The Dissertation Committee for Serita Marie Nelesen
certifies that this is the approved version of the following dissertation:

Improved Methods for Phylogenetics

Committee:

Warren A. Hunt Jr., Supervisor

Tandy J. Warnow, Supervisor

Robert S. Boyer

David M. Hillis

Craig R. Linder

Improved Methods for Phylogenetics

by

Serita Marie Nelesen, B.A.; B.S.; M.S.C.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2009

To my family: all of them!

Acknowledgments

My path to a doctorate began in Grade 3. I know this seems hard to believe, but I remember clearly the day in Math Club that Dr. Leo Jonker, a mathematics professor at Queen's University, encouraged me to pursue a doctorate. I was marvelling at Pythagorean theorem, and he told me that if loved math, as I obviously did, I should become a professor, at which point he would hire me. That was the beginning.

My path continued to junior year at Calvin College, where I was debating adding a Computer Science major to my Math major. I was chatting with friends before an Operating Systems class, saying that my overload of Math and CS classes were making me reconsider the CS major. The professor of the class, Dr. Joel Adams, overheard me, and told me that I should definitely keep the CS major and to go talk to him after class. When I did, he encouraged me to stick with Computer Science, and even to go to graduate school for CS instead of Math.

Dr. Jonker and Dr. Adams are just two of the people who have encouraged me to pursue this doctorate, or have otherwise helped me along the way. Friends and lab mates, teachers and advisers, colleagues and collaborators, each have been invaluable: Alison, Greg, Harry, Shel, Ron, Matt, Jenn, Maria, Adam, Raj, Heather, Jenny, Michael, Kevin, Sindhu, Shruthi, Rahul, Tandy, Warren, Randy, Bob Boyer, Usman, Luay, Tiffani, David, Tracy, Ruth, Shannon and many more from both the

IGERT program in Computational Phylogenetics and Applications to Biology at UT-Austin and from the CIPRES project.

But the biggest influence has been my family, each member of which has had a role in shaping me. My parents gave me confidence, and encouraged curiousity. They pushed me to do my best, all while making it clear that they will always love and respect me, no matter what. And they give great guidance and perspective. My sisters are the best friends I will ever have, regardless of where we live.

And finally, these acknowledgements would not be complete without Bob, but what can I say about my husband? He moved to Texas to be with me, and together we "make it work".

Improved Methods for Phylogenetics

Publication No. _____

Serita Marie Nelesen, Ph.D.
The University of Texas at Austin, 2009

Supervisors: Warren A. Hunt Jr.
Tandy J. Warnow

Phylogenetics is the study of evolutionary relationships. It is a scientific endeavour to discover history, and it is not easy. Massive amounts of data together with computationally difficult optimization problems mean that heuristics are prevalent, and ever better techniques are sought. New approaches are valuable if they are more accurate, but are considered even more so if they are faster than pre-existing methods. Improvements to existing algorithms, whether in terms of space requirements, or faster running times, are also worthwhile. This dissertation explores three new techniques, each of which is valuable according to the previous definitions.

The first contribution is TASPI, a system for storing collections of phylogenetic trees, and performing post-tree analyses. TASPI stores collections of trees more compactly than the previous method, and this compact structure lends itself to

post-tree analyses. This results in the ability to compute strict and majority consensus trees faster than common alternatives. As an added benefit, TASPI is written in ACL2, which allows properties of the algorithms and data structures to be formally verified.

The second contribution is an improved method to generate phylogenetic trees. A common methodology involves two steps, first estimating a Multiple Sequence Alignment (MSA), and then estimating a tree using that MSA. This method changes the way in which the MSA is estimated, and this leads to improved accuracy of the resultant trees. Also, in some cases, the time required is also reduced.

The third contribution is BLuTGEN, a method by which a phylogenetic tree is estimated from sequence data, but without ever generating an MSA for the full dataset. BLuTGEN is as accurate as one of the best published tree estimation techniques (SATé), but takes a novel approach which allows it to be applied to much larger datasets.

Table of Contents

Chapter 1

Introduction

Phylogenetics is the study of evolutionary relationships. These relationships are often represented as a phylogenetic tree, or phylogeny. Biological phylogenies represent the evolutionary relationships of a set of organisms (taxa) and are used to answer a variety of questions ranging from specific hypothesis (e.g. Did wings, flippers and arms all evolve from the same structure in some ancient organism?) to more general queries (e.g. Why does Columbia have more species of birds than any other country?).

Phylogenetic trees are also used to answer questions in fields other than biology. For example, phylogenetics have been used in forensics to determine the sources of infection from HIV [37], and to predict gene function, which can be used for drug discovery [84]. Phylogenetic trees are also used in security applications (such as artificial immune systems for computers) and historical linguistics (seeking to understand how languages have evolved) [17].

The process of creating a phylogeny is complex, from several perspectives. A biologist must procure appropriate data, and choose some analysis to perform; a computer scientist develops the analyses to be performed; a mathematician tries to understand the theoretical guarantees provided by each of the possible analyses.

Each contributes to the process of building a phylogeny. As a computer scientist, in this dissertation I will focus on the challenges inherent in designing accurate and efficient algorithms for use in phylogenetic analyses.

To begin, let us consider the high-level steps that are usually performed in generating a phylogeny. First, representative data are collected for each of the taxa being studied. These data can take many forms, although the most common include sequence data, gene order data, and historically, morphological data. Second, these data are aligned to ensure that comparable information is considered as input to the tree producing step. The third step is to produce one or more candidate trees. There are many techniques for creating these trees, some of which return a single tree, such as Neighbor-Joining [76] and other distance based-methods, and other methods that potentially return several trees, such as heuristic search methods attempting to optimize a maximum parsimony or maximum likelihood criteria. In each case, the accuracy of the tree(s) returned must be assessed. More recently, Bayesian MCMC sampling methods have been used to jointly estimate phylogeny and assess confidence in the tree produced.

When using a heuristic search method, and also in a Bayesian context, there are potentially many trees to be considered. Many programs for phylogenetic inference (e.g. PAUP [94], GARLI [102]) recommend saving more than one tree during their search for an optimal solution. A commonly used program for Bayesian inference, MrBayes [40], saves hundreds and more often thousands of trees to a file as it considers various trees as potential phylogenies.

In order to glean useful information from these possibly large collections of

trees, post-tree analyses are performed. These include consensus analyses which seek to summarize the information in the trees, other techniques to determine the common elements of the collection (e.g. MAST, MCT, clustering methods) and various filtering methods (e.g. investigating which trees agree with a pre-determined constraint).

Recently, this usual pipeline has been modified to group the alignment and tree generation steps [28,54,71,98]. Remember, this process first estimates an alignment and then uses this fixed alignment to estimate a tree. But this fixed alignment likely has errors, and so we are compounding errors when we use it as input to the tree producing algorithms. Instead, methods have been developed that attempt to reconstruct the alignment and phylogeny simultaneously. Of course, this is a much more computationally intensive endeavour since computing an alignment and a tree on that alignment are each in and of themselves hard problems.

Which brings us to just how hard this process is: very. Although sequencing advances have helped biologists gather data, this means there is an explosion of sequence data available for processing. And the algorithms in phylogenetics have not kept up. New techniques are required to manage this plethora of data and to boost our current techniques to handle the data effectively. New methods that work in entirely new ways are also needed.

This dissertation contributes to the process of discovering relationships in three ways. First, TASPI (described in Chapter 3) is a new method for handling large collections of trees. Second, Chapter 4 describes a boosting method for generating better alignments for use in reconstructing trees using existing tools. The

third contribution is BLuTGEN, a new method for constructing phylogenies which works by dividing the dataset into subproblems in novel ways (described in Chapters 5 – 9). Chapter 2 introduces a number of concepts that will be used throughout the rest of this dissertation, and Chapter 10 wraps up with a discussion of future work.

Chapter 2

Background

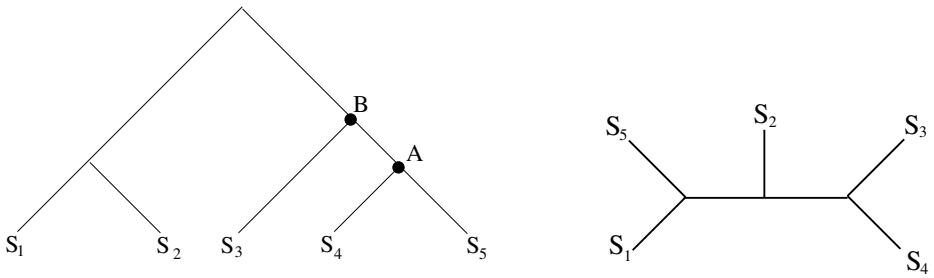
This chapter covers the terminology and concepts used throughout the rest of this dissertation.

2.1 Definitions

2.1.1 Trees

Phylogenetic Tree A phylogeny, or phylogenetic tree, is a representation of the evolutionary relationships amoung the group of taxa contained in the tree. This representation takes the form of an acyclic graph, where the nodes indicate a particular taxon, and the edges indicate evolutionary history. Terminal nodes, or leaves, usually represent extant species, whereas internal nodes represent ancestral species. A rooted tree has a special node that is marked as the root. This root allows us to associate time with the tree, such that if a node A has a path to the root through another node B in the tree, then A is considered a descendant of B. Unrooted trees have no such time axis. See Figure 2.1 for examples.

While rooted trees are more intuitive, placing the root on a tree is a difficult problem. Thus, I will usually test my algorithms on unrooted trees, since much useful information can still be gleaned from an understanding of the relationships



(a) Rooted Tree (b) Unrooted Tree

Figure 2.1: **Rooted and unrooted trees**. Figure 2.1(a) shows ancestral nodes A and B, where A is a descendant of B, as are sequences S_3, S_4 and S_5. Figure 2.1(b) shows an unrooted tree.

as presented by an unrooted tree.

Clade Defined for a rooted tree, a clade is the full set of taxa which share a most recent common ancestor that is not shared by any other taxon in the tree. For example, in Figure 2.1(a) sequences S_1 and S_2 form a clade, while S_3 and S_5 do not.

Edge (Bipartition) An edge, or branch, in the tree separates the taxa at the leaves into two sets. Each of these pairs of sets, one for each edge in the tree, is referred to as a bipartition (since it partitions the leaves into two sets). A bipartition of a tree is denoted by $x|y$, where $x \cup y$ is the complete leaf set of the tree. An edge is an internal edge if it connects two internal nodes, that is, neither of its endpoints is a leaf.

Compatibility and Conflict A collection of bipartitions is considered compatible if it is possible for a tree to have each and every bipartition in the set simultaneously. That is, if there is a single tree which contains each of the edges indicated by

a collection of bipartitions, the collection is said to be compatible. A set of compatible bipartitions uniquely defines a tree. Two bipartitions are said to conflict if they are not compatible, that is, they cannot both be present in a tree. Consider the bipartitions $x|y$ and $u|v$, both with the same leaf set. If $x \subset u$ or $x \subset v$ the two bipartitions are compatible.

Resolution Let T be a tree with n leaves and m internal edges. The resolution of T is the proportion of internal edges present in T compared to the number of possible internal edges. There are at most $n - 3$ internal branches in T, so the resolution of T is

$$\frac{m}{n-3}$$

expressed as a percentage.

2.1.2 Alignments

Multiple Sequence Alignment An alignment is also a representation of relatedness, but on a finer scale. This representation takes the form of a matrix, where each row contains information for a particular taxon, and each column indicates shared history. I will be focusing on alignments of nucleotide data, so each row will be the sequence of DNA characters (i.e. strings over the alphabet "A,C,T,G") together with a gap character, "-". The DNA characters make up the raw data collected by the scientists. By adding gaps in appropriate positions, a multiple sequence alignment puts the DNA characters into columns such that all nucleotides in a column are hypothesized to share a common ancestor. Note that the order of the DNA characters cannot be modified, so the only operation allowed going from raw sequences

```
AGGCTATCACCTGACCTCCA
TAGCTATCACGACCGC
TAGCTGACCGC
TCACGACCGACA
```

(a) Raw/Unaligned Sequences

```
-AGGCTATCACCTGACCTCCA
TAG-CTATCAC--GACCGC--
TAG-CT-------GACCGC--
-------TCAC--GACCGACA
```

(b) Aligned Sequences

Figure 2.2: **Raw (unaligned) and aligned sequences.** Using Figure 2.2(a) shows four unaligned sequences, and Figure 2.2(b) shows the same sequences aligned.

to an alignment is the addition of gaps. See Figure 2.2 for an example.

An induced alignment is a restriction of a multiple sequence alignment to a limited number of rows. If any of the columns in this restricted alignment are entirely gapped, they are disregarded.

Gappiness Gappiness is one measure of the amount of evolution in a multiple sequence alignment, where higher gappiness indicates higher rates of evolution. Setwise gappiness is the percentage of the multiple sequence alignment matrix occupied by gaps. Edgewise gappiness is defined in terms of the edges in the tree together with sequences at every node. Then, for an edge, the gappiness is the percent of the pairwise alignment that is gapped. I will usually refer to the average edgewise gappiness of an alignment, which is the average of the edgewise gappinesses for each edge in the tree.

Normalized Hamming Distance Hamming Distance is another measure of the amount of evolution in an multiple sequence alignment. Given a pairwise alignment, the classical Hamming Distance [36] is simply the number of positions in the alignment which are different. I will be using a modified version where I only

consider positions in the alignment where neither sequence has a gap. Using this definition of Hamming Distance, the normalized Hamming Distance (NHD) is the Hamming Distance, divided by the length of the sequence (not counting gapped positions). Thus, the NHD could be computed by first removing all gapped positions in the alignment. Then, using this restricted alignment as input, the NHD is the number of positions that are different divided by the length of the alignment. As with gappiness, the NHD can be computed for every edge in the tree when there are sequences at internal nodes. Taking the average of all of these edgewise NHD's results in the edgewise ANHD. The setwise ANHD is computed by instead averaging the NHD for all pairwise induced alignments from a multiple sequence alignment. Note that the setwise ANHD is calculated using only the sequences from the leaf taxa, while the edgewise ANHD requires sequences at internal nodes. Finally, the setwise maximum NHD for a multiple sequence alignment is the maximum NHD for any induced pairwise alignment.

2.2 The Phylogenetic Process

The traditional way of producing a phylogeny for a set of taxa involves four major steps:

1. Gather comparable data for the taxa of interest (for our purposes, DNA sequences).

2. Create an alignment for the sequences gathered.

3. Generate trees from the alignment.

4. Post-process the data in some way.

I will not elaborate on the techniques used to gather the source (raw) data, but some background on multiple sequence alignment techniques and tree generation will be useful. Also, consensus methods are amoung the most common post-processing techniques, and the type on which I will focus.

2.2.1 Multiple Sequence Alignment Estimation

Computing a multiple sequence alignment is a difficult problem. At times accomplished manually, where a biologist makes a best guess for positional homology, more objective criteria have also been developed. However, even the best criteria are not guaranteed to prefer the evolutionarily correct alignment.

First defined for pairwise alignments, the Sum-of-Pairs criterion is one of the most commonly used criteria for choosing between alternate alignments. Framed as an optimization problem, the best alignment for a pair of sequences is the alignment that optimizes the Sum-of-Pairs score (SOP-score). This score is defined in terms of the paired positions in the alignment, where matches improve the score, and mismatches degrade the score. Gaps can also contribute to the score, with several different models of gap treatment having been developed. With a linear gap penalty, any gap character in the alignment degrades the score equally. Alternatively, with an affine gap penalty, gaps are considered as a function of their length, with separate "gap open" and "gap extend" penalties.

Happily, given the parameters to the SOP-score, it is possible to efficiently

compute an optimal alignment for a pair of sequences using an algorithm first developed by Needleman and Wunsch [64] and refined by many others (for example, Sankoff improved the algorithm for gap treatment [80] and Hirschberg improved the space requirement of the algorithm [39]). Unhappily, extending these algorithms to multiple sequence alignment does not preserve the efficiency of the algorithms. In fact, multiple sequence alignment has been shown to be NP-hard [99] [46].

So instead of trying to solve this sort of optimization problem exactly, heuristics are used. Many of the heuristics seek to find a multiple sequence alignment that optimizes the total SOP-score (which is the sum of the scores for each of the induced pairs of sequences), though there are many other approaches as well. Heuristics tend to fall into two categories (though some methods use both, or neither): progressive and iterative. A progressive method uses an estimate of the relationships between the sequences (a guide tree) to inform the order in which sequences are added to the alignment (see Chapter 4 for further details). In a progressive method, once a sequence is part of the alignment, the induced pairwise alignments already present do not change. An iterative method is similar to a progressive method, but relaxes the restriction that the induced pairwise alignments may not change. Instead, as sequences are added, the partial multiple sequence alignment can be refined using the information added by subsequent sequences. Progressive methods are the most widely used approach since they are faster than the alternatives.

2.2.2 Tree Estimation

Even before sequence data was available, scientists attempted to estimate what the phylogenetic tree for a set of taxa should be. Over the years, many techniques for tree estimation have been developed including distance-based methods such as UPGMA [86] and NJ [76], score optimization methods such as Maximum Parsimony and Maximum Likelihood, and more recently, Bayesian methods [26] [38]. In this dissertation I will mostly focus on Maximum Parsimony and Maximum Likelihood.

Maximum Parsimony (MP) defines the parsimony score of a tree with sequences at the leaves, and seeks to find the tree that minimizes this score. The parsimony score is a measure of the minimum amount of change necessary over the tree to produce the sequences at the leaves. While determining the score of a tree is not difficult (it can be accomplished in linear time [81]), finding a tree that minimizes this score is NP-hard [18].

Maximum Likelihood (ML) also defines a score of a tree with sequences at the leaves. The likelihood score of a tree, given a model of evolution and lengths on the branches, is the probability of the sequences given the tree and the model. While finding the MP score of a tree is not difficult, even just scoring a given tree under ML is computationally intensive since branch lengths must first be optimized. Again, finding a tree that maximizes the likelihood score is an NP-hard problem [14].

Given that these optimization problems are unlikely to be solved exactly in reasonable amounts of time, heuristics have been developed. These tend to be

hill-climbing searches, where instead of considering every possible tree, a subset of trees with successively improved scores are considered until no better scoring tree can be found (using the transitions allowed by the search method). These methods are prone to local optima and tree islands, defined by David Maddison as a set of trees "all less than a specified length, each tree connected to every other tree in the island through a series of trees, and each one differing from the next by a single rearrangement of branches" [58]. In an attempt to find a globally optimal tree instead of a sub-optimal one, techniques such as using randomization in building an initial tree from which to begin the search, and methods designed to move a search away from a local optimum have been developed. Since we are unable to know for certain that we have found the optimal solution, many times a phylogeneticist will keep several of the best trees for further comparison and exploration. This is especially true when the search to find the best trees has required considerable time (weeks and months).

Thus, these techniques rarely result in a single optimal tree. Instead, there are often many trees that a phylogeneticist would like to save for further processing. These post-tree analyses take many forms, but consensus analysis, which is used to summarize a collections of trees, is one of the most common.

2.2.3 Consensus

Consensus trees are defined by Felsenstein as "trees that summarize, as nearly as possible, the information contained in a set of trees whose tips are all the same species" [26]. There are many different kinds of consensus, each stressing

a different commonality or difference between the input trees.

Consensus methods return a single tree, or an indication that no tree meeting the requirement of the method is possible. A consensus tree is created from the input trees based upon some criterion. Two of the most common types of consensus trees are strict and majority [59]. Both of these decide which edges in the input trees to keep, and then build a tree from the resulting edges.

Strict consensus requires that every edge in the consensus tree be an edge in every input tree. Majority consensus is less rigid, only requiring that every edge in the consensus tree appear in a majority of the input trees. Originally defined in terms of the usual sense of majority (i.e. where majority means more than 50%), it is also possible to define majority consensus in terms of a threshold parameter which indicates the required amount of agreement for inclusion. As long as this threshold is set above 50%, the majority consensus tree is well defined. Note that strict consensus is a special case of majority consensus; that is, it is a majority consensus with a threshold of 100%. Strict and majority consensus algorithms always return a tree (again, when the threshold is set above 50%), and can be computed efficiently. Strict consensus has an optimal O(kn) algorithm as described by Day [19] while majority consensus has an expected O(kn) randomized algorithm as described by Amenta et al. [2] (where k is the number of trees and n is the number of taxa).

It is important to note that all consensus methods, though they may return a tree, rarely return a tree that is most parsimonious, nor does the returned tree achieve the best likelihood score. Thus, there has been some debate about using consensus methods to infer phylogenies [11]. However, as long as the merits and

limitations of each consensus method remain in perspective, the trees produced by consensus methods can be very useful.

2.3 Simulation Studies

Though my goal is to create algorithms that perform well (i.e. accurately and quickly) for datasets that are collected from real biological organisms, it is essentially impossible to have perfect real data, where both the historical alignment is known, as well as the actual branching pattern of all sequences under study. Even if we have sequences collected in a wet lab over a period of time where a scientist has carefully controlled the environment (i.e. experimental evolution, for example with E. coli [7]), this still only gives partial information; the exact progression of changes to the sequences remain unknown. In the absence of these perfect real data, we instead make use of simulation studies, where the true alignment and phylogeny are known so that we can compare the results of our algorithms to truth to determine how well they work.

The simulation studies I perform usually have four steps:

1. Generate a model tree.
2. Generate sequences on the model tree.
3. Estimate alignments and trees from raw sequences.
4. Compare estimates of alignments and trees to true alignment and true tree.

In this dissertation, I use several variations of this basic process, so I will detail the specifics for each experiment in the appropriate chapter. The utility of simulation studies lies in the fact that because I create the data, there is a known true alignment and true tree with which to compare. Of course, simulation studies are also limited in usefulness since the models used to generate the data are imperfect. Thus, even if a method performs extremely well in simulation, this does not guarantee that the method will perform equally well for biological data. However, a method that works well in simulation is at least a good candidate for working with biological data. To assess the accuracy of the estimation techniques I develop, I need a way to compare the estimated alignments and trees to the known true tree and true alignment.

2.3.1 Assessing Alignment Methods

To measure alignment accuracy, I use the SP (sum-of-pairs) error rate (the complement of the SP accuracy measure, and denoted SP_{FN}), which I now define. Let $S = \{s_1, s_2, \ldots, s_n\}$, and let each s_i be a string over some alphabet Σ (e.g., $\Sigma = \{A, C, T, G\}$ for nucleotide sequences). An alignment on S inserts spaces within the sequences in S so as to create a matrix, in which each entry of the matrix contains either a dash or an element of Σ. Let s_{ij} indicate the j^{th} letter in the sequence s_i. We identify the alignment A with the set $Pairs(A)$ containing all pairs $(s_{ij}, s_{i'j'})$ for which some column in A contains s_{ij} and $s_{i'j'}$. Let A^* be the reference alignment, and let $\hat{A}$ be the estimated alignment. Then,

$$SP_{FN} = \frac{|Pairs(A^*) - Pairs(\hat{A})|}{|Pairs(A^*)|},$$

expressed as a percentage; thus SP_{FN} is the percentage of the pairs of homologous nucleotides in the reference alignment that are unpaired in the estimated alignment. Note that because pairs in the reference alignment that include a nucleotide and a gap character are not considered, it is possible for SP_{FN} to be 0, and yet have differences between the reference and estimated alignments.

The analogous SP_{FP} is similarly defined, and is the percentage of the pairs of nucleotides that are paired in the estimated alignment, but are unpaired in the reference alignment. I will tend to only consider SP_{FN}, since SP_{FP} favours alignments that are highly gapped.

2.3.2 Assessing Tree Reconstruction Methods

To measure tree accuracy, I will most often use the missing edge rate, which is the percentage of the edges of the reference tree that are missing in the estimated tree (we assume both trees have the same leaf set). This is also known as the false negative rate, and will thus be denoted by FN. To be precise, let r be the number of internal edges in the reference tree, and let j be the number of edges in the reference tree not found in the estimated tree. Then

$$FN = \frac{j}{r}$$

expressed as a percentage.

As with alignments, there is also an analogous false positive rate (FP), which is the percentage of edges in the estimated tree that are missing in the reference tree. Again, to be precise, let m be the number of internal edges in the estimated tree,

and let i be the number of edges in the estimated tree not present in the reference tree. Then

$$FP = \frac{i}{m}$$

expressed as a percentage.

Notice that if both trees are binary, that is, fully resolved, then $m = r$ and $i = j$, so $FN = FP$. However, in simulation, the reference tree is the "potentially inferable model tree" (PIMT), which is obtained by contracting the zero-event edges in the model tree. An edge is a "zero-event" edge if the sequences at each end of the edge are identical. In this case, there are no data which will support this edge, and therefore I do not expect any algorithm to be able to reconstruct such an edge.

As with alignment error, I will focus my attention on the FN rate, because a tree with no internal edges will always have a zero FP, and thus FP favours trees with low resolution.

Chapter 3

Storing Collections of Trees

As available sequence data increases, the complexity of storing these data, as well as storing the results of their analyses, also increases. With Warren Hunt and Bob Boyer, I have developed a new method for storing and retrieving phylogenetic tree data, and using this method we have implemented a consensus algorithm. Our approach permits very large data sets to be compactly stored and retrieved without any loss of precision. Also, our implementation of a post-processing technique (strict and majority consensus) provides greatly increased performance as compared to PAUP [94] and TNT [31]. The material in this chapter was first published in two conference publications [9] [45].

Our software is called the Tree Analysis System for Phylogenetic Inference (TASPI), and it is an experimental system, written from scratch. It is a stand alone tool for a few kinds of phylogenetic data manipulation. TASPI is written in the ACL2 [50] formal logic, where all operations are represented as pure functions. Using ACL2's associated mechanical theorem prover, I have proved a variety of assertions about the TASPI system.

This chapter is organized as follows: I first explain how collections of trees might arise, and then describe our representation of phylogenetic trees, and how

this format reduces the storage requirement for a collection of trees. I also note a variety of properties that I have defined for trees in the system, and show an alternate representation of trees that is less storage efficient, but useful for proving properties of the algorithms. I next consider computing strict and majority consensus trees, and show our algorithm that exhibits improved performance as compared to currently available software. Finally, I describe an empirical study exploring the storage requirements for collections of trees, and the time requirements to compute consensus trees.

3.1 Collections of Trees

In Section 2.2.2 I described a few ways in which collections of trees are produced from a single data set, from a single heuristic search. I also mentioned Bayesian methods, where a Markov chain Monte Carlo method is used to simulate a random walk through tree space. Using this method, trees are visited in proportion to their posterior probabilities. By considering the trees visited, inferences about phylogeny can be made. But this requires saving all (or some sampling) of the trees. In order to give accurate information about the phylogeny, the size of the set of trees saved can be quite large (tens to hundreds of thousands of trees).

Collections of trees can also arise from multiple data sets. For example, if a biologist is studying a group of organisms, they may consider multiple genes. A phylogenetic study using one gene may imply one phylogeny for the species, while basing the study on a different gene may give different results. Each study may result in a collection of trees, or the results from each study may be combined into

a collection.

In each of these cases, collections of trees with the same set of taxa are produced, and sometimes these collections are extremely large (particularly in the Bayesian context, but sometimes there are very large tree islands as well). A phylogeneticist would like to save all of these trees for further analysis, as well as to preserve their scientific method. Unfortunately, the traditional method for storing these trees is extremely large text files.

Indeed, the need for a database system for such collections of trees has been recognized for some time [78]. However, while databases for collections of sequence data have been well developed (e.g. GenBank [6]), limited work has been done for trees. TreeBASE [79, 97] is a first attempt at storing trees, but has been designed to store published trees, most often only an individual tree or perhaps a few, and never the entire set generated.

While saving all trees generated during an analysis is important, it is only half of the issue. Once the trees are generated, post-processing is required in order to glean information from the collection of trees. There are a variety of operations used to accomplish this task. The most often used of these are consensus methods (see Section 2.2.3), which summarize the collection of trees with a single tree on the full leaf set, but there are many others as well, such as Maximum Agreement Subtree, which instead finds the subtree common to all trees in the collection with the largest number of a taxa.

3.2 Representation

Newick format [25] is the standard way of storing a collection of phylogenetic trees. Adopted in 1986, Newick is a notation that uses commas to separate sibling subtrees, parentheses to indicate children, and a semicolon to conclude a tree. Newick outlines each tree in its entirety whether storing one tree, or a collection of trees.

On the other hand, TASPI capitalizes on common structure within a collection of trees. TASPI stores a common subtree once, and then each further time the common subtree is mentioned, TASPI references the first occurrence. This saves considerable space since potentially large common subtrees are only stored once, and the references are much smaller (for empirical results see Section 3.4.3).

There are two layers to the TASPI representation of trees. At a high-level, trees are represented as Lisp lists, similar in appearance to Newick, but without commas and semicolons. This is the format presented to the user of TASPI and on which user functions operate. At a low-level, the data are instead represented in a form that uses hash-consing [32] to achieve decreased storage requirements and improved accessing speeds. This representation makes use of Lisp-style expression identification and allows circular graphs to be built. For ease of reference in Section 3.4.3, I call this the Boyer-Hunt compression.

