5-1-2010

Efficient algorithms for phylogenetic post-analysis

Nicholas Pattengale

Nicholas Dylan Pattengale

Candidate

Computer Science

Department

This dissertation is approved, and it is acceptable in quality and form for publication:

Approved by the Dissertation Committee:

Jared Saia, Chairperson

[illegible]

[illegible]

[illegible]

Efficient Algorithms for Phylogenetic Post-Analysis

by

Nicholas Dylan Pattengale

B.S., New Mexico Institute of Mining and Technology, 2001
M.S., Computer Science, University of New Mexico, 2005

DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Computer Science

The University of New Mexico

Albuquerque, New Mexico

May, 2010

Dedication

To Mayah and April.

You inspire me to strive to fulfill my potential – this document being a case in point.

Acknowledgments

Documents like this one are made possible only by the sacrifice and support of a list of people too long to enumerate. Foremost – I would like to thank my advisor, Bernard Moret, who is perhaps most responsible for my becoming the researcher that I am today. I would also like to thank my committee for providing useful feedback along the way – most notably at my proposal defense. My academic collaborators (for the work presented in this dissertation) certainly deserve thanks – Eric Gottlieb, Krister Swenson, Alexandros Stamatakis, Andre Aberer, Olaf Bininda-Emonds, and Masoud Alipour. It should go without saying that family is integral in this whole ordeal, and mine has been incredibly supportive, especially in the final push. My employer (for the past 8 years), Sandia National Laboratories, also deserves mention, as they have financed my entire graduate education and have been wholly supportive of my academic goals. Finally, I am grateful to the entire scientific community – past and present – for gifting me with such an enjoyable discipline to call my own.

"EXTREMELY SERIOUS WARNING

Unless you are as smart as Johann Karl Friedrich Gauss, savvy as a half-blind Calcutta bootblack, tough as General William Tecumseh Sherman, rich as the Queen of England, emotionally resilient as a Red Sox fan, and as generally able to take care of yourself as the average nuclear missile submarine commander, you should never have been allowed near this document. Please dispose of it as you would any piece of high-level radioactive waste and then arrange with a qualified surgeon to amputate your arms at the elbows and gouge your eyes from their sockets. This warning is necessary because once, a hundred years ago, a little old lady in Kentucky put a hundred dollars into a dry goods company which went belly-up and only returned her ninety-nine dollars. Ever since then the government has been on our asses. If you ignore this warning, read on at your peril – you are dead certain to lose everything you've got and live out your final decades beating back waves of termites in a Mississippi Delta leper colony.

Still reading? Great. Now that we've scared off the lightweights, let's get down to business."

– Neal Stephenson, Cryptonomicon

Efficient Algorithms for Phylogenetic Post-Analysis

by

Nicholas Dylan Pattengale

ABSTRACT OF DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Computer Science

The University of New Mexico

Albuquerque, New Mexico

May, 2010

Efficient Algorithms for Phylogenetic Post-Analysis

by

Nicholas Dylan Pattengale

B.S., New Mexico Institute of Mining and Technology, 2001
M.S., Computer Science, University of New Mexico, 2005

Ph.D., Computer Science, University of New Mexico, 2010

Abstract

A variety of tasks are typically performed after a phylogenetic reconstruction proper – tasks which fall under the category *phylogenetic post-analysis*. In this dissertation, we present novel approaches and efficient algorithms for three post-analysis tasks: taking distances between (typically, all pairs in a set of) trees, bootstrapping, and building consensus trees.

For instance, it is often the case that reconstruction finds multiple plausible trees. One basic way of addressing this situation is to take distances between pairs of trees, in order to gain an understanding of the extent to which the trees disagree. The most frequently employed manner for computing the distance between a tree pair is the *Robinson-Foulds metric*, a natural dissimilarity measure between a pair of phylogenetic trees. We present a novel family of algorithms for efficiently computing the Robinson-Foulds metric.

Bootstrapping is a post-analysis technique for drawing support values on tree edges, and is often used for assessing the extent to which the underlying data (e.g., molecular sequences) supports a reconstructed tree. The basis of the approach is to reconstruct many trees, called replicates, based on random subsampling of the original data. However, to date, there has been little treatment in phylogeny regarding the question of how many bootstrap replicates to generate. We propose *bootstopping criteria* which are designed to provide on-the-fly (i.e., runtime) guidance for determining when enough bootstrap replicates have been reconstructed.

Another common post-analysis task is to build a *consensus tree*, a summary tree that attempts to capture the information agreed upon by bootstrap replicates. Unfortunately, the most popular consensus methods are susceptible to confusion by *rogue taxa*, i.e., taxa that cannot be placed with assurance anywhere within the tree. We present novel theory and efficient algorithms to identify rogue taxa, as well as a novel technique for interpreting the results (in the context of bootstrapping).

Contents

List of Figures **xiv**

List of Tables **xvi**

1 Introduction **1**

1.1 Background, Motivation, and Terminology 1

1.1.1 Efficiently Computing the Robinson-Foulds Metric 7

1.1.2 How Many Bootstrap Replicates are Necessary? 9

1.1.3 Uncovering Hidden Phylogenetic Consensus 11

2 Efficiently Computing the Robinson-Foulds Metric **14**

2.1 Background and Motivation 14

2.1.1 Day's Algorithm 16

2.1.2 Hashing – Functions and Tables 18

2.1.3 Other Applications of Hashing in Phylogeny 21

2.2 Theoretical Basis for the Algorithm 22

2.2.1 Bit-Vector Representation 23
2.2.2 Properties of $\|\cdot\|_p$-Norms of Bit-Vectors 24
2.2.3 Reducing Dimensionality 25
2.2.4 The Algorithm 25
2.3 A Framework and Implementation Tradeoffs 27
2.3.1 Direct Embedding 28
2.3.2 Improving Day's Algorithm 29
2.3.3 Probability of Collision 31
2.3.4 Speeding up Consensus Methods 32
2.4 Experiments 38
2.4.1 Validating the Approximation Bounds 39
2.4.2 Consequences of Approximation 40
2.4.3 Performance 41

3 How Many Bootstrap Replicates are Necessary? 45

3.1 Background and Motivation 45
3.2 Related Work on Bootstopping Criteria 48
3.2.1 The Phylogenetic Bootstrap 48
3.2.2 General Bootstopping Criteria 49
3.2.3 Bayesian Convergence Criteria and Tools 51
3.3 Bootstopping Criteria 52

3.3.1 Stopping Criteria 52

3.4 Implementation Considerations 57

3.4.1 Application of Bipartition Hashing 57

3.4.2 Running Time Improvement 61

3.5 Experimental Setup and Results 62

3.5.1 Experimental Setup 62

3.5.2 Results for FC and WC Methods 64

3.5.3 Robustness of Criteria 67

3.5.4 Convergence of Data Sets 69

3.5.5 Comparison to Hedges Criterion 70

4 Uncovering Hidden Phylogenetic Consensus 74

4.1 Background and Motivation 74

4.2 Relative Information Content, Consensus
Methods, and Rogue Taxa 77

4.2.1 The measure and the problem 77

4.2.2 How bipartitions change under leaf deletion 79

4.2.3 Finding subsets of leaves to drop 80

4.3 The Algorithm 81

4.4 Experimental Results 83

4.4.1 Difficult instances 83

4.4.2 The r-Cherry Trees 86

4.4.3 The Comb/Caterpillar 87

4.4.4 Results on biological data 89

4.4.5 Biological interpretation 89

4.4.6 Increasing resolution 91

4.5 Complexity Theoretic Aspects of MISC-$\mathcal{C}$ 92

4.5.1 MAST, MIST, and MR-MIST? 93

4.5.2 MFRC, MFRC-$\mathcal{C}_{\frac{m}{2}}$ and MMDS 97

5 Conclusions **104**

5.1 Robinson-Foulds Computations 104

5.2 Bootstopping 105

5.3 Uncovering Hidden Consensus 106

References **108**

List of Figures

2.1 A Summary of Day's Algorithm 19

2.2 A sketch of randomized embedding 26

2.3 Error bound between approximate and exact RF (50%) 39

2.4 Error bound between approximate and exact RF (10%) 39

2.5 RF clustering error 41

2.6 Hashing RF collision analysis 42

2.7 RF Algorithm Running Times 43

3.1 Hedges' Closed-Form Bootstop Recommendations 50

3.2 Bootstopping parameter p evaluation 54

3.3 Efficient bipartition hashing schematic 60

3.4 Bootstopping parameter ρ evaluation 65

3.5 Inherent convergence of bootstopping datasets 71

3.6 Criteria convergence on bootstopping datasets 72

3.7 Support value assessment of bootstopping 73

4.1 A difficult instance for consensus, easy for rogue identifier 84

4.2 The Cherry Tree with $k = 1$. 86

4.3 The Caterpillar Tree with 9 Taxa. 88

4.4 An instance requiring a 3-way merge. 88

4.5 Revealing hidden consensus with rogue identifier 90

4.6 MAST as lower bound (to maximize $|B'|$) counterexample 94

4.7 Swofford's MIST motivating instance 95

4.8 An instance of overlapping subsets that reduces to MMDS 101

4.9 A (multifurcating) transformed instance of MMDS 102

4.10 A (bifurcating) transformed instance of MMDS 102

List of Tables

2.1 Asymptotic Performance of Many RF Algorithms 31

2.2 RF Test Data Generation Parameters 40

3.1 Efficiency improvements for bootstopping in RAxML 62

3.2 Performance analysis of bootstopping with FC versus WC 64

3.3 More performance analysis of bootstopping with FC versus WC . . . 66

3.4 Robustness of WC bootstopping to replicate reordering 67

3.5 Robustness of FC bootstopping to replicate reordering 68

Chapter 1

Introduction

Let's set the existence-of-God issue aside for a later volume, and just stipulate that in some way, self-replicating organisms came into existence on this planet and immediately began trying to get rid of each other, either by spamming their environments with rough copies of themselves, or by more direct means which hardly need to be belabored. Most of them failed, and their genetic legacy was erased from the universe forever, but a few found some way to survive and to propagate.

– Neal Stephenson, *Cryptonomicon*

1.1 Background, Motivation, and Terminology

Novel sequencing techniques are continuing (as they have been since the 1990s) to drive a rapid accumulation of molecular data that in turn poses new challenges for the development of scalable bioinformatics tools. Such is certainly the case for phylogenetic tree reconstruction, a field concerned with inferring evolutionary relationships between organisms based on their molecular data. A *phylogenetic tree* is an unrooted binary tree where currently living organisms for which molecular data is available are located at the tips; the inner (ancestral) nodes of phylogenetic trees represent common ancestors.[1]

The central problem in phylogeny is: *given a set of extant taxa, what is the most plausible tree describing their evolutionary history?*. The term *taxa* refers to *taxonomical units* (singular is *taxon*), and is often synonymous with organism. This central problem is difficult to answer for many reasons. Perhaps the most profound reason is that the true tree is typically unknowable.[2] This is contrast to many other optimization problems in computer science, where it is verifiable whether the optimal solution equates to the desired/valid solution or not.

Another difficulty is that the space of possible trees is extremely large and seemingly unstructured. To see how large the space of tree topologies is, consider how it inductively grows as a function of the number of leaves. We proceed by only considering binary (unrooted) trees. The base case tree contains three leaves, one internal node, and no internal edges (edges other than those incident upon leaves). To place a fourth leaf (or any n^{th} leaf on a tree containing $n-1$ leaves) while retaining the property that the tree remains binary involves choosing one of the edges (internal or not), splitting it with a new internal node, and attaching the n^{th} leaf to the newly created internal node. So when adding a fourth leaf, there are three possible edges which can be split, and doing so will yield a tree with five edges. In general, a tree with n leaves can take any one of $(2n-5)\cdot(2n-7)\cdot(2n-9)\ldots(2n-(2n-3))\cdot 1 = (2n-5)!!$ unrooted binary topologies, a quantity which grows super-exponentially.

Also note that while a phylogenetic tree is indeed a tree, it is not the case that standard computer science tree algorithms always directly apply. This is because the trees here are *leaf-labeled* in the sense that the only nodes having known labels are

[1] A subtle point here – inner nodes in the *true tree* represent common ancestors, whereas internal nodes in reconstructed trees are not guaranteed (nor expected) to do so.

[2] A small few exceptions exist, for example when a bacteria is used in the lab setting to seed observable evolution, or when ancient DNA exists for a sufficiently large number of ancestral species of the taxa under consideration [53].

the leaves. This is in contrast to more standard tree settings where either all nodes have labels, or none have labels. In fact, the pertinent structure in phylogenetic trees are the bipartitions of the leaf set induced by internal edges. That is, most of our algorithms work directly with sets of bipartitions rather than trees represented in a more standard fashion (such as a parent array). As such, phylogenetics is a fruitful area for trailblazing new theory as it relates to these combinatorial structures that are closely related to, but not quite the same, as the trees computer scientists are used to treating algorithmically.

The actual tree reconstruction problem, which is typically phrased as a maximization problem over the space of all tree topologies, under widely used criteria such as Maximum Parsimony [31] (MP) and Maximum Likelihood [27] (ML) is $\mathcal{NP}$-hard [19, 32] and very computationally challenging in practice. Despite this, significant progress has been achieved in making MP and ML practical approaches for large-scale reconstruction. Programs such as TNT [37] for parsimony, and RAxML, GARLI [93], MrBayes [72] and PhyloBayes [54] for likelihood now allow for reconstruction of phylogenies that contain more than 70,000 organisms [73, 38] (organisms are also called taxa in this context) for MP and more than 10,000 taxa for ML. So called 'meta-methods,' such as Disk Covering Methods [88] can handle even larger datasets than this. One of the prohibitive factors when scaling to datasets of 10,000 or more taxa is in post-analysis. Post-analysis can refer to multiple things, three of which are considered in this dissertation: distances between (typically, all pairs in a set of) trees, bootstrapping, and consensus methods. Note that need for post-analysis does not arise simply because of a lack of agreement on the best reconstruction method. Rather, even under a single reconstruction method, there may be many tree topologies which all have optimal (or nearly optimal) score.

In other words, it is often the case that reconstruction finds multiple equally plausible trees (in the case of ML, see [36] for statistical tests that aid in making such assertions) or many (nearly) optimal trees in the case of MP. One basic way of addressing this situation is to take distances between pairs of trees, in order to gain an understanding of the extent to which the trees disagree. The most frequently employed manner for computing the distance between a tree pair is the *Robinson-Foulds metric*, a natural dissimilarity measure between a pair of phylogenetic trees. In Chapter 2 we present a novel family of algorithms for efficiently computing the Robinson-Foulds metric.

Another way to address the case of multiple equally plausible or nearly optimal trees (as well as many other similar situations) is by building a *consensus tree*, a summary tree that attempts to capture the information agreed upon by the reconstructed best trees. Unfortunately, the most popular consensus methods – the so-called *strict* and *majority rules* consensus methods – are susceptible to confusion by *rogue taxa*. That is, taxa that cannot be placed with assurance anywhere within the tree. In Chapter 4 we present novel theory and algorithms to identify rogue taxa, as well as a novel technique for interpreting the results (in the context of *bootstrapping*, which is also the subject of the next paragraph).

Finally, *bootstrapping* is a technique for assigning confidence values to edges in trees [28]. The most exercised application of bootstrapping in phylogenetics is in assessing the extent to which the underlying data (e.g., molecular sequences) support a reconstructed tree. The basis of the approach is to reconstruct many trees, called replicates, based on random subsampling of the original data. However, to date, there has been little treatment in phylogeny regarding the question of how many bootstrap replicates to generate. In Chapter 3 we propose *bootstopping criteria* [62] which are designed to provide on-the-fly (i.e., runtime) guidance for determining when enough bootstrap replicates have been reconstructed.

Most of the novel work contained in this dissertation has been presented in conference and journal venues over the past few years.

- The majority of the work in Chapter 2 was presented in a paper entitled 'A Sublinear-Time Randomized Approximation Scheme for the Robinson-Foulds Metric' [65] at *Research in Computational Molecular Biology (RECOMB) 2006*, which was held in Venice, Italy (actually on the island of Lido). This paper was invited to the journal issue for best papers from that conference, and appeared under the title 'Efficiently Computing the Robinson-Foulds Metric' in the August 12, 2007 issue of the Journal of Computational Biology [64], which also contained excellent new material contributed by Eric Gottlieb. As this work is a few years old now, we have been able to watch the citations steadily grow – perhaps of note is its recent citation in an introductory textbook [87].

- The material in Chapter 3 was presented in a paper entitled 'How Many Bootstrap Replicates are Necessary?' [62] at *RECOMB 2009*, which was held in Tucson, Arizona, USA. This paper was also invited to the journal issue for best papers from the conference, and is expected to appear under the same title (though with a much expanded section on relevant implementation details) in issue 3 of volume 17 (2010) of the Journal of Computational Biology [63].

- A large proportion of the work presented in Chapter 4 has been accepted for presentation and publication under the title 'Uncovering Hidden Phylogenetic Consensus' [66] at the *International Symposium on Bioinformatics Research and Applications (ISBRA) 2010* to be held in Storrs, Connecticut, USA in May 2010.

- The section from Chapter 2 regarding implementation speedup of consensus methods (specifically, Section 2.3.4) represents part of a paper accepted to the *International Conference on Computational Science (ICCS) 2010* under the title 'Parallel Computation of Phylogenetic Consensus Trees' [1] to be held in Amsterdam, Netherlands in May/June 2010, and has also been accepted in extended form to the Journal of Computational Science under the title 'Parallelized Phylogenetic Post–Analysis on Multi–Core Architectures' and is slated to appear in its inaugural issue.

For context, we include the abstracts from the three main papers mentioned above (RECOMB2006, RECOMB2009, ISBRA2010). Along with each abstract we include related terminology. We begin with terminology that spans more than one of the subject matters.

We use standard set and graph terminology and notation; in particular, $\cup$ refers to union, $\cap$ to intersection, $\setminus$ to set difference, and Δ to symmetric difference—i.e., $S \Delta T = (S \cup T) \setminus (S \cap T)$.

