemma 7, we know that

$$\hat{d}(x,a) < d(x,a) + g_x + g_a + \Delta/2$$

$$\hat{d}(y,b) < d(y,b) + g_y + g_b + \Delta/2$$

The result follows from Equation 4.2. □

We can now state the main result of this section.

Lemma 9. *$CheckErrorIndependence$ returns true if and only if the sequences at x, y, a, b are mutually error-independent.*

Proof. Let $T(a)$ and $T(x)$ be the subtrees that contain a and x, respectively. Furthermore, for any node z, let z_1, z_2 be its children in reconstruction order. Sequences a and b are independent and lie on the opposite sides of some edge e. If x and y join the tree at e, we

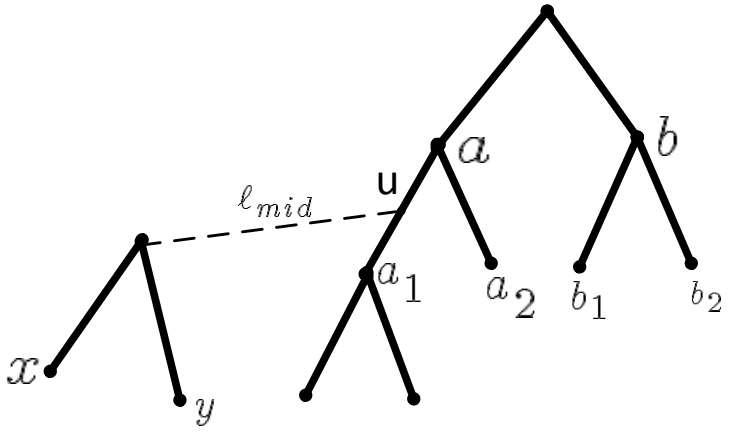

Figure 4.4: Detecting lack of error-independence between sequences. In this scenario, a is not error-independent of x and y, but both a_1 and a_2 are error-independent of both x and y. $MiddleEdge(a_1b_j|xy)$ will return ℓ_{mid}, while $MiddleEdge(a_2b_j|xy)$ will return $\ell_{mid} + d(a, u)$, causing $CheckErrorIndependence$ to return *false*.

have $ME(xy|ab) = ME(xy|a_ib_j)$ for $i, j \in \{1, 2\}$. If x, y, a, b are independent, all middle edge estimates are equal and correct with high probability.

Now suppose some pair of these sequences are not independent (say a and x). Without loss of generality, $T(x)$ joins $T(a)$ at some edge in the subtree of $T(a)$ consisting of all nodes from which we reconstruct the sequence at a. One of the sequences a_1, a_2 is then independent of x. Without loss of generality, suppose it is a_2. We have to consider two cases:

Case 1: The path from $T(x)$ to $T(a)$ joins $T(a)$ between a and a_1. In this case, both a_1 and a_2 are error-independent of x and y. If ℓ_{mid} is the length of the path between $T(x)$ and $T(a)$, $MiddleEdge(a_1b_j|xy)$ will return ℓ_{mid}, while $MiddleEdge(a_2b_j|xy)$ will return $\ell_{mid} + d(u, a)$, where u is the point where the path joins . Thus, $CheckErrorIndependence$ will return *false*.

Case 2: The path from $T(x)$ to $T(a)$ joins $T(a)$ below a_1. In this case, $MiddleEdge(a_2b_j|xy)$ will return $\ell_{mid} + d(a, a_1) + d(a_1, u)$. The sequence at a_1 is not error-independent of x and y, but Lemma 8 implies that $MiddleEdge(a_1b_j|xy)$ will return at most $\ell_{mid} + d(a_1, u)$. Thus, $CheckErrorIndependence$ will return *false*; see Figure 4.5.

□

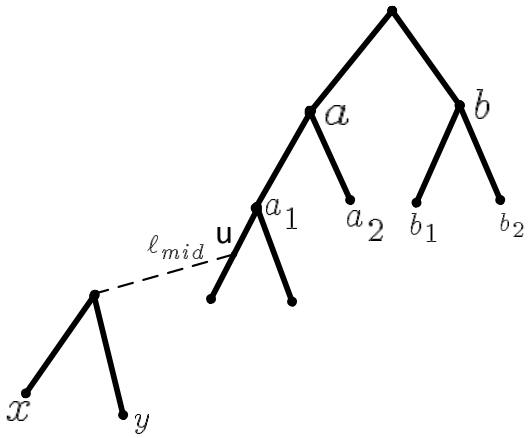

Figure 4.5: Detecting lack of error-independence between sequences. Here, a and a_1 is not error-independent of x and y, while a_2 is error-independent of both. $MiddleEdge(a_2b_j|xy)$ will return $\ell_{mid} + d(a, a_1) + d(a_1, u)$, while $MiddleEdge(a_1b_j|xy)$ will return at most $\ell_{mid} + d(a_1, u)$, causing $CheckErrorIndependence$ to return *false*.

Corollary 3. *$CheckErrorIndependence$ returns true if and only if the path between edges (a, b) and (x, y) in T does not overlap any other edges in $T(a)$ or $T(x)$.*

4.3.2 Detecting overlapping subtrees

To maintain Invariant 2, we need to prevent merging two trees if the new edge created between them would overlap some other tree in F. The procedure $CheckOverlaps$ takes the endpoints of the newly created edge as arguments, and detects overlapping trees. Let T_1 and T_2 be non-overlapping reconstructed trees in F. For any nodes a_1, b_1 in T_1 and a_2, b_2 in T_2, the correct topology of the quartet is $a_1b_1|a_2b_2$. $CheckOverlaps$ finds all trees in F in the vicinity of the newly created edge, and reconstructs quartets to determine if any of these trees overlaps the edge.

The procedure $CheckOverlaps$ is detailed in Algorithm 11. We use $R(T(x))$ to indicate the set of sequences in $T(x)$ and $d_{T'}$ for the path metric associated with some tree T' in F; see Figure 4.6.

Lemma 10. *If the edge (x, y) overlaps some other edge in F and the sequences at x and y are independent, CheckOverlaps will return true. Otherwise, it will return false.*

Algorithm 11 CheckOverlaps(x,y)

Require: x and y are error-independent and have been merged into the same tree
  $S = (FindAllClose(x, 2g + 2g_{err}) \cup FindAllClose(y, 2g + 2g_{err})) - R(T(x))$
  for each sequence a in S **do**
    for each sequence b in $T(a)$ that is error-independent of a and such that $d_{T(a)}(a, b) < 2g$ **do**
      if $Quartet(a, b, x, y) \neq ab|xy$ and $CheckErrorIndependence(Quartet(a, b, x, y)) =$ *true* **then**
        return true
      end if
    end for
  end for
  return false

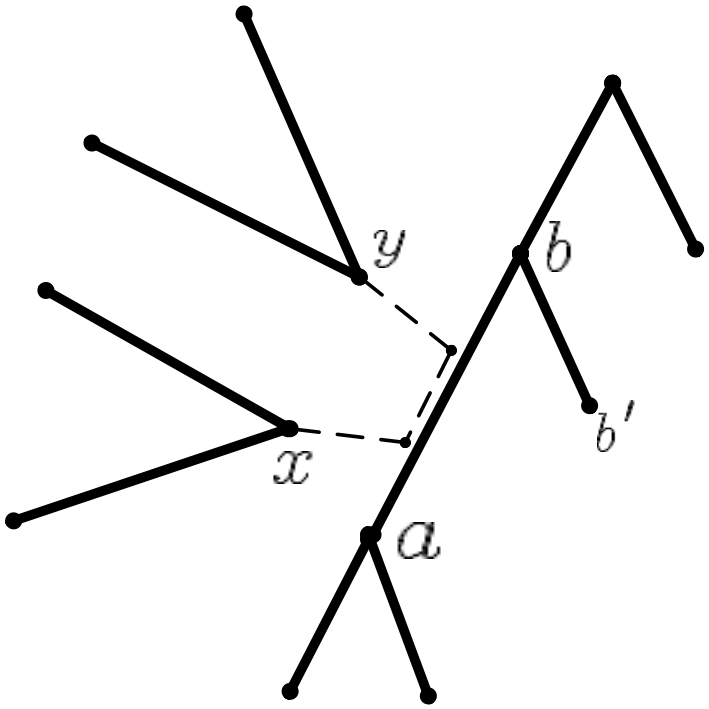

Figure 4.6: Detecting overlaps between a proposed edge and other trees in F. If the edge x, y overlaps some edge a, b

Proof. Suppose (x, y) overlaps with some other subtree T'. Let (a, b) be the lowest (in reconstruction order) of the edges in T' overlapping (x, y). We claim that the reconstructed sequence at either a or b is within distance $2g + 2g_{err}$ from the sequences at either x or y. Without loss of generality, we can assume that the true topology of the quartet on x, y, a, b is $ax|by$. Let $\ell_a, \ell_x, \ell_b, \ell_y$ be the lengths of the external edges of the quartet incident to a, x, b, y, respectively. Let ℓ_m be the length of the middle edge. Since both (a, b) and (x, y) have length at most $2g$, we have

$$d_T(a, b) = \ell_a + \ell_m + \ell_b \leq 2g$$

$$d_T(x, y) = \ell_x + \ell_m + \ell_y \leq 2g$$

Hence we have

$$\ell_a + \ell_x + \ell_b + \ell_y + 2\ell_m = d_T(a, x) + d_T(b, y) + 2\ell_m \leq 4g$$

It follows that $d_T(a, x) + d_T(b, y) < 4g$, so at least one of these distances has to be less than $2g$. The claim follows after acounting for reconstruction errors.

Moreover, we can show that either a or b must be error-independent of x, y *and* within distance $2g$ from either x or y. If (a, b) is a long edge, it is easy to see that x, y, a, b are mutually error-independent. If (a, b) is a short edge, a similar argument to the one above shows that either a or b must be error-independent and lie within $2g$ of x or y.

Suppose, without loss of generality, that node a lies within $2g$ of x and is error-independent of x and y. If node b is not error-independent of x and y, then b is the parent of a in the reconstruction order. In that case, the other child of b will be error-independent of x and y. Call that other child b'. The induced topology on x, y, a, b' is $ax|b'y$ and this quartet will pass the independence test.

On the other hand, if $T(x)$ does not overlap with any other subtree, all quartet queries that pass the independence test will return $ab|x'y$. □

Searching for b can be done by breadth-first search on $T(a)$ and takes constant time.

4.3.3 Three-way ancestral sequence reconstruction

In order to connect edges from different subtrees, we need to have independent sequence reconstructions at both ends of each of the edges we want to connect. This is needed to ensure that we can reconstruct the branch length of the connecting edge, and also of the edges that are created by creating two new internal nodes.

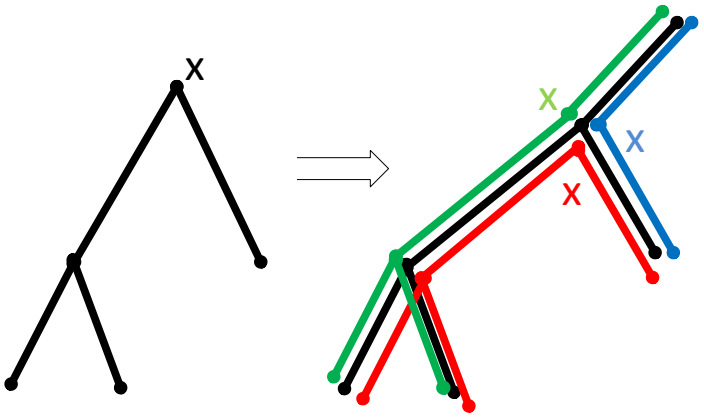

Figure 4.7: Three-way reconstruction creates 3 reconstructed sequences, conditioned on different subtrees, for each internal node.

For our algorithm to always produce the full tree, it needs to be able to create connections between trees by breaking non-root edges. To ensure this can be achieved, we will maintain, where possible, three separate sequence reconstructions at each internal node of the subtree, each based on two subtrees of $T(x)$ created by removing x, but independent of the third subtree. Figure 4.7 illustrates the idea behind three-way reconstruction.

Theorem 9 applies when all subtree edges have length less than g. We treat subtrees with a long edge as rooted at a node on that edge with a single sequence reconstruction. When that node is joined with another tree without creating a new long edge, a new core region is created; we then create new three-way reconstructions; see Figure 4.7. Trees with long edges can only be joined with another tree via the long edge, in order to maintain Invariant 3. In contrast, trees without a long edge can be joined by breaking any edge in the tree, as error-independent sequences are always available after three-way reconstruction has been performed. This means that we can reconsider some of the possible joins that had to be put on the waiting list.

The routine $ThreeWayReconstruction$ takes a tree with sequences reconstructed by maximum likelihood, adding to each vertex the two remaining reconstructions of its sequence. It must be started from a node that has no successors in reconstruction order.

Algorithm 12 ThreeWayReconstruction(r)

Let x, y, z be the neighbours of r.
Reconstruct sequences $\sigma_{xy}(r), \sigma_{yz}(r), \sigma_{xz}(r)$ conditioned on $\{x, y\}$,$\{y, z\}$ and $\{x, z\}$, respectively.
Let S be the set of vertices in $T(r)$ with only one sequence reconstruction.
Visit vertices in $T(r)$ in Breadth-First Search order, reconstructing each sequence conditioned on all choices of 2 neighbours. Stop branching if a node visited during an earlier call of $ThreeWayReconstruction$ is encountered.

4.3.4 Long edges must be joined

To maintain Invariant 3, we will ensure that any non-core region with a long edge can only be joined with another tree by breaking the long edge. For core regions, the connection to another tree can be placed on any edge. Procedure $CandidateEdges(x)$ identifies valid edges in the vicinity of x that can be broken.