Consider the following set of rooted trees in Newick format:

```
(A,((B,(C,D)),E));
(A,((E,(C,D)),B));
(A,(B,(E,(C,D))));
((A,B),(E,(C,D)));
```

The format of these trees presented to the user of TASPI is straightforward:

```
(A ((B (C D)) E))
(A ((E (C D)) B))
(A (B (E (C D))))
((A B) (E (C D)))
```

Notice that storing this set of trees involves restoring the subtree containing taxa `C` and `D` once for every tree. The Boyer-Hunt compression instead stores the `C-D` clade once, the first time it is encountered. If, subsequently, the `C-D` clade is encountered again, the first time is marked with "#n=" for the current value of a counter n that is incremented each time it is used. Then, instead of re-storing the `C-D` clade, a reference in the form "`#n#`" is stored in its place. This compression has parallels to the Lempel-Ziv data compression which is based only on characters seen so far [101]. The compressed version of the trees above is given below:

```
((A ((B #1=(C D )) E ))
(A (#2=(E #1#) B))
(A (B #2#))
((A B)#2#))
```

We use a technique sometimes called hash-consing, which ensures that no object is ever stored twice. In the context of phylogenetic trees, an object is a subtree, and consing is a tree constructor that joins subtrees. Hashing, put simply, is a technique that creates a table that allows for fast searches. In this case, hashing is used to quickly determine if a subtree was previously encountered. The format, using "`#n=`" and "`#n#`", is a standard read dispatch macro from Lisp programming [91].

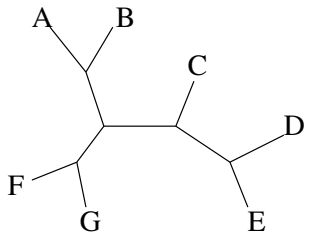


Figure 3.1: **An unrooted tree.** Each unrooted tree has many Newick representations.

Two subtleties remain to be addressed. First, though we will be presenting rooted trees, not all trees are rooted. In fact, most tree searching algorithms return unrooted trees since determining the root of a tree may itself be difficult [26]. Newick format does not distinguish between rooted and unrooted trees except through the use of auxiliary flags. By placing `[&R]` and `[&U]` just before the beginning of a tree, rooted and unrooted trees, respectively, are indicated. Without these flags, the onus is on the user to interpret the trees appropriately.

Second, Newick does not give a unique representation for a tree. Consider the tree in Figure 3.1. There are many representations for this tree in both Newick and TASPI. Possible TASPI representations include:

`((F G) ((A B) (C (D E))))` and

`((C (E D)) ((B A) (G F))).`

To ensure a unique answer in our computations, we order the output with respect to an ordering on the taxa. As far as we can tell, PAUP also does this. Thus, given an alphabetical ordering, we would order the tree above as:

`(A B ((C (D E)) (F G))).`

```
(defun taspip (flg tree)
  (if flg
      (if (atom tree)
          (or (and (symbolp tree)
                   (not (equal tree nil)))
              (integerp tree))
        (taspip nil tree))
    (if (atom tree)
        (null tree)
      (and (taspip t (car tree))
           (taspip nil (cdr tree))))))
```

Figure 3.2: **ACL2 definition of** `taspip`**.**

3.2.1 Properties of Trees

Since TASPI is built within ACL2, I have written predicates that together recognize well-formed trees. First, `taspip` (see Figure 3.2) checks that a tree is made up of symbols and integers for leaves (though `nil` is not a valid taxa name), and that any time there are sibling subtrees, they are part of a `true-listp`. Second, `ordered-taspi` (Figure 3.3) checks that a tree follows a specified ordering. Any sibling taxa are to be in the order given, and sibling subtrees must have the first taxa in each of their representations in the correct order.

I use these predicates throughout my code to ensure that the trees within the system are in a valid and consistent state. I also use these as hypotheses for theorems about algorithms which take trees as input.

```
(defun first-taxon (term)
  (if (atom term)
      term
    (first-taxon (car term))))

(defun ordered-taspi (tree order)
  (if (consp tree)
      (if (consp (cdr tree))
          (and (< (cdr
                     (assoc-equal
                       (first-taxon tree)
                       order))
                  (cdr
                     (assoc-equal
                       (first-taxon
                          (cdr tree))
                       order)))
               (ordered-taspi (car tree)
                              order)
               (ordered-taspi (cdr tree)
                              order))
        (ordered-taspi (car tree) order))
    t))
```

Figure 3.3: **ACL2 definition of** `ordered-taspi`**.**

3.2.2 Alternate Representation of Trees

Lisp programmers can easily see how the parenthetical notation given above is intuitive for representing trees. However, for several algorithms using trees as input, having a representation more directly related to the edges of the tree is useful. Biologists recognized this fact as well, and for this purpose use a "bipartition representation" of trees, which is simply a listing of the set of bipartitions in the tree. For example, the tree in Figure 3.1 can be represented as:

```
AB | CDEFG
ABFG | CDE
ABCDE | FG
ABCFG | DE
```

Notice that this gives us an unrooted tree. If we want a rooted tree we must specify a given partition as indicating the root. Also, trivial bipartitions, a single taxon versus the rest of the taxa, are not usually written, since every tree containing that taxon has that bipartition.

At times, we use bipartitions in TASPI, but we represent them slightly differently. Instead of writing down both sides of the bipartition, we only write one, since we can infer the other side from a single taxa-list. A valid bipartition is recognized by `good-partp` (Figure 3.4), which checks that there are no duplicate taxon names, that the taxon names are either a symbol or integer, and that there are at least two taxon names in the bipartition. In order to move between a bipartition representation of trees and the parenthetical notation, we have functions that transform one representation into the other: `fringes` (Figure 3.5) and `partstotaspi` (Figure 3.6).

```
(defun good-partp (x)
  (and (int-symlist x)
       (no-duplicatesp-equal x)
       (< 1 (len x))))
```

Figure 3.4: **ACL2 definition of** `good-partp`**.**

```
(defun fringes (flg tree order)
  (if flg
      (if (consp tree)
          (cons
             (ofringe t tree order)
             (append
                 (fringes
                     t (car tree) order)
                 (fringes
                     nil (cdr tree) order)))
        nil)
    (if (consp tree)
        (append (fringes
                    t (car tree) order)
                (fringes
                    nil (cdr tree) order))
      nil)))
```

Figure 3.5: **ACL2 definition of** `fringes`**.**

```
(defun partstotaspi (top under order)
  (if (consp under)
      (orderly-cons
         (partstotaspi
             (car under)
             (get-subsets (car under)
                          (cdr under))
             order)
         (partstotaspi
             (difference top (car under))
             (get-non-subsets (car under)
                              (cdr under))
             order)
         order)
    top))
```

Figure 3.6: **ACL2 definition of** `partstotaspi`**.**

`Fringes` takes a tree in parenthetical notation, and produces a list of bipartitions where each bipartition has been ordered according to the given order (the function `ofringe` creates an ordered bipartition). If given a non-nil flag, the first element of the list is not a bipartition, but instead an ordered list of the taxa in the tree. `partstotaspi` takes a list giving all taxa in the tree (`top`), the list of bipartitions (`under`) and an ordering in which to create the parenthetical notation. By recursing through the list of bipartitions specified with `get-subsets` and `get-non-subsets` we achieve the invariant that each of the bipartitions in `under` is a subset of `top`.

These functions are not tail-recursive, and as such are not the most efficient implementation. However, these simplified definitions allow us to prove the theorem shown in Figure 3.7. This theorem says that if given a good ordering of taxa in the tree, and a tree that is in a good form and appropriately ordered, then applying `partstotaspi` to the result of `fringes` gives back the original tree.

3.3 Our Consensus Algorithm

To compute a strict or majority consensus tree, our algorithm proceeds by:

1. Producing a replete association list of all of the subtrees in the original input,
2. Counting the frequencies of the non-tip subtrees,
3. Collecting the subtrees that appear as often as the designated majority threshold, and finally,

```
(defthm fringes-partstotaspi-inverse
  (implies
     (and (ordered-taspi x order)
          (good-order-list order)
          (not (and flg
                    (not (consp x))))
          (no-duplicatesp-equal
            (strip-cdrs order))
          (no-duplicatesp-equal (mytips x))
          (subset (mytips x)
                  (get-taxa-from-order order))
          (taspip flg x))
     (if flg
         (equal (partstotaspi
                  (car (fringes flg x order))
                  (cdr (fringes flg x order))
                  order)
                x)
       (equal (partstotaspi
                  (ofringe flg x order)
                  (fringes flg x order)
                  order)
              x)))
  :rule-classes :nil)
```

Figure 3.7: **ACL2 definition of** `fringes-partstotaspi-inverse`**.**

4. Constructing the consensus tree.

Step one is accomplished by our function `replete-trees-list-top`, step two by `fringe-frequencies`, step three with a simple recursion through the result of `fringe-frequencies`, and finally the consensus answer is computed by `build-term-top`.

Function `replete-trees-list-top` converts the original input list of trees into a replete association list (database). This replete database is a mapping from subtrees to every parent (sub-)tree containing the subtree. More precisely, this function takes as input a list `lst` of non-tip trees no member of which is a proper subtree of another, such as a list of trees all with the same set of taxa (non-tip simply means that there are at least two taxa). The function then returns a replete association list `db` such that:

1. `x` is a member of the domain of `db` if and only if `x` is a member of `lst` or is a non-tip proper subtree of a member of `l` and

2. If `x` is in the domain of `db`, then `db(x)` is an integer if and only if `x` is a member of `lst` and `db(x)` is the number of times `x` occurs in `lst`.

For an example execution of `replete-trees-list-top`, see Subsection 3.3.1.

Now, the function `fringe-frequencies` expects input of the form that `replete-trees-list-top` produces. This function counts the frequencies of every subtree fringe (that is, leaf set) in the replete database by iterating through the replete database. It then returns a minimal length association list that pairs each

fringe of some subtree in the list that created the replete database with the number of times that fringe occurred in the input.

By scanning through the resulting association list, we just pick out the fringes that appear as often as the desired threshold. We have no need to store the actual number of times any specific fringe appears; we simply collect the desired fringes into a list. Notice that these fringes map to bipartitions, and thus define the edges in a tree.

The function `build-term-top` takes two arguments. The first argument is a sorted list of fringes. This list is sorted using a lexicographic (normalization) order that is based both on the taxa names and the number of the elements in each fringe. The second argument is a normalization taxa list `tx`, that is used by our lexicographic ordering function so we can produce a unique representation. Remember, we represent each subtree as a list of subtrees, so to make the representation unique we sort members of each subtree. `build-term-top` constructs a consensus answer tree recursively by first building an answer based on the first non-full fringe of the input list (if the input set was a valid set for consensus, the first fringe will be for the complete set of taxa, which was common to all trees). Once the first answer subtree is computed for the first element in the list, any (sub-)subtrees required to build the first subtree are "crossed out" from the list that remain to be processed, and we continue with the next remaining element until no entries remain.

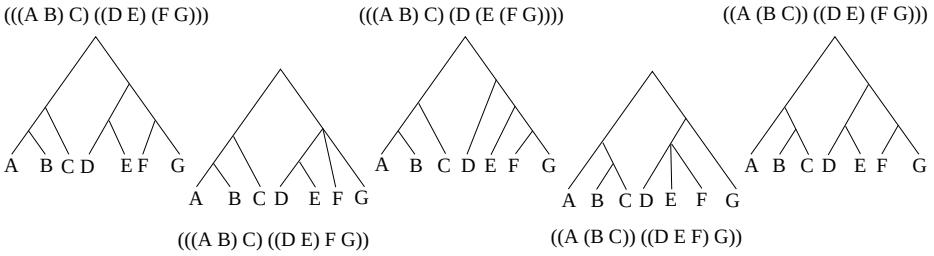


Figure 3.8: **A collection of trees together with their TASPI representations.**

3.3.1 Example

Consider the five trees in Figure 3.8. The TASPI representation of these trees is the input to the function `replete-trees-list-top`. This function returns the association list in Figure 3.9, where keys are given in boldface.

A subtree is the key for each element of the list, and the remainder of each entry (the values) is either:

1. trees or subtrees in which the key appears, or
2. an integer representing the number of times this top level tree occurs in the input collection.

Thus, this is a replete association list. This association list is now the input to the function `fringe-frequencies`, which produces the list in Figure 3.10.

This frequency list has each fringe from our replete association list, together with an integer. Remember, a fringe is simply the leaf set of a (sub-)tree, so we do not distinguish between the fringe from `(A (B C))` and the fringe from `((A B) C)`. The integer gives the number of trees that have a subtree with this fringe.

```
((A B) ((A B) C))
(((A B) C) ((D E) (F G))) . 1)
((D E) ((D E) F G)
       ((D E) (F G)))
(((D E) F G) ((A B) C) ((D E) F G)))
(((A B) C) ((D E) F G)) . 1)
(((A B) C) (((A B) C) (D (E (F G))))
           (((A B) C) ((D E) F G))
           (((A B) C) ((D E) (F G))))
((F G) (E (F G))
       ((D E) (F G)))
((E (F G)) (D (E (F G))))
((D (E (F G))) (((A B) C) (D (E (F G)))))
(((A B) C) (D (E (F G)))) . 1)
((B C) (A (B C)))
((D E F) ((D E F) G))
(((D E F) G) ((A (B C)) ((D E F) G)))
(((A (B C)) ((D E F) G)) . 1)
((A (B C)) ((A (B C)) ((D E) (F G)))
           ((A (B C)) ((D E F) G)))
(((D E) (F G)) ((A (B C)) ((D E) (F G)))
               (((A B) C) ((D E) (F G))))
(((A (B C)) ((D E) (F G))) . 1)
```

Figure 3.9: **Example replete database.**

```
((A B) . 3)
((D E) . 3)
((F G) . 3)
((E F G) . 1)
((B C) . 2)
((D E F). 1)
((A B C) . 5)
((D E F G) . 5)
((A B C D E F G) . 5)
```

Figure 3.10: **Example fringe frequency list.**

We are now prepared to sweep through this list and record the fringes that occur at least as often as the threshold for both a strict and majority consensus. In this example, for the strict majority we collect those fringes that occur five times, and for the majority, we collect those that occur at least three times. This gives us:

```
((A B C D E F G) . 5)
((D E F G) . 5)
((A B C) . 5)
```

and

```
((A B C D E F G) . 5)
((D E F G) . 5)
((A B C) . 5)
((F G) . 3)
((D E) . 3)
((A B) . 3)
```

Finally, the function `build-term-top` uses either the strict or majority fringes together with a normalization list such as `(A B C D E F G)` to create the strict and majority consensus trees. In this case we create `((A B C) (D E F G))` and `(((A B) C) ((D E) (F G)))`.

3.4 Experiments

3.4.1 Data Sets

I first obtained collections of phylogenetic trees from Dr. Usman Roshan and Dr. Tiffani Williams. These trees were created by PAUP and TNT performing maximum parsimony searches using bio-molecular data sets. We have analyzed hundreds of these collections though we only present the results from ten collections. The results presented are representative of the full set. I also generated sets of trees using MrBayes [40] that had more taxa than either PAUP or TNT can even read.

Table 3.1 gives characteristic information for each collection we present, namely, the numbers of taxa per tree, the number of trees in the collection, and the source of the collection.

3.4.2 Methods

The files we obtained often contained comments about how the trees were generated, parsimony scores, or other output from their production. TASPI does not store this information, so we began by creating files that contained only the topological tree information so that we could accurately assess our compression.

Next, we created a suite of Perl scripts that take these original files and generate appropriate input files for PAUP and TNT. In each case, the taxon list is created from the first tree in the file, and the trees themselves are collected. Then, for PAUP, a Nexus file is produced with the taxa list, the trees, and a PAUP block containing the commands to compute consensus. Similarly for TNT, an appropriate

Data Set Number	Data Set Name	Number of Taxa	Number of Trees	Source
1	Dom_2org	8506	47	Roshan
2	sRNA_mito	2587	369	Roshan
3	Will_Euk	2000	537	Roshan
4	Three567	567	2505	Williams
5	Actino	4583	301	Roshan
6	Ocho854	854	2505	Williams
7	John921	921	2505	Williams
8	t10000	500	10000	Roshan
9	Will2000	2000	2505	Williams
10	Mari2594	2594	2505	Williams
11	20000seqs	20000	1001	Nelesen
12	50000seqs	50000	1001	Nelesen

Table 3.1: **Data set statistics.** For each dataset, the number of trees in the collection and the number of taxa per tree are given.

input file is created with the taxa list, trees, and commands to compute consensus.

TASPI reads the source files directly. As with PAUP and TNT, TASPI can be run both interactively, where we submit one command at a time, or using an input file containing all commands needed for the desired computation.

Using PAUP, TNT and TASPI, we measured the time it took the software to read each collection, and the time needed to compute both a strict and majority consensus tree. For PAUP, we produced a strict consensus tree using its majority consensus command with `percent` set to 100 since the PAUP strict consensus command took considerably longer to do the same calculation. Also, by default, TNT does not include branches that are not well supported by the data used to create trees. However, we were not including any initial data other than the trees them-

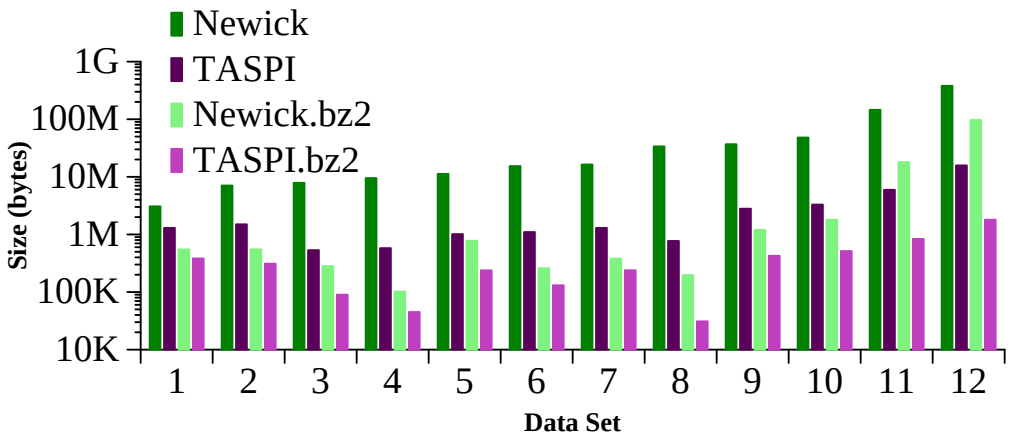


Figure 3.11: **Storage requirements.** The amount of disk space to store each collection of trees in a variety of formats is shown.

selves, so we turned this feature off using the command `collapse notemp`.

Our experiments, where we were able to compare PAUP, TNT and TASPI, were all performed on an Intel Pentium 4 CPU 3.4 Ghz computer. However, for the two largest data sets, we used an AMD Opteron CPU 2.4 Ghz computer, which has similar computational performance, but more physical memory. Either computer produces the same compressed files. The largest files are too large to be read by either PAUP or TNT due to internal limitations on the number of taxa allowed in a tree.

3.4.3 Results

The first major contribution of TASPI is the condensed format in which trees can be stored, while maintaining structural information. Figure 3.11 shows four sets

of sizes for each of our benchmark data sets. The Newick data represents the size of the trees as they were given to us, after removing information that TASPI does not currently store (e.g. comments and branch lengths) and Newick.bz2 illustrates the size of the file after compression using the algorithm implemented in bzip2 [85]. TASPI.bhz displays the size of the file after compression using the Boyer-Hunt method. Notice that this file is still in ASCII, but with redundancies removed. Unlike most compression methods, all the information present in the original files is still immediately accessible, without a decompression step. Finally, TASPI.bhz.bz2 shows the size of the file if it is compressed using the Boyer-Hunt method and then bzip2 is applied.

Using the compressed TASPI format saves considerable memory space. For the data sets we present, the storage requirement for the TASPI format ranges from 2% of the storage requirement of Newick for the t10000 (data set 8) collection, up to 26% for the Dom_2org (data set 1) collection. Over all data sets, the compressed TASPI format uses just 5% of the storage requirement of the Newick format.

The amount of storage space saved is dependent on the amount of similarity between input trees. The more similarity between input trees (i.e. the greater the number of common subtrees) the more effective the compression. Further, the greater the number of trees in the collection, the more likely there will be common structure.

It is readily apparent that bzip2 produces smaller files than the Boyer-Hunt compression on the smaller collections of trees, but for the very large data sets, the Boyer-Hunt compression produces smaller files than bzip2. Further, the Boyer-

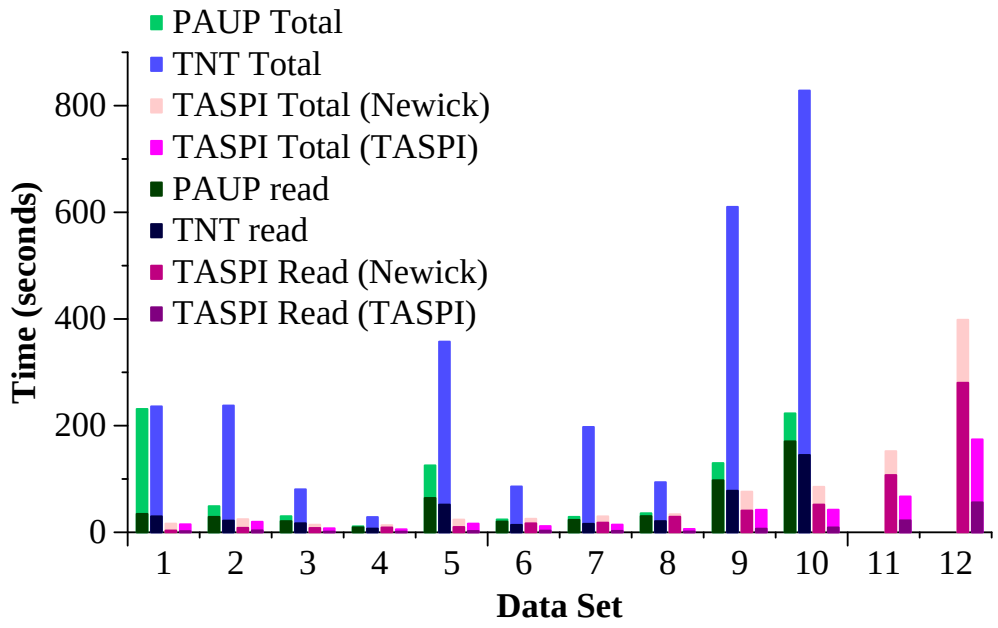


Figure 3.12: **Read and consensus times.** Time to read a collection of trees and compute strict and majority consensus trees with each of PAUP, TNT and TASPI (using both previously compressed [TASPI.bhz] and not previously compressed [TASPI] trees for TASPI). The total height of each bar is the end-to-end time, while the dark region of each bar is the portion of time spent reading the input for the calculation.

Hunt files are ASCII, and thus are ready to be used as input to analysis, such as consensus. If the data are not currently required as input to a post-tree analysis, compressed TASPI is even more useful. Boyer-Hunt files can be further compressed using bzip2 to produce even smaller files than those produced by using bzip2 on the original Newick files for sharing and transmission purposes. For our data sets, using the Boyer-Hunt compression together with bzip2 produces files that require 1% of the storage space of Newick.

The second major contribution of TASPI is its ability to read collections of trees quickly. In Figure 3.12, the darkly colored bars show average read times in seconds for each of our benchmark collections of trees. Notice that while reading trees with TNT or PAUP requires comparable times, reading the Boyer-Hunt compressed trees with TASPI is by far the fastest time for any collection. In fact, neither PAUP nor TNT is able to read the last two data sets. For the data sets which PAUP and TNT can read, reading the compressed TASPI format takes just 2% of the time to read the Newick files with PAUP. This means that loading these files takes more than 48 times longer when read with PAUP or TNT rather than using TASPI to read their compressed counterpart. Even reading the source files is faster in TASPI than it is in either PAUP or TNT – using TASPI to read the Newick files takes just 16% of the time needed to read the same files with PAUP or TNT.

The third major contribution of TASPI is a consensus implementation with improved performance. In Figure 3.12, the lightly colored portion of each bar indicates the time to compute consensus with each of TASPI, TNT and PAUP. In each case, both a strict consensus tree and a majority consensus tree are computed. Notice that the time to compute consensus includes the time to read the collection of trees since the trees are the input to a consensus calculation. Thus, the total height of the bar is the time to both read the input, and then perform the calculation. For TASPI, I show both the time to compute consensus when reading compressed trees and also the time when reading Newick trees.

In all cases, the result TASPI produces is identical to that produced by PAUP (when PAUP is able to read the input), but TASPI is faster. For the data sets PAUP

and TNT can process that we present, using TASPI to compute consensus with input trees in compressed TASPI format requires 5% of the time it takes PAUP to compute consensus with input trees in Newick format. If we factor out the improved reading time, TASPI computes these consensus trees in about 10% of the time it takes PAUP to do the same computation.

3.5 Building blocks

Several of the functions optimized for strict and majority consensus are easily reused in other operations, in particular those functions that transform a tree into a set of bipartitions and the functions that build a tree from a set of bipartitions.

Using these and other basic functions that are part of TASPI, we have implementations of several other forms of consensus including greedy [11], combinable component (otherwise known as loose or semi-strict) [10], and multipolar consensus [8]. Though a thorough exploration of the performance of these algorithms has not been conducted, their implementations are simple, and in early testing perform well. As an example, an implementation of multipolar consensus was created using the functionality already present in TASPI in the course of an afternoon, and the resulting code was forty times faster than the time mentioned in the explanatory paper.

We have also implemented several functions that are not consensus methods. These include functions to compute a Maximum Agreement Subtree [90], to determine if a set of trees is compatible [30, 35], and to compute the symmetric difference and Robinson-Foulds [72] distance rates between trees.

3.6 Conclusions

In phylogenetics, the ability to store large numbers of trees is increasingly important. Bayesian methods are considering more trees than previous methods, and are growing in popularity. Biologists are also choosing to retain additional trees visited during a search. We have shown that our format provides decreased storage requirements, while maintaining data accessibility for further processing. Further, our format together with techniques like memoization allows for improved performance in post-tree analysis. We showed this using strict and majority consensus.

As phylogeneticists continue in their quest to understand the evolutionary relationships between organisms and build the Tree of Life, they need tools to handle large collections of trees, and cull information from those trees. We have introduced our system, TASPI, for phylogenetic tree storage and manipulation. Though a prototype system we have shown that TASPI has useful properties, and good performance.

Chapter 4

The Effect of Guide Trees

In building a multiple sequence alignment, the most popular methodology uses dynamic programming to perform a progressive alignment. In this technique, an estimate of the topology of the tree for the sequences is used as input to the alignment algorithm. This input tree is referred to as a "guide tree". This guide tree determines the order in which sequences are added to the alignment. Two sequences which are siblings in the guide tree are first aligned using a pairwise alignment algorithm. Again, using the relationships from the guide tree, sequences are added to this alignment using pairwise techniques when adding a single sequence, and merging alignments when more than two sequences have already been aligned Much of the popular alignment software available today makes use of a guide tree (e.g. ClustalW [96] or one of the newer alignment methods such as MAFFT [47], ProbCons [20] or Muscle [21]).

However, since we use an alignment as input to the tree reconstruction step, there is a circularity to this methodology. In this chapter I explore the effect the guide tree has on the resulting alignment, specifically in the context of reconstructing a phylogeny. To this end, I performed a simulation study using four multiple sequence alignment tools that make use of a guide tree, and considered four dif-

ferent guide trees. Much of the material in this chapter was first published at the Pacific Symposium of Biocomputing [65].

4.1 Related Work

Progressive alignment was first proposed by Feng and Doolittle [27]. Traditional approaches computed pairwise alignments, and then shifted the sequences around by eye in order to create a multiple sequence alignment. Feng and Doolittle instead used the heuristic "once a gap, always a gap", and only ever added gaps to a set of sequences already aligned, choosing to add sequences in the order of their similarity. They found that this method helped to produce more accurate alignments as compare to the traditional approaches.

Much work has been done considering the effect of various techniques, models and data on tree estimation techniques given a fixed alignment [33, 88] and similarly robust studies have considered the accuracy of alignment tools [20, 22, 23, 47]. Though not quite as extensive, there is also work that looks at the impact of alignment tools when tree estimation is the eventual goal [13, 34, 53, 57, 61, 68, 100].

However, there has been relatively little work that looks carefully at the impact of guide trees. For example, in his explanation of how Muscle computes a guide tree [22], Edgar simply states:

> Distance matrices are clustered using UPGMA, which we find to give slightly improved results over neighbor-joining despite the expectation that neighbor-joining will give a more reliable estimate of the evolutionary tree.

A limited study by Roshan et al. [74] looked at improving maximum parsimony

trees by iteratively improving the guide trees used in the alignment step. Compared to other techniques though, they showed little improvement.

4.2 Experimental Methodology

Although there are many phylogeny estimation methods, several studies [24, 29, 41, 42, 51] suggest that maximum likelihood analyses of aligned sequences produce the most accurate phylogenies (in particular, as compared to maximum parsimony). Of the various software programs for maximum likelihood analysis, RAxML [88] and GARLI [102] are the two fastest and most accurate methods [88]. I used RAxML for my analyses.

Of the many available MSA methods, I used ClustalW, ProbCons, MAFFT and Muscle. ClustalW tends to be the one most frequently used by systematists, although several new methods have been developed that have been shown to outperform ClustalW with respect to alignment accuracy. ProbCons and MAFFT have been shown to perform well, and Muscle is included because it is very fast. Version information, as well as the commands used for each program are available in Appendix 1.

I performed a simulation study to evaluate the performance of the different MSA methods on each of several guide trees. In this experiment, I evolved DNA sequence datasets using the ROSE [92] software (because it produces sequences that evolve with site substitutions and also indels) under 16 different model conditions, half for 100 taxon trees and half for 25 taxon trees. For each model condition, I generated 20 different random datasets, and analyzed each using a variety of tech-

niques. I then compared the estimated alignments and trees to the true alignments and trees, recording the alignment error as measured by SP-error and missing edge rates (that is, false negative tree error).