A *phylogenetic tree* represents the evolutionary relationships among a collection of living organisms. Homologous molecular sequences (one for each organism) are placed at the tips of the tree—hereafter called the *leaves* or *taxa*; the internal structure of the tree—its *edges* (sometimes also called branches)—represents the evolutionary relationships. The removal of an edge disconnects the tree and partitions the set of leaves into two subsets; thus each edge corresponds to a *bipartition* of the set of leaves. Every tree includes the same *trivial bipartitions*, which separate one leaf from all others; the other bipartitions are called *nontrivial* and correspond to an *internal* edge of a tree, that is, an edge not incident on a leaf. We can thus view a phylogenetic tree as a leaf-labeled tree $T = (L, B)$, where L is the set of leaves and B is its set of nontrivial bipartitions. To describe a bipartition, we list the two sets of leaves, separated by a | symbol. To ensure an equivalence between nontrivial bipartitions and internal edges, we require that every internal node in a phylogeny have degree at least 3. The number $|B|$ of nontrivial bipartitions in a phylogeny is at most $|L| - 3$; when the two are equal, we say that the (binary) tree is *fully resolved*; otherwise, there must exist an internal node of degree at least 4 and any such node is known as a *polytomy*. Sometimes a fully resolved tree is referred to as *bifurcating* and polytomies are referred to as *multifurcations*.

1.1.1 Efficiently Computing the Robinson-Foulds Metric

Terminology

The *Robinson-Foulds (RF) metric* [71] between two trees on the same set L of taxa is simply a normalized count of the bipartitions induced by one tree, but not by the other. More precisely, given two phylogenetic trees, $T_1 = (L, B_1)$ and $T_2 = (L, B_2)$, defined on the leaf set L, the RF metric between T_1 and T_2 is

$$\begin{aligned} RF(T_1, T_2) &= \frac{1}{2}|B_1 \Delta B_2| \\ &= \frac{1}{2}|(B_1 \cup B_2) \setminus (B_1 \cap B_2)| \\ &= \frac{1}{2}\left(|B_1 \setminus B_2| + |B_2 \setminus B_1|\right) \end{aligned}$$

where the divisor of 2 accomplishes normalization by the number of trees being compared. That is, the largest possible RF distance is $|L| - 3$, which as pointed out earlier is also the maximum number of nontrivial bipartitions that can appear together in a single tree defined on $|L|$. This measure of dissimilarity is easily seen to be a metric [71] and can be computed in linear time [24]. Sometimes the RF metric is generalized in a way so as to respect edge weights in the two input trees. The corresponding *Weighted Robinson-Foulds Metric (WRF)* is defined as

$$WRF(T_1, T_2) = \frac{1}{2} \sum_{b \in B_1 \cup B_2} |w_1(b) - w_2(b)|$$

where w_1 and w_2 are functions returning the weight of edge b in the corresponding tree, or zero otherwise. Note that the RF metric is equivalent to the WRF metric with the trees having unit edge weight.

Edit distances between trees are based on one or more operators that alter the structure of a tree. Two commonly used operators are the *Nearest Neighbor In-*

terchange (NNI) and the more powerful *Tree Bisection and Reconnection (TBR)*—see [3, 16] for definitions and discussions of these operators. Applying the NNI operator to T_1 can change $RF(T_1, T_2)$ by at most 1, while applying the TBR operator to T_1 can change $RF(T_1, T_2)$ almost arbitrarily. We use these operators in generating test sets for our RF approximation routine, as discussed in Section 2.4. There is another edit distance between trees that is often presented along with NNI and TBR, namely the Subtree Prune and Regraft (SPR) distance. The SPR operator falls between NNI and TBR in terms of power. We do not use the SPR in our study here, and mention it only for completeness. Again, see [3, 16] for more details. These three measures of dissimilarity are also metrics, but unlike the RF metric are $\mathcal{NP}$-hard to calculate.

Abstract from RF Paper [64]

The Robinson-Foulds (RF) metric is the measure most widely used in comparing phylogenetic trees; it can be computed in linear time using Day's algorithm. When faced with the need to compare large numbers of large trees, however, even linear time becomes prohibitive. We present a randomized approximation scheme that provides, in sublinear time and with high probability, a $(1+\varepsilon)$ approximation of the true RF metric. Our approach is to use a sublinear-space embedding of the trees, combined with an application of the Johnson-Lindenstrauss lemma to approximate vector norms very rapidly. We complement our algorithm by presenting an efficient embedding procedure, thereby resolving an open issue from the preliminary version of this paper. We have also improved the performance of Day's (exact) algorithm in practice by using techniques discovered while implementing our approximation scheme. Indeed, we give a unified framework for edge-based tree algorithms in which implementation tradeoffs are clear. Finally, we present detailed experimental results illustrating the precision and running-time tradeoffs as well as demonstrating the

speed of our approach. Our new implementation, `FastRF`, is available as an open-source tool for phylogenetic analysis, and can be downloaded from `http://cs.unm.edu/~ejgottl/cmb.tar.bz2`

1.1.2 How Many Bootstrap Replicates are Necessary?

Terminology

Phylogenetic bootstrapping (BS) is a straightforward application of the standard statistical (nonparametric) bootstrap and was originally suggested by Joe Felsenstein [28] as a way to assign confidence values to edges in phylogenetic trees. Phylogenetic BS proceeds by generating perturbed alignments which are assembled by randomly drawing alignment columns from the original input alignment with replacement. The number of columns in the BS alignment is identical to the number of columns in the original alignment, but the column composition is different. Then, for each BS alignment, a tree is reconstructed independently. The procedure returns a collection of tree *replicates*. The replicates can then be used either to compute consensus trees of various flavors or to draw confidence values onto a reference tree, e.g., the best-scoring ML tree. Each edge/branch in such a reference tree is then assigned a confidence value equal to the number of replicates in which it appears.

Abstract from Bootstopping Paper [63]

Phylogenetic Bootstrapping (BS) is a standard technique for inferring confidence values on phylogenetic trees that is based on reconstructing many trees from minor variations of the input data, trees called replicates. BS is used with all phylogenetic reconstruction approaches, but we focus here on one of the most popular, Maximum Likelihood (ML). Because ML inference is so computationally demanding, it has proved too expensive to date to assess the impact of the number of replicates used in BS on the relative accuracy of the support values. For the same reason, a rather small number (typically 100) of BS replicates are computed in real-world studies. Stamatakis *et al.* recently introduced a BS algorithm that is 1–2 orders of magnitude faster than previous techniques, while yielding qualitatively comparable support values, making an experimental study possible.

In this paper, we propose *stopping criteria*, that is, thresholds computed at run-time to determine when enough replicates have been generated, and report on the first large-scale experimental study to assess the effect of the number of replicates on the quality of support values, including the performance of our proposed criteria. We run our tests on 17 diverse real-world DNA, single-gene as well as multi-gene, datasets, that include between 125 and 2,554 sequences. We find that our stopping criteria typically stop computations after 100–500 replicates (although the most conservative criterion may continue for several thousand replicates) while producing support values that correlate at better than 99.5 with the reference values on the best ML trees. Significantly, we also find that the stopping criteria can recommend very different numbers of replicates for different datasets of comparable sizes.

Our results are thus two-fold: (i) they give the first experimental assessment of the effect of the number of BS replicates on the quality of support values returned through bootstrapping; and (ii) they validate our proposals for stopping criteria. Practitioners will no longer have to enter a guess nor worry about the quality of support values; moreover, with most counts of replicates in the 100–500 range, robust BS under ML inference becomes computationally practical for most datasets.

The complete test suite is available at `http://lcbb.epfl.ch/BS.tar.bz2` and BS with our stopping criteria is included in the latest release of RAxML v7.2.6 available at `http://wwwkramer.in.tum.de/exelixis/software.html`.

1.1.3 Uncovering Hidden Phylogenetic Consensus

Terminology

The input to the *consensus problem* is a set $\mathcal{T}$ of m trees defined on a common set L of n taxa (leaves, singular is *taxon*). *Consensus methods* are algorithms that address the consensus problem, and are characteristic in that they return a single tree – the *consensus tree* – in an attempt to summarize $\mathcal{T}$. We focus on consensus methods based on bipartition *frequency*—see the survey of Bryant [15] for a comprehensive treatment of consensus methods. Given a threshold parameter $\frac{m}{2} < t < m$, the t-consensus tree is composed of all of the bipartitions that occur in more than t trees. The *majority rules* consensus [58] is obtained by setting t to $\frac{m}{2}$, while the *strict* consensus is obtained by setting t to $m-1$. We denote t-consensus methods by $\mathcal{C}_t$. Thus $\mathcal{C}_{m-1}(\mathcal{T})$ corresponds to taking the strict consensus tree of the set $\mathcal{T}$, whereas $\mathcal{C}_{\frac{m}{2}}(\mathcal{T})$ corresponds to taking the majority rules consensus tree of the set $\mathcal{T}$.

Another very popular consensus method, for which we do not use the $\mathcal{C}_t$ notation, is called the *extended Majority Rules (MRE)*, also referred to as the *greedy consensus method*. This consensus method begins by including the bipartitions in $C_{\frac{m}{2}}$ as a skeleton and then includes bipartitions which occur in less than or equal half the trees, in priority order by frequency, as long as they can structurally coexist with all bipartitions included so far. In order for two bipartitions to structurally coexist together in a tree, they must be *compatible*, a property that can be defined in a number of equivalent manners. One such statement is the following: two bipartitions $A|B$ and $C|D$ are compatible if and only if at least one of the intersections $A \cap C$, $A \cap D$, $B \cap C$, or $B \cap D$ is empty.

The *bipartition profile* of a set of trees $\mathcal{T}$ is the pair

$$\mathcal{P} = (B_{\mathcal{T}},\, \nu\colon B_{\mathcal{T}} \to 2^{\mathcal{T}})$$

where $B_{\mathcal{T}}$ is the set of all nontrivial bipartitions found across all m trees in the set and ν is a function mapping bipartitions to the trees in which they appear.

We denote the removal of leaves from trees through the *restriction* operator—which also uses the | symbol. For example, $\mathcal{T}|L'$ refers to restricting each tree in the set $\mathcal{T}$ to the leaf subset $L' \subseteq L$, which corresponds to removing each leaf in $L \setminus L'$ from each tree, as well as removing any nodes of degree 2 created in the process. Individual trees, tree sets, and tree profiles can appear on the left-hand side of the restriction operator.

An *agreement subtree* (of a tree set $\mathcal{T}$) occurs when restricting the tree set to a leaf subset $L' \subseteq L$ renders all of the trees in $\mathcal{T}|L'$ equal to each other. Since the leaf subset (L') taken along with $\mathcal{T}$ unambiguously defines the agreement subtree, the agreement subtree is often denoted simply by L'. When the leaf set L' is of maximum cardinality (over all $2^{|L|}$ possible leaf subsets), then L' is called a *Maximum Agreement Subtree (MAST)* [5, 26]. When the number of nontrivial bipartitions in the agreement subtree L' has maximum cardinality, then L' is called a *Maximum Information Subtree (MIST)* [14].

The *cladistic information content* is a measure for assessing the amount of information conveyed by a consensus tree T. It is defined as $CIC(T) = -log_2 \frac{|R(T)|}{|\pi(T)|}$ where R denotes the number of possible ways to resolve the polytomies of T and π denotes the number of possible trees on $|L|$ leaves. This measure is very well founded in information theory [85], especially when the consensus tree was computed using the strict consensus method [86].

Abstract from Rogue Taxa Paper [66]

Many of the steps in phylogenetic reconstruction can be confounded by "rogue" taxa, taxa that cannot be placed with assurance anywhere within the tree—whose location within the tree, in fact, varies with almost any choice of algorithm or parameters. Phylogenetic consensus methods, in particular, are known to suffer from this problem. In this paper we provide a novel framework in which to define and identify rogue taxa. In this framework, we formulate a bicriterion optimization problem that models the net increase in useful information present in the consensus tree when certain taxa are removed from the input data. We also provide an effective greedy heuristic to identify a subset of rogue taxa and use it in a series of experiments, using both pathological examples described in the literature and a collection of large biological datasets. As the presence of rogue taxa in a set of bootstrap replicates can lead to deceivingly poor support values, we propose a procedure to recompute support values in light of the rogue taxa identified by our algorithm; applying this procedure to our biological datasets caused a large number of edges to change from "unsupported" to "supported" status, indicating that many existing phylogenies should be recomputed and reevaluated to reduce any inaccuracies introduced by rogue taxa.

Chapter 2

Efficiently Computing the Robinson-Foulds Metric

A note now about the physical properties of space, as perceived by human beings imprisoned within bodies of limited physical capabilities. I have long noticed that space seems to be more compressed, more involuted, some how psychically LARGER in some places than others. Covering a distance of three or four miles in the totally open scrublands of central Washington State is a simple matter, and takes less than an hour on foot. and only a few minutes if you have some kind of vehicle. Covering the same distance in Manhattan takes much longer. It's not just that the space in Manhattan is more physically obstructed (though it definitely is) but that there is some kind of psychological impact that alters the way you perceive and experience distance. You cannot see as far, and what you do see is full of people, buildings, goods, vehicles, and other stuff that it takes your brain some amount of effort to sort through, to process. Even if you had some kind of magic carpet that would glide past all of the physical obstructions the distance would seem much longer, and would take longer to cover, simply because your mind would have to deal with more stuff.

– Neal Stephenson, *Cryptonomicon*

2.1 Background and Motivation

The need to compare phylogenetic trees is common. Many reconstruction methods (e.g., maximum parsimony and Bayesian methods) produce a large number of possible trees. Trees are also built for the same collection of organisms from different types of data (e.g., nucleotide or codon sequences for one or more genes, gene-order data, protein folds, but also metabolic and morphological data). Phylogenetic trees can be compared and the result summarized in many ways; for instance, consensus methods [15] return a single tree that best represents the information present in the entire collection, while supertree methods (typically used when the trees are built on different, overlapping subsets of organisms) [12] combine the individual trees into a single larger one. A more elementary step is to produce estimates of how much the trees differ from each other, by computing pairwise similarity or distance measures. Here again, many approaches have been used, such as computing pairwise edit distances based on tree rearrangement operators [21, 3]; the most common distance measure between two trees, however, is the Robinson-Foulds (RF) metric [71]. This measure is in widespread use because it can be computed in linear time [24], is based directly on the edge structure of the trees and their induced bipartitions, and is a lower bound on the computationally more expensive edit distances. Yet, as the size of datasets used by researchers grows ever larger, even a linear-time computation of pairwise distances becomes onerous.

In this chapter, we present the first sublinear-time algorithm to compute all pairwise RF distances among a collection of trees. Our algorithm is a randomized approximation scheme: it returns, with high probability, an approximation that is guaranteed to be within $(1+\varepsilon)$ of the true distance, where $\varepsilon > 0$ can be chosen arbitrarily small. Our approach uses a sublinear-space embedding of the trees, combined with an application of the lemma of [52] to approximate vector norms rapidly.

In [65] (we will refer to that algorithm as the P&M algorithm), we had yet to design an efficient procedure for embedding trees. Thus, while our algorithm outperformed Day's spectacularly in certain settings, we were not able to match the latter's asymptotic running time for a single pairwise distance computation, nor would our technique gracefully scale as a function of the size of the input trees. We have since designed an efficient embedding procedure, presented here, that enables our algorithm to dominate Day's algorithm in all possible settings and that provides graceful scaling in terms of all parameters.

We have reimplemented the P&M algorithm as a standalone open-source tool `FastRF`. We have used `FastRF` to improve significantly the comprehensiveness (as compared to [65]) of our experiments to assess the quality and speed of our approach. Additionally, our new implementation should better facilitate the integration of our algorithm into actual phylogenetic analyses.

In Section 2.2 we introduce the theory that underlies our approximation algorithm. In Section 2.3 we cover practical issues such as our efficient procedure for embedding, discuss our improvements to the performance of Day's (exact) algorithm [24] in practice by using techniques discovered while implementing our approximation scheme, and present a common framework for edge-based algorithms on trees. In Section 2.4 we present experimental results that address issues such as the observed quality of approximation, the consequences of the quality approximation on simulated phylogenetic data, the tradeoffs between approximation quality and running time, and the running time of our technique versus Day's algorithm applied to each pair of trees.

2.1.1 Day's Algorithm

Before presenting our approach, we detail the linear time procedure that has served as the gold standard for computing the RF metric for the past quarter century.

The linear-time algorithm of Day [24] for computing the Robinson-Foulds metric can be difficult to follow from its original presentation. Here we attempt to present the algorithm in a more accessible fashion, and then comment briefly how this algorithm relates to our approach(es).

Before presenting the details of Day's algorithm, we point out some high level organizing concepts. Specifically, in order to claim linear time, it is logical to conclude Day's algorithm must employ a highly efficient scheme for representing the bipartitions of a tree. This is because the algorithm must surely examine each bipartition (of which there are a linear number) in each of the two input trees, and thus comparing bipartitions should take no longer than constant time. Indeed, this is the case with Day's algorithm, which ingeniously represents each bipartition with a simple pair of integers.

The algorithm works as follows, and is illustrated in Figure 2.1: Given two unrooted phylogenetic trees, pick an arbitrary leaf that will serve as a root. Traverse the first tree starting at the root, in depth first fashion, assigning an integer to each leaf equal to the number of leaves encountered thus far. For each internal node, before traversing its subtree, make a note of the next leaf label to be used l, and upon exiting a subtree, make a note of the label last used u. The pair (l, u) is then used to denote this subtree. Note that the number of leaves in subtree (l, u) is equal to $u - l + 1$. Next, build the interval representation for the second tree via a similar process – but by respecting the labeling established in the first tree. For each interval (*equiv.* subtree) in the second tree, there are three cases:

1. $u - l + 1$ is equal to the size of the subtree, and (l, u) is in the first tree
2. $u - l + 1$ is equal to the size of the subtree, and (l, u) is not in the first tree
3. $u - l + 1$ is not equal to the size of the subtree

Cases 2 and 3 contribute to the RF metric, as do the subtrees in the first tree that are not found in the second tree (analogous to case 2 with the trees swapped).

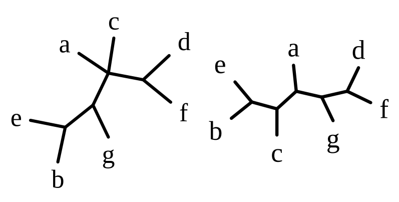
The final aspect of the algorithm admitting linear time is an efficient scheme for addressing/indexing intervals such that their occurrence can be confirmed or denied in constant time. Day provides a simple, yet clever, scheme for this as well. He recognized that for any interval/bipartition in the first tree (l, u) it holds that u is unique over all other bipartitions in the first tree unless the next node in postorder traversal is not a leaf, in which case l is unique. This uniqueness ensures that there will never be contention between two bipartitions to reside in the same row of a table having length $n - 1$. Further, given a valid bipartition (l, u) in the second tree, only two rows must be examined from the table (l and u) in order to test whether (l, u) is in the first tree.