Algorithm 13 CandidateEdges(x)

if x is in a non-core region **then**
  if x is the root of its non-core region or is incident to a long edge **then**
    Return the set containing the long edge.
  else
    Return $\emptyset$
  end if
else
  Return the set of all edges e incident to x such that three-way reconstruction has been performed on both ends of e
end if

4.3.5 Choosing hash table parameters

The running time of the algorithm is dominated by locality-sensitive hashing, as all the other operations have runtime at most proportional to the number of hits. Since the running time of $FindAllClose(q, d)$ increases with increasing radius d, we need to determine the minimum radius needed to ensure that the algorithm is able to make a true join at every iteration. To create a long edge, our algorithm must be able to find pairs of sequences at distance $2g + 2g_{err}$. However, this bound is only accurate when the endpoints of the

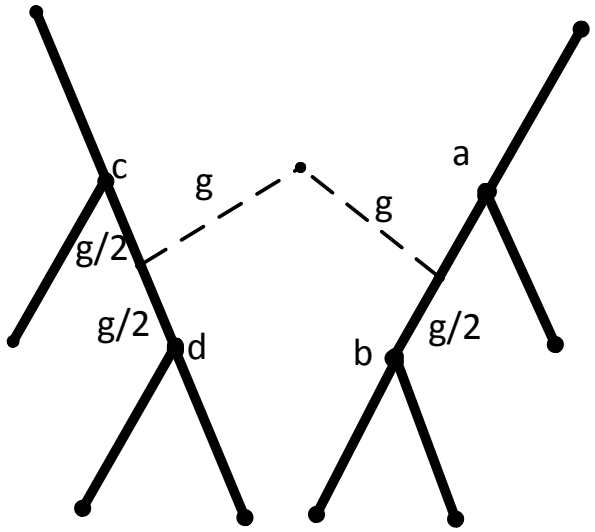

Figure 4.8: Joining two core regions with a long edge. The long edge can have length at most $2g$, while the distance from its endpoints to the nearest reconstructed sequences can be up to $g/2$.

new long edge coincide with already reconstructed ancestral sequences. This is the case when joining two root nodes. However, this is not always the case: when joining two long edges, an endpoint of the newly created edge may not be the same as the root, which is temporarily assumed to lie in the middle of the long edge. When joining two short edges, the endpoints of the newly created edges will be located somewhere between the internal nodes of the existing trees. Thus, we must take into account that the minimum distance between two reconstructed sequences from different subtrees will be greater than $2g+2g_{err}$.

For each subtree, the maximum distance between two reconstructed sequences is bounded by g. Thus, the distance from an endpoint of a newly added edge to the nearest reconstructed sequence is at most $g/2$, in both subtrees that are being joined. It follows that the minimum evolutionary distance between any two reconstructed sequences from subtrees that can be joined by a long edge is at most $d = 3g + 2g_{err}$ - see Figure 4.3.5.

Note that d can be brought arbitrarily close to $2g + 2g_{err}$ by having multiple reconstructed sequences spaced in equal intervals along each edge that has a possibility of being joined. If we ensure that the minimum distance between two reconstructed sequences at every such edge is at most g/k, we can reduce the radius d in $FindAllClose(q, d)$ to

$2g+g/k+2g_{err}$, at the cost of increasing the running time and memory usage by a constant factor. For trees with a long edge, additional reconstructions are needed only along the long edge. For trees without a long edge, we have to add extra reconstructed sequences at each edge after $ThreeWayReconstruction$ has been called. To simplify the presentation, in what follows we only discuss the basic variant of the algorithm, without the extra reconstructions.

4.3.6 The effect of reconstruction errors on locality-sensitive hashing

If two reconstructed ancestral sequences are not error-independent, their distance estimate may be biased. Here, we show this has no effect on the running time of *FindAllClose* or its ability to find all sequences within specified evolutionary distance.

Lemma 7 implies that the lack of error-independence will not cause LSH to miss internal nodes within specified evolutionary distance. Still, it could happen that the bias from error-dependence generates additional hits in the hash tables. The following lemma shows that the number of additional hits is at most $\log n$.

Lemma 11. *Let $r_2 = 1/2 - 1/2(\exp(-2cf \log\log n))$ with $c < 1$ in the LSH algorithm. The number of sequences b such that $d(a,b) > cf \log\log n$ and $d(a, A(b)) < cf \log\log n$ is at most $3 \log n$.*

Proof. We use a well-known equivalent formulation of the Cavender-Farris Markov chain on trees as a percolation process. For each edge in T, we set if to *open* with probability $1 - 2p(e)$ and *closed* otherwise. Each connected component of open edges shares the same state and states in different components are independent. Notice that conditioned on there being a closed edge between a and b, a reconstruction error in b is independent of the state at a. If the true evolutionary distance between a and b is at least $f \log\log n$, the probability of them being in the same component is at most $\exp(-2f \log\log n)$. Consequently, the expected normalized Hamming distance between a and $A(b)$ is at least

$$\frac{1}{2} - \frac{1}{2}\exp(-2f \log\log n) - \exp(-2f \log\log n)$$

Picking $c < 1$ ensures that this is bounded away from r_2 for n large enough. This, together with the fact that there are at most $3 \log n$ internal nodes within distance $f \log\log n$ of a, concludes the proof. □

4.3.7 The complete algorithm

With these issues resolved, we can present the complete algorithm as Algorithm 14. To prove that *Reconstruct* returns the correct tree with high probability, we have to prove two facts: that the algorithm never runs out of subtrees to join, and that each join maintains invariants 1-4.

Lemma 12. *During the execution of the algorithm, there always exists a pair of trees that can be merged.*

Before we prove Lemma 12, we need to introduce some additional notation. A *cherry* is a pair of leaves exactly two edges apart from each other. Consider the forest $F^c = T - F$. We will refer to components of F^c as *antitrees* to avoid confusion with reconstructed trees from F. Since all leaves in F are leaves in T, all internal vertices in F^c have degree 3. Thus, every antitree is either a single edge whose length is at most g, or it contains at least two cherries. Therefore, each antitree in F^c contains two leaves at distance less than $2g$. We will refer to the path between such close leaves (which can be a cherry or a single edge) as a *connection*. We need to show that there always exists a connection that can be used to merge two trees in F.

We will need the following auxiliary result.

Lemma 13. *Suppose trees T_1 and T_2 in F have a connection, but cannot be merged. Then either T_1 or T_2 borders another antitree incident to its long edge.*

Proof. There are two scenarios where two close trees cannot be merged. The first is when one of the endpoints of the connection (without loss of generality, in T_1) is in a non-core region and does not lie on a long edge. Since T_1 has a long edge, there exists an antitree that is incident to that long edge; otherwise T would have an edge of length more than g.

The second scenario is when the merge would create multiple long edges. For this to happen, there have to exist at least two antitrees that border a long edge; otherwise, again, T would contain an edge of length more than g. □

We can now prove Lemma 12.

Proof of Lemma 12. We perform induction on the number k of antitrees in F^c. For $k = 1$, the result follows directly from Lemma 13. For $k > 1$, consider an antitree T_1^c such that at most one tree T' in F adjacent to T_1^c is also adjacent to some other antitree in F^c; T_1^c must exist, for otherwise we would have a cycle in T. If T_1^c is a single edge, we can delete

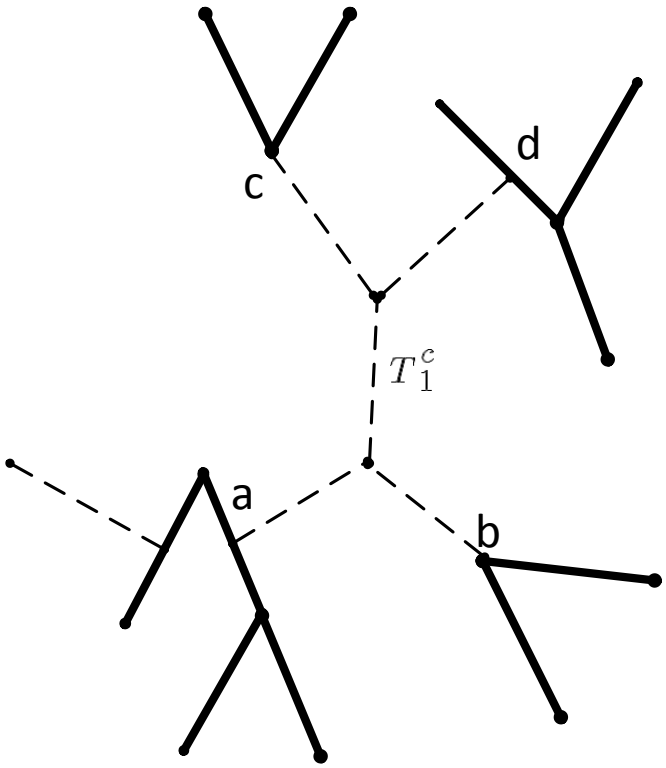

Figure 4.9: Antitree T_1^c (dashed) contains two cherries. Cherry (a, b) cannot be merged, since that would lead to the creation of a non-core region with two long edges. Cherry (c, d) can be merged, since both of its adjacent trees have other antitrees bordering them.

from T_1^c from T, creating two trees. One of those trees has $k - 1$ antitrees, so the result holds by the induction hypothesis.

On the other hand, if T_1^c contains two cherries, then by the choice of T_1^c at most one of them borders a tree in F which borders other antitrees in F^c. The second cherry can be merged by Lemma 13. □

Theorem 12. *Each iteration of the* **while** *loop maintains invariants 1-4.*

Proof. We perform induction on the number of loop iterations. Assume that at the beginning of the loop, the current forest F satisfies Invariants 1 to 4. Thus, the only edge that could possibly violate Invariants 1 and 2 is the newly added edge (r_1, r_2). Invariants

Algorithm 14 Reconstruct($\{\sigma\}$,f,g)

Start with a forest F where each sequence is in its own tree.
Initialize hash tables for nearest neighbour search
Use *FindAllClose* to identify all sequence pairs at distance less than $3g + 2g_{err}$; put these in a queue DistQueue.
$WaitList \leftarrow \emptyset$
while F has more than one tree **do**
  if DistQueue.empty() **then**
    Move all hits on $WaitList$ to $DistQueue$
  end if
  $(x, y) \leftarrow DistQueue.pop()$
  if $T(x) = T(y)$ **then**
    continue
  end if
  $X \leftarrow CandidateEdges(x), Y \leftarrow CandidateEdges(y), Joins \leftarrow X \times Y$
  if $X = \emptyset$ or $Y = \emptyset$ **then**
    $WaitList.add((x, y))$
  end if
  Filter $Joins$ to only include pairs $((i, j), (k, l))$ where $CheckErrorIndependence(i, j|k, l)$
  if $|Joins| = 0$ **then**
    $WaitList.add((x, y))$
    Continue to the next loop iteration
  end if
  if $|Joins| > 1$ **then**
    return Failure
  end if
  Let (i^*, j^*) and (k^*, l^*) be the unique pair of edges in $Joins$.
  create an edge between the two edges (i^*, j^*) and (k^*, l^*). Call the new internal nodes r_1, r_2
  use $MiddleEdge$ to calculate the lengths $d_1, \ldots, d_5$ of the five new edges
  if $\max_i d_i \geq 2g$ or $d_i > d_j > g$ for some $i \neq j$ or $CheckOverlaps(r_1, r_2)$ **then**
    undo this loop iteration
    $WaitList.add((x, y))$
  end if
  reconstruct the sequences at the new internal nodes r_1, r_2
  if the new tree has a long edge **then**
    create a root on the new long edge; reconstruct the sequence at the root
  end if
  if the new tree has no long edge **then**
    $ThreeWayReconstruction(r_1)$
  end if
  Use *FindAllClose* to add all newly-created sequence pairs whose distances are below $3g + 2g_{err}$ to *DistQueue*.
  For all hits (x, y) in *WaitList* at distances less than $3g + 2g_{err}$ from any of the newly created sequences, move (x, y) from *WaitList* to *DistQueue*
end while

Table 4.1: The approximate runtime for different values of p_g

p_g	runtime
0.01	$n^{1.10} \log^2 n$
0.02	$n^{1.19} \log^2 n$
0.05	$n^{1.47} \log^2 n$
0.075	$n^{1.67} \log^2 n$
0.10	$n^{1.85} \log^2 n$

3 and 4 could be violated by (r_1, r_2) and its adjacent edges. If Invariant 2 is violated, $CheckOverlaps(r_1, r_2)$ will return false, undoing the addition of (r_1, r_2) to the tree. Invariants 1 and 4 are maintained by Theorem 11 and the quartet query independence checks. Invariant 3 is maintained by the conditions on d_i's in line 20. □

4.3.8 Runtime analysis

At each iteration of the **while** loop, all operations except nearest neighbour search and line 33 take constant time, since the ratio g/f is constant (and thus the number of nodes within distance g of any node is a constant). Updating $WaitList$ can be performed in amortized constant time per loop iteration using a hash table. The complexity is therefore dominated by locality-sensitive hashing. The evolutionary distance $3g + 2g_{err}$ corresponds to a Hamming distance of at most $h = 1/2(1-(1-2p_g)^3(1-2p_{err})^2)$, where $p_g = (1-e^{-2g})/2$, and the bound on p_{err} is given by Theorem 10. $FindAllClose$ is used a constant number of times per loop. Each loop iteration requires $O(n^{2h+\epsilon} \log^2 n)$ time. The loop is run $O(n)$ times, since the number of sequences within distance $3g+2g_{err}$ of any sequence is constant. Overall, the runtime is bounded by

$$Cn^{2-(1-2p_g)^3(1-2p_{err})^2+\epsilon} \log^2 n < Cn^{2-(1-2p_g)^3 \frac{(1-4g)(1-8g)}{(1-2g)^4}+\epsilon} \log^2 n < Cn^{2-(1-2p_g)^3(8g-1)^2+\epsilon} \log^2 n,$$

which is always $o(n^2)$. Table 1 shows the runtime for selected values of p_g.

4.4 Experiments

4.4.1 A practical algorithm

Many assumptions made by our theoretical algorithm are impractical. The hash tables required for *FindAllClose* to work with high probability may require a prohibitive amount of memory. Using maximum likelihood for ancestral sequence reconstruction requires much memory to store conditional probabilities. We have developed a simpler and more memory-efficient practical algorithm.