4.2.1 Data Generation

In order to understand the robustness of my results, I generated data with varying characteristics. Parameters I kept constant are as follows:

- To create model trees, I generated birth-death trees of height 1.0 using the program `r8s` [77] with 100 and 25 taxa.
- I modified branch lengths to deviate each model tree moderately from ultrametricity, using the technique used by Moret *et al.* [60] with deviation factor c set to 2.0.
- I generated a random DNA sequence of length 1000 for the root of each model tree, and then evolved sequences down the tree according to the K2P+Indel+Γ model of sequence evolution. For each model tree, I set the transition/transversion ratio to 2.0, and all sites evolved at the same rate.

To obtain model coverage, I varied the remaining parameters for ROSE:

- I varied the mean substitution rate to achieve edgewise average normalized Hamming distances between (approximately) 2% and 7%.
- I used two single-gap-event length distributions, both geometric with finite tails. The "short" single-gap-event length distribution had average gap length

2.00 and a standard deviation of 1.16. The "long" single-gap-event length distribution had average gap event length 9.18 and a standard deviation of 7.19.

- I set insertion and deletion probabilities so as to produce different degrees of gappiness (S-Gap in the table).

This results in 16 model conditions (8 for 100 taxa, and 8 for 25 taxa). Table 4.1 shows the parameter settings for each model condition, and the resultant statistics.

4.2.2 Estimating Multiple Sequence Alignments and Trees

As previously mentioned, I used four multiple sequence alignment programs to create alignments from raw sequences: ClustalW, Muscle, MAFFT, and ProbCons, each of which is publicly available. I ran each of program using its default guide tree as well with guide trees that I provided. Muscle and ClustalW have options which allow the user to supply a guide tree. A member of my lab (Kevin Liu) modified ProbCons to allow it to use an input guide tree, and the authors of MAFFT provided us with a version that accepts guide trees as input. MAFFT has multiple alignment strategies built in, and I used each of L-INS-i, FFT-NS-i and FFT-NS-2. However, when there were difference between variants of MAFFT in these experiments, FFT-NS-2 usually (though not always) performed best, so I only show results using this variant.

	Model Condition Parameters				True Alignment Statistics			
MC	**Taxa**	**P(gap)**	**P(sub)**	**Gaps**	**S-MNHD**	**E-ANHD**	**S-Gap**	**E-Gap**
1	100	0.0001	0.005	long	37.5 (.2)	1.9 (.02)	40.8 (.3)	.72 (.01)
2	100	0.0001	0.01	long	56.9 (.3)	3.2 (.04)	43.7 (.6)	.69 (.01)
3	100	0.0005	0.005	long	38.0 (.3)	2.4 (.04)	81.9 (.3)	4.1 (.07)
4	100	0.0005	0.01	long	57.4 (.4)	4.9 (.03)	83.0 (.1)	4.9 (.04)
5	100	0.0005	0.005	short	36.9 (.2)	1.9 (.04)	42.6 (.6)	.76 (.01)
6	100	0.0005	0.01	short	56.7 (.2)	4.1 (.04)	46.4 (.2)	.86 (.01)
7	100	0.0025	0.005	short	38.6 (.3)	2.2 (.04)	81.4 (.2)	4.2 (.07)
8	100	0.0025	0.01	short	56.3 (.2)	4.6 (.07)	82.4 (.2)	4.6 (.07)
9	25	0.0001	0.004	long	32.2 (.2)	2.9 (.05)	22.8 (.3)	1.2 (.02)
10	25	0.0001	0.008	long	51.3 (.2)	5.9 (.09)	25.0 (.4)	1.4 (.02)
11	25	0.0005	0.004	long	31.3 (.2)	2.2 (.03)	55.1 (.5)	4.7 (.10)
12	25	0.0005	0.008	long	50.2 (.3)	5.2 (.08)	61.3 (.5)	5.9 (.12)
13	25	0.0005	0.004	short	30.0 (.1)	2.6 (.08)	26.1 (.4)	1.4 (.04)
14	25	0.0005	0.008	short	50.0 (.1)	6.4 (.07)	28.5 (.2)	1.7 (.02)
15	25	0.0025	0.004	short	31.1 (.4)	3.2 (.06)	66.7 (.4)	7.7 (.14)
16	25	0.0025	0.008	short	50.2 (.5)	5.2 (.07)	62.4 (.4)	6.6 (.08)

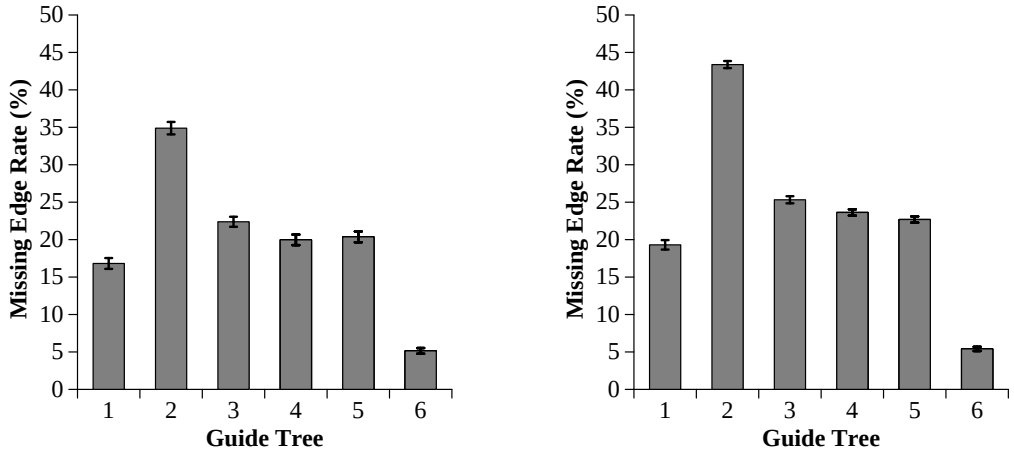
Table 4.1: **Model parameters and true alignment statistics.** "P(gap)" is gap probability, "P(sub)" is substitution probability, "Gaps" is gap distribution type, "MNHD" is maximum normalized Hamming distance, "E-ANHD" is average normalized Hamming distance on the edges, "S-Gap" is percent of the true alignment matrix occupied by gaps, and "E-Gap" is average gappiness per edge; the standard error is given parenthetically.

I considered four user-input guide trees: the true tree, and three other guide trees that I computed. The first two of the computed guide trees are UPGMA [86] trees based upon different distance matrices. For the first UPGMA guide tree ("upgma1"), I computed a distance matrix based upon optimal pairwise alignments between all pairs of sequences, using the affine gap penalty with *gap-open* $= 0$, *gap-extend* $= 1$ and *mismatch* $= 1$. For the second ("upgma2"), I computed the distance matrix based upon optimal pairwise alignments between all pairs of sequences for the affine gap penalty with *gap-open* $= 2$, *gap-extend* $= 0.5$ and *mismatch* $= 1$. In both cases, I used custom code based on the Needleman-Wunsch algorithm with the specified gap penalty to compute the distance matrices and PAUP* [94] to compute the UPGMA trees. The third guide tree ("probtree") was obtained as follows: I used the upgma1 guide tree as input to ProbCons to estimate an alignment that was then used to estimate a tree using RAxML. To keep the tree estimation step consistent, I used RAxML in its default setting.

4.3 Results

For ease of discussion, the graphs presented here have values that have been averaged over all model conditions and replicates (for the given number of taxa). The relative performance of the methods shown in the averages holds (with few exceptions) for each model condition. However, the magnitudes of the actual errors and amount of improvement based on a given guide tree vary.

To begin, let us consider the topological accuracy of the guide trees I will pass to each method. Figure 4.1 shows the accuracy of each of the guide trees.



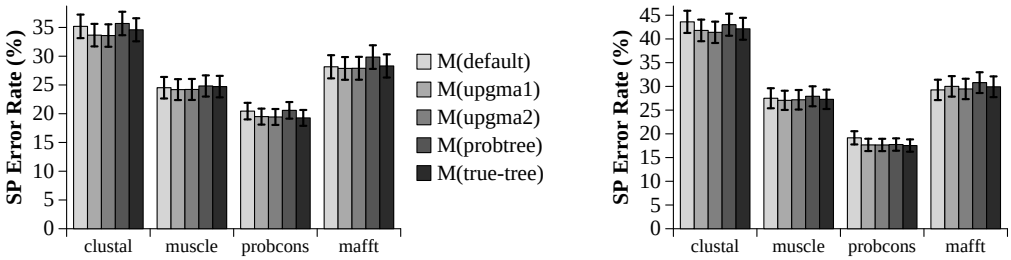
(a) 25 taxa

(b) 100 taxa

Figure 4.1: **Guide tree missing edge rates.** Tree error rates are averaged over all model conditions and replicates. (1) ClustalW default, (2) ProbCons default, (3) Muscle default, (4) upgma1, (5) upgma2, and (6) probtree.

Clearly, the accuracy differs significantly, with the ProbCons default tree generally the least accurate, and the "probtree" guide tree the most accurate; the two UPGMA guide trees have very similar accuracy levels.

Next, Figure 4.2 shows the accuracy of the alignments using different MSA methods on these guide trees. Surprisingly, despite the large differences in topological accuracy of the guide trees, alignment accuracy (measured using SP_{FN}, here referred to as "SP-error") for a particular alignment method varies relatively little between alignments estimated from different guide trees. For example, two ClustalW alignments or two Muscle alignments will have essentially the same accuracy scores, independent of the guide tree. The biggest factor impacting the SP-error of the alignment is the MSA method. Generally, ProbCons is the most



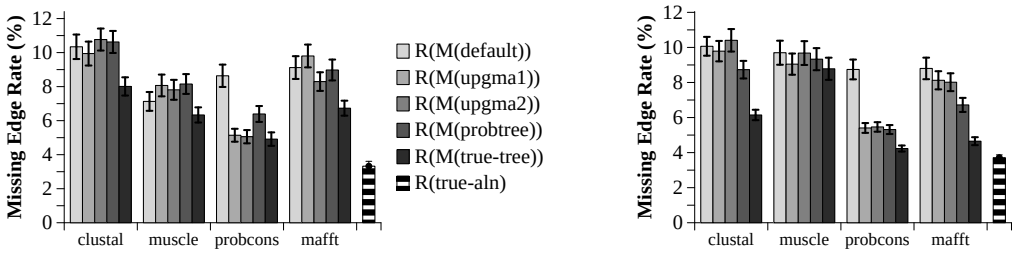
(a) 25 taxa (b) 100 taxa

Figure 4.2: **SP-error rates of alignments.** M(guide tree) indicates multiple sequence alignment generated using the indicated guide tree.

accurate and ClustalW is the least.

This is consistent with previous work which showed that guide trees have relatively little impact on the accuracy of the alignment produced [22], in part because current progressive methods attempt to limit the errors introduced by an inaccurate guide tree through the use of iteration [20, 48, 67].

But the primary purpose of this study is to consider the impact of changes in guide tree on the accuracy of the resultant RAxML-based phylogeny. Figure 4.3 shows this comparison. Not surprisingly, regardless of alignment method, the most accurate trees are obtained when the true tree is used as a guide tree. However, it is not always the case that using more accurate guide trees create alignments that result in more accurate estimated trees. For example, Muscle responded very little to improvements in the guide tree, possibly because it computes a new guide tree after the initial alignment on the input guide tree. ClustalW also responds only weakly to improvement in guide tree accuracy. In particular, using the more accurate prob-



(a) 25 taxa (b) 100 taxa

Figure 4.3: **Missing edge rate of estimated trees.** R(M(guide tree) indicates RAxML run on the alignment generated by the multiple sequence alignment method using the guide tree indicated. R(true-aln) indicates the tree generated by RAxML when given the true alignment.

tree guide tree often results in worse performance as compared to using other guide trees. On the other hand, ProbCons responds positively and significantly to improvements in guide trees. This is quite interesting, since the alignments did *not* improve in terms of their SP-error rates! Furthermore, ProbCons improves quite dramatically as compared to its performance in its default setting.

4.4 Discussion

MSA accuracy (as measured using SP-error) is not strongly correlated with guide tree accuracy. Further, for most of these MSA methods, phylogenetic accuracy is not directly predicted by the accuracy of the guide tree. Although it is common to evaluate alignments purely in terms of criteria like SP, these experiments provide clear evidence that not all errors are of equal importance, at least in terms of phylogenetic consequences. Again, this is not completely surprising, since

when Ogden and Rosenberg [68] studied the influence of tree shape on alignment and tree reconstruction accuracy they too found that alignment error did not always have a large impact on tree accuracy. Finally, it is important to realize that although alignments may have similar SP-error rates as compared to a true alignment, they can still be very different from each other.

Changes in the guide tree generally do not impact the accuracy of the estimated alignments, as measured by SP-error. However, some RAxML-based phylogenies, obtained using alignments estimated on more accurate guide trees, were *much* more accurate than phylogenies obtained using MSA methods on their default guide trees. Muscle and ClustalW were impacted the least by the choice of guide tree, and ProbCons was impacted the most. The improvement produced for ProbCons is particularly relevant to systematists, since it is one of the two best MSA methods currently available.

The experiments show clearly that tree estimation can be improved through the use of improved guide trees, though only some alignment methods seem to be able to take advantage of these improved guide trees. It is also clear that these improvements require some additional computational effort. However, as Table 4.2 shows, the increase in running time for ProbCons is not substantial, and likely worth the effort given the greatly improved performance.

4.5 Implications for Prank

Given my experiments with ProbCons, Muscle and MAFFT, when initial experiments within the lab assessing the accuracy of Prank [56] showed less than

Step	25 taxa	100 taxa
Compute UPGMA guide tree	0:01:36	0:23:49
Align with ProbCons	0:07:20	1:52:03
Run RAxML on ProbCons alignment	0:00:42	0:15:05

Table 4.2: **Running time.** Average running times across all models and replicates, given in hours:minutes:seconds. Experiments were run using a distributed system of heterogeneous machines via Condor [52].

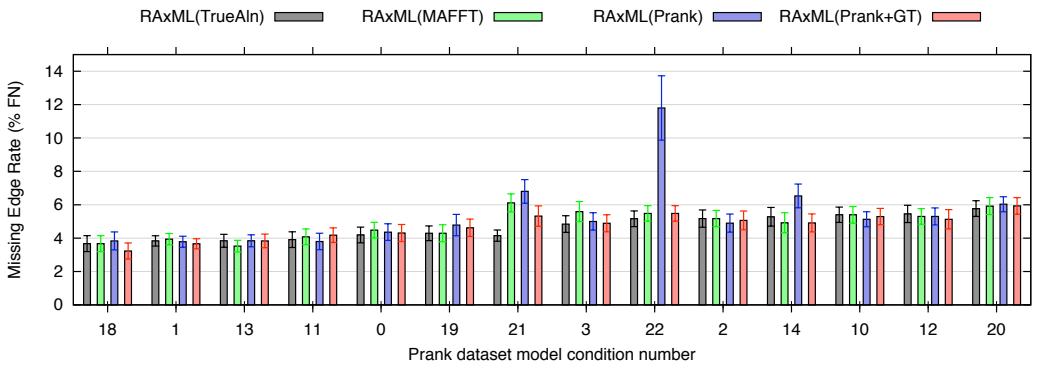
ideal performance, we applied what we had learned from the experiment just described. Prank is an alignment tool that produces not just an alignment over the leaves, but also estimates ancestral sequences (that is, sequences for every node of the tree). Prank is different from the other software studied in that it uses insertions and deletions as phylogenetic information, and thus is noted to possibly be "very sensitive to the given topology" [55].

For the experiments with Prank, we used a wide range of model conditions covering 100, 500 and 1000 taxa. The details for the 100 taxon model conditions are given in Table 4.3. As a guide tree, we used the tree estimated by RAxML given a MAFFT alignment.

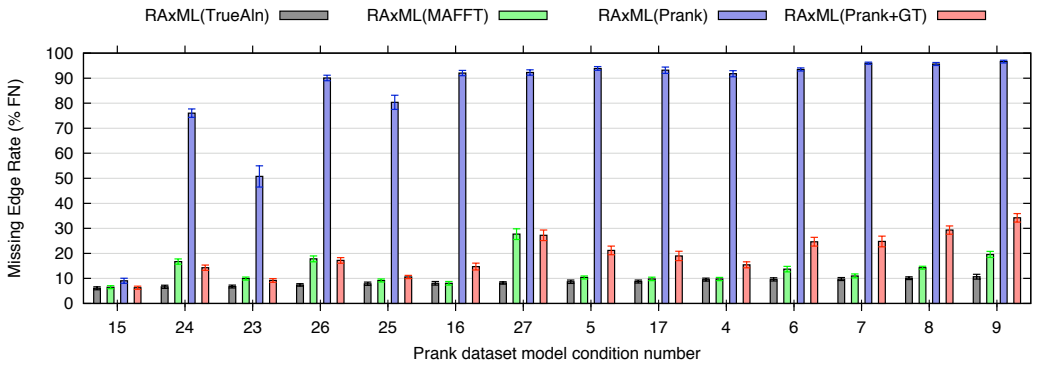
Figure 4.4 shows the dramatic improvement in the accuracy of trees estimated using Prank alignments where we have used the non-default guide tree. While it is not often the case that the tree estimated using the Prank alignment is more accurate than the RAxML(MAFFT) guide tree, it is important to remember that we are gaining additional information (i.e. ancestral sequences) by utilizing Prank. Also, by passing a guide tree to Prank, the time required by Prank to create an alignment has also been significantly reduced (see Table 4.4, but note that these

		Edgewise Statistics		Setwise Statistics			
MC	**Model**	**ANHD (%)**	**% Gap**	**MNHD (%)**	**ANHD (%)**	**Avg. Gap Len**	**% Gap**
0	100L-1	2.72 (0.08)	0.21 (0.00)	32.54 (0.20)	22.19 (0.34)	10.16 (0.28)	26.55 (0.92)
1	100L-2	2.36 (0.04)	0.34 (0.01)	32.64 (0.27)	21.93 (0.30)	9.75 (0.21)	36.08 (0.89)
2	100L-3	4.64 (0.13)	0.38 (0.01)	45.15 (0.28)	33.83 (0.36)	10.09 (0.26)	37.61 (1.11)
3	100L-4	4.33 (0.15)	0.74 (0.02)	44.37 (0.33)	32.24 (0.45)	11.41 (0.22)	53.63 (1.04)
4	100L-5	19.89 (0.37)	0.36 (0.01)	71.12 (0.19)	62.28 (0.22)	10.17 (0.25)	37.47 (0.83)
5	100L-6	20.51 (0.42)	0.84 (0.04)	70.97 (0.14)	62.30 (0.19)	11.17 (0.29)	56.41 (1.32)
6	100L-7	20.82 (0.31)	1.48 (0.04)	70.75 (0.20)	62.29 (0.25)	13.70 (0.22)	69.69 (0.65)
7	100L-8	24.08 (0.46)	0.37 (0.01)	72.81 (0.19)	65.28 (0.22)	10.01 (0.25)	38.06 (1.15)
8	100L-9	24.40 (0.44)	0.78 (0.02)	72.77 (0.18)	64.97 (0.18)	11.09 (0.24)	55.24 (0.98)
9	100L-10	24.51 (0.33)	1.63 (0.05)	72.87 (0.19)	64.92 (0.18)	14.24 (0.20)	71.48 (0.55)
10	100M-1	2.50 (0.06)	0.11 (0.00)	32.45 (0.21)	22.03 (0.33)	5.21 (0.16)	15.38 (0.74)
11	100M-2	2.43 (0.07)	0.22 (0.01)	32.33 (0.28)	22.02 (0.26)	5.33 (0.10)	24.72 (1.07)
12	100M-3	4.73 (0.10)	0.21 (0.00)	44.96 (0.23)	33.50 (0.34)	5.36 (0.16)	24.77 (0.89)
13	100M-4	4.68 (0.15)	0.45 (0.02)	44.90 (0.25)	33.67 (0.32)	6.17 (0.13)	40.13 (1.22)
14	100M-5	8.00 (0.25)	0.21 (0.01)	56.61 (0.19)	44.69 (0.40)	5.33 (0.12)	23.71 (1.03)
15	100M-6	8.13 (0.17)	0.43 (0.01)	56.41 (0.27)	44.97 (0.38)	5.89 (0.14)	38.90 (1.07)
16	100M-7	20.31 (0.43)	0.20 (0.00)	71.12 (0.24)	62.50 (0.29)	5.25 (0.13)	23.35 (0.84)
17	100M-8	19.68 (0.49)	0.48 (0.02)	71.29 (0.21)	62.06 (0.28)	6.34 (0.10)	41.41 (1.22)
18	100S-1	2.45 (0.07)	0.05 (0.00)	32.13 (0.31)	21.58 (0.38)	2.05 (0.05)	6.34 (0.36)
19	100S-2	2.44 (0.06)	0.10 (0.00)	32.49 (0.27)	21.87 (0.25)	2.20 (0.04)	11.70 (0.44)
20	100S-3	4.69 (0.11)	0.22 (0.00)	44.90 (0.28)	33.05 (0.38)	2.42 (0.03)	21.42 (0.62)
21	100S-4	4.56 (0.14)	1.14 (0.04)	43.56 (0.23)	31.95 (0.32)	4.57 (0.12)	57.24 (1.11)
22	100S-5	9.53 (0.26)	0.21 (0.00)	59.45 (0.20)	48.16 (0.50)	2.36 (0.03)	20.38 (0.75)
23	100S-6	9.77 (0.25)	1.14 (0.03)	58.74 (0.31)	47.54 (0.46)	4.50 (0.10)	57.31 (0.85)
24	100S-7	9.32 (0.28)	2.85 (0.09)	56.76 (0.20)	45.64 (0.26)	9.30 (0.30)	76.98 (0.63)
25	100S-8	15.75 (0.40)	0.21 (0.00)	67.62 (0.15)	58.23 (0.25)	2.32 (0.03)	20.41 (0.68)
26	100S-9	15.64 (0.47)	1.39 (0.05)	67.25 (0.14)	57.04 (0.40)	5.19 (0.14)	61.82 (0.98)
27	100S-10	15.31 (0.40)	2.83 (0.10)	66.11 (0.23)	55.58 (0.42)	9.07 (0.30)	76.91 (0.69)

Table 4.3: **True alignment statistics for Prank datasets.** Each dataset has 100 taxa. "ANHD" is the average normalized Hamming distance, given both edgewise and setwise. Edgewise "% Gap" is the average gappiness per edge. "MNHD" is the maximum normalized Hamming distance for the alignment. "Avg. Gap Len" is the average length of gaps in the alignment, and setwise "% Gap" is the percentage of the alignment matrix occupied by indels. Standard errors are shown in parentheses.



(a) Easier models



(b) More difficult models

Figure 4.4: **Missing edge rate of estimated trees on Prank datasets.** Model conditions have been ordered by increasing missing edge rate of RAxML(TrueAln). Note that the y-axis for the easier model conditions (Figure 4.4(a)) has a range of 0-15% while the y-axis for the more difficult model conditions (Figure 4.4(b)) has a range of 0-100%.

No. of sequences	Prank+GT	Default Prank
100	$\approx$ **.5 hours**	$\approx$ 2 hours
500	$\approx$ **2.5 hours**	$\approx$ 28 hours
1000	$\approx$ **5 hours**	$\approx$ 4 days

Table 4.4: **Comparison of running time for Prank+GT and Default Prank**. All experiments to record running time were performed on a standard desktop computer with 2 GB RAM. "Prank+GT" refers to Prank run on the RAxML(MAFFT) guide tree. The running time reported does not include the time required to compute the guide tree.

times do not include the time to estimate the guide tree).

Throughout our exploration of Prank we were in communication with the authors, who have since changed the default behaviour to include estimating the alignment twice, the second time with an improved guide tree estimated from the first alignment.

4.6 Conclusions

Taking the extra time to explicitly produce a guide tree instead of simply using the default is worthwhile. In particular, by quickly generating a UPGMA guide tree, the alignments produced by ProbCons are markedly improved for subsequent tree estimation, and the extra time required is not substantial. Similarly, by using a RAxML tree estimated from a MAFFT alignment as a guide tree, Prank alignments are dramatically improved. Though not all programs are equally impacted (for example, in these experiments trees generated from Muscle alignments do not improve) a careful consideration of the impact of the guide tree used for each

program is warranted.

Further, it is interesting to see that the standard measure of alignment accuracy, SP_{FN} or SP-error, is not indicative of subsequent phylogenetic tree estimation accuracy. Clearly new alignment metrics, which better predict their utility for tree estimation are needed.

Chapter 5

Introduction to BLuTGEN

The Assembling the Tree of Life Project (AToL) [3] is a national initiative to build a phylogenetic tree that encompasses all of life on earth, or in the projects words, "reconstruct the evolutionary origins of all living things." Thus far, however, the techniques available for this endeavour scale to at most a few thousand taxa, instead of the millions of taxa that will be required if we are to reach the ambitious goal set forth by AToL.

Advances are being made though. New alignment techniques, such as those available in MAFFT [47], can scale to many thousands of taxa, and tree estimation techniques, such as RAxML [88], are also being improved to handle ever larger datasets. But eventually these techniques fail to produce phylogenies, due to challenges such as the necessary computation time or the machine memory required for the computation.

To address these issues, I have developed a method, BLuTGEN, that attacks the problem of estimating a phylogenetic tree from sequence data in a new way. This approach has three steps:

1. Divide the sequences into overlapping subsets

2. Estimate a tree for each subset

3. Assemble a tree for the entire dataset using the subset trees as input

Thus, instead of estimating an alignment for the full dataset, BLuTGEN estimates a tree from overlapping subtrees.

In many ways, this is an obvious approach, but it has never been tried before, in particular for a single gene region as opposed to combining multiple markers. Step 2 has been the focus of research for years, and technology is now to the point that for moderately large datasets, it is often possible to estimate a tree with good accuracy. For step 3, a new supertree method, SuperFine [93], has recently been shown to quickly and accurately reconstruct phylogenies from overlapping subproblem trees. The key then lies in how the overlapping subproblems are generated.

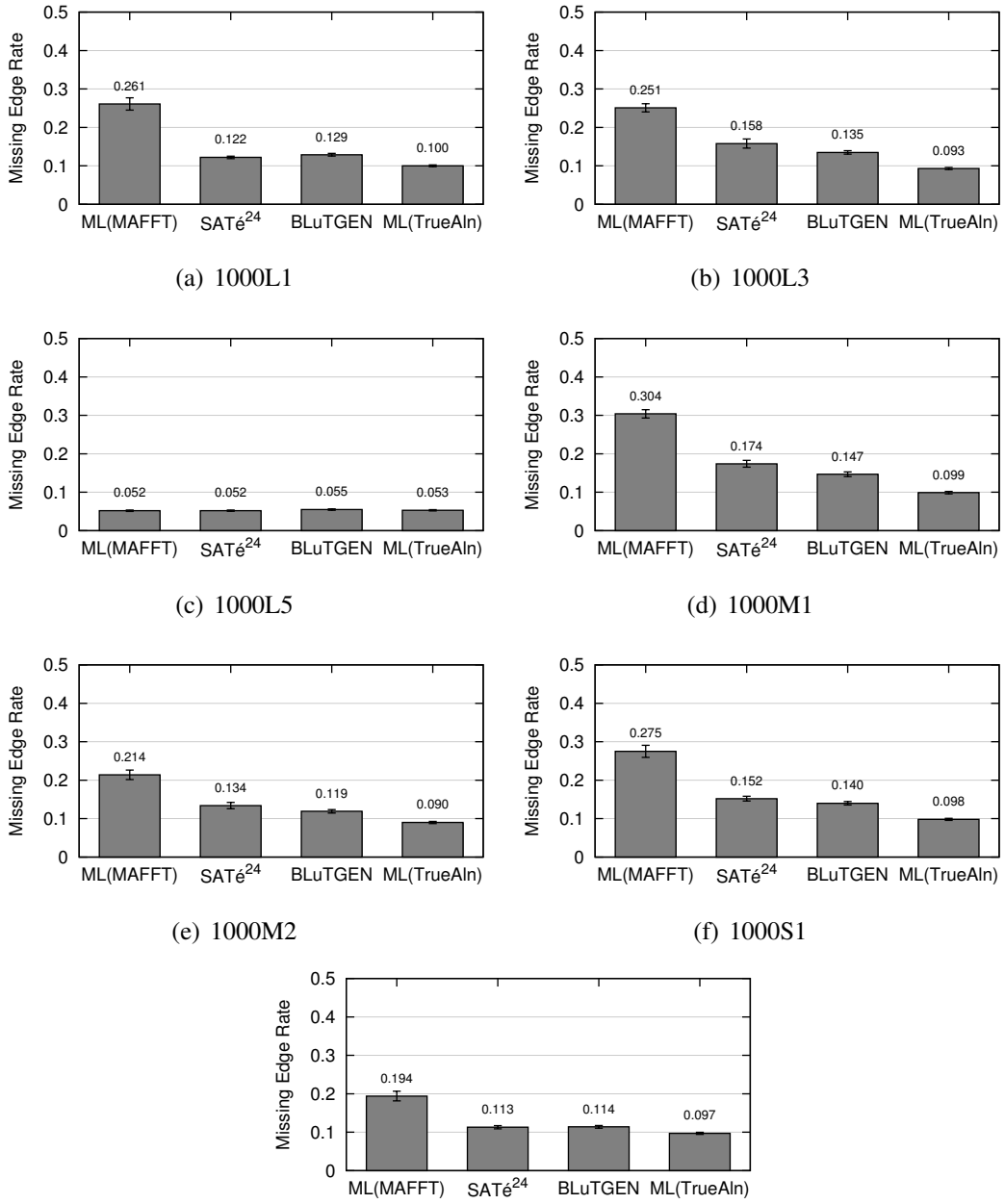
BLuTGEN uses two methods of generating overlapping subproblems. The first, a method I call BLF ("BLAST-based Fast"), uses the raw sequences together with BLAST (an algorithm that quickly finds similar sequences in a database) to generate the overlapping subproblems. The second method, a padded recursive DCM3 decomposition (pRecDCM3), uses a guide tree to generate the subproblems. The pRecDCM3 method is unpublished, and was developed by Li-San Wang and Tandy Warnow. This second method can also be used iteratively, where the result of one iteration is the input guide tree to the next. BLuTGEN uses the BLF decomposition to generate an initial estimate of the tree, and then uses the pRecDCM3 method to improve this initial estimate.