We mention, as a foreshadow, that the function mapping bipartition intervals to their position in the bipartition table can be viewed as a *perfect hash function* on the bipartitions of the first tree. The definition of hashing, its relevance in phylogeny, and the importance of Day's scheme being a perfect hash function will be made clear in subsequent two sections as well as in Section 2.3.2.

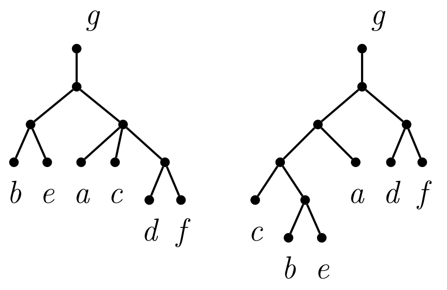
2.1.2 Hashing – Functions and Tables

The techniques surrounding *hashing* comprise the foundation of an enormous amount of efficiency gain in both algorithm performance and software performance. As such, hashing is typically found at the cornerstone of good algorithm design and software engineering. More specifically, there are innumerable situations in which a datum has been encountered once, and for some reason will need to be retrieved again. Hashing provides a well-founded framework for reducing the salient details of the datum into a space-efficient (and accordingly time-efficient to manipulate) *key* that can be used to subsequently refer to or retrieve the full datum.

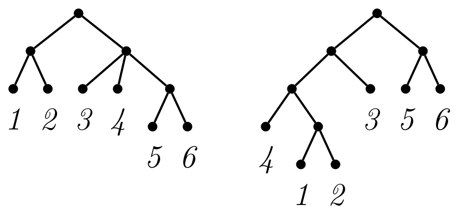
Typical manifestations of hashing are representative of two design decisions – that



(a) Original, unrooted, trees



(b) Root at a common leaf



(c) Label leaves according to depth first traversal of first tree

bipartition	interval	num leaves	table index	in tree 2
be|*acdfg*	[1,2]	2	2	yes
acdf|*beg*	[3,6]	4	3	no
df|*abceg*	[5,6]	2	5	yes

bipartition	interval	num leaves	valid interval?	in tree 1
be|*acdfg*	[1,2]	2	yes	yes
bce|*adfg*	[1,4]	3	no	no
abce|*dfg*	[1,4]	4	yes	no
df|*abceg*	[5,6]	2	yes	yes

(d) Evaluate results

Figure 2.1: A Summary of Day's Algorithm

of a suitable *hash function* and a complementary strategy for the accompanying *hash table*. The hash function is a mathematical (strictly speaking, injective) function that maps the datum in question to a key, which is typically a *word* in whatever computing architecture is being used (e.g. an integer on a 32-bit architecture). Since hash functions are injective (and not bijective, except for in a *perfect hash function*), it is possible for two or more entities to hash to the same key. Handling these so-called *collisions* is the responsibility of the hash table. A fairly typical approach to building a hash table is to use an array of linked lists, where a collision is resolved by simply adding an entry to the linked list at the index corresponding to the key. This technique is referred to as *chaining*, and is only one of very many ways to handle collisions.

The major benefit of a hashing approach is that insertions and lookups in the hash table take *expected constant time* per operation. This is a major practical improvement over nearly every other approach one can take, especially when there is no natural ordering of the data (and thus sorting is not a possibility). The interested reader can consult almost any algorithm text for an introduction to hashing.

Interestingly, the bipartitions in phylogenetic trees are amenable to a hashing approach. This observation led to the resolution of important open issues with our initial scheme for sublinear Robinson-Foulds metric computations, as well as helping to provide a unifying framework within which to view all known Robinson-Foulds metric algorithms. However, we were not the first to notice this application of hashing. In the following section we discuss other (published both before our RECOMB 2006 paper, and after) instances of bipartition hashing in phylogeny.

2.1.3 Other Applications of Hashing in Phylogeny

Perhaps the most naïve manner for representing a bipartition is to simply keep a bit vector, with one index per leaf in L, such that leaves on one side of the bipartition are represented by 0 and leaves on the other side are represented by 1. For example, if $L = \{a, b, c, d, e\}$ (and the position of a letter in the English alphabet is its index into the bit vector), the bipartition $cd|abe$ would be represented as 00110. Note that the complement (11001) also represents the same bipartition, however we adopt the convention of always denoting a single distinguished taxon, in this case a, by a 0 in order to facilitate fast comparison. This representational strategy for bipartitions has been the longstanding tradition until only very recently, and appears in most well established phylogenetic tools such as PHYLIP [29].

The first explicit application of hashing for representing bipartitions arose in the context of rapidly computing majority rules consensus trees for on–the–fly display in an interactive visualization tool [4]. In their technique, leaves are assigned hash keys according to a *universal hash function*, and internal edge (i.e., nontrivial bipartition) hashes are assigned recursively by composing the hashes of neighbor nodes using the addition operator. These are very similar to the techniques we used in resolving open issues in our RECOMB 2006 paper. However, we explored further choices regarding hash functions, and settled upon a much simpler scheme.

Subsequent to our RECOMB 2006 paper, Sul and Williams presented a program called HashCS [82] (which, as is obvious from the name, uses hashing for bipartition representation) for computing strict consensus and MR consensus trees (but not extended MR trees) and conducted a comparative performance study with consensus tree algorithms implemented in other phylogeny tools such as PAUP* [83] and MrBayes. They show that, the HashCS implementation is the fastest currently available implementation for computing strict and MR consensus trees. Note that the algorithm of HashCS is approximate. That is, it is not guaranteed to yield the correct result (see p. 105 in [82]) because not all collisions in the hash table are resolved. This is in contrast to TreeSetViz [4] and RAxML, which handle collisions via chaining, and thus represent exact approaches. That said, HashCS could no doubt be retrofitted with a collision resolution approach that would render their tool exact, although there would be some performance penalty.

Most recently, while implementing our bootstopping criteria (presented in full in Chapter 3), we deemed it worth the incremental extra effort to provide consensus method routines in RAxML. Putting into practice the knowledge obtained in the hash function evaluation of our extended RECOMB 2006 paper [64], we endowed RAxML with very efficient routines for computing the Strict, Majority Rules, and extended Majority Rules consensus trees. As of the time of writing, these stand as the fastest exact consensus method routines available. Some details regarding these routines are presented in Section 2.3.4, and the interested reader is referred to [66] where we present the first parallelization of the MRE consensus method.

We now proceed with the presentation of our RECOMB 2006 algorithm, as well as the enhancements presented in the journal version of the paper.

2.2 Theoretical Basis for the Algorithm

For the remainder of this chapter, m refers to the number of trees in a given set and n refers to the number of taxa in each tree of the set. The key to our approach is *representation*. Our approximation algorithm is a reduction to the computation of vector norms in a suitable vector space and the sublinear running time results from our ability to represent the necessary characteristics of phylogenetic trees in sublinear space. More specifically, we represent phylogenetic trees as vectors in such a way that RF distances become simply the $\|\cdot\|_1$-norms of the difference vectors, then generalize the result to arbitrary $\| \cdot \|_p$-norms for $p \geq 1$.[1] We then use a technique from high-dimensional geometry to reduce the dimensionality of tree vectors while maintaining pairwise $\| \cdot \|_2$-norms. Finally we combine these techniques to obtain a fast approximation algorithm for computing RF distances.

2.2.1 Bit-Vector Representation

Recall that in Section 1.1 we introduced B as the nontrivial bipartitions for a tree $T = (L, B)$. We continue with that notation here.

Consider an injection $f \colon \bigcup_{T=(L,B)\in\mathcal{T}_n} B \to \mathbb{N}$ that assigns a unique integer in the interval $[1, b]$ (where $b = \left|\bigcup_{T=(L,B)\in\mathcal{T}_n} B\right|$) to each bipartition.

Definition 1. *The* bit-vector representation *of a phylogenetic tree* $T = (L, B)$ *is* $v_T \in \mathbb{R}^b$ *where we have*

$$v_T[i] = \begin{cases} 1 & f^{-1}(i) \in B \\ 0 & \textit{otherwise} \end{cases}$$

Intuitively, this is a bit vector with a 1 only at the indices corresponding to bipartitions that exist in T. Obviously, this representation would be quite space-consuming (and proportionally time-consuming) to produce; fortunately, our linear-time embedding procedure (Section 2.3) completely obviates the need to compute this representation explicitly.

By construction, the $\|.\|_1$-norm (when normalized by 2) between tree vectors is the RF distance.

Theorem 1. $\forall T_1, T_2 \in \mathcal{T}_n,\ RF(T_1, T_2) = \frac{1}{2}\|v_{T_1} - v_{T_2}\|_1$

[1]The $\| \cdot \|_p$-norm of a vector $v = (v_1 v_2 \ldots v_k)$ is $\|v\|_p = \left(\sum_{i=1}^{k} |v_i|^p\right)^{\frac{1}{p}}$.

Proof. $\forall s \in B_1 - B_2$ (resp., $B_2 - B_1$), we have $v_{T_1}[f(s)] = 1$ (resp., $v_{T_2}[f(s)] = 1$) and $v_{T_2}[f(s)] = 0$ (resp., $v_{T_2}[f(s)] = 0$). Now, $\forall s \in B_1 \cap B_2$, we have $v_{T_1} = v_{T_2} = 1$ and $\forall s \in \bigcup_{T \in \mathcal{T}_n} B_T - (B_1 \cup B_2)$, we have $v_{T_1} = v_{T_2} = 0$. Thus we can conclude

$$\|v_{T_1} - v_{T_2}\|_1 = |B_1 - B_2| + |B_2 - B_1| = 2 \cdot RF(T_1, T_2)$$

$\square$

2.2.2 Properties of $\|\cdot\|_p$-Norms of Bit-Vectors

The following theorem exposes an interesting property about norms of bit-vectors and closely related vectors: it is trivial to recover the $\|.\|_p$-norm, $p \geq 1$, from the $\|.\|_q$-norm, $q \geq 1$, where $p \neq q$. This result will be useful because the Johnson-Lindenstrauss lemma (Section 2.2.3) approximates $\|.\|_2$-norms whereas, in order to compute the RF distance, we need to approximate $\|.\|_1$-norms.

Theorem 2. *For an arbitrary vector $v \in \mathbb{R}^d$ where every element is chosen from the set $\{-k, 0, k\}$ (for arbitrary $k > 0$), and for $p \geq 1$, we have $\|v\|_1 = k^{1-p} \cdot (\|v\|_p)^p$.*

Proof. Assume that v has c entries of value $\pm k$; we can write

$$\|v\|_p = \left(\sum_{i=1}^{b} (|v_i|)^p \right)^{\frac{1}{p}} = (ck^p)^{\frac{1}{p}} = c^{\frac{1}{p}} k$$

$$\|v\|_1 = \sum_{i=1}^{b} |v_i| = ck = c^{\frac{p-1}{p}} (c^{\frac{1}{p}} k) = c^{\frac{p-1}{p}} \|v\|_p$$

Raising the first result to the power $(p-1)$ and solving for $c^{\frac{p-1}{p}}$ yields

$$c^{\frac{p-1}{p}} = k^{1-p} \cdot (\|v\|_p)^{p-1}$$

and substituting into the second result finally yields

$$\|v\|_1 = k^{1-p} \cdot (\|v\|_p)^p$$

$\square$

Corollary 1. *For bit-vectors ($k = 1$) we have $\|v\|_1 = (\|v\|_p)^p$; in particular, we have $\|v\|_1 = (\|v\|_2)^2$.*

2.2.3 Reducing Dimensionality

We briefly outline a result of [52] for norm-preserving embeddings; for a more detailed treatment, see [50, 49, 55].

Consider an $m \times b$ matrix V in which we want to compute the $\|\cdot\|_2$-norm between pairs of row vectors. Naïvely calculating one pairwise norm costs $O(b)$ time. The Johnson-Lindenstrauss lemma states that, if we first multiply V by another matrix F of size $b \times \frac{4 \ln m}{\varepsilon^2}$, filled with random numbers from the normal distribution $(0, 1)$, we can use the pairwise norms between rows of $V \cdot F$ as good approximations of the pairwise norms between corresponding rows of V.

Specifically, for given ε and F, we have, with probability at least $1 - m^{-2}$,

$$\forall u, v \in V,\ (1-\varepsilon)\|u-v\|_2 \leq \|(u-v)F\|_2 \leq (1+\varepsilon)\|u-v\|_2$$

The dimensionality of $(u - v)F$ is now $\frac{4 \ln m}{\varepsilon^2}$ and thus independent of b.

Other probability distributions can also be used for populating the elements of F [2]. Figure 2.2 illustrates the basic embedding technique.

2.2.4 The Algorithm

The following theorem represents one of our main contributions. Namely, we can apply the JL lemma to tree bit-vectors in order to obtain a high-quality approximation of the RF metric between the original trees. Additionally, we can directly use the bounds from the JL lemma to establish the quality of our approximation.

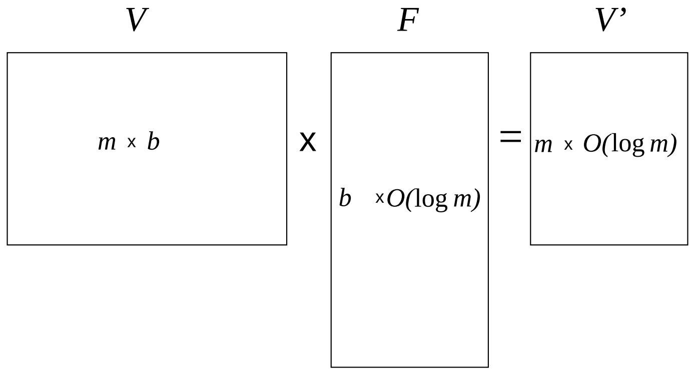


Figure 2.2: A sketch of randomized embedding. Each tree is a row in V; F is a random matrix; each row of V' is the embedded representation of the corresponding row vector in V.

Theorem 3. *Taking the square of the $\|.\|_2$-norm between embedded tree bit-vectors constitutes calculating a $(1+\varepsilon)$-approximation*[2] *of the RF distance between the original trees.*

Proof. Using the JL lemma preserves (up to the multiplicative factor of $1 \pm \varepsilon$) the $\|.\|_2$-norm between the non-embedded vectors. Corollary 1 establishes that, when dealing with bit-vectors, we need only to square the $\|.\|_2$-norm to recover the $\|.\|_1$-norm. Thus, since tree vectors are bit vectors, the JL embedding additionally preserves the $\|.\|_1$-norm. Finally, Theorem 1 states that the $\|.\|_1$-norm between tree bit-vectors is the RF-metric between the two source trees, so we are done. $\square$

Because the JL lemma is constructive, Theorem 3 provides a first algorithm. Given a set of m phylogenetic trees:

1. stack their bit-vector representations (recall that each has dimensionality b) to form an $m \times b$ matrix;

[2]Actually a $(1 + 2\varepsilon + \varepsilon^2)$ approximation, since we must square the 2-norm to recover the 1-norm

2. perform the embedding of Section 2.2.3 thereby reducing the row dimensionality of the matrix while preserving pointwise $\| \cdot \|_2$-norms between row vectors; and
3. for any pair of row vectors v_{T1}, v_{T2} (i.e., embedded trees), obtain the approximate RF distance by computing $(\|v_{T1} - v_{T2}\|_2)^2$.

However, this is the theoretical form of the algorithm. In practice, we do not compute the large matrix. Rather, we are able to incrementally embed the tree while performing a tree traversal. The manner in which this is performed is covered in Section 2.3.

Since the dimensionality of the embedded row vectors is $O(\log m)$, the time complexity of computing the approximate RF distance between two trees is also $O(\log m)$, so that our technique is asymptotically faster whenever we have $\log m = o(n)$.

2.3 A Framework and Implementation Tradeoffs

We have developed several efficient implementations of our approximation scheme. Additionally, we have improved the performance of Day's (exact) algorithm [24] in practice by using techniques discovered while implementing our approximation scheme.

We begin by presenting a general framework for which all of our algorithms (as well as Day's) are instantiations. Recall that each edge in a tree is identified by the bipartition it induces on the set of taxa. Thus implementing *any* RF algorithm invariably involves deciding upon a representation for taxa which can be efficiently accumulated into sets (of taxa). Accordingly each of our algorithms start by labeling taxa according to some scheme. The label for each taxon is used to represent the bipartition induced by its incident edge. We then provide, for each labeling scheme, an operator for accumulating two labels (i.e., two subtrees that meet at a common internal vertex) into a single label. To ensure the invariance of labeling across traversal strategies, we require that the accumulation operators be associative and commutative. The specific traversal strategy employed in our family of algorithms is a depth first traversal from a common, arbitrarily chosen, root taxon.

2.3.1 Direct Embedding

The P&M algorithm [65] used edge labels of size $O(n)$, namely a bit vector per edge. For each edge label there was one bit per leaf, and all leaves on one side of the bipartition (induced by an edge) were of the same value. The accumulation operator was bitwise-OR. Consequently, performing a tree traversal (which takes $O(n)$ time) while performing a bitwise-OR on two edge labels at each step (also incurs $O(n)$) yielded a total complexity $O(n^2)$ per tree for the embedding step.

Overcoming this problem requires eliminating the $O(n)$ length of edge labels. Recall that the essential feature of the embedding step from Section 2.2.3 is to establish a correspondence between tree edges and random vectors (of length $O(\log m)$). Also recall that an embedded tree is simply the sum of the random vectors that correspond to the edges found in the original tree. We now describe how to generate edge labels of length $O(\log m)$ space that are, in fact, the random vectors corresponding to tree edges.

In Section 2.2.3 we noted that distributions other than Gaussian can be used as elements in the random vectors. Consider one such (non-normalized) discrete distribution where $p(X = 1) = \frac{1}{6}$, $p(X = 0) = \frac{2}{3}$, and $p(X = -1) = \frac{1}{6}$. Next consider the mapping $l : \{0, 1, 2, 3, 4, 5\} \to \{-1, 0, 0, 0, 0, 1\}$ such that choosing from the interval $[0, 5]$ uniformly at random yields the aforementioned distribution by virtue of the mapping. Now, assign to each taxon an edge label consisting of a $O(\log m)$-

tuple (see Section 2.2.3 for the specific size requirement) of random numbers chosen uniformly from the interval $[0, 5]$. The accumulation operator is taken to be addition modulo 6. Thus every edge label directly maps through l into a random vector.[3]

Assuming unique labels for leaves (and the same leaf labels are used across the tree set), it is clearly the case that equivalent edges will map to the same random vector. However it is the case that two distinct edges may end up mapping to the same random value (i.e., a *collision*[4]). The probability of this occurring is equivalent to the event that two randomly chosen vectors are equivalent, which is the same probability that two rows from the embedding matrix are equivalent.