Our implementation uses a number of hash tables required to find near neighbours with known constant probability, not high probability. For reasons of memory efficiency, we also do not perform three-way reconstruction; instead, we join non-root nodes of different subtrees without requiring the sequences in quartet queries to be error-independent. Note that we still require error-independent sequences for estimating branch lengths in an existing subtree. After two subtrees are joined, the sequences in the smaller subtree are re-estimated according to an ordering compatible with that of the larger tree.

For simplicity, the practical algorithm does not use routines *CheckOverlaps* and *CheckErrorIndependence*. Instead, we first find a join that yields the shortest estimated edge length, and then perform a small number of Nearest Neighbour Interchange (NNI) operations after each join, to ameliorate the problems originally addressed by these two functions. After an edge has been added, we re-estimate the length of all edges whose length might have been affected by the merge. If any quartet gives a topology that is not consistent with the edge, we perform an NNI operation to fix the topology and re-estimate the lengths of adjacent edges. While this process might repeat several times, it is not equivalent to traditional local search algorithms using NNI's, as only the edges in the close vicinity of the new edge are affected and the associated computational cost is much lower.

The algorithm tries to merge pairs of trees, starting from the collisions with the lowest estimated evolutionary distance, using a priority queue. If no collisions are found within distance $(r_1 + r_2)/2$, for parameters r_1, r_2, a new hash table is added, until the maximum number of $2n^{r_1/r_2}$ hash tables is reached. If several subtrees remain to be joined after examining all the hits in the hash tables, we use a simple heuristic that picks representative ancestral sequences from each tree and finds closest pairs among them, so the algorithm terminates even if the tree contains a small number of long edges.

The current version of our algorithm does not attempt to find optimal LSH parameters r_1 and r_2. In our experiments, we set them to 0.2 and 0.6, respectively. This choice leads

to memory inefficiency for trees with very short edges, as we will see later. We leave the automatic adjustment of these values for future work.

4.4.2 Data sets

We used a data set from Chapter 3, where we presented QTree [162], to compare our new algorithm with QTree and FastTree. We simulated 10 trees on 20000 taxa from the pure-birth process. We then multiplied the length of each branch by a factor chosen uniformly at random from interval $[0.5, 2]$ to deviate the trees from ultrametricity. This methodology follows the previous work of Liu *et al.* [107]. We then scaled the branch lengths of the entire tree by several choices of constant factors, resulting in data sets with mean branch lengths equal to $0.0312, 0.0625, 0.1250$, and 0.25, respectively. For each choice of tree and scaling factor, we generated alignments whose length varied between 250 and 4000 positions from the Jukes-Cantor model with variable rates across sites distributed exponentially (again following Liu *et al.*). No indels were introduced in the simulation. The data sets can be downloaded at `http://cs.uwaterloo.ca/~jmtruszk/datasets.tar.gz`.

4.4.3 Results

Figure 4.10 shows the performance of our algorithm, compared with QTree and FastTree under the Robinson-Foulds metric. We only ran the Neighbour Joining phase of FastTree, without its concluding local search phase, since similar local search procedures can be applied to the output of any algorithm.

Our algorithm achieves higher accuracy than both FastTree and QTree in most settings. The main exception is trees with long branches, where its accuracy is substantially lower than both QTree and FastTree. The poor performance of our algorithm for trees with long branches is not surprising given that it relies so heavily on hash hits to reconstructed ancestral sequences, whose accuracy diminishes as branch lengths approach the phase transition. For very short sequences, FastTree appears somewhat more accurate, possibly because of aggregating information from a greater number of distance estimates.

4.4.4 Running times and scalability

On most instances, our program runs in times competitive with FastTree and somewhat longer than QTree (see Table 4.2). We believe the running times could be improved by a more careful choice of parameters, and note that our software is a preliminary prototype.

Table 4.2: The running times of the three algorithms for three representative data sets. In most cases, our algorithm is faster than FastTree, but slower than QTree. For very short branches, the number of hash table collisions is very high due to r_2 being too large, which results in a longer running time for our algorithm.

algorithm	$scale = 25, len = 1k$	$scale = 50, len = 1k$	$scale = 100, len = 1k$
QTree	4m57s	5m39s	6m28s
LSHTree	24m49s	8m27s	6m50s
FastTree (NJ phase only)	10m31s	11m01s	11m23s

For 32-bit machines with up to 4GB RAM, the original version of our program did not scale to alignments larger than $2 \cdot 10^7$ letters. This was due to memory-intensive implementations of LSH and queues. We have since implemented an improved hash table structure, enabling the algorithm to process alignments up to $6 \cdot 10^7$ letters. At present, the main obstacle to improving scalability further is the large memory footprint of the conditional probability tables required for ancestral sequence reconstruction. We believe that memory usage could be further improved by implementing more efficient strategies for storing conditional probabilities, such as those recently developed by Izquierdo-Carrasco and Stamatakis [92].

For trees where average branch length is very low compared to the r_2 parameter, vast numbers of collisions are generated, which leads to a substantial increase in running time and memory usage. We partially mitigate this problem by discarding all but top k hits from each hash table entry, but the increase in running time is still substantial, sometimes increasing by 3-fold compared to the normal scenario. We plan to solve this problem by automatically choosing r_2 in the final version of the software.

We also ran our program on the larger simulated 16S data set with 78000 sequences from the FastTree paper [134]. This data set was simulated from a tree inferred from a real 16S alignment. We created smaller data sets by randomly sampling 20000 and 40000 sequences from the full data set. For the data set with 20000 sequences, our algorithm took 15 minutes, compared with 9 minutes for both FastTree and QTree. For the data set with 40000 sequences, our algorithm took 56 minutes, compared with 26 minutes for FastTree and 19 minutes for QTree. Algorithm accuracies were as in the other experiments: our algorithm was more accurate than QTree and FastTree. The runtime of our algorithm was likely affected by the wrong choice of r_2.

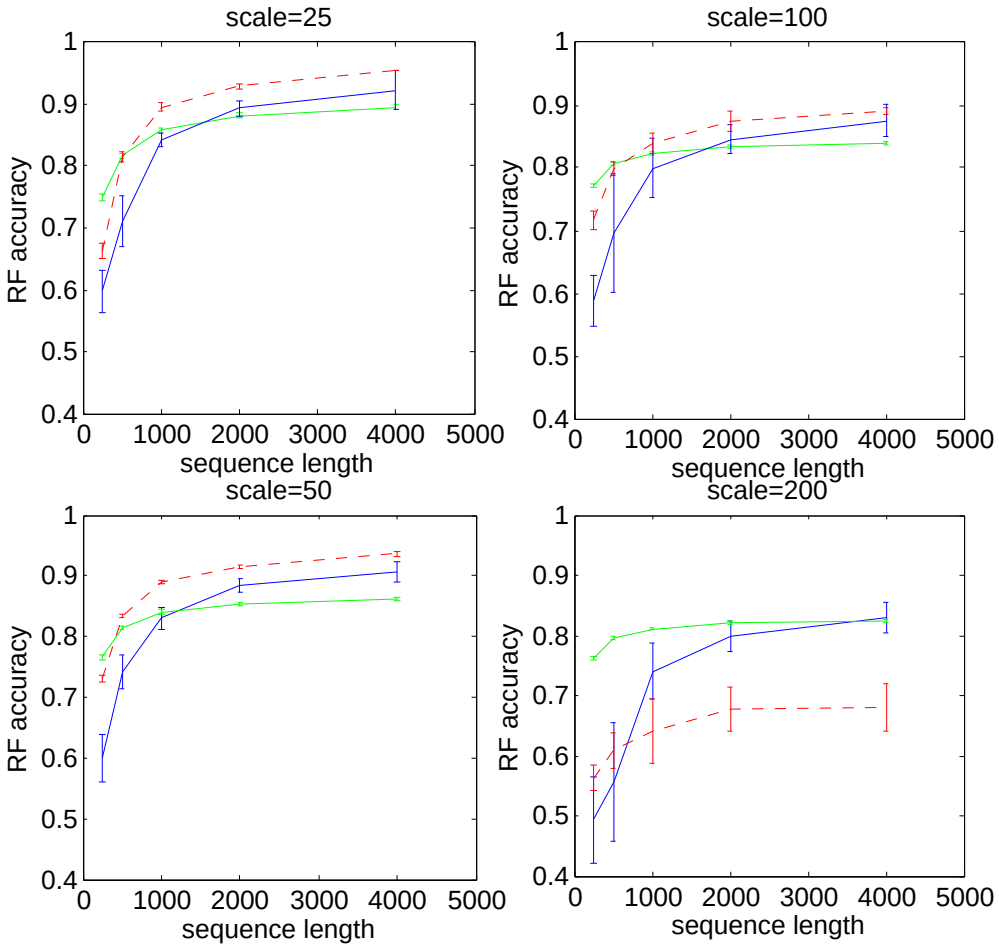

Figure 4.10: The performance of the LSH algorithm(red, dashed) compared to QTree(dark blue), and FastTree(light green), as a function of the length of the sequences. The four graphs represent the performance on 10 tree topologies with branch lengths scaled by constant factors $25, 50, 100$, and 200, corresponding to data sets with mean branch lengths equal to $0.0312, 0.0625, 0.1250$, and 0.25, respectively. The accuracy of the LSH algorithm is superior to both QTree and FastTree in most settings, except for phylogenies with very long branches (scale=200), where the LSH algorithm performs substantially worse than the other two, presumably due to poor quality of reconstructed ancestral sequences.

4.5 Conclusions and future work

We have presented the first theoretical algorithm that correctly reconstructs phylogenies whose branch lengths are short enough. This theoretical algorithm shows the possibility of reconstructing such large phylogenies in sub-quadratic time from short (logarithmic-length) sequences, without compromising the accuracy. Our prototype implementation achieves accuracies that are comparable or exceed existing algorithms, while also offering competitive running times for instances of a few tens of thousands of taxa. We believe that both the accuracy and the running time of the algorithm could be improved further.

This work could be improved in several ways. The applicability of our algorithm to diverse evolutionary scenarios will require the ability to set hash table parameters automatically. We plan to investigate, both theoretically and experimentally, whether the runtime of LSH algorithms could be improved by taking advantage of rate variability across sites, which may also offer opportunities for higher accuracy in cases where long branches are present. We also plan to incorporate rate heterogeneity information in the merging process, either by using likelihood to evaluate possible merges, or by using distance estimators that account for rate heterogeneity. Roch [126] recently developed a fast theoretical method for estimating evolutionary rates of alignment columns. Some of the techniques in that work could be useful for adapting our algorithm to varying evolutionary rates.

Some theoretical questions also remain. Felsenstein's algorithm is known to be optimal at reconstructing ancestral states, but its success probability appears hard to estimate (see e.g. [111]). Tighter bounds on reconstruction accuracy would lead to an improved running time of our algorithm. Another avenue for improvement is using a faster locality-sensitive hashing scheme. Dubiner [56] recently proposed such a scheme for very long sequences, but it is not clear whether it will be useful with sequences of only logarithmic length. Another important question is whether the hard requirement on branch lengths could be somehow relaxed.

Chapter 5

LSHPlace: fast phylogenetic placement of environmental sequences

5.1 Introduction

In the past few years, researchers have started studying microbial communities from diverse environments, such as soil [4], ocean [141], and the human body [163]. Microbial ecologists are interested in the diversity of bacteria in a given environment, their evolutionary origins, and their metabolic relationships. They answer these questions by collecting sequence data from environmental samples and then comparing them to reference sequences from known microbial lineages.

Phylogenetics provides a natural framework for investigating the microbial diversity in these environments. Most microorganisms can be approximately located on the tree of life for bacteria, and then communities or environments can be characterized by the relative abundance of certain taxa [32]. Or, unusual sequences can be a focus for further investigation and directed sequencing [110]. The first step, however, is to locate each sequence on the tree.

While many phylogenetic reconstruction algorithms have been developed over the years, the current flood of sequence data from next-generation sequencing presents new challenges for traditional methods. The massive amounts of data generated by NGS make traditional phylogenetic inference computationally prohibitive; indeed, in metagenomic contexts, a

common first step is to cluster a data set, possibly consisting of millions of reads and instead analyze just the hundreds or thousands of cluster centres, which discards much valuable data [32]. Another problem is that environmental sequencing produces short sequence reads, instead of full gene sequences. For example, reads generated by Illumina have length approximately 200 bp, which does not provide sufficiently strong phylogenetic signal for full phylogenetic inference. Moreover, if reads in the data set cover different regions of a long gene sequence, most phylogenetic algorithms tend to be biased towards grouping highly overlapping sequences together [116].

These problems have recently motivated researchers to focus on placing individual environmental sequences independently into a pre-known, fixed phylogeny, instead of performing full phylogenetic inference on the entire set of sequences. This has several advantages. The computational cost is greatly reduced, as the number of topologies that need to be considered is linear in the size of the tree for each sequence. By considering each query sequence separately, we also hope to avoid the biases associated with sequence overlaps. There are currently several programs performing this task, called *phylogenetic placement* [116, 11, 152]. Unfortunately, their speed is insufficient to place millions of reads, and they do not scale well when the reference trees grow, as we see in Section 5.4.4

In this chapter, we adapt the framework behind LSHTree, described in the previous chapter, to the problem of phylogenetic placement. More specifically, we develop the first algorithm that places sequences in a known reference phylogeny with running time sublinear in the number of taxa in the reference tree. We show that our methods are both theoretically sound and practically useful.