Since BLuTGEN never estimates an alignment on the full dataset, and uses

a fast supertree method to generate the full tree, this method is able to avoid some of the memory issues from which the usual two-phase approaches to tree estimation suffer. Further, although there may be many subproblems which need to be analyzed, through the use of parallelization BLuTGEN can also help reduce the time required to estimate the full phylogeny.

Most importantly though, this method produces accurate trees for hard datasets. I have used both simulated and biological data to assess BLuTGEN (see Chapter 6 for further details about these data). I begin by showing results where I use only one application of pRecDCM3, and then also show a few results where I have further improved the BLuTGEN result by iterating with pRecDCM3.

To begin, Figure 5.1 shows the accuracy of BLuTGEN (using only one application of the second phase) as compared to the RAxML(MAFFT) tree (one of the best two-phase methods), the SATé tree found by Liu et al. [54], and the tree estimated by RAxML given the true alignment for 140 simulated datasets (7 model conditions, 20 replicates per model condition), each with 1000 taxa. For the 6 difficult model conditions shown (where a model condition is considered difficult if the missing edge rate of the RAxML(MAFFT) tree is more than 10%), BLuTGEN is more accurate than SATé for four of the model conditions, and within a tenth of a percent for a fifth model condition. On the final difficult model condition BLuTGEN is only worse than SATé by 0.7%. Considering all seven model conditions, the greatest improvement in accuracy is for 1000L3, where BLuTGEN is 2.7% more accurate than SATé, and for the dataset where the tree returned by BLuTGEN is worse than SATé (1000L1), the difference in accuracy is less than a percent.



(a) 1000L1

(b) 1000L3

(c) 1000L5

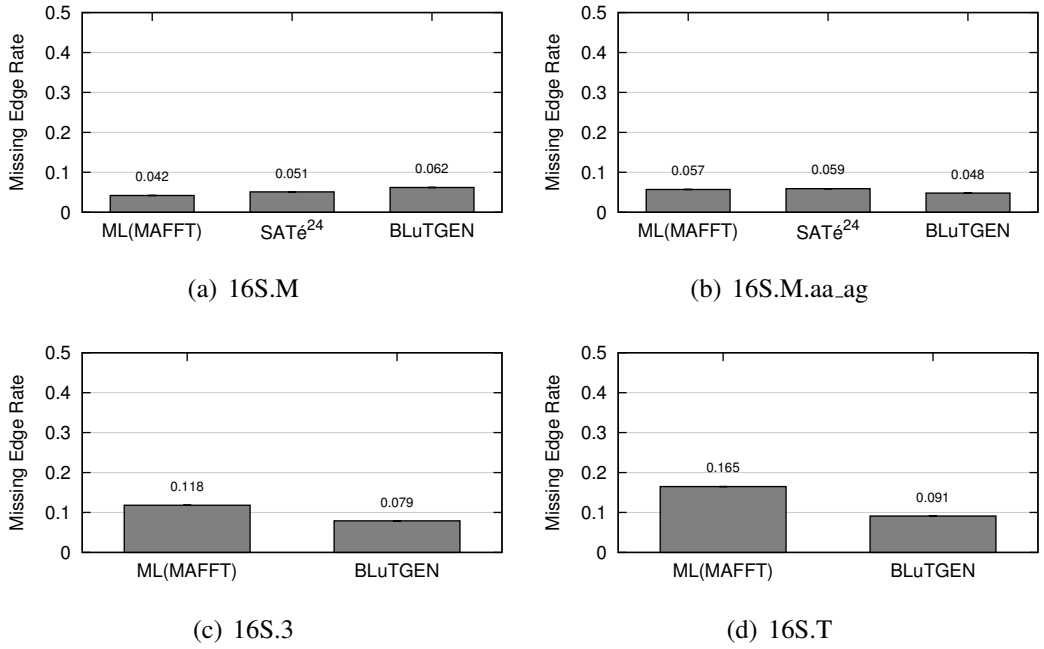
(d) 1000M1

(e) 1000M2

(f) 1000S1

(g) 1000S2

Figure 5.1: **Accuracy of BLuTGEN compared to alternative methods for ROS-Esim datasets.** The missing edge rate for each of RAxML(MAFFT), SATé, RAxML(TrueAln), and BLuTGEN is shown for seven simulated model conditions.



(a) 16S.M

(b) 16S.M.aa_ag

(c) 16S.3

(d) 16S.T

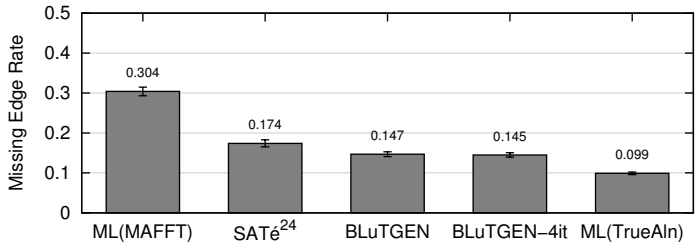
Figure 5.2: **Accuracy of BLuTGEN compared to alternative methods for Gutell datasets.** The missing edge rate for each of RAxML(MAFFT), SATé (where available), and BLuTGEN is shown for the Gutell datasets. 16S.M has 901 taxa, 16S.M.aa_ag has 1028 taxa, 16S.3 has 6263 taxa and 16S.T has 7350 taxa.In generating the RAxML(MAFFT) tree, MAFFT was run using the linsi option for 16S.M and 16S.M.aa_ag and using the parttree option for 16S.3 and 16S.T.

In Figure 5.2 I show the accuracy of four biological datasets as compared to an estimated reference tree (see Chapter 6 for further details of the reference tree generation). Again, these results use a single iteration of the pRecDCM3 technique. For three of the four datasets, the result of BLuTGEN is more accurate than the alternative, and for the one dataset where BLuTGEN is less accurate, it is only worse by 1.1 percent. Note that for 16S.3 and 16S.T, SATé results are not available.

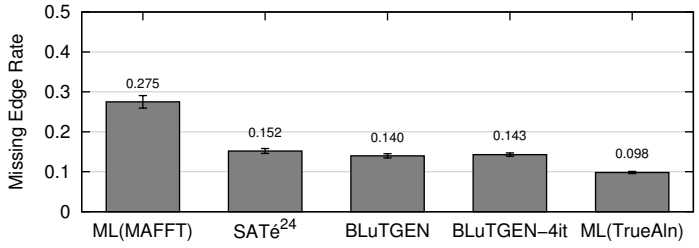
I also used the second phase of BLuTGEN, the pRecDCM3 decomposi-

tion technique, iteratively for a few of the most difficult model conditions. Figure 5.3 shows the final tree that results after four iterations for two ROSEsim datasets, and Figure 5.4 shows the results for the two Gutell datasets with more than 5000 taxa, again, as compared to the best two-phase method available (that is, RAxML(MAFFT) using the linsi option for the ROSEsim datasets, and the parttree option for the Gutell datasets). In the case of the ROSEsim datasets, I also give the SATé results. Notice that iterations generally improve the resulting tree accuracy, though minor degradations also occur (see Section 7.2 for further details).

In the next few chapters I will present the development of BLuTGEN. Chapter 6 gives background for each of the pieces involved in BLuTGEN, as well as details for the simulated and curated data on which I evaluate the performance of BLuTGEN. Chapter 7 explores the possibility of improving an existing tree, while Chapter 8 considers several early variants of BLAST-based decompositions. Finally, Chapter 9 examines some alternative versions of BLuTGEN, in particular looking at other possibilities for how to run SuperFine, as opposed to the default, to allow BLuTGEN to be applied to even larger datasets.

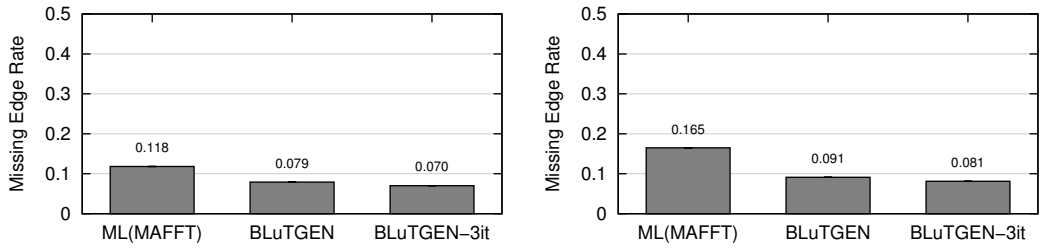


(a) 1000M1



(b) 1000S1

Figure 5.3: **Accuracy of BLuTGEN using iteration compared to alternative methods for two ROSEsim datasets.** The missing edge rate for each of RAxML(MAFFT), SATé, and BLuTGEN using five applications of pRecDCM3 (that is, four iterations) is shown for the 1000M1 and 1000S1 datasets. In generating the RAxML(MAFFT) tree, MAFFT was run using the most accurate linsi option.



(a) 16S.3 (b) 16S.T

Figure 5.4: **Accuracy of BLuTGEN using iteration compared to alternative methods for Gutell datasets.** The missing edge rate for RAxML(MAFFT) and BLuTGEN using four applications (that is, three iterations) of pRecDCM3 is shown for the two Gutell datasets with 6263 (16S.3) and 7350 (16S.T) taxa. In generating the RAxML(MAFFT) tree, MAFFT was run using the parttree option.

Chapter 6

Background and Data for BLuTGEN

As briefly described in Chapter 5, BLuTGEN makes use of several pre-existing techniques in each of its three steps. In this chapter I give the relevant background for each step, and then give details for the data I use to explore the performance of BLuTGEN.

6.1 Background

6.1.1 Step 1: Division into subproblems

For division into subproblems, I use two approaches, one based on BLAST, and another based on a disk-covering method (DCM3). I now give some background on each of these techniques.

6.1.1.1 BLAST

BLAST is an acronym for "Basic Local Alignment Search Tool". From the NCBI website: "BLAST finds regions of local similarity between sequences. The program compares nucleotide or protein sequences to sequence databases and calculates the statistical significance of matches" [63].

BLAST was designed to enable researchers to find sequences in a database

that were similar to a reference sequence very quickly. Originally designed in 1990 [1], there are now many different variations of BLAST searches. The fundamental idea though, is to compare sequences not in their entirety, but instead to only consider portions of the reference sequence. To speed the computation even further, heuristics are used to do the comparisons.

In my experiments, I use the `blastn` algorithm (available from the standard `blastall` command), which uses a database of nucleotide sequences and a nucleotide reference sequence. I create a database for each set of sequences using the command `formatdb`. I use several variations of BLAST-based clustering techniques, and give the details for each in Chapter 8.

6.1.1.2 Disk-Covering Methods

Disk-covering methods (DCMs) are a suite of tree estimation algorithms based on a divide-and-conquer approach. In each instance, the input data are divided into subsets, a tree estimation analysis is performed on each subset, and the results of each subset analysis are merged (and perhaps subsequently refined) to create a solution for the entirety of the data. The first DCM [43] was designed for use with distance-based tree reconstruction methods, while a second, DCM2 [44], was designed for use in reconstructing maximum parsimony trees.

A third decomposition method, DCM3 [75], was also first designed for reconstructing maximum parsimony trees. However, DCM3 has the beneficial property that there is a heuristic to this decomposition that makes it very fast, with smaller subproblem sizes than DCM2. I will be using this heuristic DCM3 decom-

position, which determines subsets on the basis of a user input guide tree.

I need two concepts to explain the decomposition: **centroid edge** and **short subtree**. The centroid edge of a tree is the edge that, when removed, roughly divides the leaf set of the tree into two sets of equal size. The short subtree is defined in terms of a particular edge in the tree. Given a tree T and an edge e, consider the (usually four) rooted subtrees that result from removing e and its pendant edges from the tree. Compute the distance (given by the edge lengths on the tree, or assuming that every edge has unit length) from each taxon to the root of its respective subtree. Now, for each subtree, attach to e the closest taxon (taxa in the case of ties) from each subtree in the place of the original subtree. This constructed tree is the short subtree of T on e.

The DCM3 decomposition then works as follows:

1. Determine the centroid edge of the guide tree, and set the subtrees determined by removing the centroid edge and the edges connected to the centroid edge to be A, B, C and D.

2. Set X to be the short subtree around the centroid edge.

3. Use as subsets the leaf sets of $A \cup X, B \cup X, C \cup X$ and $D \cup X$.

For my purposes, I need a fair amount of overlap between subsets, but DCM3 only has an overlap in the leaf set of X, which is usually very small (just four taxa). Padded DCM3 ("pDCM3") overcomes this limitation by adding extra taxa to X. Padded DCM3 is unpublished work by Li-San Wang and Tandy Warnow, but

an executable written by Li-San Wang is available. In a padded DCM3, the number of additional taxa is dependent on the padding factor p. Instead of only taking the closest taxon from each subtree, p of the next closest taxa are also added to X. Thus, when there are no ties, the size of the overlap is $p * 4$.

Depending on the size of the guide tree, these subsets can still be quite large. However, the algorithm can be applied recursively (an approach first successfully applied to DCM2 decompositions by Tang et al. [95], and also used for DCM3 [75]) until all subsets are smaller than a user-specified maximum subset size. This version is usually referred to as "RecDCM3". Combining this recursive version with padding is then referred to as "pRecDCM3".

6.1.2 Step 2: Estimate a tree for each subset

After subsets have been defined, I apply standard methods to generate a tree for each subset. In these experiments, I use MAFFT (using the most accurate option, linsi) to generate an alignment, and then estimate a tree from this alignment using RAxML (I refer to this tree as RAxML(MAFFT)). I chose this technique since RAxML(MAFFT) is one of the most accurate two-phase methods [54]. Note that when estimating trees using RAxML, I use the GTRCAT model as opposed to the GTRGAMMA since GTRCAT requires less memory and is considerably faster than GTRGAMMA (GTRCAT and GTRGAMMA usually return the same tree topology, though GTRCAT does not return branch lengths whereas GTRGAMMA does). Further, since the supertree method I use to assemble the full tree does not require branch lengths (see Section 6.1.3), using GTRCAT is not an impediment. In some

cases, when using simulated data, I also used RAxML to estimate a tree on the true alignment (which I refer to as RAxML(TrueAln)). The commands used are listed in Appendix 1.

6.1.3 Step 3: Assemble a tree for the full dataset

To combine the trees from each subset, I use a supertree method. Supertree methods take as input a set of trees on overlapping sets of taxa, and use these "source" trees to build a single tree that contains each of the taxa in some source tree. I use the supertree method SuperFine [93] to construct a full tree from the subtrees.

SuperFine has two phases, and has an option to report the tree at the end of each phase. The first phase constructs a strict consensus merger (SCM) tree, which the second phase then refines. I use the Matrix Representation with Parsimony (MRP) [4, 5, 70] option of SuperFine to resolve polytomies. For more complete details of SuperFine, see Swenson's dissertation [93].

As I developed BLuTGEN, I discovered that the default behaviour of SuperFine needed to be modified for very large datasets, particularly when the source trees used result in SCM trees with large polytomies. Thus, as a result of my experiments, modifications have been made to SuperFine which allow variable behaviour when resolving using MRP. I consider several of these alternatives, which include changing the number of ratchet iterations in the MRP search (default is 100), using a random result tree instead of the default greedy consensus, and limiting the time allowed for each iteration of the search.

6.2 Data

In each of the next chapters I will assess the ability of several methods of building phylogenies to accurately reconstruct a tree. I will use both simulated and biological datasets for this purpose, and will compare the methods I develop to existing techniques.

The data I present falls into three categories. The first, RNAsim [16], was generated by Junhyong Kim's lab as part of the CIPRES project [15]. These are large datasets (8192 and 16394 taxa) simulated using secondary structure information. Interestingly, these RNA datasets turn out to be very easy to analyze, perhaps because of their short gap lengths. The second, ROSEsim, was generated using the software package ROSE [92] for the introductory paper on SATé [54, 82, 83]. These datasets have 1000 taxa, and most have been found to be difficult for existing software. The third, Gutell Data, are curated empirical rRNA datasets from Robin Gutell's comparative RNA database [12]. Though there are many such datasets, I use just four, which have many taxa (ranging from 901 to 7350) and have proved difficult for existing tools (the two smallest of these were also analyzed in the SATé paper [54]). I now give further details for each set.

6.2.0.1 RNAsim

To create these datasets, a simulation was done using, for each dataset, the same starting/root RNA sequence with secondary structure, but on trees with different number of taxa. These trees were random-sampled subtrees of a 1-million-taxon binary tree, which resembles real phylogenetic trees and was created by Tracy

Dataset	Taxa	Sites	MNHD (%)	ANHD (%)	Gap (%)	Avg Gap Len
8192	8192	2420.3	40.7	20.0	36.6	1.03
16384	16384	2805.8	42.4	20.9	45.3	1.03

Table 6.1: **RNAsim data statistics.** The number of taxa, number of sites, average normalized hamming distance (ANHD), percentage of the matrix which is a gap character (Gap), and average gap length (Avg Gap Len) are shown.

Heath at Hillis/Bull lab in UTexas at Austin. The simulation parameters that generated these data were tuned such that the simulated sequences resemble real small subunit rRNA (ssu rRNA) sequences in terms of sequence identity, number of indels, the ratio between substitution and indels, etc. The resulting dataset statistics are given in Table 6.1.

I used the trees/datasets with 8192 and 16384 sequences (20 replicates each). For each of these model trees and associated datasets, I generated a potentially inferable model tree (PIMT) as discussed in Section 2.3.2 to use as a reference tree, as well as an unaligned set of sequences (by removing all gaps).

6.2.0.2 ROSEsim

These data were generated in two steps. Step 1 generated model trees with r8s [77] and then Step 2 generated sequence data using ROSE [92]. These datasets were first created by Kevin Liu and used to test the effectiveness of SATé [54]. Though there are many more such datasets, I chose the model conditions with 1000 taxa, and that had proved most difficult for existing two-phase methods. I also included one model condition which was relatively easy as a comparison point.

Step 1: Generation of model trees using r8s

We used r8s to generate random birth-death model trees with 1000 taxa using the following script:

```
begin rates;
simulate diversemodel=bdback seed=<integer random seed>
ntaxa=<1000> T=0;
describe tree=0 plot=tree_description;
end;
```

We then used a custom program (a modified version of tds2, unpublished, but available from the SATé program website [82]) that deviated the r8s trees from ultrametricity using the technique described in [62] with deviation factor c equal to 2.0. Next, we scaled all branches on a model tree by a variable factor which we call "tree height". Finally, as ROSE cannot handle fractional branch lengths for its GTR simulation, we scaled all the branches in the model tree by a factor of 100 to ensure that the branch lengths were generally greater than 1.

Step 2: Simulation of sequences using ROSE

We simulated evolution using ROSE to evolve sequences with indels and substitutions on the model trees. For each model, we generated 20 different model trees (replicates), and for each model tree, we simulated one dataset, starting with a root DNA sequence of 1000 sites.

- Model of evolution - We used the GTR+Gamma model [73] for site evolution with parameters (frequency of nucleotides at the root and the instantaneous rate matrix) obtained by estimating GTR+Gamma parameters on the NemA-TOL [66] alignment of 682 species of nematodes, using PAUP* [94]. We

used the ROSE transitional probability matrix $P(t) = e^{At}$ for $t = .001$ and A the transitional rate matrix.

The frequencies of the nucleotides at the root were given by

```
TheFreq=[.300414,.191363,.196748,.311475]
```

and the transitional rate matrix for the GTR model was given by the following off-diagonal entries:

```
A-C 1.24284
A-G 3.47484
A-T 0.48667
C-G 1.07118
C-T 4.38510
G-T 1.0
```

- Gap length distribution - We used three single-event gap-length distributions: short, medium, and long, each geometric with finite tails. The long gap distribution had expected gap length 9.2 and median gap length 7; the medium gap distribution had expected gap length 5.0 and median gap length 4; and the short gap distribution had expected gap length 2.0 and median gap length 2. The gap length distributions used in our study are given below. The first element of each list is the probability of a gap of length one, given that a gap event occurs, the second is the probability of a gap of length two given a gap event occurs, and so on.

  Long Gap Length Distribution:

```
[0.1028, 0.0899, 0.0792, 0.0702, 0.0627, 0.0565, 0.0514,
0.0470, 0.0433, 0.0400, 0.0369, 0.0341, 0.0314, 0.0289,
0.0266, 0.0245, 0.0225, 0.0206, 0.0188, 0.0171, 0.0155,
0.0141, 0.0127, 0.0114, 0.0100, 0.0087, 0.0075, 0.0063,
0.0052, 0.0042]
```

Model	Taxa	Gap Length	Tree Height	Gap Probability
1000L1	1000	long	35	4.3E-06
1000L3	1000	long	30	1E-05
1000L5	1000	long	5	7.5E-06
1000M1	1000	medium	35	8.2E-06
1000M2	1000	medium	30	1E-05
1000S1	1000	short	35	8.2E-06
1000S2	1000	short	35	4.3E-06

Table 6.2: **Parameters used to simulate datasets**. This table reports the model-specific parameters for each of the ROSEsim models used to evolve sequences on trees.

Medium Gap Length Distribution:

```
[0.2012, 0.1600, 0.1280, 0.1024, 0.0819, 0.0655, 0.0524,
0.0419, 0.0336, 0.0268, 0.0215, 0.0172, 0.0137, 0.0110,
0.0088, 0.0070, 0.0056, 0.0045, 0.0036, 0.0029, 0.0023,
0.0018, 0.0015, 0.0012, 0.0009, 0.0008, 0.0006, 0.0005,
0.0004, 0.0003, 0.0002]
```

Short Gap Length Distribution:

```
[0.4613, 0.2527, 0.1545, 0.0896, 0.0419]
```

- Probability of a gap event - We set insertion and deletion probabilities identically. The gap event probabilities we used for each model are in Table 6.2.

We set the shape parameter α of the gamma distribution, controlling rate variation across sites, to 1.0. The ROSE script used to simulate data is given in Figure 6.1. Table 6.2 gives the parameters used to simulate each dataset.

Again, for each of these model trees and associated datasets, I generated a potentially inferable model tree (PIMT) as discussed in Section 2.3.2 to use as the

```
TheInsFunc = <Insert event gap length distribution -
              long, medium or short>
TheDelFunc = <Delete event gap length distribution -
              long, medium or short>
InputType = 4
TheAlphabet = "ACGT"
TheFreq = [.300414,.191363,.196748,.311475]
ThePAMMatrix =
[[0.9948, 0.0012, 0.0035, 0.0005],
[0.0012, 0.9933, 0.0011, 0.0044],
[0.0035, 0.0011, 0.9944, 0.0010],
[0.0005, 0.0044, 0.0010, 0.9941]]
TheInsertThreshold = <Insertion event probability>
TheDeleteThreshold = <Deletion event probability>
SequenceLen = 1000
TheTree =<birth-death model tree in Newick format with
           branch lengths, deviated from ultrametricity>
ChooseFromLeaves = False
AlignmentWithAncestors = True
TreeWithAncestors = True
SequenceNum = 999
SeedVal = <random seed integer>
TheMutationProbability=<site-by-site vector listing rate
                        multipliers for that site>
```

Figure 6.1: **ROSE script.**

reference tree, and also generated the unaligned set of sequences (by removing all gaps).

6.2.0.3 Gutell Data

Gutell's alignments are obtained on the basis of RNA secondary structure – and therefore permit evaluation of the accuracy of an estimated alignment, and hence of an estimated tree. The datasets I present here are just a few of the datasets

Dataset	Sites	MNHD (%)	ANHD (%)	Gap (%)	Avg Gap Len
1000L1	3817.5	76.9	69.5	73.2	13.6
1000L3	7042.8	76.3	68.7	85.2	20.0
1000L5	1764.8	60.6	49.6	42.6	10.4
1000M1	3972.3	76.9	69.5	74.4	10.1
1000M2	2722.6	76.2	68.4	74.2	10.3
1000S1	2141.2	76.8	69.4	53.0	4.0
1000S2	1546.0	76.8	69.3	35.0	2.9

Table 6.3: **ROSEsim data statistics.** The number of taxa, number of sites, average normalized hamming distance (ANHD), percentage of the matrix which is a gap character (Gap), and average gap length (Avg Gap Len) are shown.

that Gutell has created, each of which has a highly reliable alignment. The presented datasets were chosen for their size and difficulty. Each of the chosen datasets has at least 901 taxa (and as many as 7350 taxa) and is difficult to analyze, where difficulty was assessed in terms of two-phase accuracy, as well as the computational time and memory required to generate a tree.

The datasets I use have been cleaned [54]: any taxa with more than 50% unsequenced letters have been removed, and columns that contained only ambiguous or gap characters have also been removed. The resulting reference alignment statistics are in Table 6.4.

For the two 16S.M datasets RAxML was run on the cleaned alignment with 500 bootstrap replicates. In my experiments, I considered as a reference tree the bootstrap tree where only edges with greater than 75% bootstrap support were retained. For the 16S.3 dataset, 500 bootstrap replicates were also run, and it is the 75% majority consensus tree of these 500 replicates that I use as a reference (the

Dataset	Taxa	Sites	MNHD (%)	ANHD (%)	Gap (%)	Avg Gap Len	Ref. Tree Resolut'n
16S.M	901	4722	88.7	35.9	78.1	17.2	46.9
16S.M.aa_ag	1028	4907	100	34.2	82.6	22.0	42.4
16S.3	6323	8716	83.3	31.5	82.1	9.4	50.2
16S.T	7350	11856	90.1	34.5	87.4	12.1	49.5

Table 6.4: **Cleaned Gutell data statistics.** The number of taxa, number of sites, average normalized hamming distance (ANHD), percentage of the matrix which is a gap character (Gap), and average gap length (Avg Gap Len) are shown. Also, the resolution of the reference tree is given for each dataset. The reference tree is the 75% bootstrap tree of RAxML given the curated alignment.

final RAxML search had not completed after 22 days of processing).

For the 16S.T dataset, I also use as a reference the 75% majority consensus tree, but for only 135 bootstrap replicates. These limited replicates are a result of extreme running times for RAxML: using a 16 node machine (each node 2.5GHz), a parallelized RAxML [69] using every node took more than 18 days to complete these 135 replicates. (RAxML was run using its rapid bootstrapping option [89], which estimates trees under the quick GTRCAT model [87], so branch lengths are not available.)

Chapter 7

Improving an Existing Tree: pRecDCM3

In this chapter, I explore one of the methods I used to accomplish the first step of BLuTGEN, that is, producing overlapping subsets. The approach here is to use an existing estimate of the tree to help guide the decomposition. Ideally, even starting from a quick estimate of the tree would produce a final result that was as good as or more accurate than the best methods currently available.

I considered two variants of the recursive DCM3 decomposition described in Section 6.1.1.2: an unpadded version (RecDCM3) and a padded version (pRecDCM3). The RecDCM3 decomposition results in subsets where the expected overlap is only four taxa, while the padded decomposition allows the expected overlap to vary according to the parameter p. Because I used recursive decompositions, if the decomposition resulted in a subset whose size exceeded the maximum subset size parameter, I then recursively decomposed this problem into smaller pieces, using the input tree restricted to the taxa in the subset as the new guide tree.

I applied these decompositions to a variety of guide trees, including "part-tree", a very quick clustering algorithm available in MAFFT [49], two-phase methods where MAFFT and/or ClustalW were used to generate the initial alignment and RAxML was used to estimate the tree, and also the true tree (when known in sim-

ulation). Note that the parttree algorithm in MAFFT can be used to generate both an alignment and a tree. In some cases I used this tree as an input guide tree, and at times I also used the alignment. In general, the parttree tree is quite inaccurate, and the tree estimated by RAxML from the parttree alignment is more accurate. When not computationally prohibitive, the tree estimated by RAxML from the alignment generated by the linsi version of MAFFT is most accurate.

I also examined the effect of iteration for both RecDCM3 and pRecDCM3. In both cases, iteration helps to improve the accuracy of the estimated tree, but pRecDCM3 is overall more accurate[1]. Thus, at the end of this exploration, the best method in this class is a padded Recursive Disk Covering Method (pRecDCM3), where iteration can help fine-tune the result. Note that because this work was exploratory, throughout these experiments I use several different choices for parameter values.

7.1 RecDCM3 Based Decompositions

To see if the RecDCM3 decomposition could improve upon a good estimate of the tree, I applied this decomposition to the RAxML(MAFFT) tree for two of the most difficult ROSE datasets (1000M1 and 1000S1), and an easy one as well (1000L5). As Figure 7.1 shows, using my approach resulted in an improvement to the accuracy of the estimated tree as compared to the input guide tree. Note that for

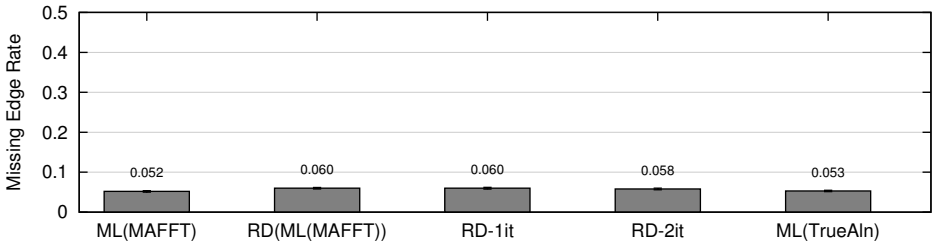
[1]The current implementation of pRecDCM3 requires completely resolved binary trees with branch lengths. Because the result of BLuTGEN is not always binary, and does not contain branch lengths, in order to iterate I randomly resolve any polytomies, and then apply approximately unit branch lengths (the branch lengths are random numbers between 0.95 and 1.05).

these decompositions I set the maximum subset size to 200 and the targeted overlap to 50. Since the RecDCM3 decomposition resulted in an improved tree, the next step was to see if further improvements could be made through the use of iteration. As Figure 7.1 also shows, while further improvements are made, they are not as substantial.