2.3.2 Improving Day's Algorithm

Because Day's algorithm runs in linear time and must traverse both trees, it follows that it must employ a constant-space edge labeling and a constant-time accumulation operator. Edge labels are in fact intervals, which need only be represented by their extrema, while the accumulation operator is simply interval union over adjacent intervals. As outlined in [24], Day's algorithm can be thought of as constructing a perfect hash function, where hashes are computed in $O(1)$ time, on the edges of one of the two trees under comparison.

It is possible to hash edges more conventionally [4]. In our implementation we begin by assigning to each taxon a random b-bit vector (for most practical purposes, a 64-bit integer). We then use the XOR operator for edge label accumulation. We then proceed, as in Day's algorithm, to compare two trees by comparing their lists of

[3]from the correct distribution since the sum (modulo k) of two uniform random vectors is, itself, a uniform random vector

[4]a term from hash functions, which this scheme turns out to embody, see Section 2.3.2 to see how recognizing this technique as hashing helps to improve the performance of Day's algorithm in practice

edge hashes. The major improvement in practice arises because in our case the hash needs only to be computed once per edge, whereas in Day's algorithm a (perfect) hash must be computed $O(m)$ times in order to perform $\binom{m}{2}$ comparisons. The risk in this approach (as with any conventional hash) is in collision, whose probability is derived in Section 2.3.3. Thus our approach carries a failure probability (albeit exponentially small).

For a fixed b, hashing in this way takes $O(n)$ time per tree, or $O(mn)$ time overall, which is optimal if every edge in every tree must be examined.

Combining Hashing with Embedding

Lessons from the previous section prompted us to investigate using conventional hashing in the approximation algorithm as well. We proceed by again using a b-bit random vector for leaf labels, and XOR as an accumulation operator. We then map (by using a conventional hash table) edge labels to random $O(\log m)$ length vectors from the appropriate distribution. This scheme turns out to be quickest in practice (of our approximation implementations) and as such is the approach employed in experimentation (Section 2.4).

Refer to Table 2.1 for a synopsis of all the algorithms presented. The column *Performance* refers to the running time of computing all pairwise RF distances among a set of m trees, where each tree is defined on the same n taxa. The algorithm denoted as naïve refers to the natural (trivial) quadratic RF algorithm, which is used in experimentation (Section 2.4).

Table 2.1: Summary of Expected Algorithm Asymptotic Performance

Algorithm	Result	Edge Label	Operator	Performance
Naïve	exact	taxa set	union	$O(m^2n^2)$
P&M	approximate	bit vector	OR	$O(m(n^2 + m \log m))$
Section 2.3.1	approximate	tuple	addition	$O(m \log m(n + m))$
Day's	exact	interval	union	$O(m(nm))$
Section 2.3.2	exact	bit vector	XOR	$O(m(nm))$
Section 2.3.2	approximate	bit vector	XOR	$O(m \log m(n + m))$

2.3.3 Probability of Collision

The primary disadvantage of a traditional hashing scheme, compared to Day's algorithm (which constructs a perfect hash), is the possibility of a hashing collision (i.e., two unequal edges being assigned the same label). Given a good hashing function, the probability of a collision decreases exponentially with the number of bits chosen for representing edge labels. The exclusive-OR function ($\oplus$), which we use, has this property.

Definition 2. *$\prod_A = \ell_1 \oplus \ell_2 \oplus \ldots \oplus \ell_k$ where taxa $A = \{a_1, a_2, \ldots, a_k\}$ and ℓ_i is a bit vector label for taxon a_i*

Theorem 4. *Given a set S of taxa with unique random b-bit vector taxon labels, and given two arbitrary subsets of taxa $A \subset S$ and $B \subset S$, we have*

$$A \neq B \Longrightarrow \prod_A = \prod_B \text{ with probability } p \leq \frac{1}{2^b}$$

Proof. Let x be a bit vector, y a random bit vector, and $z = x \oplus y$. Then z will also be a random bit vector which is uncorrelated with x (although obviously the triple x, y, z is correlated). By induction, $\prod_A$ will be random relative to $\prod_B$ if there is a greater than one taxon difference between A and B. If there is only a single taxon difference between them, then A and B are constrained to be different as a consequence of the unique labels assigned to the taxa). Thus, the probability of

$\prod_A = \prod_B$, for $A \neq B$, is either zero or the same as the probability of two random bit vectors of size b being equal, namely $1/2^b$. $\square$

Corollary 2. *Given a forest of trees with e unique edges over n taxa, and given unique taxon labels of b bits, the probability, p_f, that one or more pairs of edges will hash to the same label is $p_f < 1 - (1 - 2^{-b})^{\binom{e}{2}} < e^2/2^{b+1}$.*

Proof. Assuming that all hashing collisions in a forest of edges are uncorrelated, the expectation value of the number of collisions which might occur in a forest of e edges is $\langle c \rangle = \binom{e}{2}/2^b < e^2/2^{b+1}$. In general, hashing collisions need not be uncorrelated. A lower bound on $\langle c \rangle$ will occur when collisions are negatively correlated such that no more than one collision may occur in a forest. In this case $p_f = \langle c \rangle$. We have already seen that 2^{-b} is an upper bound on the probability an arbitrary pair of distinct edges will hash to the same value. As a consequence $1 - (1 - 2^{-b})^{\binom{e}{2}}$ is an upper bound on the probability that one or more collisions will occur in a forest of edges. Therefore, regardless of the distribution of edges in a forest, $p_f \leq 1 - (1 - 2^{-b})^{\binom{e}{2}} < e^2/2^{b+1}$. $\square$

2.3.4 Speeding up Consensus Methods

Before proceeding to the experimental evaluation of our RF techniques, we briefly present an unanticipated application of the material presented thus far – that of computing consensus trees. This application arises because bipartition hashing turns out to be very generally applicable, as well as effective. Namely, the methods detailed in this section have given rise to the fastest exact consensus method routines currently (as of the time of writing) in use. The text here refers to “improvements” which simply correspond to differences between the version of RAxML released in conjunction with our RECOMB 2009 paper (RAxML v7.2.1) versus the optimizations detailed in our ICCS 2010 paper (RAxML v7.2.6).

The implementation of consensus tree methods in RAxML was initially motivated by our work on bootstrap convergence criteria [62], which used the bipartition hashing techniques detailed earlier in this chapter. Specifically, we decided it worth the incremental effort to enhance RAxML such that it can compute strict, majority rules (MR), and extended majority rules (MRE) consensus trees. In this section we mainly focus on the construction of an MRE consensus tree, which is computationally more challenging due to compatibility checking.

The process of computing a consensus tree can be broken down into four steps,

1. **Tree Parsing:** Loading the collection of trees into memory and parsing them.
2. **Extraction and Addition of Bipartitions:** Extracting bipartitions from each parsed tree and inserting them into a hash table.
3. **Selection of Candidate Bipartitions:** Selecting candidate bipartitions for the final MRE tree and storing them in an array according to their frequency of occurrence and compatibility with the bipartitions that have already been added to the array.
4. **Reconstruction of the MRE tree:** Using the array of bipartitions to build the MRE tree and print it to file.

The fraction of execution time spent in the different phases largely depends on the tree size. For trees with 2,500 organisms it spends an approximately equal proportion of time in each phase, while for larger trees with 30,000-55,000 organisms, run times are dominated by the selection of candidate bipartitions (phase 3.) which requires more than 95% of total run time.

Tree Parsing The tree parsing procedure for the Newick tree format (see [10]) is straightforward, but has benefitted from some focused modifications. We replaced the parsing routine that directly reads and parses the tree file by a function that first loads the trees into main memory and then parses them as strings to avoid unnecessary I/O overhead. In addition, an optimization of the taxon name lookup procedure yielded significant speedups. In order to check for consistency in the taxon names, i.e., if a taxon name in the tree is contained in the set of taxon names, and also to consistently enumerate taxa, the parser needs to look up every taxon name that is encountered in a tree. The original implementation used a linear $O(n)$ time taxon name lookup which was replaced by a simple constant time lookup using a hash table. These optimizations yielded significant speedups of more than an order of magnitude, especially for trees with more than 1,000 taxa. Note that these optimizations are beneficial to many parts (other than MRE computation) of RAxML.

Extraction and Addition of Bipartitions Once an input tree has been parsed, we extract the $n-3$ nontrivial bipartitions and store them in a hash table using the simple hash function described earlier in Section 2.3.2. Note that while we use random labels as hash keys, we do not throw away the bit vector representations of bipartitions, as they are needed for compatibility checking. As such, the bit vectors are stored as one of the *values* of a hash entry (in the standard associative array terminology where a hash key maps to a hash value). Also as described earlier in Section 2.3, bipartitions of a tree can be extracted in $O(n)$ time. The pertinent RAxML structure is a hash table entry:

```
struct ent {
  unsigned int *bitVector;
  unsigned int *treeVector;
  ...
  struct ent *next;
};
```

where `bitVector` is a representation of the bipartition such that all leaves on one side of the bipartition (the side that contains the first taxon) have value 0 (resp. 1), and `treeVector` is a (bit vector) where an index i has value 1 when this bipartition occurs in tree i, and is 0 otherwise. Note that, `treeVector` could essentially be replaced by a simple integer counter for computing MR and MRE. However, other functions that operate on bipartitions of trees in RAxML require to also store the tree which generated a bipartition; for software engineering reasons we decided to keep the code as simple and generic as possible. In order to obtain the frequency of occurrence of a specific bipartition in the tree set, one needs to count the bits that are set in `treeVector`. This can be done by using one of the fast counting routines discussed at `http://gurmeetsingh.wordpress.com/2008/08/05/fast-bit-counting-routines/`. Based on extensive computational experiments with bit counting functions, we find that functions that use lookup tables perform best on modern CPUs.

Selection of Candidate Bipartitions Once the bipartitions of all trees are stored in the hash table, and the respective frequency of occurrence of every bipartition is computed (using a fast bit count on `treeVector`), it is trivial to select the bipartitions that form the MR consensus tree. One simply needs to iterate through all bipartitions in the hash table and retain every bipartition (or rather a pointer to every bipartition) in an array, that occurs in more than half of the input trees, i.e., whose frequency of occurrence is > 0.5. At most $n - 3$ bipartitions will be stored in this array in the case that the MR tree is a fully resolved binary tree.

Computing the MRE tree is only marginally more complicated, but significantly more compute intensive. One starts by constructing the MR tree, i.e., by adding those bipartitions that occur in more than 50% of the trees to the array of bipartitions of the consensus tree. Next, all bipartitions not occurring in the MR tree are sorted (in descending order) by their frequency of occurrence. Finally, the sorted list is scanned, and a bipartition is added to the bipartition array if it is compatible with *all* other bipartitions that already form part of the consensus tree. This operation is of time complexity $O(n^2)$, where n is the number of taxa, and hence dominates the run time of this step. However, we have found that the order in which the array of bipartitions that already form part of the consensus tree is traversed for checking compatibility has a notable effect on execution times. It turns out that, the number of pairwise compatibility checks between bipartitions is greatly reduced if the array is traversed from the end, i.e., starting at the most recently added entry. This reversal of the compatibility check order yielded a run time improvement for the bipartition selection phase of approximately 90% with respect to the original implementation on a tree with 2,554 taxa. The speedup is obtained because bipartitions that are located at the end of the array have a lower frequency of occurrence than bipartitions that are located at the start of the array. As such, there is a higher probability that the bipartition under consideration occurs together in a tree with bipartitions earlier in the array, and thus are compatible. It follows then that the probability that the candidate bipartition to be added is incompatible with the bipartitions located at the end is higher and we therefore, on average, need to conduct less compatibility checks per candidate bipartition. Thus, the biggest gains are achieved by very diverse input tree sets that do not give rise to a fully resolved binary MRE tree. Bipartitions with low occurrence frequencies can often be rejected after the first compatibility check against the accepted bipartition with lowest frequency.

All of the steps just described (for computing MRE) are trivial, with perhaps the only exception being compatibility checking. For compatibility checking, again, we initially followed a very straightforward approach. We used the well known property that for two bipartitions $A|B$, $C|D$ to be compatible, it must be the case that at least one of the intersections $A \cap C$, $A \cap D$, $C \cap B$, $B \cap D$ is empty [42]. If we have stored A and C in the canonical form described above (Section 2.1.3), the computation of the intersection of $B \cap D$ can be omitted because B and D will both contain the distinguished taxon and their intersection will therefore never be empty.

This formulation of compatibility gives rise to a straightforward implementation using bit-vector based bipartitions. Since the function for compatibility checking dominates the execution times of the $O(n^2)$ time pairwise compatibility check, we optimized it by testing different implementation options for the compatibility function. The optimization of this function yielded an additional run time improvement of approximately 50% on trees with more than 35,000 taxa and of 70% on trees with 2,554 taxa for the compatibility check.

Reconstruction of the MRE tree We have found that when large trees are involved, a nontrivial amount of time can be spent transforming the consensus tree in bit-vector bipartition form back into a Newick representation. Our basic approach for this transformation is similar to the approach taken in HashCS [82]. First, we sort the bit vector bipartitions according to number of bits set in ascending order. Then, for each bipartition, we search the sorted list for the first occurrence of a bipartition that has a superset of its bits set with respect to the bipartition under examination. In this manner we build for each bipartition, a list of the bipartitions that are its most strict subset bit vectors. It is then straightforward to traverse these lists in depth-first order and output the tree.

The sorting and the ascertained compatibility of all of the consensus bipartitions allow for a rapid test on whether bipartition A is a superset of bipartition B. Due to the sorting A is either a proper superset of B or the set shares an empty intersection with B (direct consequence of the compatibility check). Thus, for the superset test we only need to check, if any element of B is also contained in B. This optimization yields a run time improvement of 70% for the tree reconstruction phase on a tree set with more than 35.000 taxa.

2.4 Experiments

We have implemented the algorithms from Section 2.3 (with the exception of the Section 2.3.1 algorithm), in order to evaluate their performance experimentally, both in terms of speed and in terms of accuracy. We have run a large series of experiments, all on the CIPRES[5] cluster at the San Diego Supercomputing Center, a 16-node Western Scientific Fusion A8 running RedHat Linux, in which each node is an 8-way Opteron 850 system with 32GB of memory.

In the following experiments we generated forests of trees according to the following procedure (for various values of `numClusters`,`treesPerCluster`,j,k):

1. Generate a phylogenetic tree T_{seed} uniformly at random from $\mathcal{T}_{numTaxa}$
2. do `numClusters` times
    - (a) create a new tree $T_{clusterSeed}$ by doing a random number ($0 \leq k <$ maxTBR) of TBR operations to T_{seed}.
    - (b) write $T_{clusterSeed}$ to file
    - (c) do `treesPerCluster` times
        - i. create a new tree T' by doing a random number ($0 \leq j <$ maxNNI) of NNI operations to $T_{clusterSeed}$.
        - ii. write T' to file

This procedure creates the classic "islands" of trees [56] by providing pairwise distant trees as seeds and generating a cluster of new trees around each seed tree.

[5]The Cyber Infrastructure for Phylogenetic Research project, at `www.phylo.org`, is a major NSF-sponsored project involving over 15 institutions and led by B.M.E. Moret.

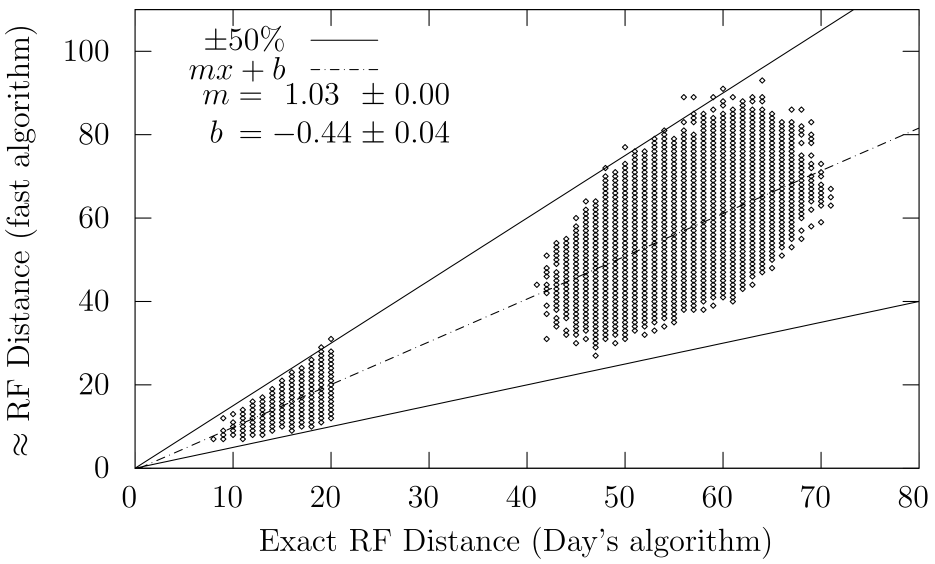


Figure 2.3: 50% error bounds approximate versus exact RF distance

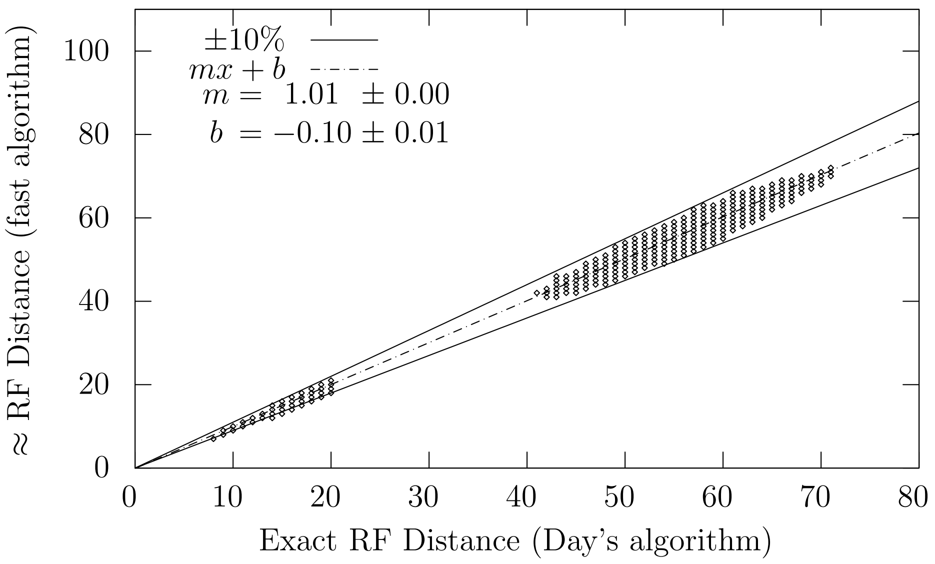


Figure 2.4: 10% error bounds approximate versus exact RF distance

2.4.1 Validating the Approximation Bounds

In this experiment, we focused on the difference between the exact and the approximate distances. 10 clusters of 100 trees each were constructed using 10 TBR operations per cluster and 5 NNI operations per tree. A 1000×1000 matrix of Robinson-Foulds pairwise distances was then constructed first using Day's algorithm and then twice using the Section 2.3.2 algorithm (once with $\varepsilon = 0.5$ and once with $\varepsilon = 0.1$). Figures 2.3 and 2.4 are scatter plots using the results from Day's algorithm *vs.* the results from the Section 2.3.2 algorithm. The variation due to the embedding is easily seen to obey the $1 \pm \varepsilon$ constraint.