Our algorithm follows the principle behind LSHTree, with some important modifications. As in LSHTree, we use locality-sensitive hashing to locate parts of the tree that are probably close to the location of the query sequence. We reconstruct ancestral sequences in the tree to ensure that close sequences can be found even if the query sequence is distant from the reference taxa. Again, we assume evolution functions as a Markov chain on sequences with independent columns. Then, we use local search to determine the exact edge of the reference tree from which the read diverged. The running time of this procedure depends on both the maximum branch length in the reference tree, and the evolutionary distance from the query sequence to the reference tree. For Cavender-Farris models, if all edge lengths in the reference tree are below $\frac{1}{4}\ln 2 \approx 0.17$, the running time is $O(n^{\gamma_2(g,g_q)}\log^2 n)$ per sequence, which is always sublinear in n. The total runtime to place k new sequences onto a reference tree is thus $O(kn^{\gamma_2(g,g_q)}\log^2 n)$. In practice, we use a constant number of hash tables, and the runtime is dominated by local search. Still, the runtime for the algorithm grows sublinearly in n in the practical algorithm.

A novel algorithmic idea in this algorithm is to choose the most informative columns for hash table keys based on the Shannon mutual information between the column and the location of the read on the tree. We also use likelihood, rather than distance estimates, to guide the final phase of the local search.

We evaluate our algorithm on a number of synthetic and real data sets. The accuracy of our algorithm is slightly lower than that of pplacer [116], while its running time is around 8-25 times faster, making it a useful tool for handling large data sets. The previous version of LSHPlace [20] only used distance-based local search. In the current version, distance-based local search is followed by by maximum likelihood search, which results in substantially higher accuracy, at the cost of increased running time. We will show the effect of this change in Section 5.4.4.

An earlier version of the results in this chapter appeared in a conference paper by Brown and Truszkowski [20]. The current version of the algorithm has not yet been published.

5.2 Related work

Many tools determine the taxonomic origin of environmental sequences. These tools can be roughly divided into three categories: those based on phylogenetic modelling, those based on sequence composition, and those based on homology search. Gerlach [77] provides a recent survey of these methods.

Recently, researchers have developed several tools for placing environmental sequences onto a reference phylogeny. These methods generally take $O(n)$ time to insert a sequence in a tree of n taxa. MLTreeMap [152] was the first tool designed for this task. The Evolutionary Placement Algorithm [11] and pplacer [116] offer more efficient implementations of the same approach, which places a sequence optimally at each edge of the phylogeny, and then assigns the overall maximum likelihood placement as the answer for each individual sequence. For increased efficiency, both of these algorithms initially compute the approximate likelihood corresponding to placing a read at the midpoint of each edge of the reference tree. They then perform full likelihood optimization for a subset of edges with highest likelihoods. This leads to a substantial speedup, but the overall runtime remains $O(n)$ per read.

Some software pipelines for analyzing metagenomic data sets employ full phylogenetic reconstruction. These include TreePhyler [144] and CARMA [78]. While much progress has been made in fast phylogeny reconstruction in recent years [134, 22, 19], reconstructing the full tree remains much slower than other classification methods.

Another approach is to build a classifier to discriminate between taxonomic groups at different levels of the Linnaean hierarchy. These classifiers do not attempt to model phylogenetic relationships between different taxa, but instead treat the problem as a supervised classification problem at each level of the hierarchy. The features used are usually derived from the k-mer composition of the sequence, which bypasses the need for aligning sequences. Many such classifiers have been developed, including PhyloPythia [118], TACOA [52], and PhyMM [100]. Taxy [119] uses mixture models and k-mers to estimate the relative abundance of different taxonomic groups in a set of sequences, without attempting to classify each sequence in detail. Taxonomic classifiers are often faster than phylogeny-based methods, but they do not offer the same explanatory power. Moreover, classifiers based on k-mers tend to behave badly on short sequences, as they lack sufficient information to distinguish different clades.

Yet another class of approaches uses BLAST [5] to determine evolutionary proximity of sequence reads to known species. Here, environmental sequences are expected to generate BLAST hits with the sequences they are closely related to. Several algorithms exist for mapping sets of BLAST hits to taxonomic classifications, including MEGAN [90] and SOrt-ITEMS [86]. Unfortunately, if the only close relatives to a sequence in a tree are internal nodes, this approach will fail; our method will avoid this problem due to our inference of internal sequences.

5.3 The algorithm

5.3.1 Overview

The input to the algorithm consists of three parts: the reference phylogeny T, on n taxa, which includes tree edge lengths; the multiple alignment A of the n sequences in the reference phylogeny; and a set of k query sequences, each aligned to that reference alignment. Our goal is to assign each query sequence to the edge where it joins the tree so that the phylogeny of $n + 1$ taxa is correct.

Our algorithm consists of the following steps. The first three are a preprocessing phase, and the resultant data structures could be stored for use, if a given tree and multiple alignment were to be used to analyze many different sets of reads.

1. Estimate evolutionary rates for each column of the reference alignment.

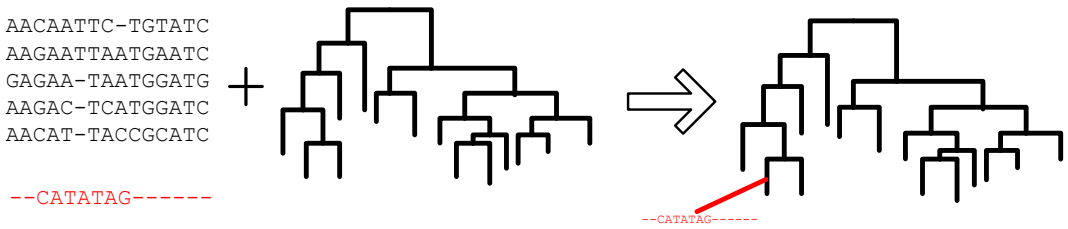

Figure 5.1: An illustration of phylogenetic placement. For each query sequence q (shown in red), the algorithm finds an edge in the reference tree from which q diverged.

2. Reconstruct ancestral sequences at each internal node of the reference phylogeny using maximum likelihood.
3. Build a collection of hash tables, keyed on informative columns of A. Add the keys for both leaf sequences and ancestral inferences into the hash tables.
4. For each query sequence y:
   (a) Look for collisions between y and the hash tables. Choose the closest sequence x (in Hamming distance) colliding with y.
   (b) Examine the neighbourhood of x in T. For each edge e in that neighbourhood, estimate the distance between e and y. Find the closest edge e_y to y.
   (c) Starting from e_y, use greedy local search to find the maximum likelihood placement of y.

In what follows, we explain the details and the motivation behind each of these steps.

5.3.2 Estimating evolutionary rates

Different positions in a gene evolve at different rates. This phenomenon is known as *rate heterogeneity across sites*, and can lead to lower reconstruction accuracy when the algorithm assumes constant speed of evolution across columns. For long sequences, the effect of rate heterogeneity on the accuracy of inferred trees can be modest [27], but it is more serious for short sequences where there is no sufficient phylogenetic signal. To increase accuracy, we model each alignment column as having its own evolutionary rate. Formally, for a substitution model with rate matrix M, the transition matrix for column

i is $e^{r_i t M}$, where r_i is the evolutionary rate for column i. We then use the inferred rates when computing the likelihood of each placement, and when choosing informative columns for hash tables (see Section 5.3.4).

We infer evolutionary rates for each column by maximum likelihood. We compute column likelihoods using Felsenstein's algorithm [68], given a given rate r_i. We optimize likelihoods by finding the root of the derivative of the likelihood with respect to evolutionary rate. We use a numerical approximation of the derivative, and we find the root by bisection.

5.3.3 Reconstructing ancestral sequences

We reconstruct the ancestral sequences using maximum likelihood. To facilitate rapid likelihood search, we do three-way reconstruction of ancestral sequences, and we maintain the probability distributions of reconstructed characters conditioned on each direction. We denote by $\pi^{v,i,e}$ the probability vector representing the *a posteriori* distribution of the ancestral character at the i-th site of internal node v, conditioned by the subtree of T created by deleting edge e incident to v.

We also record the probability distribution of the ancestral character conditioned on the whole reference tree T. We denote the probability vector for the i-th site at node v as $\pi^{v,i}$. These vectors will be helpful when choosing informative columns for hashing (see Section 5.3.4), and they will also speed up local search, as we will see in Section 5.3.5.

All in all, we store 4 probability distributions per column per internal node of the tree. This greatly increases the memory footprint of the program compared to LSHTree. As a result, the current implementation can only handle reference trees of size up to 20000 taxa. This can be improved by clever memory management, such as the methods of Izquierdo-Carrasco and Stamatakis [92]. We leave that as future work.

5.3.4 Locality-sensitive hashing

We use locality-sensitive hashing in a manner similar to LSHTree (see Chapter 4). Unlike in LSHTree, columns for hash table keys are chosen at random from the set of *most informative* columns. This is to avoid using nearly-constant columns which would result in a large number of collisions, and also to avoid using very fast-evolving columns that are very likely to contain a mismatch between a read and its closest sequence in the reference tree, and whose ancestral reconstructions are likely to be less accurate.

Choosing informative sites

We identify informative characters by estimating the mutual information between a read location and a character. For simplicity, we assume that the read diverged from the reference tree at one of the nodes of the tree (as opposed to somewhere along an edge), and we treat that divergence point as a random variable with uniform distribution across nodes. Furthermore, we assume that the distance between the read and the divergence point is fixed at $\ell_{read} = 0.05$. The Shannon mutual information between read location R and the character Y_i at site i can be written as

$$I(R, Y_i) = H(Y_i) - H(Y_i|R)$$

where $H(Y_i)$ is the global Shannon entropy of Y_i and $H(Y_i|R)$ is the conditional entropy of Y_i given the location [112]. The conditional entropy $H(Y_i|R)$ is

$$H(Y_i|R) = \sum_{r \in V(T)} H(Y_i|R = r) = -\sum_{r} \Pr[R = r] \sum_{y \in \{A,C,G,T\}} \Pr[Y_i = y|R = r] \log_2 \Pr[Y_i = y|R = r]$$

where r ranges over all nodes in T. The probability $\Pr[Y_i = y|R = r]$ can be computed by multiplying the probability vector π_r^T by the transition matrix $\mathbb{P}(\ell_{read} r_i)$. The global Shannon entropy $H(Y_i)$ can be calculated as follows

$$H(Y_i) = -\sum_{y \in \{A,C,G,T\}} \Pr[Y_i = y_i] \log_2 \Pr[Y_i = y_i]$$

where $\Pr[Y_i = y]$ can be calculated by summing over all values of R.

After computing the mutual information for each column, we choose the columns with highest mutual information. An important question is how many informative columns to choose. If too many columns are chosen, many of them will not be very informative. On the other hand, if we choose too few columns, the amount of information about the true evolutionary distances between sequences will be insufficient. We choose to randomly choose from the 200 most informative columns (not including the near-constant columns), or the top 50% most informative columns, for short alignments below 400 columns.

Theoretical runtime analysis

In theory, the asymptotic running time of our method is determined by the number of hash tables needed to ensure that a read can be mapped to its nearest neighbour sequence in the

reference tree. In Chapter 4, we presented the running time analysis for LSHTree, under the Cavender-Farris model of evolution with equal evolutionary rates at all sites, and assuming that all branch lengths were below the phase transition. Under these assumptions, a similar bound applies to LSHPlace. Let g_{err} be the distance corresponding to the probability p_{err} of incorrectly reconstructing an ancestral character. If the evolutionary distance between y and the node that joins it to the tree is g_{query}, the effective evolutionary distance between the query and the nearest sequence in the tree is at most $g_{query} + g/2 + g_{err}$, where g is the maximum edge length in the reference tree. The total branch length $g_{query} + g/2 + g_{err}$ corresponds to a mutation probability of $\frac{1}{2} - \frac{1}{2}(1 - 2p_{query})(1 - 2p_g)^{1/2}(1 - 2p_{err})$. The number of hash tables required is thus bounded by

$$n^{1-(1-2p_{query})(1-2p_g)^{1/2}(1-2p_{err})}$$

Based on the bound on p_{err} given by Theorem 10, we bound the number of hash tables for any value of p_g and p_{err}. Table 1 shows the running times of the hashing time for a single query, for different values of p_g, assuming for simplicity that $p_{query} \leq p_g$.

In practice, we do not know the upper bound on the distance of query sequences to the tree. The values in Table 5.1 should be understood only as illustration of the theoretical basis for our method. In the current version of the software, the number of hash tables is fixed at 8, as this value gave good results for phylogenetic tree reconstruction, and maintains moderate memory use. In the current version of the software, the running time is dominated by local search; changing the number of hash tables has a moderate effect on the runtime.

Table 5.1: The runtime of inserting a new taxon into a tree with n taxa, as a function of the maximum edge mutation probability in CF models.

p_g	0.01	0.02	0.05	0.075	0.10
runtime	$n^{0.05} \log^2 n$	$n^{0.10} \log^2 n$	$n^{0.27} \log^2 n$	$n^{0.43} \log^2 n$	$n^{0.61} \log^2 n$

5.3.5 Local search

Our near-neighbour search procedure should find sequences that are in the vicinity of the correct placement for a query sequence y. However, it does not guarantee that the optimal

placement of y is one of the edges adjacent to the node found. We use a heuristic two-phase local search procedure to find the maximum likelihood placement in the vicinity of the node that generated the collision. We first use distance-based local search, followed by likelihood optimization.

Distance-based local search

For the distance-based local search phase, we estimate the distance between y and each edge $e = (x_1, x_2)$ near of the colliding sequence x as:

$$\hat{d}(y, e) = \frac{1}{2}(\hat{d}(x_1, y) + \hat{d}(x_2, y) - \hat{d}(x_1, x_2)),$$

estimating $\hat{d}(a, b)$ via the probabilistic model of evolution.

If x_1 and x_2 are error-independent reconstructed ancestral sequences, they will not bias the estimate $\hat{d}(y, e)$, as the additive terms in distance estimates associated with the reconstruction error will approximately cancel out (see Section 2.4.6). We examine all edges within distance $\hat{d}(x, y)$ of node x. The edge e^* with smallest estimate is chosen as the placement of y, with pendant edge length $\hat{d}(x, e^*)$. If T has many very long and very short edges, this will prove slow; in practice this situation is quite rare. If we assume a minimum length for each edge, and a maximum length beyond which we consider $\hat{d}(x, y)$ too long to be meaningfully estimated, the neighborhood is of constant size.