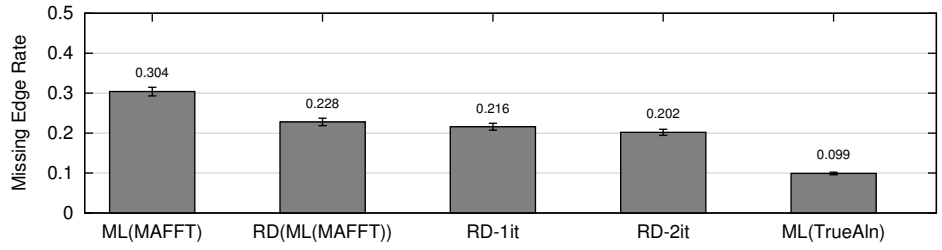
For these experiments, I saved both the final result of using SuperFine, and the SCM tree, which was an intermediate step. One of the interesting things about the RecDCM3 decomposition is that the SCM tree and the SuperFine tree have very similar errors (see Figure 7.2). This means that there is either very little overlap between the subsets, or all subsets agree on the relationships – either way there are not many polytomies in the SCM tree for SuperFine to resolve (see Table 7.1 for polytomy information). Given the nature of a DCM3 decomposition, the expected overlap is only four taxa, and that is before any recursion. To address this, and see if more overlap could help SuperFine produce a more accurate tree, I next considered a padded RecDCM3, where instead of the expected four taxa overlap, I instead have much more. The intuition here is that with more overlap, more accuracy (from the source trees) can be maintained.

7.2 pRecDCM3

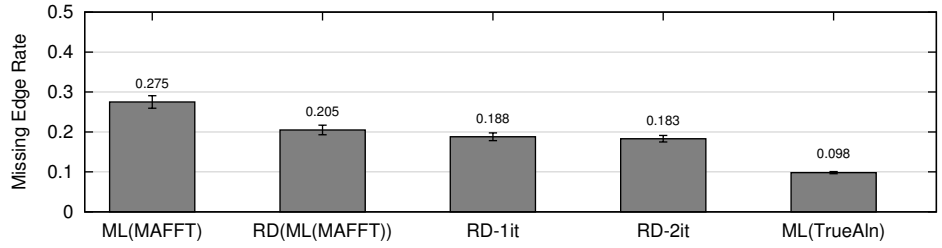
In Figure 7.3 I show the results of using a padded RecDCM3 decomposition, where I aim to have a common subset size of 50 taxa at each level of the recursion. Though not a huge improvement, only a few percentage points, it is clear that using the padded RecDCM3 decomposition results in more accurate trees as compared to



(a) 1000L5

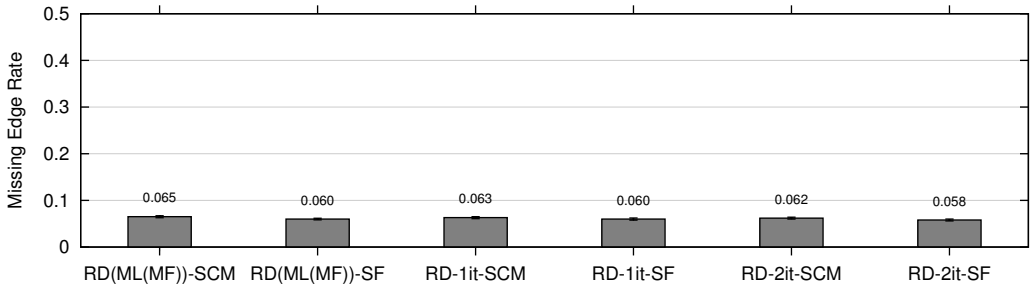


(b) 1000M1

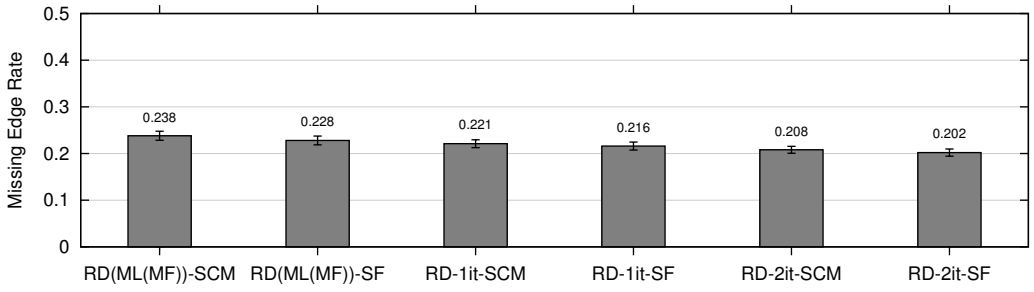


(c) 1000S1

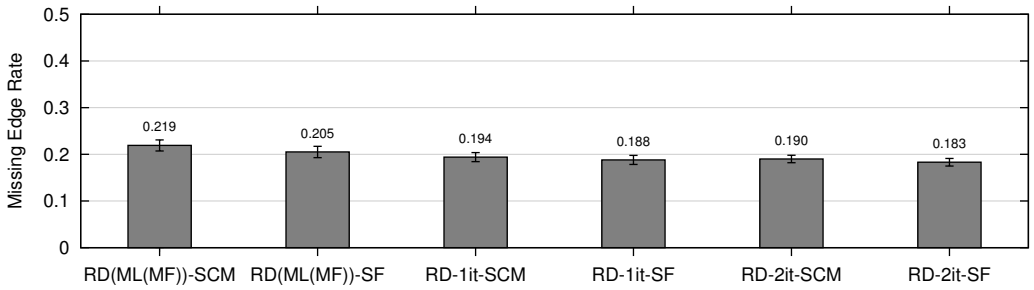
Figure 7.1: **RecDCM3 iterative results on ROSEsim data.** The missing edge rates for trees resulting from RecDCM3 decompositions, starting with the RAxML(MAFFT) tree, and then iterating using the SuperFine result as the new guide tree. ML(MAFFT) indicates the RAxML(MAFFT) starting tree. RD(ML(MAFFT)) indicates the result of using the RAxML(MAFFT) tree as a guide tree for the RecDCM3 decomposition. "RD-1it" indicates using the result of RD(ML(MAFFT)) as a guide tree (i.e. 1st iteration). "RD-2it" indicates using the result of the 1st iteration as a guide tree. ML(TrueAln) indicates RAxML(TrueAlignment).



(a) 1000L5



(b) 1000M1



(c) 1000S1

Figure 7.2: **Comparison of SCM and SuperFine results for RecDCM3 iterative results on ROSEsim data.** The missing edge rate for trees resulting from RecDCM3 decompositions, starting with the RAxML(MAFFT) tree, and then iterating using the SuperFine result as the new guide tree. RD(ML(MF))-SCM indicates the SCM result of using the RAxML(MAFFT) tree as a guide tree to the RecDCM3 decomposition while RD(ML(MFT))-SF indicates the SuperFine result. The iterative trees are referred to analogously.

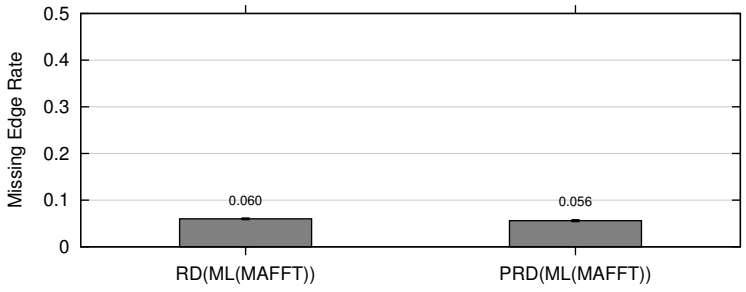
Dataset	Tree	Max Poly Deg.	# Polytomies
1000L5	RD(ML(MAFFT))-SCM	7.9 (1.1)	2.6 (0.3)
	RD-1it-SCM	6.4 (0.4)	3.8 (0.4)
	RD-2it-SCM	6.2 (0.2)	4.3 (0.3)
1000M1	RD(ML(MAFFT))-SCM	24.6 (1.1)	9.6 (0.6)
	RD-1it-SCM	37.5 (2.4)	10.2 (0.6)
	RD-2it-SCM	31.3 (3.1)	9.6 (0.5)
1000S1	RD(ML(MAFFT))-SCM	25.0 (12.4)	9.6 (0.6)
	RD-1it-SCM	32.7 (2.7)	8.9 (0.6)
	RD-2it-SCM	30.4 (2.6)	8.7 (0.4)

Table 7.1: **Polytomy information for iterative RecDCM3 decompositions of ROSEsim data.** The maximum polytomy degree and number of polytomies are given for the SCM trees from each of several iterations of RecDCM3. Standard errors are given in parentheses.

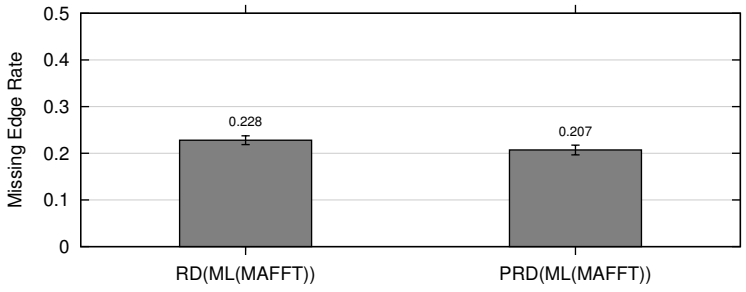
the non-padded version.

As with the non-padded RecDCM3, iteration seems like a possible help. Figure 7.4 shows that there is again a small improvement as we iterate with the padded RecDCM3 decomposition technique. However, within a few iterations, keeping parameter values constant, no further improvements are made. However, as Table 7.2 shows, the pRecDCM3 decomposition does result in more overlap as measured by maximum polytomy degree and the number of polytomies in the SCM tree that results as the first step of SuperFine.

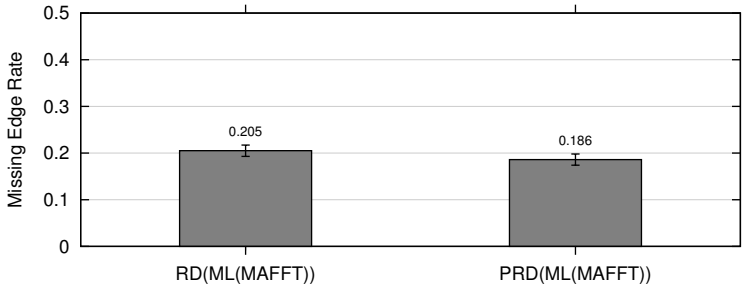
Finally, in Figure 7.5 I directly compare pRecDCM3 and RecDCM3. This shows that using padded RecDCM3 decompositions results in more accurate trees than the unpadded version, at each step of iteration. Interestingly, after a few iterations, the padded version has made greater gains in accuracy as compared to the



(a) 1000L5

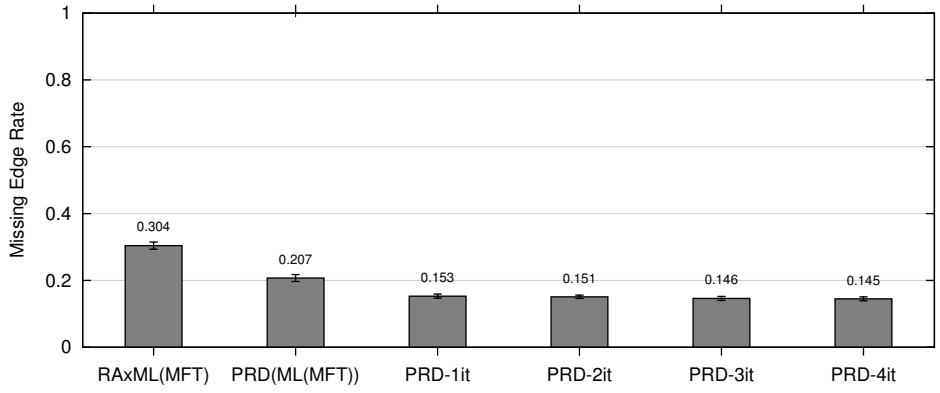


(b) 1000M1

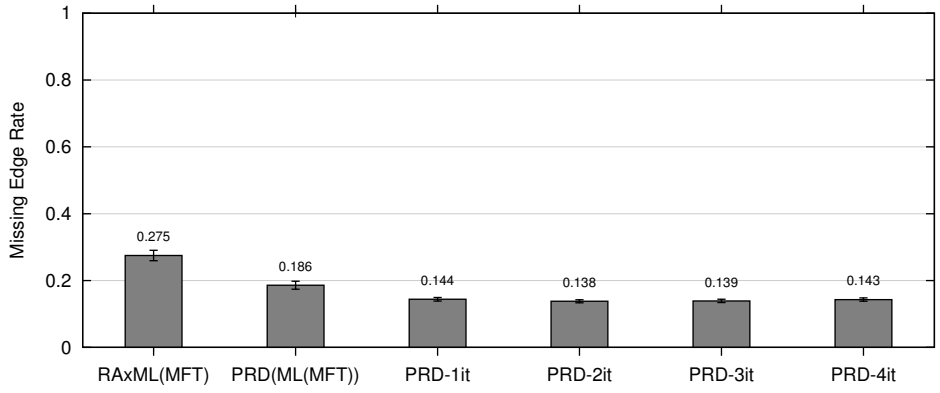


(c) 1000S1

Figure 7.3: **Comparison of RecDCM3 and pRecDCM3 decomposition results for ROSEsim data.** The missing edge rate for the SuperFine trees resulting from RecDCM3 and pRecDCM3 decompositions, using the RAxML(MAFFT) tree as a guide tree. RD indicates the RecDCM3 result while PRD indicates the padded RecDCM3 result.



(a) 1000M1



(b) 1000S1

Figure 7.4: **Effects of pRecDCM3 iteration on ROSEsim data**. Results of iterating using pRecDCM3 with maximum subset size 250 and targeting an overlap of 50 taxa.

Dataset	Tree	Max Poly Deg.	# Polytomies
1000L5	PRD(ML(MAFFT))-SCM	5.1 (0.2)	7.15 (0.4)
	PRD-1it-SCM	5.5 (0.2)	8.2 (0.6)
1000M1	PRD(ML(MAFFT))-SCM	41.0 (3.2)	23.2 (1.4)
	PRD-1it-SCM	55.9 (2.4)	15.1 (0.9)
	PRD-2it-SCM	47.8 (2.1)	12.6 (0.8)
1000S1	PRD(ML(MAFFT))-SCM	36.9 (3.7)	21.6 (1.6)
	PRD-1it-SCM	46.0 (2.4)	12.4 (0.6)
	PRD-2it-SCM	41.4 (3.2)	13.4 (0.8)

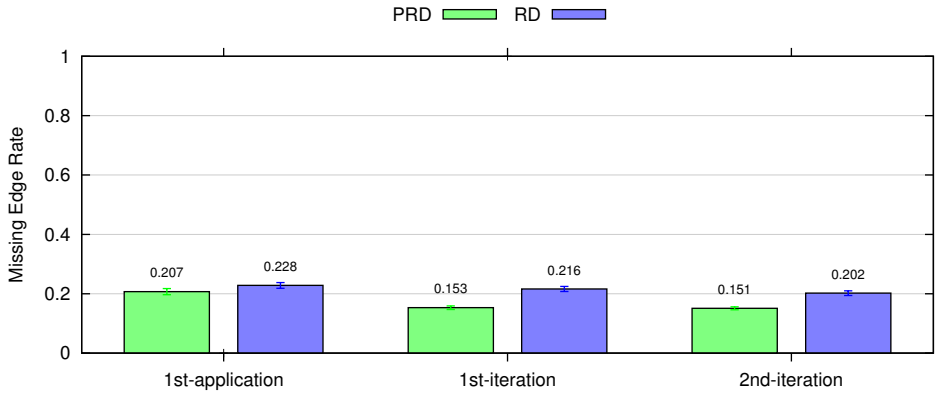
Table 7.2: **Polytomy information for iterative pRecDCM3 decompositions of ROSEsim data.** The maximum polytomy degree and number of polytomies are given for the SCM trees from each of several iterations of pRecDCM3. Standard errors are given in parentheses.

unpadded version. That is, even though the initial difference in accuracy between the two versions was only about two percent, after two iterations, the difference was about five percent.

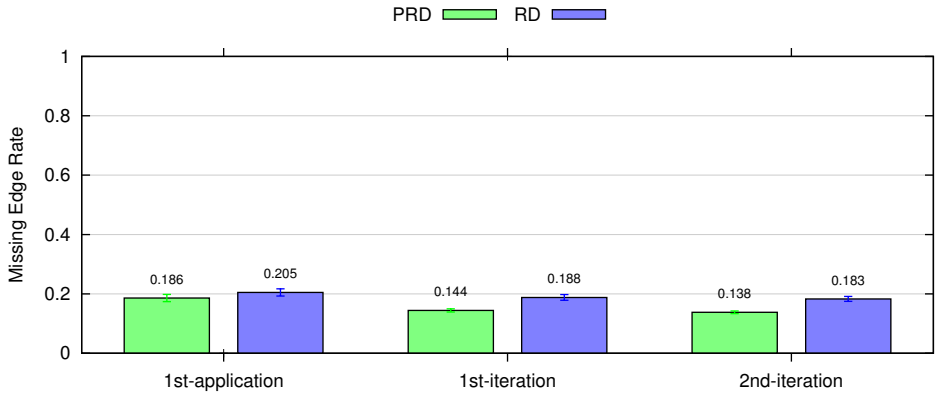
Having developed a method that works well for the 1000 taxon ROSEsim datasets, I then explored even larger datasets.

7.2.1 Gutell Datasets

Working with the 16S.3 (6263 taxa) and 16S.T (7350 taxa) datasets presents a new challenge. Unlike the ROSEsim datasets, running a reasonable two-phase method is computationally difficult in terms of both running time and required machine memory. Thus, I was unable to produce a highly accurate estimated tree from which to begin a padded RecDCM3 decomposition technique. Further, since SATé relies on estimating a tree from a full alignment, running SATé for these datasets



(a) 1000M1



(b) 1000S1

Figure 7.5: **Comparison of RecDCM3 (RD) and pRecDCM3 (PRD) iteration on ROSEsim data**. Missing edge rates for results of iterating using RecDCM3 and pRecDCM3 with maximum subset size 250 and targeting an overlap of 50 taxa. 1st-application indicates the result of applying the indicated decomposition to the RAxML(MAFFT) tree.

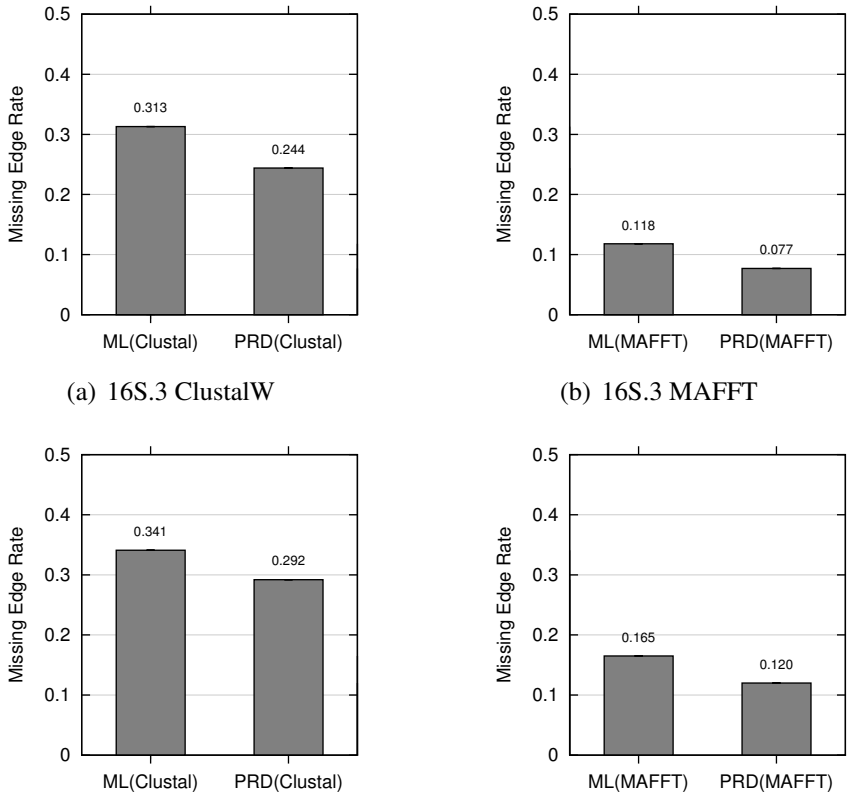
was also infeasible. Therefore, the goal with these datasets was to produce a tree that is as accurate as possible.

Since the padded RecDCM3 requires an input guide tree, I began by considering two alignment methods that quickly produce an alignment, though that alignment need not be very good. I used the `quicktree` option of ClustalW, and the `parttree` option of MAFFT to quickly generate an alignment for each dataset, and then used RAxML to estimate a tree from each of these alignments. Figure 7.6 shows the accuracy of these estimated trees, as well as the result of using each of these trees as the guide tree for pRecDCM3. As a first observation, the MAFFT alignments result in more accurate trees as compared the the ClustalW alignment, but in each case, applying the pRecDCM3 decomposition is able to boost the accuracy of the initial result.

Since the RAxML(MAFFT) trees were more accurate than the RAxML(ClustalW) trees, I used RAxML(MAFFT) trees as the starting trees for iteration with pRecDCM3. As Figure 7.7 shows, continued improvements are made using this technique. In fact, for the 16S.T dataset, it appears that the iterations may not yet have reached a fixed point, and further iterations might improve the result.

7.3 Conclusions

When an initial estimate of a tree is available, it is possible to improve upon this initial tree's accuracy using a divide-and-conquer technique that uses the initial tree as a guide tree. In this chapter I explored the use of a Recursive DCM3 decomposition, and compared this to a padded Recursive DCM3 decomposition. My



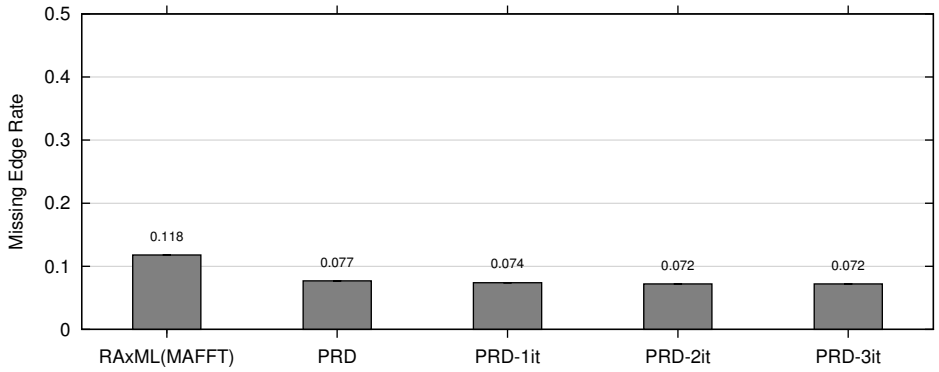
(a) 16S.3 ClustalW

(b) 16S.3 MAFFT

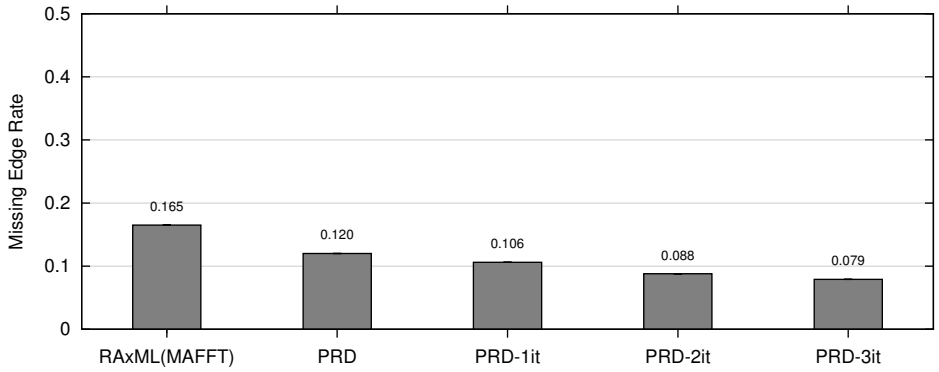
(c) 16S.T ClustalW

(d) 16S.T MAFFT

Figure 7.6: **Improvements to missing edge rate for Gutell datasets.** The missing edge rate are shown for each of several trees. ML(Clustal) is the tree estimated by RAxML on a ClustalW quicktree alignment. PRD(Clustal) is the tree estimated using a single application of the pRecDCM3 decomposition of the ML(Clustal) tree. ML(MAFFT) is the tree estimated by RAxML on the MAFFT parttree alignment. PRD(MAFFT) is the tree estimated using a pRecDCM3 decomposition of the ML(MAFFT) tree. For pRecDCM3, source trees were estimated using MAFFT alignments, and polytomies resolved using MRP with the default 100 ratchet iterations.



(a) 16S.3



(b) 16S.T

Figure 7.7: **Effects of pRecDCM3 iteration on Gutell data**. Missing edge rates of iterating using pRecDCM3 with maximum subset size 250 and targeting an overlap of 50 taxa. Source trees were estimated using MAFFT alignments, and polytomies resolved using MRP with the default 100 ratchet iterations.

results show that the padded decomposition results in improved accuracy as compared to the non-padded version. Further, iterating with this technique results in ever more accurate trees, even for large datasets with more than 5000 taxa.

There are still many things to be learned about these decompositions though. A much more thorough exploration of the parameter space, in particular the maximum subset size and desired overlap parameters, is required. In addition, techniques are needed to determine when "enough" iteration has been performed, and more experiments are desired to discover if changing parameters in the course of iteration might help move the technique away from local optima.

Chapter 8

Phylogenies Without a Full Alignment

In this chapter I explore another method by which to accomplish the first step of BLuTGEN, that is, dividing the original dataset into overlapping subsets. Chapter 7 showed that if given a reasonable initial estimate of the phylogeny, it is possible to improve the accuracy of the tree using a padded Recursive DCM3 decomposition. In this chapter, I consider what is possible when an initial estimate of the tree is not available. The goal here is to estimate a tree on par with existing two-phase techniques, but without ever generating an alignment for the entire dataset. Then, once I have a reasonable estimate, I can apply the padded Recursive DCM3 method to improve the result, and thus estimate an accurate tree without ever generating an alignment for the full dataset.

I develop several decomposition techniques each of which clusters the sequences into subsets based on their BLAST hit ordering. Each decomposition is named "BL" for "BLAST-based decomposition," and then subsequent letters indicate a variant. The first variant, simply "BL", does not work very well, and after a few initial experiments was discarded. The first successful decomposition was BLD ("D" for "distinct"), with BLS ("S" for "simplified") and BLF ("F" for "fast") improving upon this methodology in terms of speed, and to some extent, accuracy.

As I developed these methods I tested them on a variety of the datasets described in Chapter 6 choosing to attempt to understand the limitations of each method as opposed to exhaustively testing each method on all datasets.

In each of these approaches, I build a BLAST database from the set of sequences (using `formatdb`) and then BLAST individual sequences against this database (using `blastall`). The particulars of which sequences I use to BLAST against the database, and which hits form a set, result in several different variants. In every variant, the first set is created by taking the first sequence in the raw sequence file and using it to BLAST against the database. The top hits, up to the maximum subset size specified, make up the first set. The variants differ according to how they create subsequent hits.

At the end of this exploration, the BLF decomposition is the best in terms of speed, and also accuracy.

8.1 BL Decomposition

The first decomposition I tried, BL, has two additional parameters in addition to the maximum subset size: low and $high$. These parameters are used to bound the amount of overlap between the current set being created, and a previously generated set. As I create each set, I keep track of the number of taxa not previously part of some set added to the current set, as well as the amount of overlap with a previous set. Figure 8.1 gives the pseudocode for this decomposition.

To create subsequent sets, I consider the previous ordering, and use as a

```
1. Generate first set from firstOrdering
2. Generate currentOrdering from first taxon \
                 following those in first set
3. While some taxon is not assigned to a set
   3a. Reset variables: numberNew = 0
                        numberInCurSet = 0
                        setToMatch = unknown
                        seedForNextOrdering = unknown
   3b. foreach currentHit in currentOrdering
       If setList[currentHit] is undefined
         If numberNew > (#maxSize# - #low#)
            if seedForNextOrdering is unknown
                 seedForNextOrdering = currentHit
            else
                 move on to next hit
         else
            Add currentHit to currentSet
            update numberInCurSet, numberNew
       else
         If setToMatch is unknown:
             setToMatch = first set of setList[currentHit]
             Add currentHit to currentSet
         else
             if setList[currentHit] does not match \
                                        setToMatch
                move on to next hit
             else
                if (numberInCurSet - numberNew) < #high#
                   move on to next hit
                else
                   Add currentHit to currentSet
                   Update numberInCurSet
      If numberInCurSet = #maxSize#
         set currentOrdering to hit ordering from \
            BLASTing database using seedForNextOrdering
         break out of foreach loop
```

Figure 8.1: **BL decomposition.**

seed the first sequence from the ordering that is not yet contained in any set. I again consider the hits in order. If the hit is not yet contained in any set, and I have no more than $maxSize$–low new hits added to the current set, I add it to the current set. If I have already added the maximum number of new hits, and have not yet saved a hit as the next seed, I mark the current hit as the next seed. If the hit is contained in some set, and I have not yet found such a hit, I mark the set that contained the hit as the target overlap set, and add the hit to the current set. If I have already found a target overlap set, and this hit is not a member of the overlap set, or I have already found $high$ overlap hits, I discard the hit. This continues until I have found a next seed, and I have at least low overlap with a previous set, and $maxSize$–$high$ new hits.