Table 2.2: Parameter Values Used to Generate Clustering Data

Parameter	Values	Parameter	Values
TBRs	$5, 8, 11, 14$	clusters	$2, 3, \ldots, 10$
NNIs	$10, 20, \ldots, 100$	trees per cluster	$50, 100$
taxa	100		

2.4.2 Consequences of Approximation

In this experiment, we generated 7200 forests using all permutations of various parameter values, as described in Table 2.2. Each permutation of parameters was used to generate 10 different forests (for a total of 72000). A distance matrix was then constructed for each forest using the Section 2.3.2 algorithm (which, if no hashing collisions occur, is exact, like Day's algorithm) and the Section 2.3.2 algorithm using values of $\varepsilon = 0.1, 0.2, \ldots, 1.0$. 64-bit edge labels were used to avoid hash collisions. These distances were then used to cluster the trees using a hierarchical agglomerative clustering algorithm based on cluster distances defined as the distance between the farthest separated cluster members. As a stopping criterion, the same number of clusters were generated as existed in the original data. The Rand index [69] (the most commonly used measure of clustering quality) was then computed and plotted as a function of ε. Figure 2.5 shows the result. (Note that $\varepsilon = 0$ implies the use of the Section 2.3.2 algorithm.)

From these data, it appears that an approximation bound of 10% to 20% may be

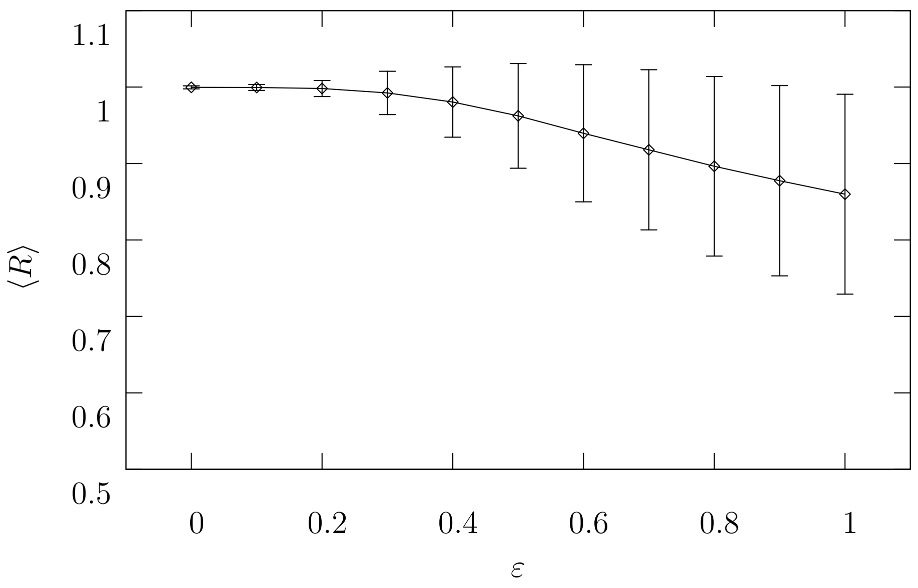


Figure 2.5: Clustering error as a function of distance error bounds

acceptable in some cases (note the tight bounds on the curve at these values of ε), although of course results will depend on the distribution of trees in the data and on the analysis methods used.

2.4.3 Performance

Hash Collisions

Both of our implementations under consideration carry the risk of collision. To evaluate the actual rate of collision, we used 16-bit labels to hash edges from over 300,000 forests and plotted as a function of the number of edges in a forest, $\langle c \rangle$ (the average number of collisions per forest), and p_f (the probability of one or more collisions occurring in a forest). The results, in Figure 2.6, support our derived upper bounds; namely: $p_f < 1 - (1 - 2^{-b})^{\binom{e}{2}} < e^2/2^{b-1}$.

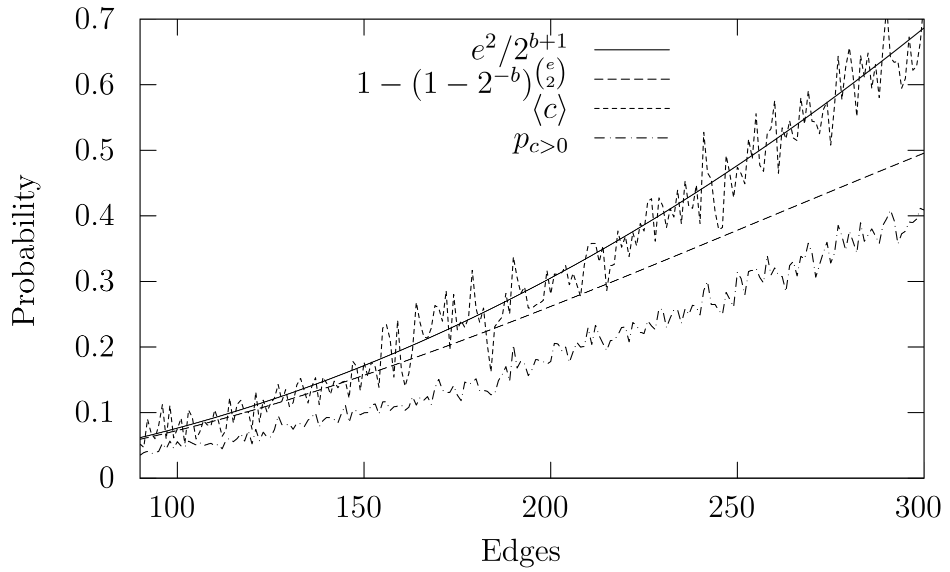
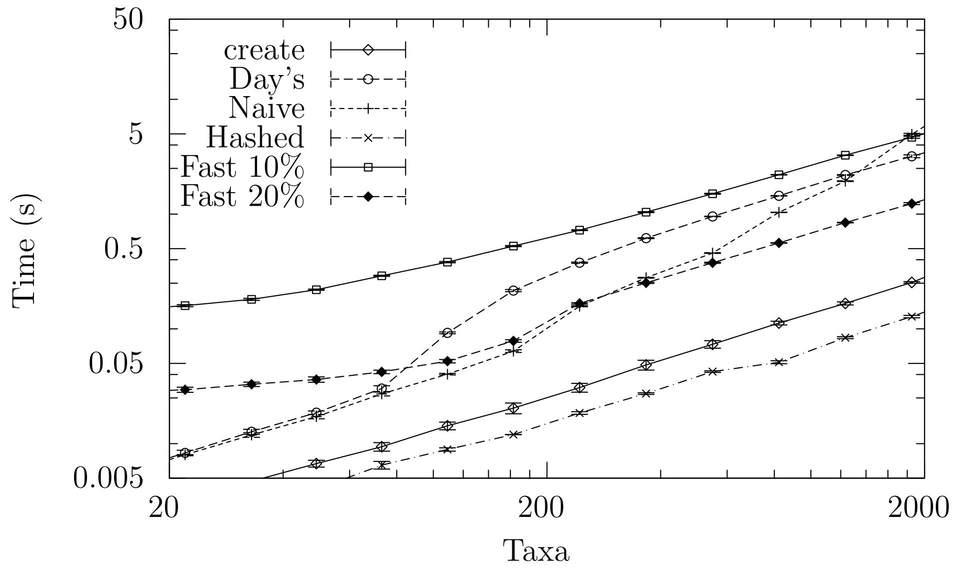


Figure 2.6: Results of using 16 bits to hash the edges of over 300,000 forests. $\langle c\rangle$ is the expected number of resulting hashing collisions as a function of the number of edges in a forest; $p_{c>0}$ is the probability that one or more hashing collisions will occur.

Speed

Figure 2.7 (top) displays running time as a function of tree size for generating pairwise distances using each algorithm presented here with the exception of that of Section 2.3.1. In Figure 2.7 (bottom), the running time is displayed as a function of forest size while the number of taxa is held constant. All other parameters remain invariant.

It can be seen that, as the number of taxa grows, both Day's algorithm and the naïve algorithm quickly become onerous. As predicted, Section 2.3.2 performs much better, even with an error bound of just 10%. The Section 2.3.2 algorithm, which can be made arbitrarily probabilistically exact, outperforms all others in these tests. For growth with respect to the number of trees, the findings are similar, although, in this case, all curves are of course quadratic—but the coefficients of the curve for



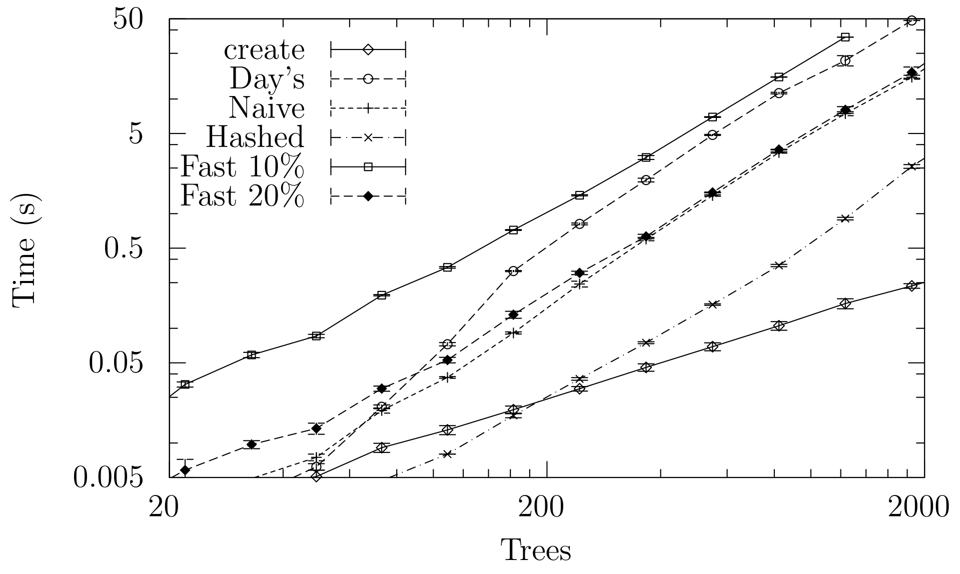


Figure 2.7: Performance of various algorithms as a function of number of taxa (top) and number of trees (bottom). Variable-length bit vectors were used for the naïve algorithm. The other algorithms used 64-bit hashed edge labels. The create curve shows the time used to generate the random forests. The labels Day's, Hashed (i.e., Section 2.3.2), and Naïve refer to exact distance computations, whereas Fast 10% and Fast 20% (i.e., Section 2.3.2) refer to approximate distance computations.

the hashed implementation of the exact method lead to a drastically slower growth curve.

Chapter 3

How Many Bootstrap Replicates are Necessary?

An idea springs out of his forehead fully formed, with no warning. This is how all the best ideas arrive. Ideas that he patiently cultivates from tiny seeds always fail to germinate or else grow up into monstrosities. Good ideas are just there all of a sudden, like angels in the Bible. You cannot ignore them just because they are ridiculous. Waterhouse stifles a giggle and tries not to get overly excited. The dull, tedious, bureaucratic part of his mind is feeling testy, and wants a few shreds of supporting evidence.

– Neal Stephenson, *Cryptonomicon*

3.1 Background and Motivation

As has been mentioned earlier, significant progress has been achieved in the field of heuristic ML search algorithms with programs such as PHYML [40], GARLI [93], LeaPhy [89], and RAxML [76]. However, there is still a major bottleneck in computing bootstrap support (BS) values on these trees, which can require more than one month of sequential execution time for a likely insufficient number of 100 replicates [75] on a reasonably fast CPU. To date, it has proved infeasible to assess empirically the convergence properties of BS values, much less to evaluate means for dynamically deciding when a set of replicates is sufficiently large—at least on the size of trees where computing BS values is an issue.

Recently, Stamatakis *et al.* [77] introduced a fast BS algorithm that yields a run time acceleration of one to two orders of magnitude compared to other current algorithms while returning qualitatively comparable support values. This improvement makes possible a large-scale experimental study on bootstrap stopping criteria, the results of which are the topic of this chapter.

We propose two stopping criteria. Both split the set of replicates computed so far into two equal sets and compute statistics on the two sets. The frequency criterion (FC) is based on the observed frequencies of occurrences of distinct bipartitions; the more conservative weight criterion (WC) computes the consensus tree for each subset and scores their similarity. Both criteria can be computed efficiently and so a stopping test can be run every so many replicates until stopping is indicated. We test these criteria and the general convergence properties of BS values on 17 diverse real-world DNA, single-gene, as well as multi-gene datasets, that include between 125 and 2,554 taxa. We find that our stopping criteria typically stop computations after 100–500 replicates (although the most conservative criterion may continue for several thousand replicates) while producing support values that correlate at better than 99.5% with the reference values on the best ML trees. Unsurprisingly, differences tend to occur mostly on branches with poor support—on branches with support values of at least 0.75, over 98% of the values returned after early stopping agree with the reference values to within 5%.

Our results show that the BS convergence speeds of empirical datasets are highly dataset-dependent, which means that bootstopping criteria can and should be deployed to determine convergence on a per alignment basis. The criteria help to conduct as many BS replicates as *necessary* for a given accuracy level and thus help to reduce the computational costs for phylogenetic analyses. Practitioners will no longer have to enter a guess nor worry about the quality of support values; moreover, with most counts of replicates in the 100–500 range, robust BS under ML inference becomes computationally practical for most datasets.

The remainder of this chapter is organized as follows: In Section 3.2, we review the bootstrap concept and related work on stopping criteria for (mostly non-phylogenetic) bootstrap procedures, including a brief overview of convergence criteria for MrBayes [72]. In Section 3.3 we describe our family of stopping criteria. In Section 3.5 we describe our experimental study, give detailed results, and discuss their implications.

Over and above the preliminary version of the paper upon which this chapter is based [62], we have added the following content: The criteria are now fully implemented in the current release version 7.2.6 of RAxML which required a large re-engineering effort. We have added an entirely new section (Section 3.4) which details our major undertaking to improve the runtime performance of our technique. Specifically, we discuss and assess the applicability of bipartition hashing [64] to bootstopping (Section 3.4.1), and present timing data showing the speedup RAxML has enjoyed via the application of these techniques (Section 3.4.2). These techniques have also been integrated with all other functions in RAxML that operate on bipartitions, such as a fast implementation of the Robinson-Foulds distance. We present new results on stability properties of our criteria (Section 3.5.3), namely that they appear tolerant to reordering bootstrap replicates, as well as seemingly independent of the bootstrap procedure (standard versus Stamatakis' rapid bootstrap [77]). Finally, in Section 3.5.5 we have also included a comparison to Hedges equation that demonstrates that the number of replicates required to achieve a certain accuracy level is indeed highly dataset-dependent

3.2 Related Work on Bootstopping Criteria

3.2.1 The Phylogenetic Bootstrap

Phylogenetic bootstrapping is a fairly straightforward application of the standard statistical (nonparametric) bootstrap and was originally suggested by Joe Felsenstein [28] as a way to assign confidence values to edges/clades in phylogenetic trees. Phylogenetic BS proceeds by generating perturbed BS alignments which are assembled by randomly drawing alignment columns from the original input alignment with replacement. The number of columns in the bootstrapped alignment is identical to the number of columns in the original alignment, but the column composition is different. Then, for each BS alignment, a tree is reconstructed independently. The procedure returns a collection of tree *replicates*. The replicates can then be used either to compute consensus trees of various flavors or to draw confidence values onto a reference tree, e.g., the best-scoring ML tree. Each edge/branch in such a reference tree is then assigned a confidence value equal to the number of replicates in which it appears. The question we address in this chapter is: how many replicates must be generated in order to yield accurate confidence values? By accurate confidence values we mean relative accuracy of support values (the “true” support values are unknown for empirical datasets) with respect to support values obtained by a very large number ($\geq$ 10,000 in our experiments) of reference replicates. The extent to which the question about the appropriate number of BS replicates has been answered in other applications of the (non-phylogenetic) bootstrap is the subject of the following subsection.

3.2.2 General Bootstopping Criteria

Most of the literature addressing (whether theoretically or empirically) the issue of ensuring a sufficient number of replicates stems from the area of general statistics or econometrics. However, they are difficult to apply to phylogenetic BS due to the significantly higher computational and theoretical complexity of the estimator [47]. In addition, the problem is more complex since the number of entities (bipartitions) to which support values are assigned grows during the BS procedure, i.e., adding more BS replicates increases the number of unique bipartitions. This is not commonly the case for other application areas of the general Bootstrapping procedure and general bootstopping criteria that have recently been proposed (for instance see [41]).