For simplicity of implementation, the current implementation of the algorithm ignores possible dependencies between reconstruction errors. These dependencies could be ameliorated by using some of the techniques discussed in our previous paper [19]. We leave this as future work.

Likelihood local search

Our likelihood local search procedure starts from the edge found by the distance-based seeks to find the maximum likelihood placement in the local neighbourhood. A naive approach to likelihood optimization is to find, for each edge, the optimum divergence point along that edge, together with the optimum pendant branch length leading to the read (see Figure 5.2). This involves maximizing a function of two variables, and takes much longer than distance-based search. Instead, we use a faster, simplified local search procedure that first places a read onto internal nodes of the tree, rather than edges. The benefit of this procedure is that we only need to optimize one variable for each placement,

the pendant length. Once the maximum likelihood node placement is found, we find optimal placements onto each edge incident to that node and choose the one with the highest likelihood. This approach is not mathematically guaranteed to find the maximum likelihood edge placement, but we have found that it gives very similar results compared to edge-based local search, while being three to five times faster.

All likelihood calculations in LSHPlace were done under the Jukes-Cantor substitution model. This is different from pplacer [116], which uses a more sophisticated general time-reversible (GTR) model. The flexibility of the GTR model often leads to higher accuracy in phylogenetic inference [68], at the cost of slightly increased running time. We chose to use the Jukes-Cantor model for simplicity of implementation. In future versions of LSHTree, we plan to switch to GTR.

5.3.6 Accommodating for different read locations

In some cases, the sequences being placed are short reads, much smaller than the total length of the alignment, and their starting positions are distributed across the alignment [160]. We need multiple sets of hash tables to make sure that each read is covered by at least one set of LSH tables. We solve this problem by using a sliding window approach. We construct groups of hash tables, each corresponding to short regions of the alignment, so that each query sequence maps to at least one set of hash tables. Specifically, if all reads have at least k' bases, we divide the alignment into blocks of length $k'/2$, and build a separate set of hash tables for each block. We bin the reads by which block of tables overlaps them in the alignment and progressively process each block of hash tables, from the beginning to the end of the alignment. This ensures that we never have to store more than one set of hash tables in memory.

5.4 Experiments

We have investigated the performance of LSHPlace on several simulated data sets and one real metagenomic data set from an Arctic soil sample [9]. We report accuracies and running times for the current version of LSHPlace, referred to as LSHPlace 2, as well as the initial version of the program, which used only distance-based local search. We also compare both versions with pplacer [116], the current most commonly used phylogenetic placement program. The newest version of LSHPlace is 8-25 times faster than pplacer, at a modest cost in accuracy.

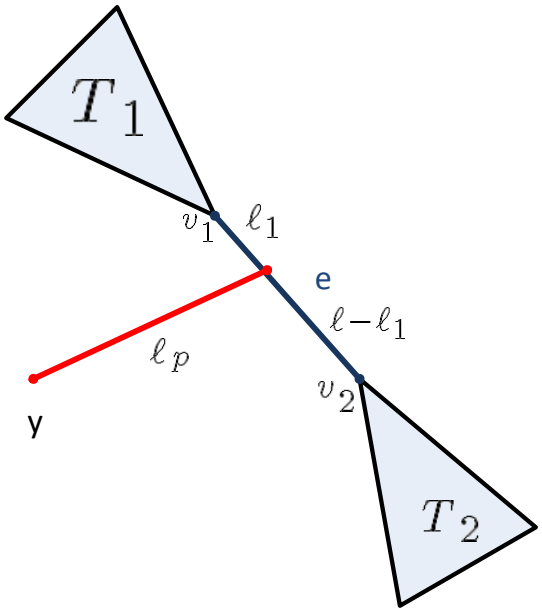

Figure 5.2: Finding the maximum likelihood placement along a single edge. The procedure optimizes two variables: the length of the pendant edge ℓ_p, and the distance ℓ_1 from the divergence point to one of the endpoints of the edge. To compute the likelihood contribution for column i, we use the precomputed probability vectors $\pi^{v_j,i,e}$ corresponding to the a posteriori distribution of characters at v_1 and v_2 conditional on the sequences in T_1 and T_2, respectively. We multiply these vectors by the appropriate transition matrices corresponding to the lengths of each of the three branches. More precisely, if y_i is the probability vector corresponding to the character at the read, the likelihood of the placement is

$$\left[(\pi^{v_1,i,e}\mathbb{P}(\ell_1 r_i)) \otimes (\pi^{v_2,i,e}\mathbb{P}((\ell - \ell_1) r_i)) \otimes (y_i \mathbb{P}(\ell_p r_i))\right]\vec{1}$$

where $\otimes$ denotes entry-wise multiplication of two vectors and $\vec{1}$ represents the all-ones column vector.

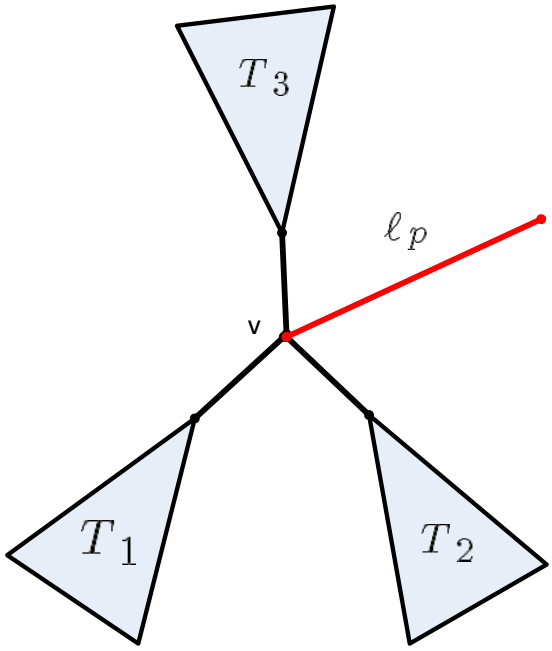

Figure 5.3: Finding the maximum likelihood placement diverging from a node. The procedure finds the maximum likelihood pendant edge length ℓ_p. To compute the likelihood contribution for column i, we use the precomputed probability vector $\pi^{v,i}$ corresponding to the a posteriori distribution of the character at v conditional on all the sequences in the tree. Formally, the likelihood is

$$\left[(\pi^{v,i}\mathbb{P}(\ell_p r_i)) \otimes y_i\right]\vec{1}$$

5.4.1 Data sets

Our experiments used both simulated and real data. We generated synthetic short read data sets from a simulated 16S rRNA alignment on $n = 78132$ sequences from the FastTree paper [134]. The length of the alignment was $m = 1287$ bases. The data were generated as follows. First, k taxa were chosen at random from the alignment, and the reference subtree induced by these taxa on the true tree was recorded. We then generated 100,000 simulated short reads. Each short read was generated by sampling with replacement from the $n-k$ sequences not included in the reference tree, and keeping m' contiguous positions, starting at a uniformly-chosen position. The read length m' was chosen from a Gaussian distribution with mean 200 and standard deviation 20.

For the real data set, we used a metagenomic 16S rRNA Illumina library from Alert, Nunavut, Canada [9]. The reads were located in the V3 variable region of the ribosomal RNA. The sequences were clustered at 97% identity, which resulted in 27848 sequences with a mean length of 152 bases, aligned to a reference alignment of 2759 sequences using SSU-Align v. 0.1 [130]. For placement, we used a reference 16S tree from the Living Tree Project [172]. The diameter of the tree was 1.46 mutations per site, and the average distance between two nodes was 0.54.

5.4.2 Accuracy

Table 5.2 shows the results of our algorithm and pplacer on the synthetic data sets. The distance-based version of our algorithm, LSHPlace 1, was considerably less accurate than pplacer, placing roughly half of the reads in their correct locations, compared to 79% for pplacer. This improved dramatically after adding likelihood search to our program; LSHPlace 2 is just 6 percentage points less accurate than pplacer, but is much faster (see Table 5.4).

For larger trees, all algorithms tended to place sequences farther from their correct edges in terms of the topological distance (TD). However, the evolutionary distance (ED) between placements and correct locations tended to be lower for larger trees. Larger reference trees contained more short edges which were hard to distinguish for both algorithms. The magnitude of these effects was similar for both versions of LSHPlace and pplacer. This suggests that using larger trees can provide more information about the evolutionary origin of the read, even if placements are less likely to be exactly correct for larger trees.

Table 5.2: Accuracy of LSHPlace and pplacer on three simulated data sets. LSHplace is less accurate, but still reasonably close for all data set sizes.

	data set								
	huge.1, 1000 taxa			huge.1, 5000 taxa			huge.1, 10000 taxa		
method	% correct	ED	TD	% correct	ED	TD	% correct	ED	TD
LSHPlace 1	51	0.054	1.12	48	0.033	1.17	46	0.028	1.34
LSHPlace 2	73	0.013	0.45	68	0.008	0.52	63	0.007	0.65
Pplacer	79	0.011	0.30	74	0.007	0.39	69	0.006	0.51

5.4.3 Results on the real data set

While both programs produced generally similar predictions on synthetic data sets, they disagreed much more on the real data set. Only 10% of reads were placed on the same edge by both Pplacer and LSHPlace 1. LSHPlace 2 agreed with pplacer 32% of the reads. The median topological distance between a pplacer prediction and an LSHPlace 1 prediction for the same sequence was 10 edges, with median evolutionary distance being 0.15 mutations per site. For LSHPlace 2, these medians were 4 edges and 0.08 mutations per site, respectively. We suspect that the accuracy of LSHPlace 1 was adversely impacted by widely varying evolutionary rates across sites in the alignment. The accuracy might also have been impacted by the presence of many near-constant sites in the alignment, which could have had an adverse effect on the accuracy of distance estimates.

Even with the likelihood search procedure in LSHPlace 2, around two out of three reads were classified differently by LSHPlace 2 and pplacer. More worryingly, some reads were placed in vastly different places by the two programs; for 15% of the reads, the distance between pplacer and LSHPlace 2 predictions was more than 20 edges. We performed several experiments to investigate the cause of these differences.

In the first experiment, we took a subset of randomly chosen 1000 reads from the data set and wrote a program that found the optimal placement under the Jukes-Cantor model by exhaustive search. We then compared these placements with the predictions of LSHPlace 2 and pplacer. In doing so, we wanted to see how efficient LSHPlace was at finding the maximum likelihood placement under Jukes-Cantor. We also wanted to see to what extent the differences in predictions were due to pplacer using the GTR model instead of Jukes-Cantor.

The results are shown in Table 5.3. LSHPlace 2 predictions are more likely to agree exactly with those computed by exhaustive search; LSHPlace finds the most likely placement 52% of the time, compared to 45% for pplacer. This is not surprising, since pplacer

Table 5.3: Comparing LSHPlace 2 and pplacer to the maximum Jukes-Cantor likelihood placement on the real data set. LSHPlace 2 is more likely produce the same placement as exhaustive search, but also more likely to place reads very far from the global maximum.

method	$TD = 0$	$TD \leq 3$	$TD \leq 5$	$TD \leq 20$
LSHPlace 2	51	56	60	86
Pplacer	45	66	70	93

optimizes a different (though similar) likelihood function. However, LSHPlace 2 was also more likely to produce predictions that were vastly different from those found by exhaustive search; 14% of its predictions wound up being 20 edges or further apart from the maximum likelihood placement, compared to 8% for pplacer. The larger number of severely misplaced reads in LSHPlace predictions is due to choosing an initial hash table hit that is far away from the global maximum. This is not surprising: both versions of LSHPlace choose the hash table hit based on the Hamming distance from the read, which is a poor predictor for the likelihood when evolutionary rates vary widely across sites. As a result, LSHPlace 2 only finds a locally optimal placement under Jukes-Cantor. In contrast, pplacer's exhaustive search is more likely to correctly identify the region containing the maximum likelihood, since GTR and Jukes-Cantor likelihoods are strongly correlated. The optimal placements under the two models will often be slightly different.

To improve the odds of finding the globally optimal placement by LSHPlace 2, we experimented with two improvements in the way hash table hits are handled. The first improvement is to use multiple hash table hits for local search, rather than just one. We did this by sorting the hits for each read by their Hamming distances, and evaluating each hit whose Hamming distance to the read was less than twice that of the closest hit. For each such hit, we calculated the likelihood of the starting point of the search, and discarded the hit if the log-likelihood it differed from the best starting point by more than $t = 6$ (we used base 2). For most reads, this resulted in discarding all but the closest hit. In some cases, new higher likelihood areas were discovered, which led to improving the fraction of predictions that coincided with the global maximum Jukes-Cantor likelihood placement to 57%.

The second improvement was to use likelihood, not Hamming distance, to evaluate all hits. For each node in the tree that generated a collision with the read, we calculated the likelihood corresponding to that read having diverged from the node at a fixed evolutionary distance $d = 0.05$. The program sorted the hits according to decreasing likelihood, and examined all hits whose log-likelihood was within $t = 6$ of the maximum. This improved the agreement with exhaustive search to 65%, at the cost of increasing the runtime approx-

imately by a factor of 3. We think this increase in runtime could be dramatically lowered by clever pre-processing of likelihoods. We leave this for future work.

To further investigate the apparent poor performance of LSHPlace 2 on some reads, we performed a visual examination of the likelihood landscape for several representative reads. We chose 10 reads for detailed investigation: 5 reads were chosen among those whose LSHPlace 2 predictions were within 5 edges from the optimal placement under the Jukes-Cantor model; the other 5 had their LSHPlace 2 predictions at least 20 edges apart from the optimum. We then visualized the likelihood landscape by colouring higher likelihood parts of the tree with brighter colours. We have found that each of the reads from the accurate subset produced a landscape with one dominant peak whose likelihood was at least 2^5 times higher than that of the second-highest local optimum. In contrast, reads that were placed far from the global maximum were likely to contain multiple local optima in distant parts of the tree. These optima often had similar likelihoods; the second-highest local maximum for those reads was only 2 to 4 times lower than the global maximum.