Since my original goal was to create a method that could scale to large datasets, I began by considering the RNAsim datasets which have 8192 and 16384 taxa. I set the maximum subset size to 250, low to 25 and $high$ to 50. Unfortunately, using this BL decomposition as the first step to BLuTGEN results in rather inaccurate trees (see Figure 8.2), even when using the true alignment for subtree generation. Looking at the SCM and SuperFine tree accuracy though (Figure 8.3), I noted that each had very similar error, indicating that perhaps not enough overlap was being generated. In an attempt to address this, I used the same technique, but doubled the low and $high$ parameters. Figure 8.4 shows that the parameter change did result in a substantial improvement. However, the resulting trees still had much more error than desired, especially considering that the tree estimated by RAxML from the default MAFFT alignment indicated that these datasets are actually quite

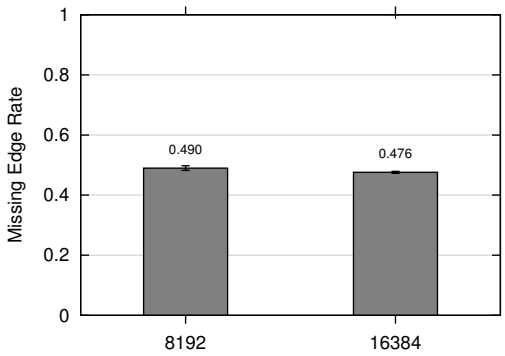


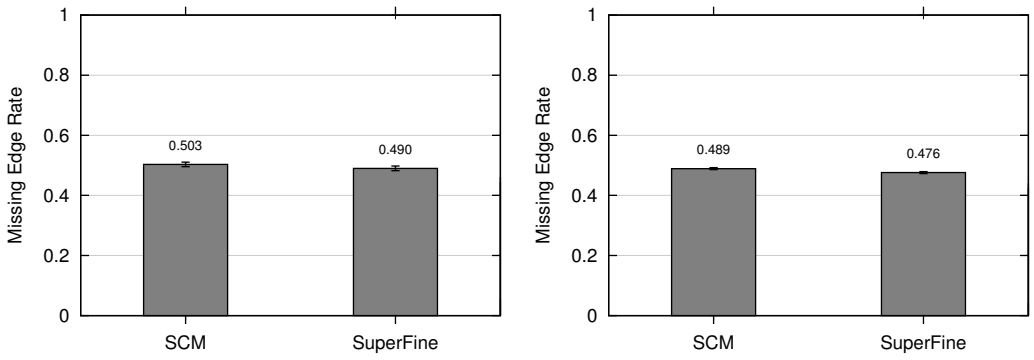
Figure 8.2: **BL decomposition on RNAsim data.** The missing edge rate is shown for the BL decomposition method, using true alignments for subset generation, and resolving polytomies with MRP. The maximum subset size is 250, $low = 25$ and $high = 50$.

easy.

8.2 BLD Decomposition

The "D" of this decomposition's name is for "distinct". This BLAST-based decomposition technique gets its name from the fact that it tries to create distinct subsets, that is, subsets with enough, but not too much overlap. It does this through the use of three new parameters, two of which (low and new) bound the amount of overlap between sets. The third new parameter, $seed$, gives a new starting position from which to find subsequent sets. Figures 8.5–8.7 give pseudocode for this variant.

As with all BLAST-based decompositions, the first set is simply the top hits from the ordering creating by BLASTing the first sequence in the file against the



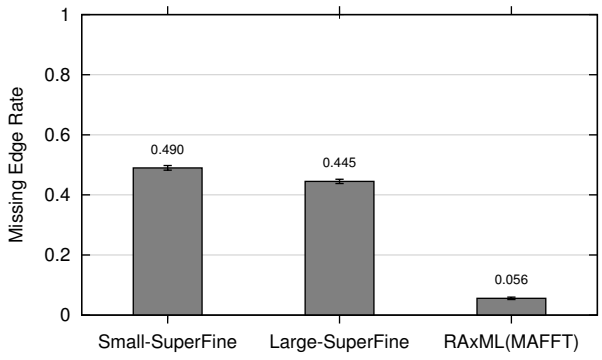
(a) 8192 (b) 16384

Figure 8.3: **Comparison of SCM and SuperFine trees for BL decomposition on RNAsim data.** The missing edge rate is shown for the SCM and SuperFine trees resulting from the BL decomposition method, using true alignments for subset generation, and resolving polytomies with MRP. The maximum subset size is 250, $low = 25$ and $high = 50$.

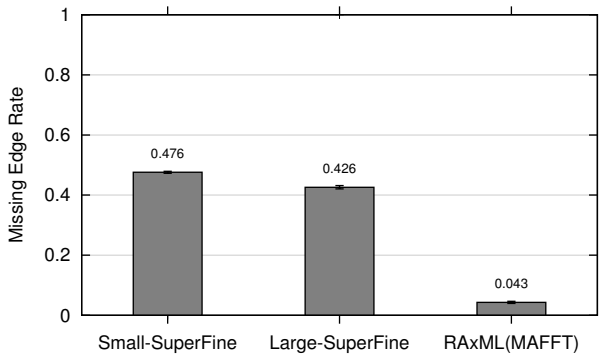
database. Creating subsequent sets proceeds in three phases:

1. Strict: Consider the top $maxSize$ hits using the hit from the previous set in position $seed$ as the seed. If the set has low overlap with some previously defined set, and has at least new number of hits not already in some set, accept the set. If this new set under consideration does not meet the criteria, consider in turn as seeds the hits from the previous set following the initial seed hit until some seed results in an accepted set, which then becomes the previous set in the next loop iteration. If at any time all taxa have been put in some set, the decomposition completes.

2. Relaxed: If some taxa are not in a set after the strict phase, I relax the requirement that a new set must have a least new number of hits not already



(a) 8192



(b) 16384

Figure 8.4: **Two BL decompositions on RNAsim data.** The missing edge rates are shown for two parameter choices for the BL decomposition method, using true alignments for subset generation, and resolving polytomies with MRP. Both Small and Large have a maximum subset size of 250. Small sets $low = 25$ and $high = 50$ while Large sets $low = 50$ and $high = 100$.

```
1. Generate first set from firstOrdering
2. Set previousOrdering to firstOrdering
3. Generate currentOrdering from taxon in position seed \
                                        in first ordering
4. While possible seeds have not been exhausted
   4a. Consider set of top #maxSize# hits in \
                               currentOrdering
       If consideredSet has #low# overlap with some \
                        previous set and \
                        has at least #new# new hits
           Accept consideredSet
           Set possibleSeed to taxon in position seed of \
                                          currentOrdering
       else
           Set possibleSeed to taxon in position one \
                 further past current seed (if possible) \
                 in previousOrdering
       If each taxon is in some set
           Break from while loop
```

Figure 8.5: **Strict phase of BLD decomposition.**

```
1. Set previousOrdering to be what Strict phase last had
2. Generate currentOrdering from taxon in position #seed# \
                                      in previousOrdering
3. While possible seeds have not been exhausted
   3a. Consider set of top #maxSize# hits in \
                               currentOrdering
       If consideredSet has #low# overlap with some \
                                    previous set and \
                        has at least 2 new hits
           Accept consideredSet
           Set possibleSeed to taxon in position #seed# \
                                    of currentOrdering
       else
           Set possibleSeed to taxon in position one \
           further past current seed (if possible) in \
                                    previousOrdering
       If each taxon is in some set
           Break from while loop
```

Figure 8.6: **Relaxed phase of BLD decomposition.**

```
1. Foreach taxon not currently in some set
   1a. Generate currentOrdering using current taxon as \
                                                    seed
   1a. Consider set of top size hits in currentOrdering
       If consideredSet has low overlap with some \
                                      previous set
           Accept consideredSet
       If each taxon is in some set
           Break from loop
```

Figure 8.7: **Loose phase of BLD decomposition.**

in some set. Instead, I simply require at least two hits not in some previous set. I consider again all hits after the $seed$ hit in the most recent ordering that resulted in an accepted set.

3. Loose: If some taxa are still not in a set, consider each as a seed. If the resulting set has low overlap with some previously defined set, accept the set, and update the list of taxa not yet in a set accordingly so as not to consider them as a seed.

In the Loose phase, a single pass through the set of taxa not assigned to a subset is made. If at the end of the Loose phase the completed sets do not contain all taxa, the decomposition fails. Note that it is possible that further passes through the taxa not assigned to a subset could result in a valid decomposition that otherwise fails, but I do not currently pursue this option.

I began by testing this method on the ROSEsim datasets. Figure 8.8 shows

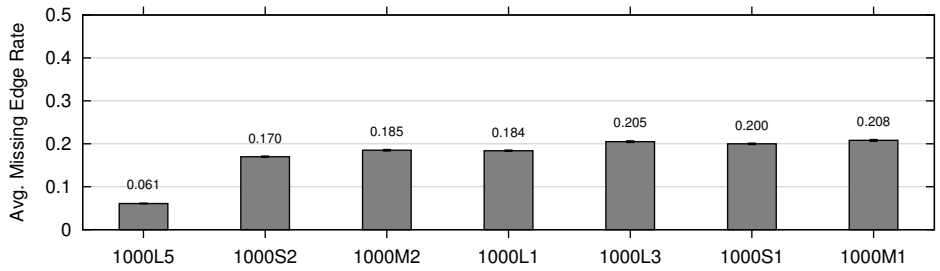


Figure 8.8: **BLD decomposition average source tree error for ROSEsim data.** The average missing edge rate is shown for the source trees estimated from the BLD decomposition method, using MAFFT alignments for subset generation.

the average subset tree error for this decomposition, which indicates that I have managed to create reasonably accurate source trees (though they could be improved) and Figure 8.9 shows the resultant full tree errors. The BLD errors are still significantly greater than RAxML(MAFFT), but the BLD errors are more reasonable than those the BL decomposition returned. Further, the ROSEsim datasets are more difficult than the RNAsim datasets, so even though I do not give a direct comparison of the BL and BLD decompositions, the BLD decomposition is preferred.

Though it is possible that the BLD decomposition could fail to return a successful decomposition, for the ROSEsim datasets it always returned a valid decomposition, and results in trees with a more reasonable accuracy. Unfortunately it cannot be applied to larger datasets due to computation time. Attempts to use the BLD decomposition for the RNAsim datasets failed to complete after more than 15 hours of processing. Looking for the bottleneck of the algorithm, I soon discovered

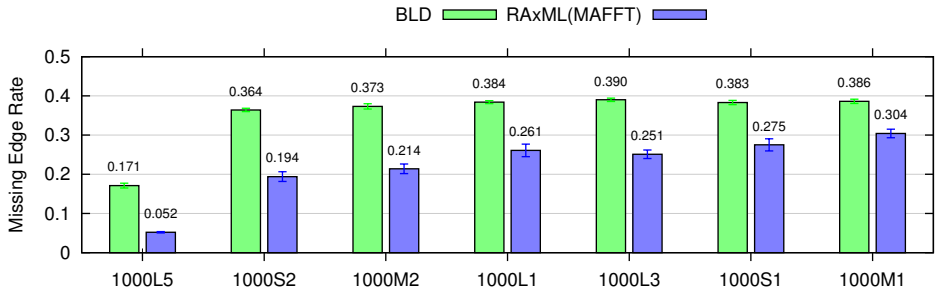


Figure 8.9: **Missing edge rate of BLD decomposition on ROSEsim data.** The missing edge rate is shown for the BLD decomposition method, using MAFFT alignments for subset generation, and resolving polytomies with MRP as well as for the RAxML(MAFFT) tree.

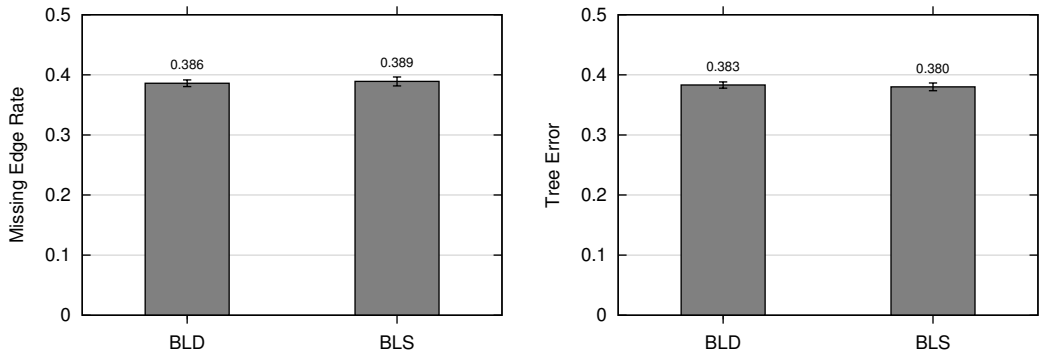
that the many calls to `blastall` were to blame. On a decent desktop machine (3.4GHz processor), it takes approximately 0.07 seconds to create a hit ordering from a database with 1000 taxa, 26 seconds for a database with 8192 taxa, and 54 seconds when the database has 16384 taxa. Despite the fact that 54 seconds seems a trivial amount of time, the BLD decomposition in the worst case considers every taxon as a seed, which would mean more than 10 days of processing for datasets with 16K taxa.

8.3 BLS Decomposition

To see if I could speed up the BLD decomposition, without degrading performance, I created the BLS decomposition ("S" for "simplified"), which is the same as BLD, but removes Phase 2 (thus the name). Using the BLS decomposition resulted in much faster decompositions for the 1000 taxon datasets (see Table 8.1), and maintained about the same accuracy (see Figure 8.10).

Model Condition	Method	Decomposition Time (secs)
1000M1	BLD-50-250	223.1 (23.7)
1000M1	BLS-50-250	25.5 (1.5)
1000S1	BLD-50-250	223.4 (20.3)
1000S1	BLS-50-250	22.0 (1.3)

Table 8.1: **BLD and BLS decomposition times.** For simulated data, all times are given in seconds, and are averages over 20 replicates per model condition, using a heterogeneous mix of computers available through the UTCS condor system. Standard errors shown in parentheses.



(a) 1000M1 (b) 1000S1

Figure 8.10: **Comparison of BLD and BLS decomposition missing edge rates.** The missing edge rates are shown for the BLD and BLS decomposition methods, using MAFFT alignments for subset generation, and resolving polytomies with MRP.

Unfortunately, it still takes unacceptably long to generate the BLS decomposition for larger datasets, taking about seven hours for the 16S.3 dataset with 6263 taxa, and over eight hours for the 16S.T dataset with 7350 taxa (exact hardware unknown, since run through the UTCS condor system). Even more troubling, the decomposition was not successfully generated! Instead, after the last phase, the union of the taxa in all sets was a proper subset of the taxa in the original dataset. The BLS decomposition is the same as the BLD decomposition, just less restrictive in the generation of early sets, indicating that if the BLD decomposition could be run to completion for these datasets, it too would likely fail to produce a valid decomposition. Of course, since the early sets might be different, it is possible that the BLD decomposition could succeed where the BLS decomposition fails. Either way though, the BLS decomposition is still slower than desired.

8.4 BLF Decomposition

The BLF decomposition ("F" for "Fast", since that was the goal of this variant) overcomes the limitations of the BLD and BLS decompositions by reducing the number of calls to `blastall` and guaranteeing that source trees generated as a result of the decomposition will be an acceptable input to a supertree method. The pseudocode for this method is given in Figure 8.11.

The BLF decomposition creates the first set just as all BLAST-based decomposition method do, and has just one additional parameter: low. In this variant, to create subsequent sets, I consider the previous order, and use as a seed the first sequence after the hits that made up the previous set that is not yet in a set. I con-

```
1. Generate first set from firstOrdering
2. Generate currentOrdering from first taxon \
                  following those in first set
3. While some taxon is not assigned to a set:
   3a. Reset variables: numberInCurSet = 0
                        seedForNextOrdering = unknown
   3b. Foreach currentHit in currentOrdering
       If numberInCurSet < #maxSize#
            Add currentHit to currentSet
            Update numberInCurSet
       else
         If setList[currentHit] is unknown
             Set seedForNextOrdering to currentHit
         else
             Move on to next hit
      If numberInCurSet = #maxSize# and\
                          seedForNextOrdering is known
         set currentOrdering to hit ordering from \
            BLASTing database using seedForNextOrdering
         break out of foreach loop
4. Create overlap graph
5. Foreach set, use overlap graph to determine if more \
                                overlap should be added
   5a. If overlap is too low, determine highest \
                                 overlapping set
   5b. Using a common taxon to highest overlapping set,\
                              and to create hitOrdering
   5c. Loop through hitOrdering
       If currentHit adds overlap to highest overlapping set
          Add currentHit to currentSet
```

Figure 8.11: **BLF decomposition.**

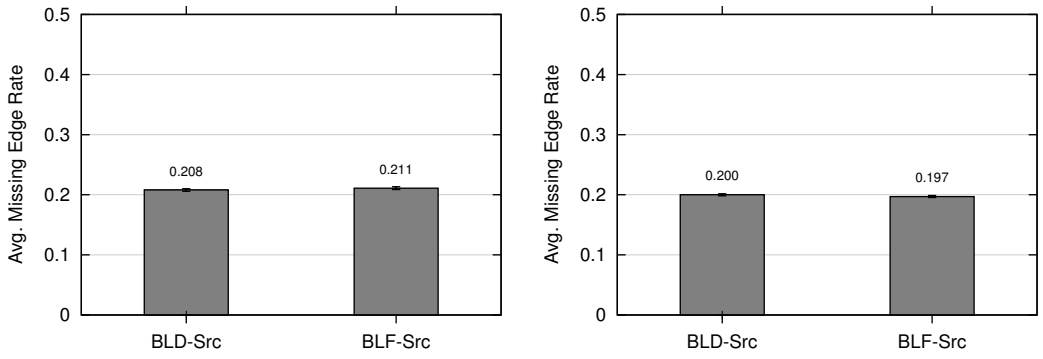
Dataset/Model	BLS Decomposition Time	BLF Decomposition Time
1000M1	25.5 (1.47) secs	< 1 sec
1000S1	21.95 (1.33) secs	< 1 sec
16S.3	7 hours	45 min
16S.T	8+ hours	45 min

Table 8.2: **BLS and BLF decomposition times.** For the simulated data, all times are averages over 20 replicates per model condition (single data point for real data). All times are the result of using a heterogeneous mix of computers available through the UTCS condor system. Standard errors (where applicable) are given in parentheses.

tinue until every sequence is in some set. However, at this point, there may not be sufficient overlap, so I post-process the sets. I create an overlap graph, where the nodes of the graph are the sets, and there is an edge between the nodes if the corresponding sets have at least low overlap. I then add hits to sets as necessary to ensure low overlap with some other set, and to ensure that the overlap graph is connected.

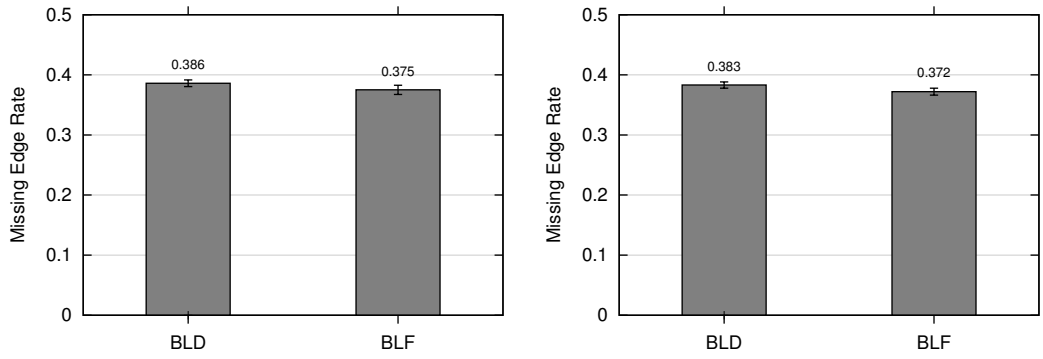
Figure 8.12 compares the BLD and BLF decompositions source tree error rates for the ROSEsim datasets, while Figure 8.13 compares the resulting full tree error rates. It is clear that the BLF decomposition results in trees that are just as accurate, and perhaps even better, than those generated with the BLD decomposition. Further, as Table 8.2 shows, it is also a win in terms of running time, making BLAST-based decompositions feasible for datasets with over 5000 taxa.

To assess the impact of the low and $maxSize$ parameters, I decreased these parameters, naively thinking that smaller subsets would translate to more accurate



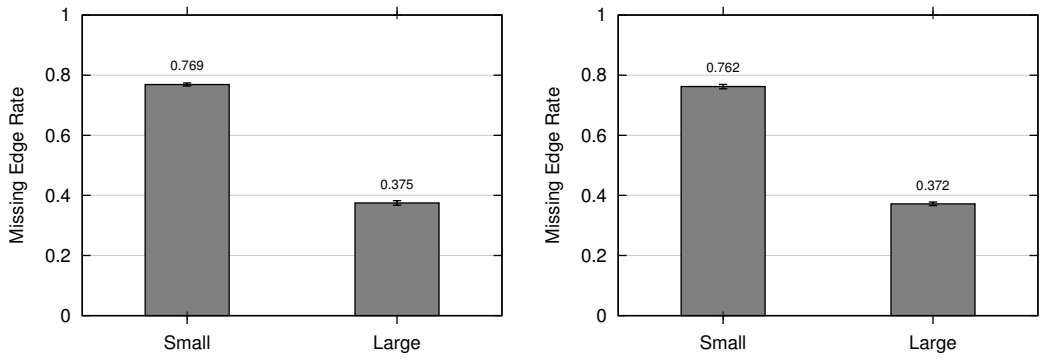
(a) 1000M1 (b) 1000S1

Figure 8.12: **Comparison of BLD and BLF decomposition average source tree error rates for ROSEsim datasets.** The average missing edge rates for source trees are shown for the BLD and BLF decomposition methods, using MAFFT alignments for subset tree generation. Both methods use a maximum subset size of 250, and low set to 50.



(a) 1000M1 (b) 1000S1

Figure 8.13: **Comparison of BLD and BLF decomposition error rates for ROSEsim datasets.** The missing edge rates are shown for the BLD and BLF decomposition method using source trees estimated using MAFFT alignments, and polytomies resolved using MRP. Both methods use a maximum subset size of 250, and low set to 50.

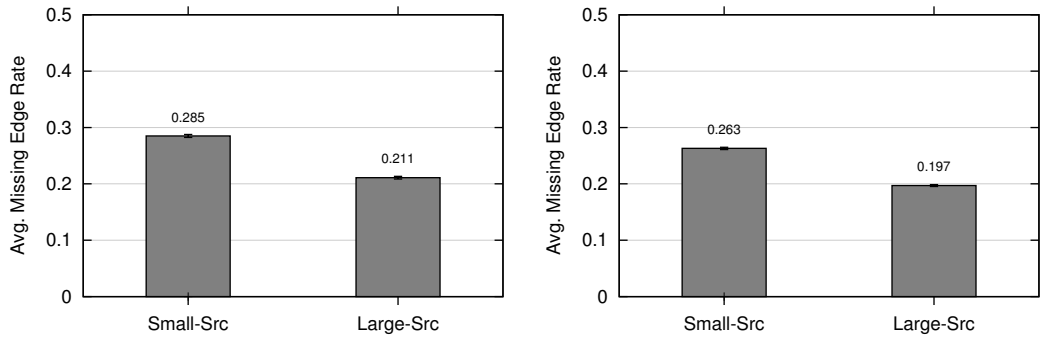


(a) 1000M1 (b) 1000S1

Figure 8.14: **Comparison of BLF decomposition error rates given different parameters for ROSEsim datasets.** The missing edge rate is shown for two versions of the BLF decomposition method, using MAFFT alignments for subset generation, and resolving polytomies with MRP. Small sets $low = 25$ and the maximum subset size to 100 while Large sets $low = 50$ and the maximum subset size to 250.

source trees, and thus more accurate full trees. However, as Figure 8.14 shows, this was not the case. Instead, the source tree accuracy was reduced (see Figure 8.15), perhaps because of insufficient taxon sampling, resulting in significantly less accurate full trees.

Now, to determine if this BLF decomposition would be practical for even larger datasets, I used it on the RNAsim datasets, setting the maximum subset size to 250, and minimum overlap parameter low to 50. Figure 8.16 compares the BL and BLF decompositions (since BL was the last successful decomposition for these datasets). The BLF decomposition resulted in trees with much lower error for the RNAsim datasets even though the BL decomposition had several advantages. First, the results for the BL decomposition use true alignments to generate subtrees, while

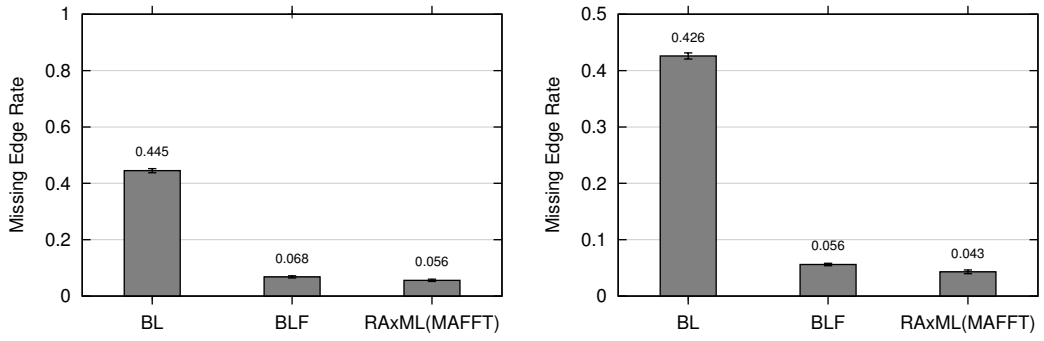


(a) 1000M1 (b) 1000S1

Figure 8.15: **BLF decomposition average source tree error.** The average missing edge rates for source trees produced by two versions of the BLF decomposition method. Source trees were estimated using MAFFT alignments. Small sets $low = 25$ and the maximum subset size to 100 while Large sets $low = 50$ and the maximum subset size to 250.

the results for the BLF decomposition use alignments estimated by MAFFT. Second, the results for the BLF decomposition of the 16384 dataset are based on a non-standard (fast, but less accurate) version of SuperFine due to the extreme running time of the usual method (see Section 9.1 for further details).

To conclude this section, Figures 8.17 and 8.18 show the accuracy of the BLF method on the full set of ROSEsim and Gutell datasets. Though the BLF method does not result in trees more accurate than RAxML(MAFFT) for the ROSEsim datasets, given that no alignment is ever generated these results are positive. Further, for the two smaller Gutell datasets (16S.M and 16S.M.aa_ag) the BLF decomposition results in a tree with accuracy on par with that of the RAxML(MAFFT) tree (where MAFFT was run using the most accurate linsi option). For the two



(a) 8192 (b) 16384

Figure 8.16: **Comparison of BL and BLF decomposition error rates for RNAsim datasets.** The missing edge rate is shown for the BL and BLF decomposition methods. The BL method uses true alignments for subtree generation, and resolves polytomies with the default of SuperFine (QMC). The BLF method uses alignments estimated by MAFFT for subtree generation, and resolves polytomies using a time-limited MRP.

larger Gutell datasets (16S.3 and 16S.T), where the most accurate version of MAFFT is no longer feasible, the BLF decomposition result is considerably better than the tree estimated by RAxML using the alignment generated by the parttree version of MAFFT.

8.5 Future Directions

While the final tree error using the BLF decomposition method is still quite high, I now have a "reasonable", or at the very least no longer horrible, starting tree, for which no full alignment was ever generated. I can now use this result as the guide tree to a pRecDCM3 decomposition, and thereby estimate an accurate tree without generating a full alignment.

As with the pRecDCM3 decomposition, there is still much to be discovered and optimized about this approach. First, as Figure 8.14 indicated, appropriate parameters are required in order for BLF to work well. A thorough exploration of the impact of parameter choice on the resulting full trees would be enlightening. Also, as will be explored further in Chapter 9, the BLF decomposition tends to create large polytomies in the first part of SuperFine that then must be resolved. It would be preferable to find an alternative to BLF that maintained its speed and accuracy, but did not result in such large polytomies.