Standard textbooks on Bootstrapping such as [22, 25] suggest to choose a sufficiently large number B of BS replicates without addressing exact bounds for B. This does not represent a problem in most cases where the BS procedure is applied to simple statistical measures such as the mean or variance of univariate statistics. Efron and Tibshirani [25] suggest that $B = 500$ is sufficient for the general standard bootstrap method in most cases. [57] propose a simple approach to determine B *a priori*, i.e., before conducting the BS analysis, based on a worst-case scenario by approximating the standard deviation of BS statistics. The analysis in [57] concludes that a general setting of $B = 200$ provides a relatively small error margin in BS estimation. This approximation can only be applied to standard BS procedures, based on simple, univariate statistics. However, a larger number of BS replicates is required for other applications of the Bootstrap such as the computation of confidence intervals or tests of significance. P. Hall [43] proposes a general method for stopping the BS in a percentile-t confidence interval. In the area of econometrics, Davidson and MacKinnon [23] propose a two-step procedure to determine B for BS P-values based on the most powerful test. Andrews *et al.* [6, 7, 8] propose and evaluate a general three-step algorithm to specify B in the bootstrap procedure. Andrews and

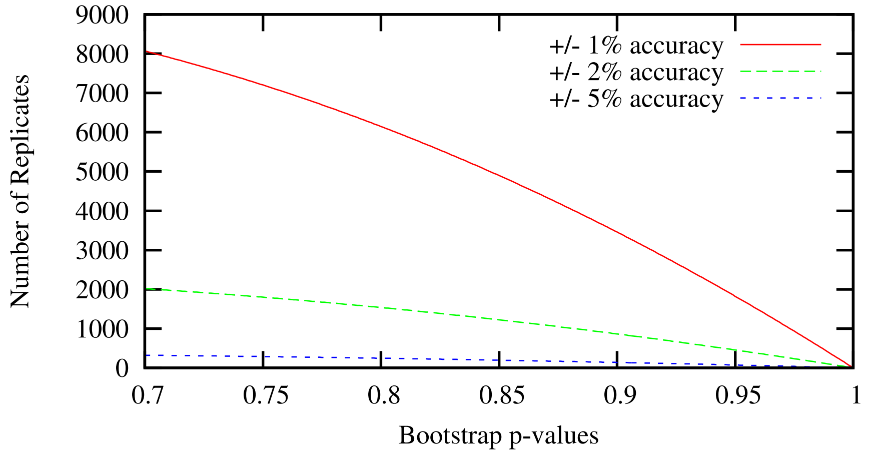


Figure 3.1: Number of required Replicates for various confidence intervals according to Hedges

Buchinsky [9] then further extend their algorithm to bootstrap BCA intervals.

With respect to phylogenetics Hedges [44] suggests a method to specify B *a priori* for a given level of significance. In the approach of Hedges', a bipartition is assumed to occur in bootstrap replicates according to a binomial distribution, with binomial parameter p equal to its true support. As such, it is possible (under the binomial assumption) to calculate an upper bound B on the number of replicates needed to achieve a specified accuracy. Figure 3.1 shows the estimation of B for three accuracy thresholds.

This approach does not take into account the number of sequences and hence the number of potential alternative tree topologies, or the number of base-pairs or distinct patterns in the alignment. However, as underlined by our experimental results, important alignment-specific properties such as the "gappyness" (percentage of gaps) of the alignment, the quality of the alignment, and the respective phylogenetic signal strength greatly influence the estimator (the tree search algorithm) and hence the stability of BS replicates. We conclude that an adaptive stopping criterion which is computed on the fly at regular intervals during the actual BS search is best suited to take into account the particularities of real-world datasets and to determine a useful trade-off between accuracy and inference time. We are convinced that such trade-offs will become increasingly important for analysis on large phylogenomic datasets under computational resource constraints, as a recent collaborative study [45] with biologists already required 2,000,000 CPU hours on an IBM BlueGene/L supercomputer. Therefore, we assess our approach empirically, via a large number of computational experiments on diverse real datasets.

3.2.3 Bayesian Convergence Criteria and Tools

There exists some work on convergence criteria and tools for Bayesian phylogenetic analyses, most probably because the convergence of the actual search as opposed to a sufficient number of BS replicates in ML represents a more serious methodological problem for MCMC in general and phylogenetic MCMC searches in particular [60, 74, 78]. Gelman, Rubin, and Brooks [13, 35] provide general frameworks to determine convergence of iterative simulations, with a focus on MCMC methods. MrBayes implements convergence diagnostics for multiple Metropolis-coupled MCMC chains that use the average standard deviation in partition frequency values across independent analyses. One potential drawback is that these statistics take into account all partition frequencies and not only the important, highly supported ones. In addition, there exist tools for graphical exploration of convergence such as AWTY [61] to visualize convergence rates of posterior split probabilities and branch lengths or Tracer [68] that analyzes time-series plots of substitution model parameters. AWTY also offers bivariate plots of split frequencies for trees obtained via independent chains. Note that both AWTY and Tracer require the user to visually inspect the respective output and determine whether the MCMC chains have converged. We are not aware of any computational experiments to assess the performance and accuracy of the above methods.

3.3 Bootstopping Criteria

In this section, we introduce stopping criteria for bootstrapping procedures, which we call "bootstopping" criteria. These are measures that are computed and used at run time, during the replicate inference phase, to decide when enough replicates have been computed. The frequency-based criterion (FC) is based upon Pearson's correlation coefficient, whereas the Weighted Robinson-Foulds criterion (WC) is based upon the weighted Robinson-Foulds metric – i.e., the weighted version of the dissimilarity metric treated so thoroughly in Chapter 2.

3.3.1 Stopping Criteria

The two criteria we present in the following are both based on the same underlying mechanism. Initially, the set of replicates to be tested for convergence is randomly split into two equal halves. Then we compute statistics between the bipartition support values induced by these halves. If the difference between the splits of the replicates are small this indicates that adding more replicates will not significantly change the bipartition composition of the replicate set. In addition, we compute the statistics not only for one but for 100 random splits of the replicate sets, i.e., we draw a sample from all possible random splits of the replicates by applying a permutation test.

Frequency Criterion (FC)

The frequency-based criterion uses the bipartition frequencies of all replicates computed up to the point at which the test is conducted, for example every 50 replicates, i.e., at 50, 100, 150, 200, ... replicates. One major design goal is to devise stand-alone criteria that do not rely on a previously computed best-known ML tree for the original alignment. This is partially due to the rapid BS algorithm (and future extensions thereof) in RAxML that uses information gathered during the BS search to steer and accelerate the search for the best-scoring ML tree on the original alignment. Another important goal is to avoid a heavy dependency on the spacing (e.g., every 10, 20, or 50 replicates) of two successive steps of the test, i.e., we do not want to compute statistics that compare 20 with 30 replicates. Therefore, we have adopted a procedure, that is in some sense similar to the aforementioned convergence tests for MCMC chains implemented in MrBayes. There are two main differences though: (i) we do not use the test to determine convergence of the tree search itself, and (ii) we do not apply the test to only one single random or fixed split of the replicate tree set.

Our FC test works as follows: Assume that the test is conducted every 50 replicates, i.e., after the computation of 50, 100, 150, ... BS replicates. This spacing of 50 has been chosen empirically, in order to achieve a reasonable computational trade-off between the cost of the test and the cost for computing replicates (future work will cover the development of adaptive spacing strategies). The empirical setting also fits the typical range of bootstopped tree topologies, which range between 150 and 450 in our FC-based experiments, depending on the strength of the signal in the respective alignment. For the sake of simplicity, assume that we conduct the test for 50 replicates. At the top level of our procedure we perform a permutation test by randomly splitting up those 50 tress p =100 times (p =100 permutations) into disjoint sets s_1,s_2 of equal size with 25 trees each. The advantage of 100 random

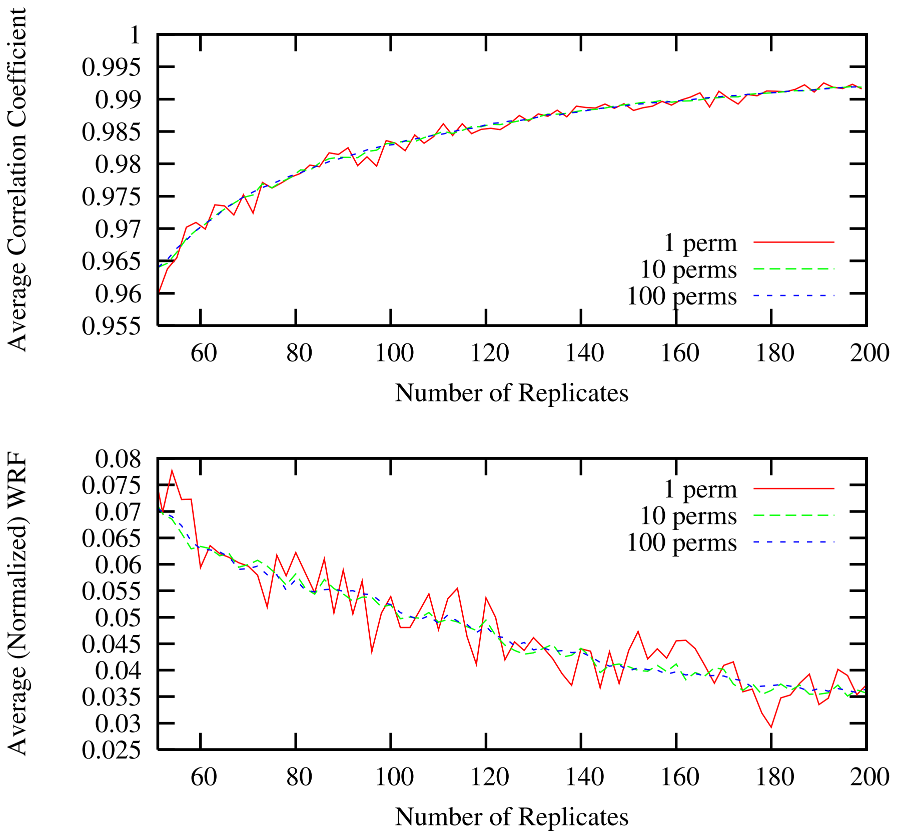


Figure 3.2: FC (top) and WC (bottom) criteria for various p settings on dataset 500

splits over a single random split or a fixed split into, e.g., replicates with even and odd numbers, is that the curve is smoothed and depends to a far lesser degree on a by chance favorable or unfavorable single split of the data.

In Figure 3.2 we depict the impact of using p =1, 10, and 100 permutations on the FC and WC criteria (see Section 3.3.1) for a dataset with 500 sequences. As expected the curve becomes smoother for larger p settings; a setting of $p = 10$ appears to be sufficient to smooth the curve and reduce the cost of the test. Though statistically more stable, the disadvantage of this approach is clearly the significantly increased computational cost of the test. Nonetheless, an initial, highly optimized at a technical level, yet algorithmically naïve implementation requires only 1 minute to conduct all 6 tests on 50, 100, ..., 300 replicates on a 1,481 taxon dataset (2 minutes, 40 seconds for $p = 1000$ random splits), compared to roughly 27 hours for the computation of 300 rapid BS replicates.

For each of the aforementioned 100 random splits we compute the support vectors v_1 for s_1 and v_2 for s_2 for all bipartitions b_{ALL} found in $s_1 \cup s_2$, i.e., all bipartitions contained in the original 50 trees. Note that both vectors v_1, v_2 have length b_{ALL}. Given those two vectors for each permutation (random split) i, where, $i = 0, ..., 99$ we simply compute Pearson's correlation coefficient ρ_i on the vectors. Our procedure stops if there are at least 99 ρ_i with $\rho_i \geq 0.99$ (only one possible parameter setting). We henceforth denote the Pearson's threshold used as ρ_{FC}. A potential drawback of this method is that the support frequencies on the best-scoring tree or for all bipartitions found during the BS search might not follow a normal distribution. Nonetheless, the FC method appears to work reasonably well in practice (see Section 3.5). Another potential drawback is that the FC criterion is based on the bipartition frequencies of all bipartitions found. However, from a biological point of view, one is only interested in the "important" bipartitions, i.e., the bipartitions induced by the best-scoring ML tree or the bipartitions that form part of a strict, majority rule, or extended majority rule consensus tree. We address the design of a criterion that only takes into account important bipartitions in the next section. Nonetheless, the FC test can easily be extended in the future to take into account the important bipartitions by providing a user-defined best-scoring ML tree using either Pearson's correlation or, e.g., the mean square error between corresponding bipartition support values.

Weighted Robinson-Foulds distance-based Criterion (WC)

The Weighted Robinson-Foulds (WRF) distance criterion (WC) is employed similarly to the FC criterion (i.e., every 50 trees and uses $p = 100$ permutations per test).

Rather than computing a vector correlation, we compute the majority rules consensus trees for s_1 and s_2 and then assess the (dis)similarity between the two consensus trees. We then use the respective consensus trees, which only contain support values for "important" biologically relevant partitions, to calculate the WRF distance between the consensus tree $c(s_1)$ of tree set s_1 and the consensus tree $c(s_2)$ of tree set s_2.

As a distance measure and hence convergence criterion we use the weighted Robinson-Foulds distance (WRF). This weighted topological distance measure between consensus trees takes into account the support values and penalizes incongruent subtrees with low support to a lesser extent. When RF distances are significantly larger than their weighted counterparts (WRF), this indicates that the differences in the consensus trees are induced by subtrees with low support. When WRF≈RF this means that the differences in the tree topologies under comparison are due to differently placed clades/subtrees with high support. From a biological perspective the WRF distance represents a more reasonable measure since systematists are typically interested in the phylogenetic position of subtrees with high support. In real-world studies the typical empirical threshold is set to 75%, i.e., clades with a BS support of $\geq 75\%$ are usually considered to be monophyletic (see [75] for a summary). As for the FC criterion, the WC stopping rule can be invoked with varying numbers of permutations and threshold settings. One might for example stop the BS procedure, if for $p = 99$ out of 100 permutations, the relative WRF between $c(s_1)$ and $c(s_2)$ is $\leq 5\%$. For reasons of consistency we also denote the threshold parameter for WC as ρ_{WC}, a ρ_{WC} setting of 0.97 means that the BS search is stopped when p WRF distances are $\leq (1.0 - 0.97) = 3\%$.

3.4 Implementation Considerations

In the following, we address an important issue that had been completely omitted from the original paper upon which this chapter is based, i.e., that of efficiently implementing our criteria which also entails several interesting algorithmic problems. In the following we will focus on implementation and performance of the WC criterion which we consider as being the biologically more meaningful criterion. The algorithmic problems associated with the FC criterion are analogous.

3.4.1 Application of Bipartition Hashing

The efficient computation of our bootstrap convergence criteria (see Section 3.3.1) is closely related to efficiently computing the Robinson-Foulds (RF) metric [71] and handling bipartitions induced by a large collection of trees. The main computational challenge lies in the design of efficient methods to extract, maintain, and operate on lists that contain all nontrivial bipartitions (splits) induced by a collection of trees. Apart from computing the RF distances such lists of bipartitions are also required for computing consensus trees [51] or implementing convergence assessment mechanisms for Bayesian inference programs [61]. While the theoretically optimal RF algorithm is well-described [24], important technical details are often not considered and rarely assessed experimentally, such as, e.g., the choice of the hash function.

A bipartition of a tree T, $A|B$ can be represented by two presence/absence bit vectors v_A, v_B of length n, where every bit denotes the presence/absence of a taxon in the subtree to the left (T_a) and to the right (T_b) of the edge/branch that is being cut. Clearly, v_A is the bit-wise complement of v_B. Because of this property it suffices to either store v_A or v_B. In order to ensure consistency of this choice between v_A and v_B and avoid computational overhead to check whether two bit-vectors are bit-wise complements of each other, one may chose to always store the bit-vector that contains (or does not contain) a specific taxon, e.g., the first taxon in the input alignment. This is important, to ensure consistency among bipartitions extracted from two distinct trees, T_1, T_2, because a bipartition that is shared between the trees may be stored as v_A for T_1 and v_B for T_2.

Let us now consider how to efficiently extract bipartitions from an unrooted tree that is already stored in memory, i.e., we do not consider how to efficiently read in trees in the standard NEWICK format (see `http://evolution.genetics.washington.edu/phylip/newicktree.html`) from file. The algorithm for efficient computation of the bipartitions at each inner branch is conceptually very similar to Felsenstein's pruning algorithm for computing the ML score on a tree [27]. It relies on a rooted view of the otherwise unrooted tree by the bipartition bit vectors as well as on a cyclic organization of inner node pointers as used for Maximum Likelihood computations (for details about this data structure organization, see, e.g., [79]). Initially, we will assign bit vectors of length n to all $2n-2$ nodes of the tree and initialize the bipartition vectors at the tips accordingly, i.e., just set the bit that corresponds to the respective taxon number.

Thereafter, we place a virtual root into the branch that leads to the first taxon in the input alignment and recursively compute all bipartition vectors bottom-up towards the virtual root via a depth-first traversal. Keep in mind that all inner bipartition vectors will be oriented towards the virtual root of the tree. Every time we compute the bipartition vector at an inner node that is connected to another inner node, we can directly store the bipartition in a hash table. This means that we are always storing only those bipartitions that do not contain the selected taxon and thereby ensure consistency. The complexity of this operation is $O(n^2)$, since we need to compute $n-3$ bipartition vectors and the computation of each bipartition vector is a `for` loop over n bits. However, in practice 32, 64, or even 128 (if SSE-vectorized code is used) bit vector entries can be computed in one CPU cycle, such that a more accurate approximation for the actual number of instructions is, e.g., $n \cdot (n/32)$.

Given this efficient method for extracting bipartitions from trees, we can now consider the appropriate data structure for storing these bipartitions. The usage of a hash table is a straight-forward and represents an efficient choice. However, the question arises how to select a hash function for the hash key which in our case is simply the bipartition vector. The usage of universal hash functions [18] as advocated in some more theoretical papers [81, 80, 4] is highly questionable: *Firstly*, because the computation of a universal hash function given a bit vector of length n is slow, and *secondly* universal hash functions only work well, when hash keys are equally randomly distributed [18], which is not very likely for hash keys that are induced by a hierarchical data structure such as a tree. Those two practical performance considerations have not been addressed in the aforementioned papers.

In contrast to this we have experimentally assessed several highly-tuned open-source hash functions that are nicely summarized at `http://burtleburtle.net/bob/hash/doobs.html` adopting an algorithmic engineering approach [59]. In addition to this collection of hash functions, we also tested a phylogeny-specific hash key proposed by Pattengale, Gottlieb, and Moret [64]. This method takes advantage of the tree structure and uses 32 or 64-bit integer values as hash keys instead of the entire bipartition vector. Initially, each taxon is initialized by a random unsigned 64-bit integer number. Then, the hash numbers for the bipartitions are also computed bottom up towards the virtual root by performing a bit-wise exclusive or on the respective child numbers (hash-keys). This procedure can be conveniently integrated into the depth-first traversal that is used to compute the bipartition vectors. Extensive tests on large collections of trees have revealed that this method slightly outperforms all other tested hash functions in terms of speed and generates the same amount of collisions that are resolved by chaining in the current RAxML implementation. The procedure is outlined in Figure 3.3.