We hypothesized that the existence of distant local optima with similar likelihood values are due to the lack of phylogenetic signal in the data. To assess the amount of phylogenetic signal, we bootstrapped the alignment 10 times and computed the global maximum likelihood placement for each read, for each bootstrap sample. We then compared the location of the maximum likelihood placement for each bootstrap sample to the optimal placement for the original alignment. Bootstrapping alignments is a well-established statistical method of assessing statistical support for phylogenetic hypotheses [68]. Reads with robust phylogenetic signal should give similar predictions after bootstrapping.

For many reads, the optimal placements for bootrapped alignments were quite different. The average distance between from a bootstrapped placement to the original optimal placement was 9.47 edges, with wide variability across reads ranging from 0 to 81 edges. We found that the average distance between the placement inferred the bootstrap sample and that inferred from the original alignment is strongly correlated (0.58 Pearson correlation) to the accuracy of LSHPlace 2 predictions. Of the LSHPlace 2 placements that were within 2 edges from the maximum Jukes-Cantor likelihood placement, 78% had average bootstrap variability below 10 edges. Of the placements above 20 edges from the maximum, 80% had bootstrap variability above 10 edges. This suggests that the poor performance of LSHPlace on some reads is largely due to insufficient signal in those reads.

5.4.4 Running times and scalability

The running times for both programs are shown in Table 3. Each runtime corresponds to placing 100,000 reads into a tree of the given size. LSHPlace 1 is around 25 to 60 times faster than pplacer, while LSHPlace 2 is 8 to 25 times faster.

Table 5.4: The time to place 100000 reads into a tree, as a function of the number of taxa in the tree

taxa	1000	5000	10000
LSHPlace 1	7m	12m	17m
LSHPlace 2	35m	41m	49m
pplacer	3.8h	6.4h	18.8h

5.5 Conclusion

We have presented LSHplace, a new algorithm for phylogenetic placement. By using locality-sensitive hashing, and including inferred ancestral sequences in the hash tables, our algorithm allows us to approximately locate new sequences onto an existing phylogenetic tree extremely rapidly; a local-search procedure allows us to then find an optimal placement quickly for each new sequence. Our work can be used in a variety of domains, but we expect it will be especially useful in the context of metagenomic sampling, where millions of sequence reads are generated and analyzed at the same time to characterize environments. Experimental results on simulated data sets show that the current version of LSHPlace is often almost as accurate as programs based on exhaustive search such as pplacer, while being several times faster. On real data sets, our program tends to give similar results to exhaustive search, except for sequences with low phylogenetic signal.

There are several future directions we plan to follow with this work. First, we plan to extend the program to more sophisticated evolutionary models than Jukes-Cantor. We also plan to devise a fast method for assessing the reliability of LSHPlace predictions based on bootstrapped alignments and evaluating multiple hits. Such a method would be very useful, as we could exclude sequences with insufficient signal from further analysis. Finally, we are thinking of extending our method to placing unaligned sequences. Two recent programs, PaPaRa [12] and SEPP [122] have combined alignment estimation with placement by using

placement information to only align the read to sequences that are closely related to it. It is an interesting question whether a locality-sensitive hashing strategy could be adapted to that domain. Given the widespread success of tools like BLAST [5], which use similar techniques for unaligned sequences, it is conceivable that this could be achieved.

Chapter 6

Conclusions

In this thesis, we have investigated faster algorithms for building phylogenetic trees from large sequence alignments. Such algorithms are needed to handle massive amounts of sequence data that are being produced by today's sequencing projects. The recent surge of interest in microbial diversity, metagenomics, and a number of "Tree of Life" projects currently under way have created a need for methods able to reconstruct phylogenies from hundreds of thousands of aligned sequences. Our algorithms help address that need in a way that is both principled and practical. Furthermore, our theoretical results show that such methods can be both fast and accurate in many scenarios. This sound theoretical foundation is what distinguishes our work from previous, more heuristic approaches to building large phylogenies [134, 59]. In the following paragraphs, we summarize our contributions and outline directions for future research.

In Chapter 3, we presented QTree, a very fast quartet-based algorithm that relies on a novel data structure whose goal is to facilitate rapid insertions of new taxa into an existing tree. The runtime of the algorithm is $O(n \log n)$, which is asymptotically optimal for any phylogenetic reconstruction algorithm. When quartet errors occur independently and with known probability, our algorithm is guaranteed to reconstruct the correct phylogeny with probability that approaches 1 for large data sets. While these assumptions are not true in practice, our algorithm can still produce reasonably accurate trees. Our experimental results indicate that the algorithm has similar accuracy to Neighbour Joining, while being several times faster than any other algorithm currently in use. However, we also see evidence that there are fundamental limits to improving the accuracy. The data structure used in the algorithm may be of independent interest, as it may be useful in other domains where hierarchical clustering is used.

In Chapter 4, we introduced LSHTree, a fast algorithm with mathematical accuracy guarantees under Markov models of evolution. Given long enough sequences, LSHTree is guaranteed to reconstruct the correct phylogeny with high probability, provided that the branch lengths of the generating tree are not too long. Our algorithm is the fastest among all algorithms with such guarantees: its runtime is $O(n^{1+\gamma(g)} \log^2 n)$, where the exact running time depends on the branch lengths, but is always $o(n^2)$. The minimum number of columns required for accurate reconstruction is $\Omega(\log n)$, which matches the guarantees of the most efficient algorithms up to a constant [43, 139]. Unlike previous theoretical algorithms, LSHTree produces reasonably accurate phylogenies in practical scenarios where the theoretical guarantees do not necessarily apply. In experiments, LSHTree is more accurate than other algorithms, except in circumstances foreseen by our theoretical analysis. In practice, the algorithm is only slightly slower than QTree, and faster than all the other algorithms for phylogenetic reconstruction. These experimental results are encouraging, but there is still work to be done on running times and memory usage in practice.

Finally, in Chapter 5, we have built on the techniques behind LSHTree to develop a fast algorithm for placing anonymous sequences onto a known phylogenetic tree. LSHPlace is the first algorithm that places query sequences onto a reference tree whose runtime grows sublinearly with the number of taxa in the reference tree. In practice, this means that the running time of our algorithm is 8-25 times lower than that of its leading competitor, pplacer, while its accuracy is only slightly lower on simulated data. On real data sets, LSHPlace and pplacer tend to give very similar predictions, except for sequences whose evolutionary origin is highly uncertain.

Our results on fast phylogeny algorithms naturally lead to further research questions, both in theory and in practice. We have discussed them in the previous chapters and we summarize them briefly below.

In the theoretical realm, we would like to extend the LSHTree framework to alignments with rate heterogeneity across sites. This could be aided by a recent result of Roch [126], who provided a theoretical method for inferring evolutionary rates of sites without the knowledge of the tree topology. Such an extension of LSHTree would be significant in two ways. Firstly, it would be the first algorithm with theoretical guarantees for alignments whose columns evolve at different rates. Secondly, choosing slow-evolving columns for hashing would also improve the running time.

Another theoretical question is whether LSHTree can be adapted to reconstruct accurate phylogenies from unaligned sequences. Alignment-free phylogenetic reconstruction has recently attracted some renewed interest [6, 44], but it is not known whether fast algorithms can be developed in this domain. This question is particularly important since

current methods for inferring large alignments are widely considered to be less accurate than their slower counterparts. An algorithm that is not sensitive to alignment errors is likely to have higher accuracy for many data sets where there is a lot of uncertainty about the alignment.

There are also several practical issues that we want to address in the near future. Most urgently, we would like to continue working on the scalability of LSHTree and LSHPlace, to ensure that they can handle trees of hundreds of thousands to millions of sequences. We hope to achieve this by carefully exploring the balance between memory usage and accuracy. We would also like to improve the accuracy of both tools by using more sophisticated models of nucleotide substitution, such as HKY or GTR. To that end, we plan to use one of several open-source libraries for phylogenetic likelihood that are currently under development in other research groups (e.g. BEAGLE by Ayres *et al.* [8]). In the longer run, we would like to develop a version of LSHTree that directly optimizes likelihood at every merging step, without relying on distances for choosing the best merge. We hope that such a strategy could eliminate the need for time-consuming local search procedures, such as those currently used by RAxML and FastTree.

References

[1] Assembling the Tree of Life Initiative, 2013.

[2] The Los Alamos HIV Sequence Database, 2013.

[3] Open Tree of Life Project, 2013.

[4] Heather K Allen, Luke A Moe, Jitsupang Rodbumrer, Andra Gaarder, and Jo Handelsman. Functional metagenomics reveals diverse [beta]-lactamases in a remote Alaskan soil. *ISME Journal*, 3(2):243–251, 2009.

[5] Stephen F. Altschul, Thomas L. Madden, Alejandro A. Schffer, Jinghui Zhang, Zheng Zhang, et al. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402, 1997.

[6] Alexandr Andoni, Mark Braverman, and Avinatan Hassidim. Phylogenetic reconstruction with insertions and deletions. *Preprint*, 2010.

[7] Kevin Atteson. The performance of neighbor-joining methods of phylogenetic reconstruction. *Algorithmica*, 25(2-3):251–278, 1999.

[8] Daniel L Ayres, Aaron Darling, Derrick J Zwickl, Peter Beerli, Mark T Holder, Paul O Lewis, John P Huelsenbeck, Fredrik Ronquist, David L Swofford, Michael P Cummings, et al. Beagle: an application programming interface and high-performance computing library for statistical phylogenetics. *Systematic biology*, 61(1):170–173, 2012.

[9] Andrea K. Bartram, Michael D. J. Lynch, Jennifer C. Stearns, Gabriel Moreno-Hagelsieb, and Josh D. Neufeld. Generation of Multimillion-Sequence 16S rRNA Gene Libraries from Complex Microbial Communities by Assembling Paired-End Illumina Reads. *Applied and Environmental Microbiology*, 77(11):3846–3852, 2011.

[10] Alex Bateman, Ewan Birney, Lorenzo Cerruti, Richard Durbin, Laurence Etwiller, Sean R. Eddy, Sam Griffiths-Jones, Kevin L. Howe, Mhairi Marshall, and Erik L. L. Sonnhammer. The Pfam Protein Families Database. *Nucleic Acids Research*, 30(1):276–280, 2002.

[11] Simon A. Berger, Denis Krompass, and Alexandros Stamatakis. Performance, accuracy, and web server for evolutionary placement of short sequence reads under maximum likelihood. *Systematic Biology*, 60(3):291–302, 2011.

[12] Simon A. Berger and Alexandros Stamatakis. Aligning short reads to reference alignments and trees. *Bioinformatics*, 27(15):2068–2075, 2011.

[13] Vincent Berry and Olivier Gascuel. Inferring evolutionary trees with strong combinatorial evidence. *Theoretical Computer Science*, 240(2):271–298, 2000.

[14] Vincent Berry, Tao Jiang, Paul Kearney, Ming Li, and Todd Wareham. Quartet cleaning: improved algorithms and simulations. In *Proceedings of ESA 1999*, pages 313–324.

[15] Mathieu Blanchette, Eric D Green, Webb Miller, and David Haussler. Reconstructing large regions of an ancestral mammalian genome in silico. *Genome Research*, 14(12):2412–2423, 2004.

[16] Pavel M. Bleher, Jean Ruiz, and Valentin A. Zagrebnov. On the purity of the limiting Gibbs state for the Ising model on the Bethe lattice. *Journal of Statistical Physics*, 79(1/2):473–482, 1995.

[17] Alexandre Bouchard-Côté, Sriram Sankararaman, and Michael I. Jordan. Phylogenetic Inference via Sequential Monte Carlo. *Systematic Biology*, 2012.

[18] Gerth Stølting Brodal, Rolf Fagerberg, and Christian N. S. Pedersen. Computing the Quartet Distance between Evolutionary Trees in Time $O(n \log n)$. *Algorithmica*, 38(2):377–395, 2003.

[19] Daniel G. Brown and Jakub Truszkowski. Fast reconstruction of phylogenetic trees using locality-sensitive hashing. In *Proceedings of WABI 2012*, pages 44–56.

[20] Daniel G. Brown and Jakub Truszkowski. LSHPlace: Fast phylogenetic placement using locality-sensitive hashing. In *Proceedings of PSB 2013*, pages 310–319. to appear.

[21] Daniel G. Brown and Jakub Truszkowski. Fast error-tolerant quartet phylogeny algorithms. In *Proceedings of CPM 2011*, pages 147–161, 2011.

[22] Daniel G. Brown and Jakub Truszkowski. Towards a practical $O(n \log n)$ phylogeny algorithm. In *Proceedings of WABI 2011*, pages 14–25, 2011.

[23] Daniel G. Brown and Jakub Truszkowski. Fast error-tolerant quartet phylogeny algorithms. *Theoretical Computer Science (special issue on selected papers from CPM 2011)*, 483:104–114, 2012.

[24] William J Bruno, Nicholas D Socci, and Aaron L Halpern. Weighted neighbor joining: a likelihood-based approach to distance-based phylogeny reconstruction. *Molecular Biology and Evolution*, 17(1):189–197, 2000.

[25] David Bryant. A classification of consensus methods for phylogenetics. *DIMACS series in Discrete Mathematics and Theoretical Computer Science*, 61:163–184, 2003.

[26] David Bryant. On the uniqueness of the selection criterion in neighbor-joining. *Journal of Classification*, 22(1):3–15, 2005.

[27] David Bryant, Daniel Huson, Tobias Kloepper, and Kay Nieselt-Struwe. Distance corrections on recombinant sequences. In *Proceedings of WABI 2003*, pages 271–286.

[28] David Bryant, John Tsang, Paul E. Kearney, and Ming Li. Computing the quartet distance between evolutionary trees. In *Proceedings of SODA 2000*, pages 285–286.

[29] Jeremy Buhler and Martin Tompa. Finding motifs using random projections. *Journal of Computational Biology*, 9(2):225–242, 2002.