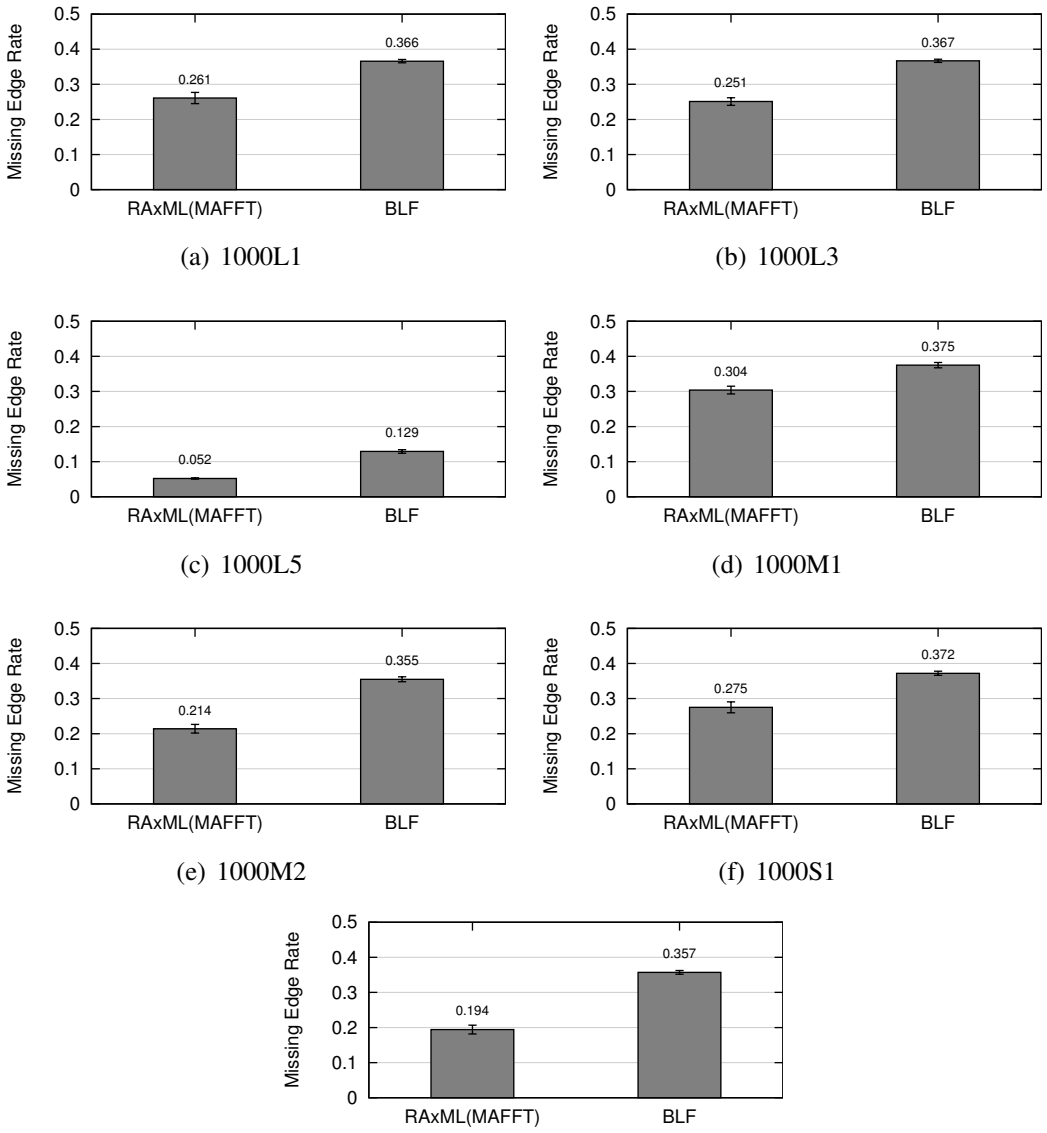
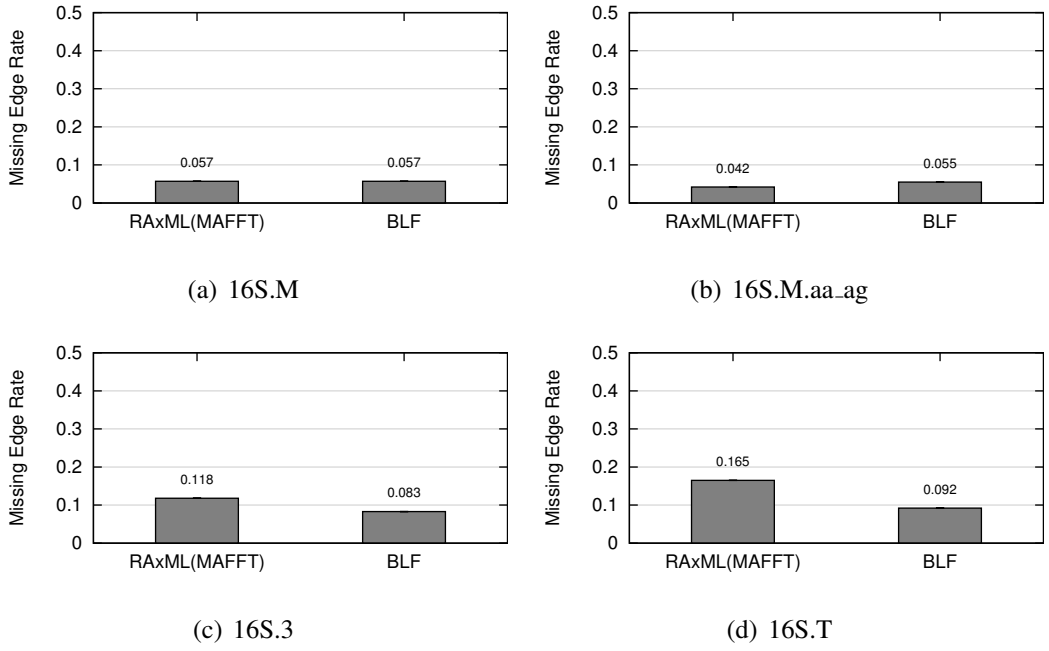


Figure 8.17: **BLF decomposition error for ROSEsim datasets.** The missing edge rates are shown for the BLF decomposition method using source trees estimated using MAFFT alignments, and polytomies resolved using MRP (maximum subset size of 250, and low set to 50. The accuracy of RAxML(MAFFT) is shown for comparison, where MAFFT was run in its most accurate setting, linsi.



(a) 16S.M

(b) 16S.M.aa_ag

(c) 16S.3

(d) 16S.T

Figure 8.18: **BLF decomposition error for Gutell datasets.** The missing edge rates are shown for the BLF decomposition method using source trees estimated using MAFFT alignments, and polytomies resolved using MRP (maximum subset size of 250, and low set to 50. The accuracy of RAxML(MAFFT) is shown for comparison. MAFFT was run in its most accurate setting, linsi, for 16S.M and 16S.M.aa_ag and using the fast parttree algorithm for the remaining two datasets.

Chapter 9

Extending BLuTGEN

In the past few chapters I have described the development of BLuTGEN, which works by dividing the dataset into subsets, estimating trees for each subset, and then assembling a full tree from the subtrees. In order to get a tree without ever generating an alignment, BLuTGEN uses a BLF decomposition to build an initial estimate of the phylogeny for a set of unaligned sequences, and then improves this estimate using a padded recursive DCM3 decomposition. In this chapter I consider an improvement to the basic algorithm that allow BLuTGEN to be used for large datasets, and also consider the impact of starting tree for iterative pRecDCM3 decompositions. Finally, I apply the pRecDCM3 decomposition to a very large dataset (27643 taxa) where even generating a reference tree is problematic.

9.1 Improving Merge Times

To assess the effectiveness of BLuTGEN on large datasets, I used the BLF decomposition method for the 16S.3 Gutell dataset, and attempted to use it for the 16S.T dataset. Unfortunately, after over 5 days of processing, SuperFine had failed to produce a tree for the 16S.T dataset, though I did manage to recover the SCM tree produced in the first phase of SuperFine. This long computation time stemmed

from the impact of the polytomy degrees in the SCM tree which is an intermediate result of SuperFine.

When the largest degree polytomy in the SCM tree is very big, resolving the polytomy of the SCM tree becomes a bottleneck. The default behaviour of SuperFine, when resolving with MRP, is to do a careful search for the best resolution, and then take a greedy consensus of the best trees that resulted from the search. Unfortunately, the time for this search can be extreme if the polytomy is large.

Given that I often have large degree polytomies when using a BLAST-based decomposition (see Section 9.3), and that I then randomly refine this result before using it as input to a padded RecDCM3 decomposition (since the current implementation of pRecDCM3 requires a binary input tree), I wondered if reducing the accuracy of the resolution step could reduce the time required by BLuTGEN, without substantially degrading performance.

The 16S.3 BLF SCM result has 478 polytomies, where the largest degree polytomy has degree 1339. The 16S.T BLF SCM result has 533 polytomies, with the largest degree polytomy having degree 2495. Figure 9.1 compares the accuracy of the SCM tree, the SCM tree randomly-refined, and the result of using SuperFine resolving with MRP in a very fast manner (a single two hour MRP tree search). Figure 9.1(a) also shows the accuracy of the SuperFine result for 16S.3 when a more thorough search for the MRP tree was completed (the analogous result for 16S.T was not available after more than 5 days of processing). Though the SuperFine result with a careful ratchet search is indeed the best, using the much faster (see Table 9.1) time-limited search does not hurt accuracy enough to warrant the

Dataset	Resolution Type	Time
16S.3	Random refinement	3.5 seconds
	Time limited fast MRP search	1.95 hours
	Default MRP search	19.7 hours
16S.T	Random refinement	4.7 seconds
	Time limited fast MRP search	10.07 hours
	Default MRP search	> 5 days

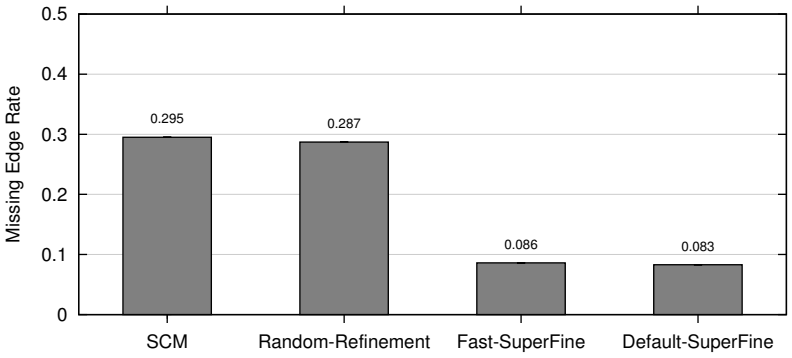
Table 9.1: **Resolution times.** For the large 16S datasets, a comparison of running times for variations of the refinement step of SuperFine. Note that times are not directly comparable as several different machines were used.

considerable increase in computation time.

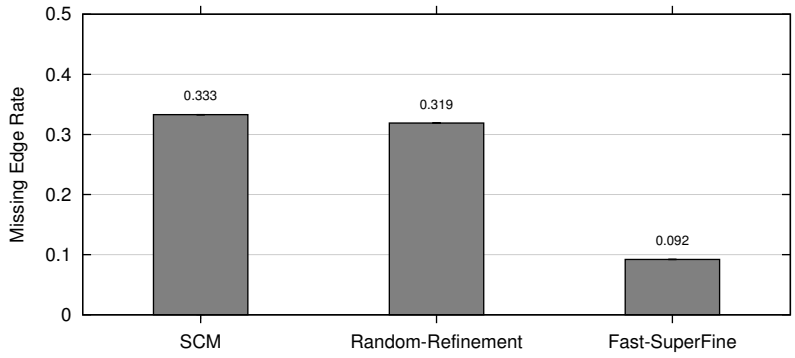
9.2 Starting Tree Choice

The BLAST-based BLF decomposition works well to generate an tree without ever generating an alignment for the full dataset. But if a reasonable two-phase method could be applied (such as a MAFFT alignment followed by estimating a tree with RAxML), the question becomes whether or not using the BLAST-based decomposition is worthwhile. To explore this, I used the 16S.3 and 16S.T Gutell datasets and considered starting an iterative pRecDCM3 approach starting from a quick and dirty alignment (in this case the parttree alignment available in MAFFT) as compared to starting from the BLF decomposition result (using the fast MRP searches). Figure 9.2 shows that starting from either of the BLF-based or MAFFT-based trees quickly converges to trees with very similar accuracy.

I also used two of the hard ROSEsim datasets for a similar comparison, where I used a more accurate version of MAFFT (linsi) to generate the RAxML(MAFFT)

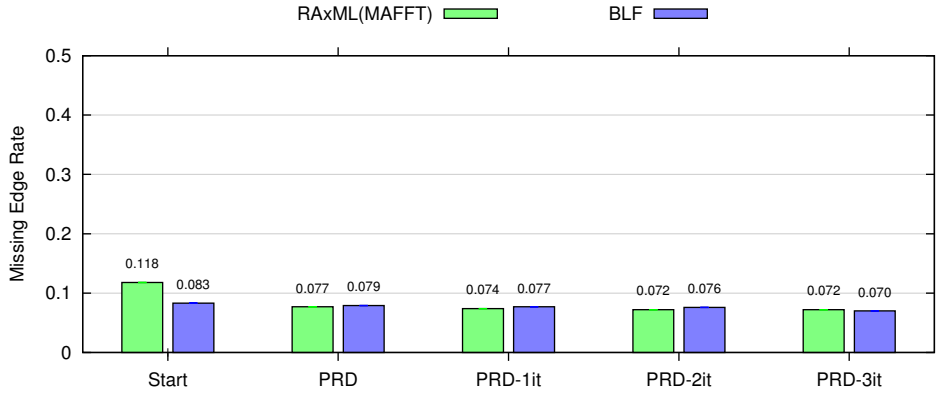


(a) 16S.3

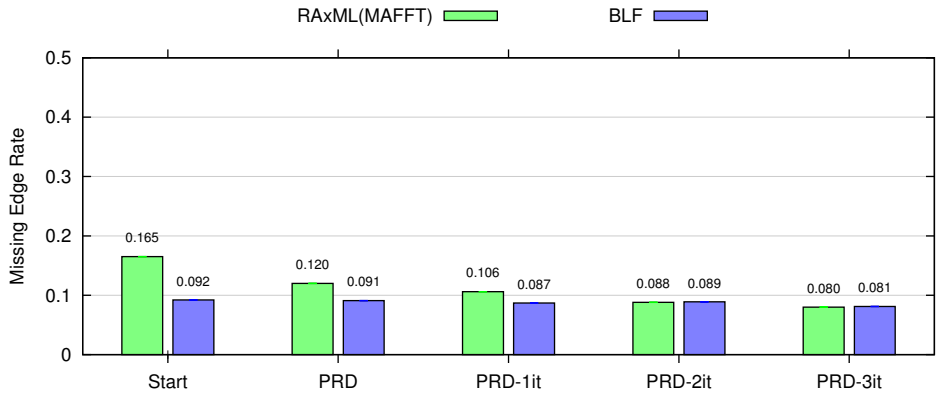


(b) 16S.T

Figure 9.1: **Comparison of accuracy of several resolutions of SCM tree for Gutell datasets.** Missing edge rates are given. SCM is the strict consensus merger tree that is the result of the first stage of SuperFine. Random-Refinement is the SCM tree randomly refined. Fast-SuperFine is the SCM tree resolved using MRP, but using a fast search for each MRP tree. Default-SuperFine resolves the SCM tree in the default manner (using the greedy consensus of 100 ratchet iterations for the MRP tree with no time limits for each polytomy).



(a) 16S.3



(b) 16S.T

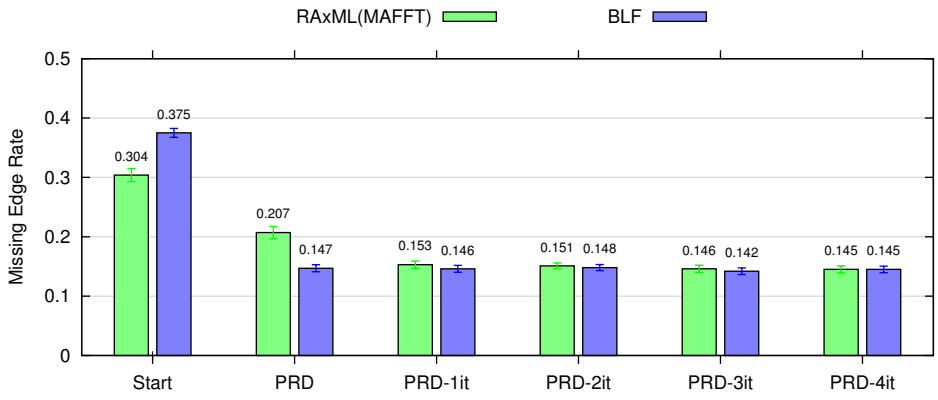
Figure 9.2: **Impact of starting tree on iterative pRecDCM3 decompositions on Gutell datasets.** The accuracy of each starting tree (RAxML(MAFFT) and BLF result) are shown together with four iterative applications of the pRecDCM3 decomposition (maximum subset size of 250, targeted overlap of 50). All decompositions used MAFFT subset alignments and default MRP searches to resolve polytomies.

starting tree (see Figure 9.3), and used the more thorough MRP search option to resolve polytomies. Again, starting from either the BLF or RAxML(MAFFT) trees quickly converges to a result tree with similar accuracy, but starting from the BLF tree reaches convergence in fewer iterations.

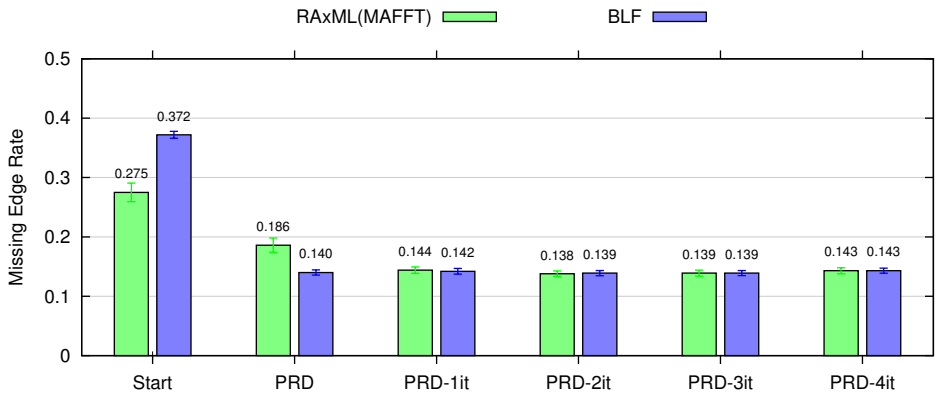
9.3 Understanding Decompositions

To further understand the decomposition methods, Table 9.2 shows a variety of the statistics for two ROSEsim model conditions, the 16S.3 and 16S.T datasets, and several decomposition methods.

For each dataset size, each of the BLAST-based decomposition methods generate about the same number of subsets, but the pRecDCM3 decomposition results in much fewer. Also, the average size of the pRecDCM3 decompositions are smaller, which translates to faster MAFFT alignments, and faster RAxML tree estimations, as compared to the BLAST-based decompositions (see Table 9.3). A smaller average subset size, and a smaller number of subsets means that the pRecDCM3 decompositions have much less overlap than the BLAST-based decompositions, which is reflected in the maximum polytomy size resulting in the SCM tree. This can also been seen by comparing the time required to merge the source trees from the various decompositions: merging from pRecDCM3 decompositions is much faster than merging BLAST-based decompositions.



(a) 1000M1



(b) 1000S1

Figure 9.3: **Impact of starting tree on iterative pRecDCM3 decompositions for ROSEsim datasets.** The accuracy of each starting tree (RAxML(MAFFT) and BLF result) are shown together with five iterative applications of the pRecDCM3 decomposition (maximum subset size of 250, targeted overlap of 50). All decompositions used MAFFT subset alignments and default MRP searches to resolve polytomies.

Model Condition	Method	Number of Src Trees	Avg Src Size	SCM Max Poly degree
1000M1	BLD-50-250	31.4 (0.3)	250 (0)	758.4 (5.8)
	PRD-BLD-50-250	11.1 (0.3)	131.3 (3.2)	53.9 (3.4)
	BLS-50-250	29.1 (0.4)	250 (0)	417.5 (25.3)
	PRD-BLS-50-250	10.7 (0.3)	135.1 (3.4)	54.2 (4.3)
	BLF-25-100	49.0 (0.6)	123.8 (0.1)	657.1 (24.3)
	BLF-50-250	30.3 (0.4)	250 (0)	407.0 (34.9)
	PRD-BLF-50-250	10.9 (0.3)	133.2 (3.4)	53.5 (4.1)
1000S1	BLD-50-250	30.1 (0.5)	250 (0)	752.8 (5.5)
	PRD-BLD-50-250	11.1 (0.3)	132.9 (3.4)	48.0 (3.2)
	BLS-50-250	28.3 (0.5)	250 (0)	330.9 (24.6)
	PRD-BLS-50-250	10.8 (0.3)	134.6 (3.6)	47.8 (3.4)
	BLF-25-100	48.9 (0.6)	123.7 (0.1)	613.2 (24.8)
	BLF-50-250	29.3 (0.6)	250 (0)	363.1 (27.3)
	PRD-BLF-50-250	10.7 (0.3)	135.1 (3.5)	48.2 (3.1)
16S.3	BLF-50-250	245	252.0 (0.6)	1339
	PRDF-50-250	70	133.7 (5.4)	124
16S.T	BLF-50-250	291	252.1 (0.6)	2495
	PRDF-50-250	75	139.9 (6.2)	284

Table 9.2: **A comparison of decomposition statistics.** The number of source trees resulting from the decomposition, the average number of taxa per subset, and the maximum polytomy degree in the SCM tree are given. The low parameter and maximum subset size are given for each decomposition. PRD-BLD indicates a padded RecDCM3 decomposition of the BLD result. PRD-BLS indicates a padded RecDCM3 decomposition of the BLS result. PRD-BLF indicates a padded RecDCM3 decomposition of the BLF result. Standard errors are given in parentheses.

Model Condition	Method	Decomp	Avg Src Align	Avg Src Tree	Merge	Serial Total (hours)
1000M1	BLD-50-250	223.1 (23.6)	1228.5 (10.1)	1661.0 (26.5)	1174.9 (357.3)	25.6
	PRDD-50-250	0.35 (0.16)	386.1 (18.8)	465.6 (21.2)	14.6 (1.3)	2.6
	BLF-25-100	<1	262.0 (1.6)	431.6 (6.2)	2544.5 (224.8)	10.2
	BLF-50-250	<1	1183.1 (8.2)	1830.8 (38.2)	529.4 (81.1)	25.5
	PRDF-50-250	0.3 (0.14)	383.8 (18.9)	464.2 (23.2)	3.0 (0.5)	2.6
1000S1	BLD-50-250	223.4 (20.3)	1121.5 (7.7)	1424.8 (27.4)	785.5 (125.0)	21.6
	PRDD-50-250	0.2 (0.13)	349.2 (16.7)	462.7 (24.3)	4.35 (0.6)	2.5
	BLF-25-100	<1	247.2 (1.7)	374.7 (5.4)	3372.2 (309.9)	9.4
	BLF-50-250	<1	1096.1 (7.4)	1294.1 (21.4)	760.5 (158,0)	19.7
	PRDF-50-250	0.3 (0.12)	378.0 (19.6)	421.2 (20.2)	2.0 (0.6)	2.4
16S.3	BLF-50-250	2710	2027.8 (34.0)	433.8 (12.5)	8257	170.6
	PRDF-50-250	136	789.5 (73.6)	260.3 (28.3)	337	20.5
16S.T	BLF-50-250	2710	1845.2 (35.2)	481.6 (13.7)	13620	192.6
	PRDF-50-250	820	806.0 (81.4)	349.6 (40.2)	1579	24.7

Table 9.3: **Running times.** All times are given in seconds unless otherwise indicated. For simulated data, all times are averages over 20 replicates per model condition; analyses were run using a heterogeneous mix of computers available through the UTCS condor system. The low parameter and maximum subset size are given for each decomposition. PRDD indicates a padded RecDCM3 decomposition of the BLD result. PRDF indicates a padded RecDCM3 decomposition of the BLF result. Standard errors are given in parentheses. Serial total was computed using the average times for each subset analysis. Times for PRD analyses do not include the time to generate the tree used as input to the decomposition.

Dataset	Taxa	Sites	MNHD (%)	ANHD (%)	Gap (%)	Avg Gap Len	Ref. Tree Resolut'n
16S.M	901	4722	88.7	35.9	78.1	17.2	46.9
16S.M.aa_ag	1028	4907	100	34.2	82.6	22.0	42.4
16S.3	6323	8716	83.3	31.5	82.1	9.4	50.2
16S.T	7350	11856	90.1	34.5	87.4	12.1	49.5
16S.B.ALL	27643	6857	76.9	21.0	80.0	4.9	?

Table 9.4: **Gutell dataset statistics with 16S.B.ALL.** The number of taxa, number of sites, average normalized Hamming distance (ANHD), percentage of the matrix which is a gap character (Gap), and average gap length (Avg Gap Len) are shown. Also, the resolution of the reference tree is given for each dataset.

9.4 And now for something really big

The Gutell data I have presented thus far are just a few of the curated rRNA datasets available from Robin Gutell's comparative RNA database [12]. One more dataset available here is the 16S.B.ALL dataset, which, after being "cleaned" like the others (see Section 6.2.0.3), has 27643 taxa. Table 9.4 gives further details about this alignment in addition to the previous Gutell datasets.

For this 16S.B.ALL dataset, a single estimate of the tree from the curated alignment took over a week of processing (though the wall-clock time was less since a threaded version of RAxML was used). Thus, no bootstrapping was accomplished for this dataset, and therefore the only reference tree that exists for this dataset is the single RAxML tree from the curated alignment. While this reference tree is actually fully resolved, I list the resolution of the reference tree as "?". This is due the fact that if I could do bootstrapping, I would not expect a highly resolved tree; the average normalized Hamming Distance of the curated alignment – 21%

– is small enough that there are likely short edges in the true tree. Further, the number of characters/sites in the dataset, 6857, is so small in comparison to the number of taxa that a highly supported fully resolved tree is very unlikely – each character would need to be informative for more than four edges in the tree. Thus, given the curated alignment parameters, I would not expect the 75% bootstrap tree to be more than 30% or 40% resolved. This means that, using the current reference tree, missing edge rates of 60-70% *might* be the best possible, even using a highly accurate alignment.

I computed the BLF decomposition for this dataset, and successfully generated a tree for each subset. Unfortunately, the SCM tree from the first step of SuperFine has 1174 polytomies, and a resulting resolution of 26%. Though most of these polytomies have degree less than 200 (easily handled by an MRP resolution), the maximum polytomy degree is 15533. To resolve this polytomy with MRP, the SuperFine algorithm creates a matrix to be analyzed using maximum parsimony. In particular, SuperFine relabels source trees to reduce the problem size [93] and codes the resulting edge information as binary characters, but this results in over 471K characters. Thus, to resolve this 15533 degree polytomy using MRP requires writing an input file for PAUP that takes more than the available 2G of space in the /tmp directory on the machine I used. Even if SuperFine were changed so that the PAUP file could be written to a place with enough space for the entire file, it is unclear if PAUP could handle the required analysis of over 15K taxa and more than 471K characters.

However, on this very large dataset I was able to use pRecDCM3 to gener-

Method	Number of Src Trees	Avg Src Size	SCM Max Poly degree
BLF-50-250	2311	250.9 (0.1)	15533
PRD-50-250	302	133.4 (3.0)	427
PRD-75-500	139	261.9 (9.2)	465
PRD-100-1000	80	435.7 (22.4)	392

Table 9.5: **Decomposition statistics for 16S.B.ALL (27643 taxa).** The number of source trees resulting from the decomposition, the average number of taxa per subset, and the maximum polytomy degree in the SCM tree are given. The low parameter and maximum subset size are given for each decomposition.

ate trees. To create the guide tree, I used the fast MAFFT parttree alignment, and used RAxML to quickly estimate a maximum parsimony tree (the starting tree for its search). I used the parttree alignment technique since it was the one alignment technique that seemed likely to complete (other MAFFT variations may be possible, as might the `quicktree` option of ClustalW, but I was trying to quickly find a starting point). I used the quick estimate of the maximum parsimony tree as opposed to the result of the RAxML search since the search failed after approximately 5 days of processing on a machine with 32G of RAM.

I started by considering three variants of the pRecDCM3 decomposition:

- PRD-50-250 sets low to 50, and maximum subset size to 250
- PRD-75-500 sets low to 75, and maximum subset size to 500
- PRD-100-1000 sets low to 100, and maximum subset size to 1000

Table 9.5 shows decomposition statistics for each of these decompositions.

Figure 9.4 shows the comparison of the resulting trees as well as the initial MP tree that was used as a guide tree, each as compared to the reference tree. Note that the missing edge rate decreases by one percent each time the minimum overlap (low) and maximum subsize size are increased. Given that this tree has over 27K taxa, this represents an improvement of more than 270 edges for each change. Thus, though these are small numbers in terms of percentage, they are not as small in terms of absolute numbers, suggesting that increasing the subset size can lead to improvements in accuracy.

Note also that even the best tree here is missing about 70% of the reference tree edges. However, as previously discussed, we do not expect many of the reference tree edges to have high bootstrap support. In comparing the empirical statistics of the Gutell datasets, in fact, we would expect this dataset to have proportionally fewer edges with high bootstrap support, with perhaps only 30-40% of the edges having bootstrap support at the 75% level or higher. Thus, missing edge rates of somewhere between 60-70% may be the best that can be obtained with respect to the current reference tree, even with the best analyses (RAxML on the true alignment, if computational requirements were not an issue). This means that the analyses using pRecDCM3 may in fact be quite good.

Table 9.6 gives the running time information for each of the pRecDCM3 decompositions for this very large dataset. Note that by far the most time is spent in estimating trees for subsets (since a source alignment and source tree must be esimtated for each subset), but that this step is also highly parallelizable.