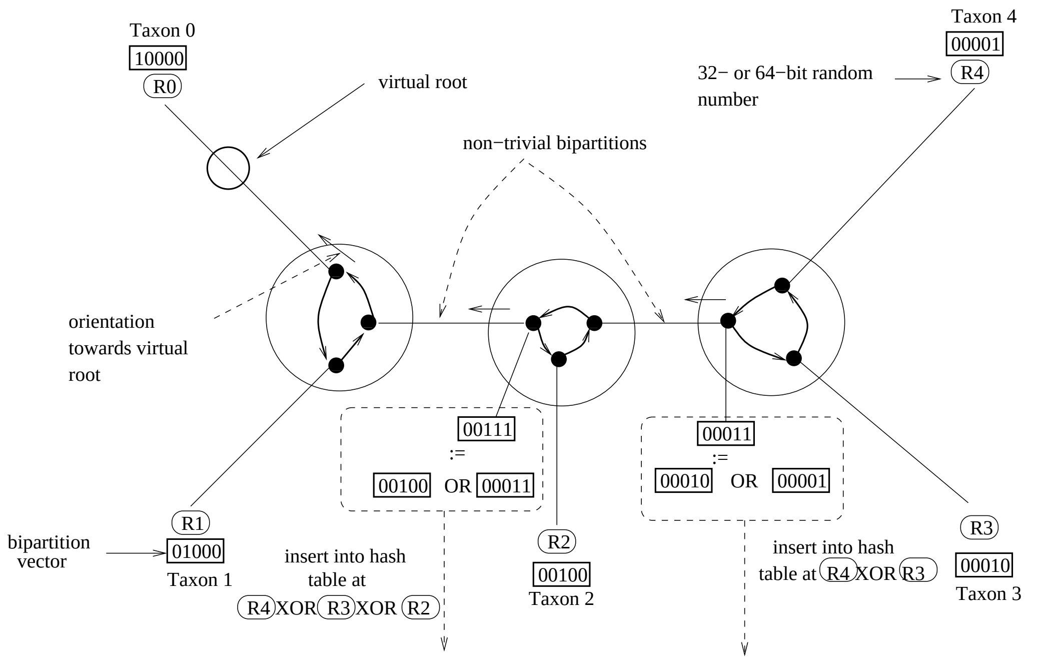


Figure 3.3: Outline of the procedure to efficiently extract bipartitions and generate bipartition hash numbers on an unrooted binary tree.

For performing performing the splits of our permutation tests for FC and WC (see Sections 3.3.1 and 3.3.1) we also need to keep track of the trees that contain a bipartition that is stored in the hash table. For this we deploy an additional presence/absence bit vector of length r, where r is the number of trees/replicates. Hence, if we add an entry to the hash table and the respective slot is already occupied we initially need to compare the bipartition vector (or list of bipartition vectors) in that slot with the bipartition vector to be added. If it matches one of the stored bipartition vectors, we simply set the respective bit for the replicate number to 1, otherwise we resolve by chaining.

3.4.2 Running Time Improvement

The initial implementation of our bootstopping criteria [62] did not perform bipartition hashing (as described in Sect 3.4.1). Instead, a list of bipartitions were accumulated, such that determining whether a bipartition had been previously encountered required a (worst-case linear time) scan of the list. To this end, we have integrated and thoroughly assessed two alternative implementations that deploy bipartition hashing. The first implementation, which was written from scratch in RAxML, is based upon hashing with chaining as described above. The second implementation invokes an appropriately adapted version of `FastRF` (from [64]) that has been integrated into RAxML (currently unreleased). The latter implementation ignores collisions and thus is an inexact technique, but within a tolerable error. While operations on bipartitions of trees represent an interesting algorithmic problem, one must keep in mind that the respective execution times of the bootstop test are insignificant, compared to the actual replicate inference times under Maximum Likelihood. Nonetheless, they may become a limiting factor for parallel scalability on massively parallel machines because of Amdahl's law. Within this context speed as well as a potential future parallelization *are* important issues.

In Table 3.1 we report the speedup achieved by RAxML for the optimized bootstopping functions. The column *DATA* labels each data set and corresponds to its number of taxa (see Section 3.5.1). Column *CON-WC* indicates how many trees were processed for a setting of $\rho_{WC} = 0.03$. Finally, columns *[62] impl.*, *RAxML 7.2.6*, and *RAxML+FastRF* correspond to running times (in seconds) of the WC implementation for the preliminary version of the paper upon which this chapter is based [62], the publicly available open-source version of RAxML 7.2.6, and the integration of `FastRF` [64] with RAxML, respectively.

We observe that the new bipartition hashing approach has yielded a dramatic

Table 3.1: Performance improvements for the Bootstopping function in RAxML. Columns 3-5 are reported in seconds of CPU time.

DATA	CON-WC	[62] impl.	RAxML 7.2.6	RAxML+FastRF
150	650	2.48	2.52	3.29
218	700	5.21	5.30	6.16[2]
500	400	3.70	3.42	4.28
994	300	4.84	4.25	3.97
1481	450	38.71	34.56	37.58
2000	600	90.20	76.78	85.69
2554	500	64.80	52.42	52.51
4114	100[1]	24.45	11.32	8.65
6718	100[1]	122.18	26.92	17.00
7764	100[1]	264.43	37.09	23.09
37381	250[1]	dnf	1700.29	453.86

[a]Bootstopping did not converge, and the indicated number of replicates reflects all that were available.

[b]Bootstopping for this sample actually converged after 550 trees (when $\rho_{WC} = 0.03$), however we adjusted ρ_{WC} to 0.0297 (thereby requiring 700 replicates to stop) to enable a meaningfully comparable run time.

speedup over the preliminary implementation, especially as the number of taxa grows. Further, if one is willing to sacrifice exactness (`FastRF` has a failure probability, and is thus inexact), the third implementation is particularly desirable for datasets with huge number of taxa. See [64] for a discussion of the accuracy of the `FastRF` approach.

3.5 Experimental Setup and Results

3.5.1 Experimental Setup

To test the performance and accuracy of FC and WC we used 17 real-world DNA alignments containing 125 up to 2,554 sequences. The number of distinct alignment patterns ranges between 348 and 19,436. For the sake of simplicity, alignments will henceforth be referenced by the number of taxa as provided in Table 3.2. The experimental data spans a broad range of mostly hand-aligned sequences including rbcL genes (500, 2,554), mammalian sequences (125, 1,288, 2,308), bacterial and archaeal sequences (714, 994, 1,481, 1,512, 1,604, 2,000), ITS sequences (354), fungal sequences (628, 1,908), and grasses (404). The 10,000 reference BS replicates on each dataset were inferred on two AMD-based Linux clusters with 128 and 144 CPUs, respectively. All result files and datasets used are available for download at `http://lcbb.epfl.ch/BS.tar.bz2` We make this data available in the hope that it will be useful as a basis for further exploration of stopping criteria as well as general properties of BS.

Computational experiments were conducted as follows. For each dataset we computed a minimum of 10,000 BS replicates using the rapid Bootstrapping (RBS [77]) algorithm implemented in RAxML. We then applied stand-alone bootstopping tests (either FC or WC) that take the set of 10,000 BS reference replicates as input and only execute the tests described in Section 3.3 without performing the actual BS search. Returned is a file containing the first k trees from the full set, where k is determined by the stopping criterion (FC or WC, along with appropriate parameter values). We refer to these first k trees as the 'bootstopped' trees.

We then computed a number of (dis)similarity metrics between the reference replicates and the bootstopped replicates, including: correlation coefficient, RF between MRE consensus trees of the two sets, and WRF between the MRE consensus trees of the two sets. Additionally, support values from the bootstopped and full replicate sets were drawn on the best-scoring ML tree and the resulting support values compared.

Table 3.2: Performance analysis of FC ($p = 99$, $\rho_{FC} = 0.99$) vs. WC ($p = 99$, $\rho_{WC} = 0.97$) for three metrics: number of trees to converge, WRF between MRE consensus trees and Correlation Coefficient. Column # Patterns indicates the number of distinct column patterns in each alignment. The last line depicts the respective averages.

DATA	CON-FC	CON-WC	WRF-FC	WRF-WC	P-FC	P-WC	# Patterns
125	150	50	0	0	0.9997	0.9994	19,436
150	250	650	0.03	0.01	0.9984	0.9994	1,130
218	300	550	0.04	0.01	0.9977	0.9988	1,846
354	450	1200	0.03	0.01	0.9979	0.9992	348
404	250	700	0.04	0.01	0.9965	0.9988	7,429
500	200	400	0.03	0.01	0.9982	0.9991	1,193
628	250	450	0.03	0.01	0.9975	0.9987	1,033
714	200	400	0.03	0.02	0.9977	0.9989	1,231
994	150	300	0.04	0.02	0.9964	0.9974	3,363
1,288	200	400	0.03	0.02	0.9967	0.9985	1,132
1,481	300	450	0.04	0.02	0.9968	0.9979	1,241
1,512	250	350	0.03	0.02	0.9977	0.9983	1,576
1,604	250	600	0.04	0.02	0.9975	0.9990	1,275
1,908	200	400	0.03	0.02	0.9975	0.9987	1,209
2,000	300	600	0.03	0.01	0.9976	0.9989	1,251
2,308	150	200	0.03	0.02	0.9980	0.9985	1,184
2,554	200	500	0.03	0.01	0.9975	0.9991	1,232
1,102	238	482	0.03	0.01	0.9976	0.9987	2,771

3.5.2 Results for FC and WC Methods

In Table 3.2 we provide basic performance data for FC and WC. Column *DATA* lists the alignments, *CON-FC* the FC bootstop convergence number, and column *CON-WC* the WC bootstop convergence number. Columns *WRF-FC* and *WRF-WC* provide the WRF distance between the MRE consensus tree for the bootstopped trees and the MRE consensus tree induced by the reference replicates for FC and WC respectively. Finally, columns *P-FC* and *P-WC* provide Pearson's correlation coefficient between support values from the bootstopped trees and the reference trees on the best-scoring ML tree for FC and WC respectively.

We observe that WC tends to be more conservative, i.e., stops the BS search after more replicates, except for dataset 125. Dataset 125 is a particularly long phylogenomic alignment of mammals and exhibits a surprisingly low variability for the bipartitions it induces. The 10,000 reference replicates only induce a total of

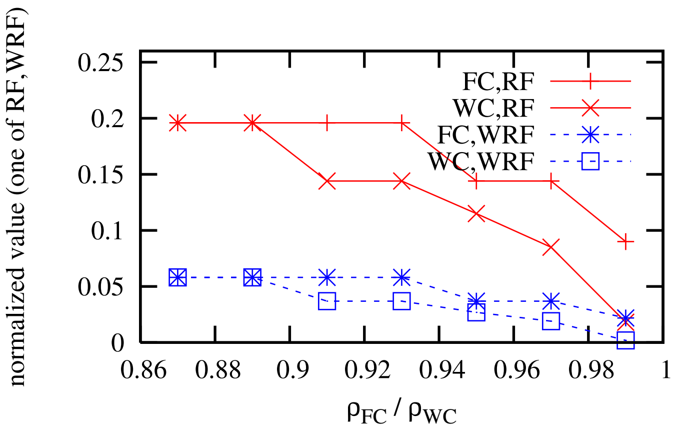


Figure 3.4: Plot showing convergence of WC over FC for various threshold settings (ρ_{FC} and ρ_{WC} respectively) on dataset 1418.

195 distinct bipartitions, which is extremely low given that a single BS tree for this dataset induces $125 - 3 = 122$ nontrivial bipartitions. The WC method appears to capture this inherent stability of the BS trees sooner than FC, while the WRF to the MRE tree is 0 in both cases, i.e., the consensus trees for 50, 150, and 10,000 replicates are exactly identical. This also underlines our claim that our criteria help avoid needless computation (and needless energy expenditures, as large clusters tend to be power-hungry), in particular on such large and challenging phylogenomic datasets. Due to the general trend for WC to stop later, both *WC* metrics (P/WRF) are higher than the respective values for FC. For WC, a setting of $\rho_{WC} = 0.97$ always returns a bootstopped set with a WRF $< 2\%$ to the MRE consensus of the reference replicates. The results also clearly show that there is a significant alignment-dependent variability in the stopping numbers, as these range between 150 and 450 replicates for FC and between 50 and 1,200 for WC.

In Table 3.3 we provide additional metrics for the bootstopped trees. Columns μ_x and σ_x^2 provide the mean error and the mean squared error between support values induced by the x ={FC,WC}-bootstopped trees and by the reference trees on the best-scoring ML tree. Columns *SUPPLOSS-FC* and *SUPPLOSS-WC* quantify the

Table 3.3: Performance analysis of FC ($p = 99$, $\rho_{FC} = 0.99$) vs. WC ($p = 99$, $\rho_{WC} = 0.97$) for three metrics: mean error, mean squared error, and loss of support. The last line depicts the respective averages.

DATA	μ-FC	σ^2-FC	μ-WC	σ^2-WC	SUPPLOSS-FC	SUPPLOSS-WC
125	0.303279	0.637530	0.483607	1.807108	0.001066	0.004672
150	1.544218	2.941922	1.074830	1.402564	0.009252	0.003605
218	1.865116	3.205062	1.297674	1.836971	0.005070	0.004674
354	1.364672	1.912598	0.886040	0.864506	0.002009	0.002835
404	2.553616	6.626178	1.384040	2.386179	0.012357	0.007170
500	1.792757	3.532503	1.239437	1.936634	0.010020	0.006841
628	2.030400	4.531876	1.398400	2.175677	0.013400	0.008408
714	2.129395	4.973412	1.424754	2.396237	0.010858	0.008833
994	2.498486	11.178353	2.068618	9.014464	0.013895	0.010575
1,288	2.477821	8.308652	1.700389	3.752257	0.013899	0.009864
1,481	1.845061	5.082219	1.496617	3.243223	0.008562	0.007287
1,512	1.762094	3.958643	1.552684	3.176317	0.008403	0.006289
1,604	1.898813	3.891073	1.229232	1.746953	0.008120	0.005721
1,908	1.961680	4.209030	1.377528	2.298479	0.009711	0.007113
2,000	1.773160	3.323105	1.184276	1.504350	0.008488	0.005020
2,308	1.951410	6.626706	1.703254	4.919317	0.010330	0.009681
2,554	2.063897	4.639194	1.248530	1.793192	0.011319	0.006370
1,102	1.871522	4.681062	1.338230	2.720849	0.009221	0.006762

deviations of support values in the best scoring ML tree.

In Figure 3.4 we graphically depict, for one dataset (1481), the convergence of FC versus WC. We plot the RF and WRF distances between the MRE consensus of the bootstopped trees and reference trees over distinct settings (0.87, 0.88,...,0.99) for ρ_{FC} and ρ_{WC}. For all but two datasets we observed that WC yielded a better convergence (while it required almost 50% more replicates on average) toward replicate sets whose consensi are more congruent (i.e., have lower RF and WRF distances) with the full replicate sets, as a function of ρ. This favorable property is due to the fact that WC is exclusively based on the "important" bipartitions. Therefore, WC allows to more precisely specify the desired degree of accuracy with respect to the biologically relevant information via an appropriate setting of ρ. As can be derived from Table 3.2 a setting of $\rho = 0.97$ for WC induces a WRF toward the reference dataset consensus that is $\leq 2\%$ in all cases for all of our datasets. Hence, the usage of a WC threshold will also be more meaningful, because it appears to be strongly correlated with the final WRF distance to the 10,000 reference replicates.

Table 3.4: Data supporting the robustness of WC to reordering replicates as well as the method for generating bootstrap replicates. The notable column is σ_ε which indicates strong agreement across criterion applications while shuffling replicates.

DATA	μ_ε	σ_ε	μ_μ	σ_μ	SBS_ε
125	7.9	2.39	3.86	1.63	
150	3.2	1.08	1.04	0.23	
218	2.9	0.54	0.90	0.14	3.0
404	3.7	0.64	1.03	0.18	
500	4.9	0.70	1.62	0.22	4.0
628	4.7	0.78	1.38	0.16	
354	3.0	0.63	0.88	0.16	5.0
714	5.5	0.92	1.96	0.23	6.0
994	7.2	0.75	2.71	0.56	
1481	4.6	0.92	1.39	0.19	
2000	4.9	0.70	1.23	0.13	
1288	6.1	0.70	1.94	0.20	
1604	5.0	0.77	1.30	0.08	
1908	6.4	0.92	1.90	0.11	7.0
2554	5.8	0.87	1.58	0.09	
2308	9.0	1.61	3.12	0.33	
1512	6.5	0.92	2.04	0.14	

3.5.3 Robustness of Criteria

To conclude that our criteria are robust, we investigated the sensitivity of our criteria to two factors: the ordering of bootstrap replicates and the method used to create the bootstrap replicates. In tables 3.4 (for WC, $\rho_{WC} = 0.03$) and 3.5 (for FC, $\rho_{FC} = 0.99$)) we report on the results. For each full set of bootstrap replicates we generated 10 random permutations of the order of trees. We then applied our bootstop procedures again on each of the copies (permutations).

In each table, μ_ε refers to the mean of the worst support value error (for bipartitions from the best ML tree with support $\geq 75\%$) across 10 permutations. While the selected threshold settings for the stopping criteria yield certain accuracy errors, the standard deviation of the same quantity – σ_ε, is small, which underlines the robustness of our stopping criteria under permutations of the input replicates that we

Table 3.5: Data supporting the robustness of FC to reordering replicates as well as the method for generating bootstrap replicates. The notable column is σ_ε which indicates strong agreement across criterion applications while shuffling replicates.