[30] Michael Bulmer. Use of the method of generalized least squares in reconstructing phylogenies from sequence data. *Molecular Biology and Evolution*, 8(6):868, 1991.

[31] O. Peter Buneman. The recovery of trees from measures of dissimilarity. *Mathematics in the Archaeological and Historical Sciences*, 1971.

[32] J.G. Caporaso, J. Kuczynski, J. Stombaugh, K. Bittinger, F. D. Bushman, et al. Qiime allows analysis of high-throughput community sequencing data. *Nature Methods*, 7(5):335–336, May 2010.

[33] James A Cavender. Taxonomy with confidence. *Mathematical Biosciences*, 40(3):271–280, 1978.

[34] Belinda SW Chang, Karolina Jönsson, Manija A Kazmi, Michael J Donoghue, and Thomas P Sakmar. Recreating a functional ancestral archosaur visual pigment. *Molecular Biology and Evolution*, 19(9):1483–1489, 2002.

[35] François Chevenet, Christine Brun, Anne-Laure Bañuls, Bernard Jacq, and Richard Christen. Treedyn: towards dynamic graphics and annotations for analyses of trees. *BMC Bioinformatics*, 7:439, 2006.

[36] Benny Chor, Amit Khetan, and Sagi Snir. Maximum likelihood on four taxa phylogenetic trees: analytic solutions. In *Proceedings of RECOMB 2003*, pages 76–83.

[37] Benny Chor and Tamir Tuller. Maximum likelihood of evolutionary trees is hard. In *Proceedings of RECOMB 2005*, pages 296–310.

[38] Melissa Cline, Richard Hughey, and Kevin Karplus. Predicting reliable regions in protein sequence alignments. *Bioinformatics*, 18(2):306–314, 2002.

[39] Richard Cole, Lee-Ad Gottlieb, and Moshe Lewenstein. Dictionary matching and indexing with errors and don't cares. In *Proceedings of STOC 2004*, pages 91–100.

[40] Miklós Csűrös. Fast recovery of evolutionary trees with thousands of nodes. *Journal of Computational Biology*, 9(2):277–297, 2002.

[41] Charles Darwin. *The Origin of Species by Means of Natural Selection: or, the Preservation of Favoured Races in the Struggle for Life*. John Murray, London, 1859.

[42] Constantinos Daskalakis, Elchanan Mossel, and Sébastien Roch. Phylogenies without branch bounds: Contracting the short, pruning the deep. In *Proceedings of RECOMB 2009*, pages 451–465.

[43] Constantinos Daskalakis, Elchanan Mossel, and Sébastien Roch. Evolutionary trees and the Ising model on the Bethe lattice: a proof of Steels conjecture. *Probability Theory and Related Fields*, 149(1-2):149–189, 2011.

[44] Constantinos Daskalakis and Sebastien Roch. Alignment-free phylogenetic reconstruction. In *Proceedings of RECOMB 2010*, pages 123–137.

[45] William HE Day. Computational complexity of inferring phylogenies from dissimilarity matrices. *Bulletin of Mathematical Biology*, 49(4):461–467, 1987.

[46] William H.E. Day, David S. Johnson, and David Sankoff. The computational complexity of inferring rooted phylogenies by parsimony. *Mathematical Biosciences*, 81:33–42, 1986.

[47] Alexis Dereeper, V. Guignon, G. Blanc, Stéphane Audic, S. Buffet, François Chevenet, Jean-François Dufayard, S. Guindon, V. Lefort, M. Lescot, Jean-Michel Claverie, and Olivier Gascuel. Phylogeny.fr: robust phylogenetic analysis for the non-specialist. *Nucleic Acids Research*, 36(Web-Server-Issue):465–469, 2008.

[48] Todd Z DeSantis, Philip Hugenholtz, Neils Larsen, Mark Rojas, Eoin L Brodie, Keith Keller, Thomas Huber, Daniel Dalevi, Ping Hu, and Gary L Andersen. Greengenes, a chimera-checked 16S rRNA gene database and workbench compatible with ARB. *Applied and environmental microbiology*, 72(7):5069–5072, 2006.

[49] Richard Desper and Olivier Gascuel. Fast and accurate phylogeny reconstruction algorithms based on the minimum-evolution principle. *Journal of Computational Biology*, 9(5):687–706, 2002.

[50] Christophe Dessimoz and Manuel Gil. Phylogenetic assessment of alignments reveals neglected tree signal in gaps. *Genome Biology*, 11(4):R37, 2010.

[51] Luc Devroye. A note on the height of binary search trees. *Journal of the ACM*, 33(3):489–498, 1986.

[52] Naryttza N. Diaz, Lutz Krause, Alexander Goesmann, Karsten Niehaus, and Tim W. Nattkemper. TACOA - taxonomic classification of environmental genomic fragments using a kernelized nearest neighbor approach. *BMC Bioinformatics*, 10, 2009.

[53] Chuong B Do, Mahathi SP Mahabhashyam, Michael Brudno, and Serafim Batzoglou. Probcons: Probabilistic consistency-based multiple sequence alignment. *Genome Research*, 15(2):330–340, 2005.

[54] Alexei J. Drummond and Andrew Rambaut. Beast: Bayesian evolutionary analysis by sampling trees. *BMC Evolutionary Biology*, 7:214, 2007.

[55] Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge Univ. Press, 2009.

[56] Moshe Dubiner. Bucketing coding and information theory for the statistical high-dimensional nearest-neighbor problem. *IEEE Transactions on Information Theory*, 56(8):4166–4179, 2010.

[57] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[58] Robert C Edgar. Muscle: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 32(5):1792–1797, 2004.

[59] Isaac Elias and Jens Lagergren. Fast neighbor joining. In *Proceedings of ICALP 2005*, pages 1263–1274.

[60] David Eppstein. Fast hierarchical clustering and other applications of dynamic closest pairs. *Journal of Experimental Algorithmics (JEA)*, 5:1, 2000.

[61] Péter L. Erdős, Michael Anthony Steel, László A. Székely, and Tandy Warnow. A few logs suffice to build (almost) all trees (I). *Random Structures and Algorithms*, 14(2):153–184, 1999.

[62] Péter L. Erdős, Michael Anthony Steel, László A. Székely, and Tandy Warnow. A few logs suffice to build (almost) all trees: Part II. *Theoretical Computer Science*, 221(1-2):77–118, 1999.

[63] Jason Evans, Luke Sheneman, and James Foster. Relaxed neighbor joining: a fast distance-based phylogenetic tree construction method. *Journal of molecular evolution*, 62(6):785–792, 2006.

[64] William Evans, Claire Kenyon, Yuval Peres, and Leonard J. Schulman. Broadcasting on trees and the ising model. *The Annals of Applied Probability*, 10(2):410–433, 2000.

[65] Scott Federhen. The ncbi taxonomy database. *Nucleic acids research*, 40(D1):D136–D143, 2012.

[66] U. Feige, D. Peleg, P. Raghavan, and E. Upfal. Computing with unreliable information. In *Proceedings of STOC 1990*, pages 128–137.

[67] Joseph Felsenstein. Evolutionary trees from dna sequences: a maximum likelihood approach. *Journal of Molecular Evolution*, 17(6):368–376, 1981.

[68] Joseph Felsenstein. *Inferring Phylogenies*. Sinauer, 2001.

[69] Robert D Finn, Jody Clements, and Sean R Eddy. Hmmer web server: interactive sequence similarity searching. *Nucleic Acids Research*, 39(suppl 2):W29–W37, 2011.

[70] Johannes Fischer and Volker Heun. Theoretical and practical improvements on the rmq-problem, with applications to lca and lce. In *Proceedings of CPM 2006*, pages 36–48.

[71] Walter M. Fitch. Towards defining the course of evolution: Minimum change for a specific tree topology. *Systematic Zoology*, 20:406–416, 1971.

[72] Dan Frumkin, Adam Wasserstrom, Shai Kaplan, Uriel Feige, and Ehud Shapiro. Genomic variability within an organism exposes its cell lineage tree. *PLoS Computational Biology*, 1(5):e50, 2005.

[73] Robert G Moyle, Jérôme Fuchs, Eric Pasquet, and Ben D Marks. Feeding behavior, toe count, and the phylogenetic relationships among alcedinine kingfishers (alcedininae). *Journal of Avian Biology*, 38(3):317–326, 2007.

[74] F. Gao and *et al.* Origin of HIV-1 in the chimpanzee *Pan Troglodytes Troglodytes* . *Nature*, 367:436–441, 1999.

[75] Olivier Gascuel. BIONJ: an improved version of the NJ algorithm based on a simple model of sequence data. *Molecular Biology and Evolution*, 14(7):685–695, 1997.

[76] Olivier Gascuel. *Mathematics of evolution and phylogeny*. Oxford University Press, 2005.

[77] Wolfgang Gerlach. Taxonomic classification of metagenomic sequences, 2012. PhD thesis. Bielefeld University.

[78] Wolfgang Gerlach, Sebastian Jünemann, Felix Tille, Alexander Goesmann, and Jens Stoye. WebCARMA: a web application for the functional and taxonomic classification of unassembled metagenomic reads. *BMC Bioinformatics*, 10:430, 2009.

[79] Morris Goodman, Calvin A Porter, John Czelusniak, Scott L Page, Horacio Schneider, Jeheskel Shoshani, Gregg Gunnell, and Colin P Groves. Toward a phylogenetic classification of primates based on DNA evidence complemented by fossil evidence. *Molecular Phylogenetics and Evolution*, 9(3):585–598, 1998.

[80] Dilan Görür and Yee Whye Teh. An Efficient Sequential Monte Carlo Algorithm for Coalescent Clustering. In *Proceedings of NIPS 2008*, pages 521–528, 2008.

[81] Sam Griffiths-Jones, Alex Bateman, Mhairi Marshall, Ajay Khanna, and Sean R. Eddy. Rfam: an RNA family database. *Nucleic Acids Research*, 31(1):439–441, 2003.

[82] Ilan Gronau and Shlomo Moran. Neighbor joining algorithms for inferring phylogenies via lca distances. *Journal of Computational Biology*, 14(1):1–15, 2007.

[83] Ilan Gronau and Shlomo Moran. Optimal implementations of UPGMA and other common clustering algorithms. *Information Processing Letters*, 104(6):205–210, 2007.

[84] Ilan Gronau, Shlomo Moran, and Sagi Snir. Fast and reliable reconstruction of phylogenetic trees with very short edges. In *Proceedings of SODA 2008*, pages 379–388.

[85] Stphane Guindon, Jean-Franois Dufayard, Vincent Lefort, Maria Anisimova, Wim Hordijk, and Olivier Gascuel. New Algorithms and Methods to Estimate Maximum-Likelihood Phylogenies: Assessing the Performance of PhyML 3.0. *Systematic Biology*, 59(3):307–321, 2009.

[86] M. Monzoorul Haque, Tarini Shankar Ghosh, Dinakar Komanduri, and Sharmila S. Mande. SOrt-ITEMS: Sequence orthology based approach for improved taxonomic estimation of metagenomic sequences. *Bioinformatics*, 25(14):1722–1730, 2009.

[87] Bernhard Haubold, Peter Pfaffelhuber, Mirjana Domazet-Lošo, and Thomas Wiehe. Estimating mutation distances from unaligned genomes. *Journal of Computational Biology*, 16(10):1487–1500, 2009.

[88] Leo J Hickey and Jack A Wolfe. The bases of angiosperm phylogeny: vegetative morphology. *Annals of the Missouri Botanical Garden*, pages 538–589, 1975.

[89] Kevin L. Howe, Alex Bateman, and Richard Durbin. QuickTree: building huge Neighbour-Joining trees of protein sequences. *Bioinformatics*, 18(11):1546–1547, 2002.

[90] title = Integrative Analysis of Environmental Sequences using MEGAN4 Huson, Daniel H and Mitra, Suparna and Weber, Nico and Ruscheweyh, Hans-Joachim and Schuster, Stephan C. *Genome Research*, 21:1552–1560, 2011.

[91] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of STOC 1998*, pages 604–613, New York, 1998.

[92] Fernando Izquierdo-Carrasco and Alexandros Stamatakis. Computing the phylogenetic likelihood function out-of-core. In *IPDPS Workshops*, pages 444–451, 2011.

[93] Tao Jiang, Paul Kearney, and Ming Li. Orchestrating quartets : Approximation and data correction. In *Proceedings of FOCS 1998*, pages 416–425.

[94] Katherine St. John, Tandy Warnow, Bernard M. E. Moret, and Lisa Vawter. Performance study of phylogenetic methods: (unweighted) quartet methods and neighbor-joining. *Journa of Algorithms*, 48(1):173–193, 2003.

[95] Sampath K. Kannan, Eugene L. Lawler, and Tandy J. Warnow. Determining the evolutionary tree using experiments. *Journal of Algorithms*, 21(1):26–50, 1996.

[96] Ming-Yang Kao, Andrzej Lingas, and Anna Östlin. Balanced randomized tree splitting with applications to evolutionary tree constructions. In *Proceedings of STACS 1999*, pages 184–196.

[97] Richard M. Karp and Robert Kleinberg. Noisy binary search and its applications. In *Proceedings of SODA 2007*, pages 881–890.

[98] Kazutaka Katoh, Kazuharu Misawa, Kei-ichi Kuma, and Takashi Miyata. Mafft: a novel method for rapid multiple sequence alignment based on fast fourier transform. *Nucleic Acids Research*, 30(14):3059–3066, 2002.

[99] Paul E. Kearney. The ordinal quartet method. In *Proceedings of RECOMB 1998*, pages 125–134.

[100] David R. Kelley and Steven L. Salzberg. Clustering metagenomic sequences with interpolated markov models. *BMC Bioinformatics*, 11:544, 2010.

[101] Valerie King, Li Zhang, and Yunhong Zhou. On the complexity of distance-based evolutionary tree reconstruction. In *Proceedings of SODA 2003*, pages 444–453.

[102] Carla Kuiken, Karina Yusim, Laura Boykin, and Russell Richardson. The Los Alamos Hepatitis C Sequence Database. *Bioinformatics*, 21(3):379–384, 2005.