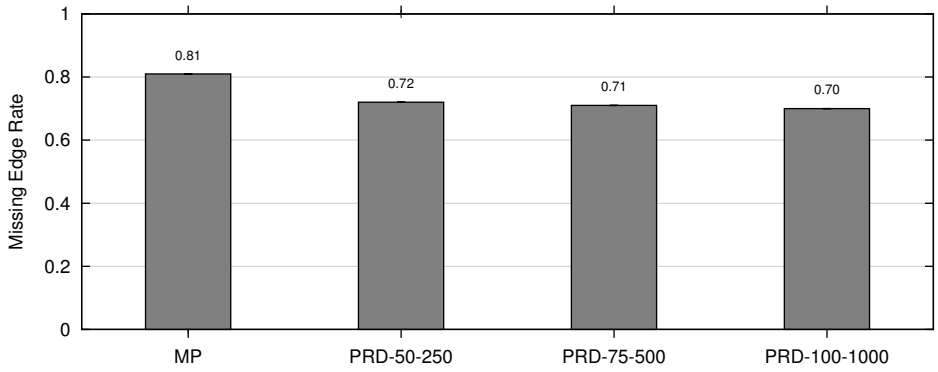


Figure 9.4: **pRecDCM3 applied to 16S.B.ALL Gutell dataset (27643 taxa).** Each of several estimated trees are compared to a single tree estimated by RAxML on the curated alignment. MP indicates the maximum parsimony guide tree used for the pRecDCM3 decomposition, PRD indicates the pRecDCM3 result. Targeted overlap and maximum subset size parameters are given for each decomposition. In each case subset alignments were estimated using MAFFT (with the linsi option) and polytomies were resolved using the default MRP search.

Method	Decomp	Avg Src Align	Avg Src Tree	Merge	Serial Total
PRD-50-250	4.5 h	478.0 s (23.0)	173.9 s (9.8)	8.9 h	68 h
PRD-75-500	2.7 h	1835.5 s (118.1)	653.8 s (48.2)	9.5 h	108.3 h
PRD-100-1000	3.0 h	5390.9 s (506.1)	1851.1 s (193.3)	4.4 h	168.3 h

Table 9.6: **Running times for pRecDCM3 decompositions of 16S.B.ALL (27643 taxa).** Both the low and maximum subset size parameters are given. All times are approximate, with "h" indicating hours and "s" indicating seconds. Decomposition and merge steps run on machines with 32G RAM. Subset analyses performed using the UTCS condor pool. When applicable, standard errors are given in parentheses. Serial total was computed using the average times for each subset analysis and does not include the time required to generate the tree used as input to the PRD decomposition.

```
1. Generate BLF decomposition tree by:
   1a. Create BLF decomposition
   1b. Estimate a tree for each subset
   1c. Assemble tree on full dataset using SuperFine
2. Generate PRD decomposition tree by:
   2a. Create PRD decomposition
   2b. Estimate a tree fore ach subset
   2c. Assemble tree on full dataset using SuperFine
3. Iterate step 2 if desired.
```

Figure 9.5: **BLuTGEN**

9.5 Conclusions

Generating a tree without ever generating an alignment for the complete dataset is a lofty goal. But by combining approaches I have developed a new method, BLuTGEN, which does just that, with good accuracy on hard datasets. The pseudocode for BLuTGEN is given in Figure 9.5. The algorithm begins by using the BLF decomposition to estimate a tree without generating a full alignment, and then improves this result by using the pRecDCM3 decomposition.

Though the BLF decomposition often creates large polytomies, changes to the heuristics SuperFine uses to resolve polytomies allow this potential bottleneck to be overcome – up to a point. Using the fast MRP search heuristic effectively trades time and accuracy, allowing the BLF decomposition to generate a good starting point for pRecDCM3 for large datasets. For extremely large datasets though, the BLF decomposition results in polytomies which are too large to be handled in

the usual way. Thus, new clustering techniques, which do not produce such large polytomies will be required.

Finally, when an initial estimate of the tree can be generated in an alternative way, using pRecDCM3 effectively improves the accuracy of the initial estimate, and as I showed with the 16S.B.ALL dataset with 27643 taxa, this technique works well even for very large datasets.

Chapter 10

Conclusions and Future Work

This dissertation has contributed to the solutions of several of the problems inherent to reconstructing phylogenetic trees. The work on TASPI gave a structure to reduce the storage size requirements of large collections of trees. It also showed that working with improved data structures can greatly reduce the computation time required for strict and majority consensus analysis. Further, by developing the code base in ACL2, and proving properties of the code, I have shown that confidence in code can be attained. Given the ways in which phylogenetics is used, for example in health care and law, ideally all phylogenetic code would be verified in some way.

There are still many ways that TASPI could be improved and extended, particularly as post-tree analyses continue to be developed. For one specific example, the data structures developed could lend themselves well to tree-profiling, especially if an ordering element were also maintained. Also, the current representation is limited when branch lengths are considered. Though TASPI can correctly read and store branch lengths, their effect on the storage requirements has not been fully explored. It seems likely that branch lengths for a subtree, particularly those with 20 decimal places such as those output by RAxML, will not be identical even if the topology is identical. It would be enlightening to consider the effect of a bin-

ning algorithm for these branch lengths, since two or three decimal places are likely sufficient to make comparisons between trees.

The work on guide trees showed that improving the guide tree for MSA methods can improve estimated phylogenies, provided that appropriate multiple alignment methods are used. Trees estimated from ProbCons and Prank alignments were significantly more accurate when the alignments were generated using a pre-estimated guide tree. One of the interesting questions that arose from this study has to do with alignment-error metrics. Given that SP-error fails to predict the ability of an alignment to generate an accurate phylogeny, might there be a better measure of alignment error?

And finally, BLuTGEN provides a means by which to estimate a tree from sequence data, without generating an alignment on the full dataset. It does this by combining two approaches to decomposing the full dataset into overlapping subproblems, and using a recent supertree method to combine the subproblems into a full solution.

There are many ways that BLuTGEN might be improved and optimized. The most readily apparent would be in the use of parallelization. In my experiments, I made use of a condor pool of machines to generate source trees in parallel, but this parallelization could be built directly into an executable. Also, there are several parameters to the decompositions of BLuTGEN that could be explored to determine if there are optimal settings that balance computation time with source tree accuracy. In particular, finding a BLAST-based decomposition that maintained speed and necessary overlap without causing very large polytomies could greatly

reduce the running time of BLuTGEN, and allow even larger datasets to be analyzed without generating an alignment on the full dataset.

There are also several ways SuperFine could be changed (for example varying the merge and polytomy resolution options), perhaps resulting in even more accurate trees, or more speed. One more future direction considers again the use of iteration. If the second phase of BLuTGEN is to be run several times, developing an appropriate stopping criterion will be necessary (for example, one possibility might be to stop when the consensus tree stopped changing, an instance where using TASPI to generate the consensus could prove useful).

Though there are still many challenges in reconstructing the Tree of Life, my dissertation includes a few steps towards the goal of understanding the relationships that surround us. In particular, the development of BLuTGEN will allow very large datasets to be accurately analyzed in reasonable time frames, bringing us closer to a fully assembled tree of life.

Appendix

Appendix 1

Commands for Software

1.1 Commands from Chapter 3

- PAUP v 4.0b10 for Unix

```
contree all/strict=no majrule=yes percent=<percent> \
treefile=<outfile> replace;
```

- TNT version 1.0

  Strict consensus:

```
tsave * <outfile>;
nelsen *;
save /;
tsave /;
```

  Majority consensus:

```
tsave * <outfile>;
majority = <percent> *;
save /;
tsave /;
```

1.2 Commands from Chapter 4

- Clustal v 2.0.9

  Default alignment:

```
clustalw -align -infile=<raw sequence file> \
   -outfile=<output file> -output=fasta \
   -newtree=<file for default guide>
```

  Passing guide:

```
clustalw -infile=<raw sequence file> \
-outfile=<output file> -output=fasta \
-usetree=<guide tree file name>
```

- Muscle v3.7

  Default alignment:

```
muscle -in <raw sequence file> \
-fastaout <output file> \
-tree1 <file for default guide>
```

  Passing guide:

```
muscle -in <raw sequence file> \
-fastaout <output file> \
-usetree_nowarn <guide tree file name>
```

- ProbCons v1.12 (modified to take a user input guide tree)

  Default alignment:

  ```
  probcons <raw sequence file> <output file>
  ```

  Passing guide:

  ```
  probcons <raw sequence file> \
  -g <guide tree file name>
  ```

- MAFFT v 6.504alpha (modified by Gotoh to accept guide trees)

  - MAFFT-linsi

    Default alignment:

    ```
    mafft --localpair \
    --maxiterate 1000 <raw sequence file>
    ```

    Passing guide:

    ```
    mafft --topin <guide tree file name> \
    --localpair --maxiterate 1000 <raw sequence file>
    ```

  - MAFFT-fftnsi

    Default alignment:

    ```
    mafft --retree \
    --maxiterate 2 <raw sequence file>
    ```

    Passing guide:

```
mafft --topin <guide tree file name> \
--retree 2 --maxiterate 2 <raw sequence file>
```

- MAFFT-fftns2

  Default alignment:

  ```
  mafft --retree 2 --maxiterate 0 <raw sequence file>
  ```

  Passing guide:

  ```
  mafft --topin <guide tree file name> \
  --retree 2 --maxiterate 0 <raw sequence file>
  ```

- Prank v.080707

  Default alignment:

  ```
  prank -d=<raw sequence file> -o=<output file>
  ```

  Passing guide:

  ```
  prank -t=<guide tree file name> \
  -d=<raw sequence file> -o=<output file>
  ```

- RAxML

  ```
  raxmlHPC -m GTRGAMMA -w ./ -n <output file> \
  -s <aligned sequence file in phylip format>
  ```

- PAUP v4.0b10

  UPGMA:

```
upgma brlens=yes showtree=no \
treefile=<upgma tree file name>
```

Midpoint:

```
roottrees rootmethod=midpoint userbrlens=yes;
```

1.3 Commands from Chapters 5 – 9

- MAFFT v 6.240

  Parttree:

  ```
  mafft --retree 0 --treeout --parttree <infile>
  ```

  Subproblems:

  ```
  mafft-linsi --quiet <infile> > <outfile>
  ```

- Clustal v 2.0.9

  ```
  clustal -align -infile=<infile> -oufile=<outfile> \
  -output=fasta -quicktree -newtree=<guidetree>
  ```

- SuperFine

  Resolving with MRP:

  ```
  runReup.py -w <uniqueName> -n <# ratchet iterations> \
  -r <rmrp|gmrp> <sourceTrees>
  ```

Resolving with QMC (the default):

```
runReup.py -w <uniqueName> <sourceTrees>
```

- RAxML v 7.0.4

  Subproblems:

  ```
  raxmlHPC -m GTRCAT -w <workDir> -n <output file> \
  -s <aligned sequence file in phylip format>
  ```

  Parallelized:

  ```
  raxmlHPC-PTHREADS -s <PHYLIP input alignment> \
  -n <name for the run> -m GTRCAT -T 8
  ```

  Boostrapping:

  ```
  raxmlHPC-PTHREADS -s <PHYLIP input alignment> \
  -n <name for run> -f a -m GTRGAMMA \
  -x <random number> -p <random number> -N 500 -T 16
  ```

Bibliography

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[2] N. Amenta, K. St. John, and F. Clarke. A linear-time majority tree algorithm. In G. Benson and R. D. M. Page, editors, *Proceedings of the 3rd International Workshop on Algorithms in Bioinformatics (WABI 2003)*, volume 2812 of *Lecture Notes in Computer Science*, pages 216–227. Springer-Verlag, 2003.

[3] ATOL — Assembling the Tree of Life. Website, 2009. `http://atol.sdsc.edu/`.

[4] B. R. Baum. Combining trees as a way of combining data sets for phylogenetic inference, and the desirability of combining gene trees. *Taxon*, 41:3–10, 1992.

[5] B. R. Baum and M. A. Ragan. The MRP method. In O. R. P. Bininda-Emonds, editor, *Phylogenetic Supertrees: Combining Information To Reveal The Tree Of Life*, page 1734. Kluwer Academic, Dordrecht, the Netherlands, 2004.

[6] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, and D. L. Wheeler. GenBank. *Nucleic Acids Research*, 33 (Database Issue):D34–D38, 2005.

[7] Z. D. Blount, C. Z. Borland, and R. E. Lenski. Historical contingency and the evolution of a key innovation in an experimental population of escherichia coli. *Proceedings of the National Academy of Sciences*, 105:7899–7906, 2008.

[8] C. Bonnard, V. Berry, and N. Lartillot. Multipolar consensus for phylogenetic trees. *Systematic Biology*, 55(5):837–843, 2006.

[9] R. S. Boyer, W. A. Hunt Jr., and S. M. Nelesen. A compressed format for collections of phylogenetic trees and improved consensus performance. In R. Casadio and G. Meyers, editors, *Proceedings of the 5th Workshop on Algorithmics in Bioinformatics (WABI 2005)*, Lecture Notes in Bioinformatics 3692. Springer, 2005.

[10] K. Bremer. Combinable component consensus. *Cladistics*, 6:369–372, 1990.

[11] D. Bryant. A classification of consensus methods for phylogenetics. In M. Janowitz, F.J. Lapointe, F. McMorris, B. Mirkin, and F. Roberts, editors, *BioConsensus*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pages 163–184. DIMACS.AMS., 2003.

[12] J. J. Cannone, S. Subramanian, M. N. Schnare, J. R. Collett, L. M. D'Souza, Y. Du, B. Feng, N. Lin, L. V. Madabusi, K. M. Muller, N. Pande, Z. Shang,

N. Yu, and R. R. Gutell. The Comparative RNA Web (CRW) Site: An Online Database of Comparative Sequence and Structure Information for Ribosomal, Intron and Other RNAs. *BMC Bioinformatics*, 3(15), 2002. http://www.rna.ccbb.utexas.edu.

[13] B. L. Cantarel, H. G. Morrison, and W. Pearson. Exploring the relationship between sequence similarity and accurate phylogenetic trees. *Molecular Biology and Evolution*, 23(11):2090–2100, 2006.

[14] B. Chor and T. Tuller. Finding the maximum likelihood tree is hard. In *Proceedings of the 9th Annual International Symposium on Research in Computational Biology (RECOMB 2005)*, 2005.

[15] CIPRES: Cyberinfrastructure for Phylogenetic Research. Website. http://www.phylo.org/.

[16] CIPRES simulation data. Website, 2009. http://kim.bio.upenn.edu/wiki/html/Public/Software.htm.

[17] CIPRES: Cyberinfrastructure for Phylogenetic Research - Outreach: Utilizing Phylogenetic Information. Website. http://www.phylo.org/sub_sections/outreach/outreach_b.html.

[18] W. Day, D. Johnson, and D. Sankoff. The computational complexity of inferring rooted phylogenies by parsimony. *Mathematical Biosciences*, 81:33–42, 1986.

[19] W. H. E. Day. Optimal algorithms for comparing trees with labeled leaves. *Journal of Classification*, 2(1):7–28, 1985.

[20] C. B. Do, M. S. P. Mahabhashyam, M. Brudno, and S. Batzoglou. PROBCONS: Probabilistic consistency-based multiple sequence alignment. *Genome Research*, 15:330–340, 2005.

[21] R. C. Edgar. MUSCLE: A multiple sequence alignment method with reduced time and space complexity. *BMC Bioinformatics*, 5(113), 2004.

[22] R. C. Edgar. MUSCLE: A multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 32(5):1792–1797, 2004.

[23] R. C. Edgar and S. Batzoglou. Multiple sequence alignment. *Current Opinion in Structural Biology*, 16:368–373, 2006.

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

[25] J. Felsenstein. The newick tree format. Website, 1986. `http://evolution.genetics.washinton.edu.phylip/newicktree.html`.

[26] J. Felsenstein. *Inferring Phylogenies*. Sinauer Associates, Inc., 2004.

[27] D. Feng and R. Doolittle. Progressive sequence alignment as a prerequisitetto correct phylogenetic trees. *Journal of Molecular Evolution*, 25(4):351–360, 1987.

[28] R. Fleissner, D. Metzler, and A. von Haeseler. Simultaneous statistical multiple alignment and phylogeny reconstruction. *Systematic Biology*, 54:548–61, 2005.

[29] S. R. Gadagkar and S. Kumar. Maximum likelihood outperforms maximum parsimony even when evolutionary rates are heterotachous. *Molecular Biology and Evolution*, 22(11):2139–2141, 2005.

[30] G. Ganapathysaravanabavan and T. Warnow. Finding a maximum compatible tree for a bounded number of trees with bounded degree is solvable in polynomial time. In O. Gascuel and B. M. E. Moret, editors, *Algorithms in Bioinformatics: First International Workshop, WABI 2001, Aarhus, Denmark*, volume 2149 of *Lecture Notes in Computer Science*, pages 156–163. Springer Berlin / Heidelberg, August 2001.

[31] P. A. Goloboff, J. S. Farris, and K. C. Nixon. TNT (Tree analysis using new technology) (BETA) ver. 1.0. Published by the authors, Tucumán, Argentina, 2000.

[32] E. Goto, T. Soma, N. Inade, T. Ida, M. Idesawa, K. Hiraki, M. Suzuki, K. Shimizu, and B. Philpov. Design of a Lisp machine - FLATS. In *LFP '82: Proceedings of the 1982 ACM Symposium on LISP and functional programming*, pages 208–215, 1982.

[33] S. Guindon and O. Gascuel. A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. *Systematic Biology*, 52(5):696–704, 2003.

[34] B. G. Hall. Comparison of the accuracies of several phylogenetic methods using protein and DNA sequences. *Molecular Evololution and Biology*, 22(3):792–802, 2005.

[35] A. M. Hamel and M. A. Steel. Finding a maximum compatible tree is NP-hard for sequences and trees. *Applied Mathematics Letters*, 9(2):55–59, 1996.

[36] R. W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2):147–160, 1950.

[37] D. M. Hillis, J. P. Huelsenbeck, and C. W. Cunningham. Application and accuracy of molecular phylogenies. *Science*, 264:671–677, 1994.

[38] D. M. Hillis, C. Moritz, and B. K. Mable, editors. *Molecular Systematics*. Sinauer Associates, Inc., Sunderland, Massachusetts, 2nd edition, 1996.

[39] D. S. Hirschberg. A linear-space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18:341–343, 1975.

[40] J. P Huelsenbeck and F. Ronquist. MrBayes: Bayesian inference of phylogeny. *Bioinformatics*, 17:754–755, 2001.

[41] J.P. Huelsenbeck. The performance of phylogenetic methods in simulation. *Systematic Biology*, 44:17–48, 1995.

[42] J.P. Huelsenbeck and D.M. Hillis. Success of phylogenetic methods in the four-taxon case. *Systematic Biology*, 42(3):247–265, 1993.

[43] D. Huson, S. Nettles, and T. Warnow. Disk-covering, a fast converging method for phylogenetic tree reconstruction. *Journal of Computational Biology*, 6(3):369–386, 1999.

[44] D. Huson, L. Vawter, and T. Warnow. Solving large scale phylogenetic problems using DCM2. In *Proceedings of the 7th International Conference on Intelligent Systems for Molecular Biology (ISMB'99)*, pages 118–129. AAAI Press, 1999.

[45] W. A. Hunt Jr. and S. M. Nelesen. Phylogenetic trees in ACL2. In P. Manolios and M. Wilding, editors, *Proceedings of the Sixth International Workshop on the ACL2 Theorem Prover and its Applications*, 2006.

[46] W. Just. Computational complexity of multiple sequence alignment with SP-score. *Journal of Computational Biology*, 8(6):615–623, 2001.

[47] K. Katoh, K. Kuma, H. Toh, and T. Miyata. MAFFT version 5: Improvement in accuracy of multiple sequence alignment. *Nucleic Acids Research*, 33(2):511–518, 2005.

[48] K. Katoh, K. Misawa, K. Kuma, and T. Miyata. MAFFT: A novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nucleic Acids Research*, 30(14):3059–3066, 2002.

[49] K. Katoh and H. Toh. Parttree: An algorithm to build an approximate tree from a large number of unaligned sequences. *Bioinformatics*, 23(3):372–374, 2007.

[50] M. Kaufmann, P. Manolios, and J S. Moore. *Computer-Aided Reasoning: An Approach*. Kluwer Academic Publishers, 2000.

[51] M. K. Kuhner and J. Felsenstein. A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates. *Molecular Biology and Evolution*, 11(3):459–68, 1994.

[52] M. Litzkow. Remote UNIX - Turning idle workstations into cycle servers. In *Proceedings of Summer 1987 Usenix Conference*, pages 381–384, 1987.

[53] K. Liu, S. Nelesen, S. Raghavan, C. R. Linder, and T. Warnow. Barking up the wrong treelength: The impact of gap penalty on alignment and tree accuracy. *IEEE Transactions on Computational Biology and Bioinformatics*, 6(1):7–21, Jan.-Mar. 2009.

[54] 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, June 2009.

[55] A. Löytynoja. Goldman Group Software - PRANK: Probabilistic Alignment Kit. Website, 2009. `http://www.ebi.ac.uk/goldman-srv/prank/prank/`.

[56] A. Löytynoja and N. Goldman. An algorithm for progressive multiple alignment of sequences with insertions. *Proceedings of the National Academy of Sciences*, 102:10557–10562, 2005.

[57] A. Löytynoja and N. Goldman. Phylogeny-aware gap placements prevents errors in sequence alignment and evolutionary analysis. *Science*, 320(5883):1632–1635, 2008.

[58] D. R. Maddison. The discovery and importance of multiple islands of most-parsimonious trees. *Systematic Zoology*, 40(3):315–328, 1991.

[59] T. Margush and F. R. McMorris. Consensus n-Trees. *Bulletin of Mathematical Biology*, 43(2):239–244, 1981.

[60] B. M. E. Moret, U. Roshan, and T. Warnow. Sequence length requirements for phylogenetic methods. In *Proceedings of the 2nd International Workshop Algorithms in Bioinformatics (WABI'02)*, volume 2452 of *Lecture Notes in Computer Science*, pages 343–356. Springer-Verlag, 2002.

[61] D. A. Morrison and J. T. Ellis. Effects of nucleotide sequence alignment on phylogeny estimation: A case study of 18S rDNAs of apicomplexa. *Molecular Biology and Evolution*, 14:428–441, 1997.

[62] L. Nakhleh, B. M. E. Moret, U. Roshan, K. St. John, J. Sun, and T. Warnow. The accuracy of fast phylogenetic methods for large datasets. In *Proceedings of the 7th Pacific Symposium on BioComputing (PSB02)*, pages 211–222. World Scientific Pub, 2002.

[63] BLAST: Basic Local Alignment Search Tool. Website, 2009. `http://blast.ncbi.nlm.nih.gov/Blast.cgi`.

[64] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

[65] S. Nelesen, K. Liu, D. Zhao, C. R. Linder, and T. Warnow. The effect of the guide tree on multiple sequence alignments and subsequent phylogenetic analyses. In *Pacific Symposium on Biocomputing*, volume 13, pages 15–24, 2008.

[66] The Nematode Branch of the Assembling the Tree of Life Project: NemA-TOL. Website, 2008. `http://nematol.unh.edu/tree/tree1/vIch20ct682_c1w1.aln`.

[67] C. Notredame, D. G. Higgins, and J. Heringa. T-Coffee: A novel method for fast and accurate multiple sequence alignment. *Journal of Molecular Biology*, 302:205–217, 2000.

[68] T. H. Ogden and M. S. Rosenberg. Multiple sequence aligment accuracy and phylogenetic inference. *Systematic Biology*, 55(2):314–328, 2006.

[69] M. Ott, J. Zola, S. Aluru, and A. Stamatakis. Large-scale maximum likelihood-based phylogenetic analysis on the IBM BlueGene/L. In *Proceedings of ACM/IEEE Supercomputing Conference 2007*, 2007.

[70] M. A. Ragan. Phylogenetic inference based on matrix representation of trees. *Molecular Phylogenetics and Evolution*, 1:5358, 1992.

[71] B. D. Redelings and M. A. Suchard. Joint Bayesian estimation of alignment and phylogeny. *Systematic Biology*, 54(3):401–418, 2005.

[72] D. F. Robinson and L. R. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53:131–147, 1981.

[73] F. Rodriguez, J. L. Oliver, A. Marin, and J. R. Medina. The general stochastic model of nucleotide substitution. *Journal of Theoretical Biology*, 142:485–501, 1990.

[74] U. Roshan, D. R. Livesay, and S. Chikkagoudar. Improving progressive alignment for phylogeny reconstruction using parsimonious guide-trees. In *Proceedings of the IEEE 6th Symposium on Bioinformatics and Bioengineering*. IEEE Computer Society Press, 2006.

[75] U. Roshan, B. M. E. Moret, T. L. Williams, and T. Warnow. Rec-I-DCM3: A fast algorithmic technique for reconstructing large phylogenetic trees. In *Proceedings of the 3rd Computational Systems Biology Conference (CSB'05)*, pages 98–109. Proceedings of the IEEE, 2004.

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

[77] M. J. Sanderson. r8s: Inferring absolute rates of molecular evolution and divergence times in the absence of a molecular clock. *Bioinformatics*, 19(2):301–302, 2003.

[78] M. J. Sanderson, B. G. Baldwin, G. Bharathan, C. S. Campbell, D. Ferguson, J. M. Porter, C. Von Dohlen, M. F. Wojciechowski, and M. J. Donoghue. The growth of phylogenetic information and the need for a phylogenetic database. *Systematic Biology*, 42:562–568, 1993.

[79] M. J. Sanderson, M. J. Donoghue, W. Piel, and T. Eriksson. TreeBASE: A prototype database of phylogenetic analyses and an interactive tool for browsing the phylogeny of life. *American Journal of Botany*, 81(6):183, 1994.

[80] D. Sankoff. Matching sequences under deletion/insertion constraints. *PNAS*, 69:4–6, 1972.

[81] D. Sankoff. Minimal mutation trees of sequences. *SIAM Journal of Applied Mathematics*, 28(1):35–42, January 1975.

[82] SATé data and programs. Website. `http://www.cs.utexas.edu/users/kliu/public/sate_journal.html`.

[83] Large-scale simultaneous multiple alignment and phylogeny estimation. Website. `http://www.cs.utexas.edu/users/tandy/ATOL-MSA.html`.

[84] D. B. Searls. Pharmacophylogenomics: Genes, evolution and drug targets. *Nature Reviews Drug Discovery*, 2:613–623, 2003.

[85] J. Seward. bzip2. Website, 2002. `http://sources.redhat.com/bzip2/`.

[86] R. R. Sokal and C. D. Michener. A statistical method for evaluating systematic relationships. *University of Kansas Scientific Bulletin*, 28:1409–1438, 1958.

[87] A. Stamatakis. Phylogenetic models of rate heterogeneity: A high performance computing perspective. In *Proceedings of IPDPS2006*, 2006.

[88] A. Stamatakis. RAxML-VI-HPC: Maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics*, 22(21):2688–2690, 2006.

[89] A. Stamatakis, P. Hoover, and J. Rougemont. A rapid bootstrap algorithm for the raxml web servers. *Systematic Biology*, 57(5):758–771, 2008.

[90] M. Steel and T. Warnow. Kaikoura tree theorems: Computing the maximum agreement subtree. *Information processing letters*, 48(2):88–82, 1993.

[91] G. L. Steele. *Common Lisp the Language*, chapter 22.1.4. Digital Press, 2nd edition, 1990.

[92] J. Stoye, D. Evers, and F. Meyer. Rose: Generating sequence families. *Bioinformatics*, 14(2):157–163, 1998.

[93] M. Swenson. *Phylogenetic Supertree Methods*. PhD thesis, The University of Texas at Austin, May 2009.

[94] D. L. Swofford. *PAUP*: Phylogenetic Analysis Using Parsimony (and Other Methods) 4.0 Beta*. Sinauer Associates, Sunderland, Massachusetts, 2002.

[95] J. Tang and B. M. E. Moret. Scaling up accurate phylogenetic reconstruction from gene-order data. In *Bioinformatics*, volume 19 (Suppl. 1), pages i305–i312, 2003. Proceedings of the 11th International Conference on Intelligent Systems for Molecular Biology ISMB'03.

[96] J. D. Thompson, D. G. Higgins, and T. J. Gibson. CLUSTAL W: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22:4673–4680, 1994.

[97] TreeBASE: A Database of Phylogenetic Knowledge. Website. `http://www.treebase.org/treebase/index.html`.

[98] A. Varón, L.S. Vinh, I. Bomash, and W.C. Wheeler. POY software, 2007. Documentation by A. Varón, L. S. Vinh, I. Bomash, W. Wheeler, K. Pickett, I. Temkin, J. Faivovich, T. Grant, and W. L. Smith. Available for download at `http://research.amnh.org/scicomp/projects/poy.php`.

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

[100] K. M. Wong, M. P. Suchard, and J. P. Huelsenbeck. Alignment uncertainty and genomic analysis. *Science*, 319(5862):473–476, January 2008.

[101] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23:337–342, 1977.

[102] D. Zwickl. GARLI download page. Website, 2006.

http://www.zo.utexas.edu/faculty/antisense/Garli.html.

Vita

Serita Marie Nelesen was born in Missouri on October 17, 1979, to Bill and Connie Van Groningen. She grew up in Kingston, Ontario, Canada, before attending Calvin College in Grand Rapids, Michigan. In 2001 she received her Bachelor of Arts in Computer Science, and her Bachelor of Science in Mathematics from Calvin College. She wasted no time in starting her graduate studies at The University of Texas at Austin, beginning there in the fall of 2001, and completing her Masters in May 2006. Serita is married to Bob Nelesen, and has a daughter, Adelae.

Permanent address: 3356 Bloomington Hills Ave SE
Ada MI 49301

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.