[103] Michelle R. Lacey and Joseph T. Chang. A signal-to-noise analysis of phylogeny estimation by neighbor-joining: Insufficiency of polynomial length sequences. *Mathematical Biosciences*, 199(2):188–215, 2006.

[104] David A Liberles. *Ancestral sequence reconstruction*. Oxford University Press USA, 2007.

[105] Carolus Linnaeus. *Systema Naturae per Regna Tria naturae, Secundum Classes, Ordines, Genera, Species, cum Characteribus, Differentiis, Synonymis, Locis*. 1758.

[106] Konstantinos Liolios, I-Min A. Chen, Konstantinos Mavromatis, Nektarios Tavernarakis, Philip Hugenholtz, Victor M. Markowitz, and Nikos C. Kyrpides. The Genomes On Line Database (GOLD) in 2009: status of genomic and metagenomic projects and their associated metadata. *Nucleic Acids Research*, 38(Suppl 1):D346–D354, 2009.

[107] K. Liu, S. Raghavan, S. Nelesen, C.R. Linder, and T. Warnow. Rapid and accurate large-scale coestimation of sequence alignments and phylogenetic trees. *Science*, 324(5934):1561–1564, 2009.

[108] Kevin Liu, C Randal Linder, and Tandy Warnow. Raxml and fasttree: comparing two methods for large-scale maximum likelihood phylogeny estimation. *PloS One*, 6(11):e27731, 2011.

[109] Ari Löytynoja and Nick Goldman. Phylogeny-aware gap placement prevents errors in sequence alignment and evolutionary analysis. *Science*, 320(5883):1632–1635, 2008.

[110] M.D.J. Lynch, Andrea K. Bartram, and Josh D. Neufeld. Targeted recovery of novel phylogenetic diversity from next-generation sequence data. *ISME Journal*, 6:2067–2077, 2012.

[111] Bin Ma and Louxin Zhang. Efficient estimation of the accuracy of the maximum likelihood method for ancestral state reconstruction. *Journal of Combinatorial Optimization*, 21(4):409–422, 2011.

[112] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge University Press, 2003.

[113] Wayne P Maddison. Gene trees in species trees. *Systematic biology*, 46(3):523–536, 1997.

[114] Thomas Mailund, Gerth Brodal, Rolf Fagerberg, Christian Pedersen, and Derek Phillips. Recrafting the neighbor-joining method. *BMC Bioinformatics*, 7(1):29, 2006.

[115] Thomas Mailund and Christian N. S. Pedersen. QDist-quartet distance between evolutionary trees. *Bioinformatics*, 20(10):1636–1637, 2004.

[116] Frederick A. Matsen, Robin B. Kodner, and E. Virginia Armbrust. pplacer: Linear-time maximum-likelihood and bayesian phylogenetic placement of sequences onto a fixed reference tree. *BMC Bioinformatics*, 11:538, 2010.

[117] Atsushi Matsui and Masami Hasegawa. Molecular phylogeny and evolution in primates. In *Post-Genome Biology of Primates*, pages 243–267. Springer Tokyo, 2012.

[118] Alice Carolyn McHardy, Hctor Garca Martn, Aristotelis Tsirigos, Philip Hugenholtz, and Isidore Rigoutsos. Accurate phylogenetic classification of variable-length dna fragments. *Nature Methods*, 4(1):63–72, 2007.

[119] Peter Meinicke, Kathrin Petra Aßhauer, and Thomas Lingner. Mixture models for analysis of the taxonomic composition of metagenomes. *Bioinformatics*, 27(12):1618–1624, 2011.

[120] Radu Mihaescu, Cameron Hill, and Satish Rao. Fast phylogeny reconstruction through learning of ancestral sequences, December 08 2008.

[121] Radu Mihaescu, Dan Levy, and Lior Pachter. Why neighbor-joining works. *Algorithmica*, 54(1):1–24, 2009.

[122] Siavash Mirarab, Nam Nguyen, and Tandy Warnow. SEPP: SATé-enabled phylogenetic placement. In *Proceedings of PSB 2012*, pages 247–58.

[123] Navodit Misra, Guy E. Blelloch, R. Ravi, and Russell Schwartz. An optimization-based sampling scheme for phylogenetic trees. In *Proceedings of RECOMB 2011*, pages 252–266.

[124] Camilo Mora, Derek P Tittensor, Sina Adl, Alastair GB Simpson, and Boris Worm. How many species are there on earth and in the ocean? *PLoS Biology*, 9(8):e1001127, 2011.

[125] Elchanan Mossel. Phase transitions in phylogeny. *Transactions of the American Mathematical Society*, 356:2379–2404, 2004.

[126] Elchanan Mossel and Sebastien Roch. Identifiability and inference of non-parametric rates-across-sites models on large-scale phylogenies, July 30 2011.

[127] Rajeev Motwani and Prabakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[128] Fionn Murtagh. A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal*, 26(4):354–359, 1983.

[129] Eric P Nawrocki, Diana L Kolbe, and Sean R Eddy. Infernal 1.0: inference of RNA alignments. *Bioinformatics*, 25(10):1335–1337, 2009.

[130] Eric Paul Nawrocki. *Structural RNA Homology Search and Alignment Using Covariance Models*. Washington University, 2009.

[131] Minh Anh Thi Nguyen, Steffen Klaere, and Arndt von Haeseler. Misfits: evaluating the goodness of fit between a phylogenetic model and an alignment. *Molecular Biology and Evolution*, 28(1):143–152, 2011.

[132] Cédric Notredame. Recent progress in multiple sequence alignment: a survey. *Pharmacogenomics*, 3(1):131–144, 2002.

[133] David D Pollock, Derrick J Zwickl, Jimmy A McGuire, and David M Hillis. Increased taxon sampling is advantageous for phylogenetic inference. *Systematic Biology*, 51(4):664, 2002.

[134] Morgan N. Price, Paramvir S. Dehal, and Adam P. Arkin. FastTree: Computing large minimum evolution trees with profiles instead of a distance matrix. *Molecular Biology and Evolution*, 26(7):1641–1650, 2009.

[135] Morgan N Price, Paramvir S Dehal, and Adam P Arkin. FastTree 2–approximately maximum-likelihood trees for large alignments. *PloS One*, 5(3):e9490, 2010.

[136] J. Raes, K. U. Foerstner, and P. Bork. Get the most out of your metagenome: computational analysis of environmental sequence data. *Current Opinion in Microbiology*, 10(5):490–498, 2007.

[137] V. Ranwez and O. Gascuel. Quartet-based phylogenetic inference: Improvements and limits. *Molecular Biology and Evolution*, 18(6):1103–1116, 2001.

[138] Sébastien Roch. A short proof that phylogenetic tree reconstruction by maximum likelihood is hard. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(1):92–94, 2006.

[139] Sebastien Roch. Toward extracting all phylogenetic information from matrices of evolutionary distances. *Science*, 327(5971):1376–1379, 2010.

[140] Y. Roskov, T. Kunze, L. Paglinawan, T. Orrell, D. Nicolson, A. Culham, N. Bailly, P. Kirk, T. Bourgoin, G. Baillargeon, F. Hernandez, A. De Wever, and eds. Species 2000 and ITIS Catalogue of Life, 2013 Annual Checklist. *Species 2000*, 2013.

[141] Douglas B Rusch, Aaron L Halpern, Granger Sutton, Karla B Heidelberg, et al. The *Sorcerer II* Global Ocean Sampling Expedition: Northwest Atlantic through Eastern Tropical Pacific. *PLoS Biology*, 5(3):e77, 2007.

[142] Naruya Saitou and Masatoshi Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular biology and evolution*, 4(4):406–425, 1987.

[143] Michael J. Sanderson. R8s: inferring absolute rates of molecular evolution, divergence times in the absence of a molecular clock. *Bioinformatics*, 19(2), 2003.

[144] Fabian Schreiber, Peter Gumrich, Rolf Daniel, and Peter Meinicke. Treephyler: fast taxonomic profiling of metagenomes. *Bioinformatics*, 26(7):960–961, 2010.

[145] C. Semple and M. Steel. *Phylogenetics*, volume 24 of *Lecture Series in Mathematics and its Applications*. Oxford University Press, 2003.

[146] Martin Simonsen, Thomas Mailund, and Christian NS Pedersen. Rapid neighbour-joining. In *Algorithms in Bioinformatics*, pages 113–122. Springer, 2008.

[147] Gregory E. Sims, Se Ran Jun, G.A. Wu, and S.H. Kim. Alignment-free genome comparison with feature frequency profiles (FFP) and optimal resolutions. *Proceedings of the National Academy of Sciences*, 106(8):2677–2682, 2009.

[148] Sagi Snir, Tandy Warnow, and Satish Rao. Short quartet puzzling: A new quartet-based phylogeny reconstruction algorithm. *Journal of Computational Biology*, 15(1):91–103, 2008.

[149] Robert R Sokal and Peter HA Sneath. *Principles of numerical taxonomy.* Freeman, 1963.

[150] Erik L. L. Sonnhammer and Volker Hollich. Scoredist: A simple and robust protein sequence distance estimator. *BMC Bioinformatics*, 6:108, 2005.

[151] Alexandros Stamatakis. RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics*, 22(21):2688–2690, 2006.

[152] Manuel Stark, Simon A. Berger, Alexandros Stamatakis, and Christian von Mering. Mltreemap - accurate maximum likelihood placement of environmental dna sequences into taxonomic and functional reference phylogenies. *BMC Genomics*, 11:461, 2010.

[153] Michael Steel. The complexity of reconstructing trees from qualitative characters and subtrees. *Journal of Classification*, 9(1):91–116, 1992.

[154] Mike A. Steel. Distributions on bicoloured evolutionary trees., 1989. PhD thesis. Massey University.

[155] Mike A. Steel and Laszlo Szekely. Teasing apart two trees. *Combinatorics, Probability and Computing*, 16(6):903–922, 2007.

[156] Jens Stoye, Dirk Evers, and Folker Meyer. Rose: generating sequence families. *Bioinformatics*, 14(2), 1998.

[157] K. Strimmer, N. Goldman, and A. von Haeseler. Bayesian probabilities and quartet puzzling. *Molecular Biology and Evolution*, 14(2):210–211, 1996.

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

[159] James A Studier and Karl J Keppler. A note on the neighbor-joining algorithm of Saitou and Nei. *Molecular Biology and Evolution*, 5(6):729–731, 1988.

[160] Marc Sultan, Marcel H. Schulz, Hugues Richard, Alon Magen, Andreas Klingenhoff, et al. A global view of gene activity and alternative splicing by deep sequencing of the human transcriptome. *Science*, 5891(321):956–960, 2008.

[161] HJG Tolou, P Couissinier-Paris, J-P Durand, V Mercier, J-J de Pina, P De Micco, F Billoir, RN Charrel, and X De Lamballerie. Evidence for recombination in natural populations of dengue virus type 1 based on the analysis of complete genome sequences. *Journal of General Virology*, 82(6):1283–1290, 2001.

[162] Jakub Truszkowski, Yanqi Hao, Daniel G Brown, et al. Towards a practical $O(n \log n)$ phylogeny algorithm. *Algorithms for Molecular Biology*, 7:32, 2012.

[163] Peter J. Turnbaugh, Ruth E. Ley, Micah Hamady, Claire M. Fraser-Liggett, Rob Knight, and Jeffrey I. Gordon. The human microbiome project. *Nature*, 449:804–810, 2007.

[164] Igor Ulitsky, David Burstein, Tamir Tuller, and Benny Chor. The average common substring approach to phylogenomic reconstruction. *Journal of Computational Biology*, 13(2):336–350, 2006.

[165] Andrew M Waterhouse, James B Procter, David MA Martin, Michèle Clamp, and Geoffrey J Barton. Jalview version 2a multiple sequence alignment editor and analysis workbench. *Bioinformatics*, 25(9):1189–1191, 2009.

[166] Travis J. Wheeler. Large-scale neighbor-joining with NINJA. In *Proceedings of WABI 2009*, pages 375–389. Springer.

[167] Chung-I Wu and Wen-Hsiung Li. Evidence for higher rates of nucleotide substitution in rodents than in man. *Proceedings of the National Academy of Sciences*, 82(6):1741–1745, 1985.

[168] Gang Wu, Ming-Yang Kao, Guohui Lin, and Jia-Huai You. Reconstructing phylogenies from noisy quartets in polynomial time with a high success probability. *Algorithms for Molecular Biology*, 3, 2008.

[169] Gang Wu, Jia-Huai You, and Guohui Lin. A lookahead branch-and-bound algorithm for the maximum quartet consistency problem. In *Proceedings of WABI 2005*, pages 65–76.

[170] Gang Wu, Jia-Huai You, and Guohui Lin. Quartet-based phylogeny reconstruction with answer set programming. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(1):139–152, 2007.

[171] Ziheng Yang. Maximum likelihood estimation on large phylogenies and analysis of adaptive evolution in human influenza virus A. *Journal of Molecular Evolution*, 51(5):423–432, 2000.

[172] Pablo Yarza, Michael Richter, Jörg Peplies, Jean Euzeby, and Rudolf Amann others. The all-species living tree project: A 16s rrna-based phylogenetic tree of all sequenced type strains. *Systematic and Applied Microbiology*, 31(4):241 – 250, 2008.

[173] Leonid Zaslavsky and Tatiana A Tatusova. Accelerating the neighbor-joining algorithm using the adaptive bucket data structure. In *Proceedings of ISBRA 2008*, pages 122–133.

[174] Louxin Zhang, Jian Shen, Jialiang Yang, and Guoliang Li. Analyzing the Fitch method for reconstructing ancestral states on ultrametric phylogenetic trees. *Bulletin of Mathematical Biology*, 72:1760–1782, 2010.

[175] Derrick J Zwickl and David M Hillis. Increased taxon sampling greatly reduces phylogenetic error. *Systematic Biology*, 51(4):588–598, 2002.