DATA	μ_ε	σ_ε	μ_μ	σ_μ	SBS_ε
125	5.8	1.78	1.74	0.70	
150	5.5	1.75	2.98	0.49	
218	5.5	1.36	2.47	0.43	4.0
404	6.1	0.94	2.44	0.32	
500	7.4	1.36	3.45	0.42	5.0
628	6.8	1.33	2.95	0.27	
354	4.3	0.90	1.60	0.14	5.0
714	7.3	1.10	3.54	0.31	7.0
994	9.5	1.63	4.20	0.57	
1481	6.1	1.22	2.21	0.21	
2000	6.8	1.08	2.26	0.11	
1288	8.4	1.43	3.41	0.19	
1604	7.9	1.58	2.78	0.20	
1908	9.2	1.33	3.32	0.17	7.0
2554	9.1	1.87	3.44	0.32	
2308	10.0	1.18	3.99	0.19	
1512	7.3	1.10	2.90	0.20	

intended to demonstrate. We have also included (for completeness) the mean and standard deviation of the average error in support (again, of bipartitions from the best ML tree with support $\geq 75\%$) as μ_μ and σ_μ.

Regarding robustness to the method used to generate replicates, we generated standard bootstrap replicates for five of our datasets, and subsequently ran them through our bootstopping criteria. The results are also listed in tables 3.4 and 3.4, under the column SBS_ε. Clearly, the error in bipartition support for the bootstopped set of standard BS sets agrees nicely with the rapid BS case.

3.5.4 Convergence of Data Sets

In addition to assessing our stopping criteria, we have also comprehensively assessed the inherent convergence properties of our replicate sets. Doing so has enabled us to understand a number of quantities that tend to reflect bootstrap support and may help in the design of improved stopping criteria. We have plotted a number of (dis)similarity measures between a subset (i.e., the first m trees) and full replicate ($\geq 10,000$ trees) set. In Figure 3.5 we plot the RF and WRF (in the two lower plots) between the MRE consensus of each tree set restricted to the first m trees versus its respective full set of replicates ($\geq 10,000$ trees). This plot shows the differences in convergence speeds among datasets. In addition, it underlines that WRF introduces less noise than RF as replicates are added, so that WRF is a more reliable measure for convergence. An extreme example for this is dataset 354, a short (348 alignment columns) alignment of maple tree sequences from the ITS gene that is known to be hard to analyze [39]. A comparison between the development of RF and WRF over the number of trees for this alignment shows that there are many sequences with low support that are placed in different parts of the tree and essentially reflect unresolved nodes. The slight increase of distance metrics around 1,000 replicates and consecutive decrease observed for dataset 125 might be minor artifacts of the RAxML RBS algorithm.

Also in the upper part of Figure 3.5 we plot the development of the mean error between support values of m replicates and all replicates on the best-scoring tree. The three plots in Figure 3.5 clearly show that the development of WRF distances over the number of replicates is highly congruent to the development of the mean error on the best-scoring tree. Thus, WRF can be used as a criterion to determine convergence without an external reference tree. Accordingly, Figure 3.6 shows the development of WC and FC over number of replicates, which as desired tracks nicely with Figure 3.5. Designing such a criterion has been a major goal of the phylogenetic community; WRF is the first good answer. Moreover, the plots can help to determine an appropriate threshold setting for ρ_{WC}, depending on the desired degree of accuracy.

Finally, in Figure 3.7 we plot the support values of FC/WC-bootstopped trees against the support values from the reference replicates on the best-scoring ML tree for dataset 628. The comparison clearly shows a decrease in deviations from the diagonal for the WC criterion.

3.5.5 Comparison to Hedges Criterion

We also experimentally assess the accuracy of the formula proposed by Hedges ([44] that is also covered briefly in Section 3.2.2 on real datasets. As already mentioned it can be used to compute an upper bound for the number of replicates that are required to achieve a certain accuracy. In our experiments, we set the upper bound such that the theoretical error for support values of 75% (or greater) lies between +/- 2%. This upper bound is roughly 2000 replicates as can be derived from Figure 3.1. We chose this threshold of accuracy because biologists typically employ this threshold when deciding whether a bipartition is supported or not. This empirical setting is also suggested by an in-depth study on real and simulated datasets [46].

Therefore, we performed experiments in order to determine how many replicates were truly necessary (for our data) to meet the desired accuracy of +/- 2 % on our datasets for all bipartitions supported by $\geq$ 75% by 10,000 replicates on the best-scoring ML tree. The results of our experiments, i.e., the number of replicates per datasets to achieve the desired accuracy are indicated as follows:

Dataset	125	1288	1481	150	1512	1604	1908	2000	218
# Replicates	950	dnf[1]	1400	1600	1500	1400	1650	1650	1200

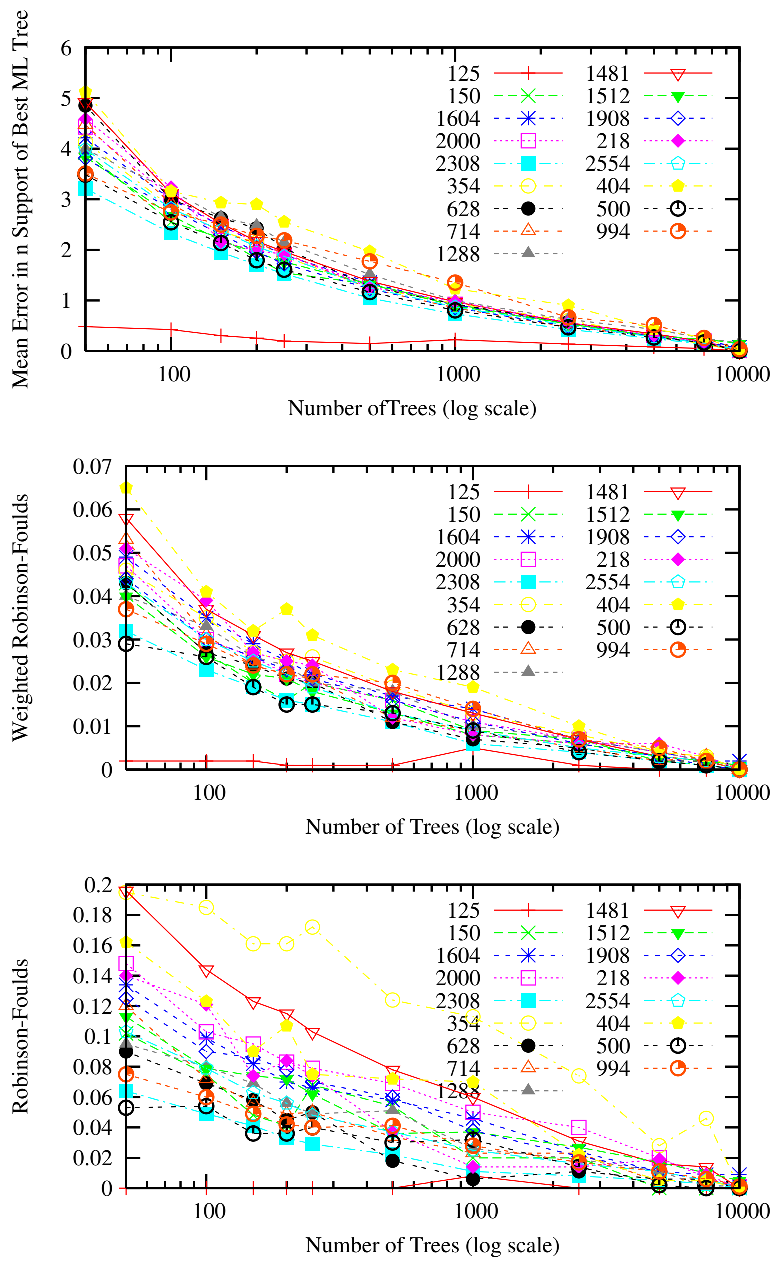
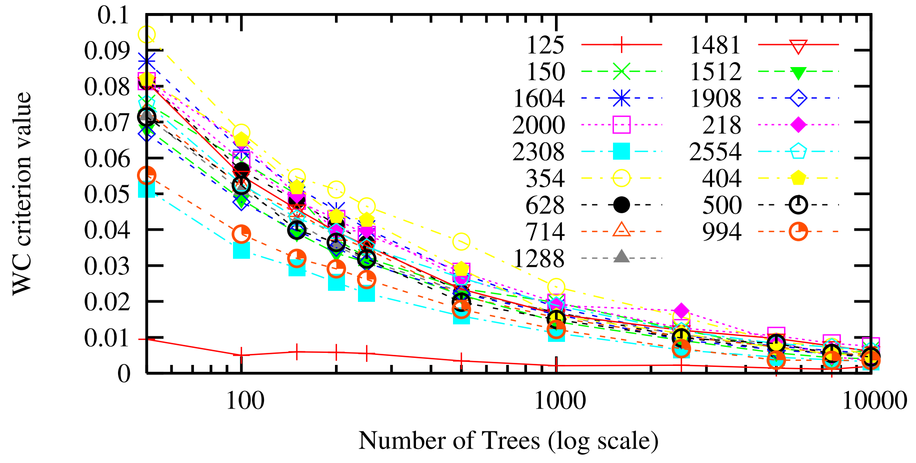


Figure 3.5: Inherent convergence of replicate sets scored by (top) error in support of best ML tree (middle) WRF and (bottom) RF distances between the first m trees and the entire (10,000 tree) set.

Dataset	2308	2554	354	404	500	628	714	994
# Replicates	1550	1900	1550	1550	1700	1850	1200	1550

[1]Dataset 1288 had not beat the threshold by 2000 replicates, but had by 2500.



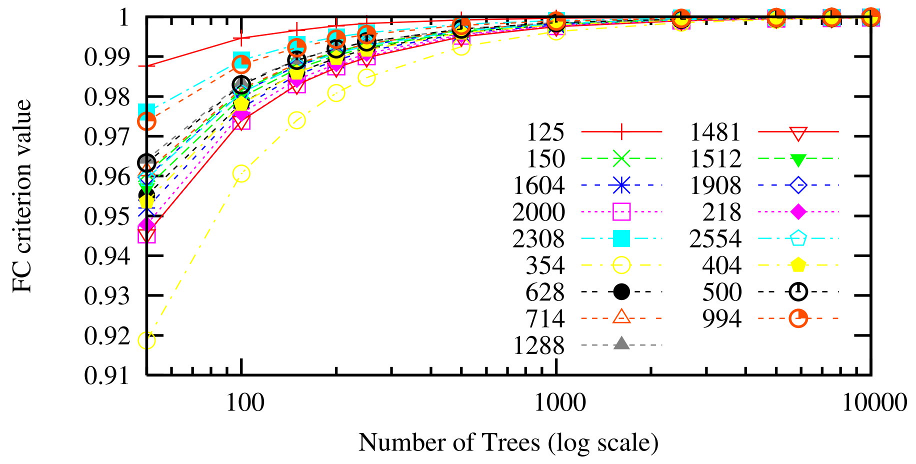


Figure 3.6: Values of FC and WC criteria for tree subsets consisting of the first m trees.

As such, we conclude that Hedges' estimate provides a reasonable upper bound for the accuracy, meaning that it is not a gross overestimate. To our knowledge this data represents the first empirical assessment of Hedges formula. Nonetheless, the number of replicates required is highly dataset-dependent. Thus, given the above stopping numbers, there is a potential for computing far too many replicates and wasting 30% or more CPU hours when deploying the formula. Interestingly, dataset 1288 requires more replicates than indicated by Hedges formula to achieve the desired accuracy level.

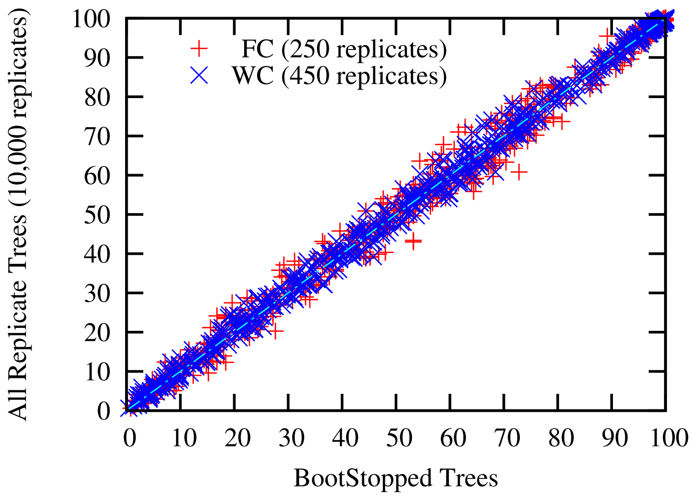


Figure 3.7: Support values drawn on the best ML tree for FC (blue) and WC (red) versus full replicate set, for data set 628

Chapter 4

Uncovering Hidden Phylogenetic Consensus

Then there is some talk about secrecy. A great deal of talk about it. They run through drills intended to test their ability to throw things away properly. This goes on for a long time and the longer it continues, without an explanation as to why, the more mysterious it becomes. The musicians, who were at first a little put out by their chilly reception, start to speculate amongst themselves as to what kind of an operation they have gotten themselves into now.

– Neal Stephenson, *Cryptonomicon*

4.1 Background and Motivation

Phylogenetic consensus methods are used for combining a set of trees defined on the same set of leaves into a single tree that summarizes the information found in the set. By their very nature, these methods discard information, typically structural elements not prevalent in the set. However, the most popular consensus methods (strict and majority rule) are susceptible to so-called *rogue taxa* [90]. That is, while the tree set may agree very strongly on the structure relating a large subset of the leaves, the remaining few leaves (the rogue taxa) can effectively prevent this underlying structure from appearing in the strict or majority consensus tree. In other words, these methods end up discarding structural elements that are, in fact, prevalent in the set.

Much work has been done on the problem of summarizing a set of trees and on the issue of rogue taxa in particular. The pioneering work of Wilkinson [90, 91, 92] addresses the problem by returning *sets* of trees, some of which are missing leaves, with the aim of conveying the prevalent structural elements in at least one of the returned trees. While theoretically satisfying, this approach suffers from computational complexity problems (he provides an exponential time algorithm) and, more importantly, from difficulties in interpretation.

A problem closely related to both consensus and rogue taxa is the *Maximum Agreement Subtree (MAST)*. A MAST on a set of input trees is the subtree of largest leaf-set cardinality common to all input trees. While the general problem of finding the MAST of three or more trees is $\mathcal{NP}$-hard [5], it can be solved efficiently when at least one of the input trees has bounded degree [26]. Another agreement subtree optimization problem *Maximum Information Subtree (MIST)* was proposed by Bryant[14] to overcome a key deficiency of MAST, namely that the maximization of leaf-set cardinality can entirely obscure important internal structure revealed by a smaller, suboptimal for MAST, leaf subset. Bryant's algorithm for solving MIST, whose complexity mirrors that of MAST algorithms, actually affords the practitioner an option to weight the importance placed on leaf-set cardinality versus internal structure in the solution. As such, the optimization function for MIST has a striking resemblance to the MISC optimization problem we propose below. Unfortunately, *all* agreement subtree approaches tend to be too conservative for our purpose; most notably, there exist instances where the strict consensus tree (without dropping any leaves) has more internal edges than any MAST or MIST (Section 4.5.1).

Cranston and Rannala recently presented a Markov Chain Monte Carlo (MCMC) method for identifying a version of rogue taxa in the context of Bayesian phylogenetic reconstruction [20]. Their approach identifies subsets of leaves for which the posterior distribution strongly supports the structure of the induced subtree—leaves left out can be viewed as rogue taxa, albeit in the narrow context of a sampling of trees in a Bayesian search, rather than in the general context of a consensus of trees. All of the approaches mentioned thus far fall into the category of "leaf-dropping methods," in the terminology of Redelings [70]. In contrast, Redelings presents, again in the context of Bayesian phylogenetics, a method that returns a "multi-connected tree," which includes all leaves, but does not summarize the information through a single tree and thus again raises issues of interpretation—an issue plaguing all approaches producing non-trees [11, 17, 34, 48].

In this chapter we contribute another leaf-dropping method, one based on a rigorous definition of the tradeoff involved between dropping leaves and uncovering additional consensus structure. Most existing measures and methods discard leaves in order to uncover *any* underlying structure; in contrast, our approach sets up a bicriterion problem, in which leaves should be discarded only if the gain in uncovered internal edges outweighs the loss incurred by discarding the leaves. We are not the first researchers to define some notion of relative information content for consensus trees [86], but our definition is the first to both *explicitly* take into account the loss incurred by dropping taxa, and generalize outside the setting of agreement subtrees. We provide an effective greedy heuristic to compute a good (if not necessarily optimal) set of rogue taxa and apply it to both pathological examples from the literature and a collection of large biological datasets that we used in a prior study of bootstrapping. As the presence of rogue taxa in a set of bootstrap replicates can lead to deceivingly poor support values, we propose a procedure to recompute support values in light of the rogue taxa identified by our algorithm; applying this procedure to our biological datasets caused a large number of edges to change from "unsupported" to "supported" status, indicating that many existing phylogenies should be recomputed and reevaluated to reduce any inaccuracies introduced by rogue taxa.

We also present, which is new material respective to the paper upon which most of this chapter is based, a section on theoretical aspects of our proposed optimization problem – Maximum (Relative) Information Subtree Consensus (MISC-$\mathcal{C}$). This section shows that the strong ties between agreement subtree problems (i.e., MAST, MIST) and MISC-$\mathcal{C}_{m-1}$ appear to break down when generalizing to a non–strict setting. Further, it appears that (in the non–strict setting) a simpler problem, namely finding a maximum cardinality leaf subset L' so that *some* bipartition of L' appears in $\mathcal{C}_t(\mathcal{T}|L')$, may be intractable even for trees with bounded degree.

The rest of the chapter is organized as follows. In Section 4.2 we define our measure of relative information content, formalize the bicriterion optimization problem for consensus and rogue taxa (MISC-$\mathcal{C}$), and present some theoretical results that underlie our approach. In Section 4.3 we present an efficient greedy heuristic for our bicriterion problem. In Section 4.4 we present the results of experiments performed with our greedy heuristic. Finally, in Section 4.5 we explore complexity theoretic aspects of MISC-$\mathcal{C}_t$, especially where $t < m - 1$.

4.2 Relative Information Content, Consensus Methods, and Rogue Taxa

4.2.1 The measure and the problem

The general problem we study can be phrased as follows: given a set $\mathcal{T}$ of trees on a common leaf set L and given a frequency-based consensus method $\mathcal{C}_t$, we want to find a leaf subset L' that optimizes the relative information content of the consensus returned by $\mathcal{C}_t$ on the set of subtrees induced by L'. The crucial notion here is that of relative information content. Formally, if $\mathcal{C}_t(\mathcal{T}|L')$ yields $T' = (L', B')$, then the *relative information content* is

$$I(T', L, \mathcal{C}_t) = \frac{|L'| + |B'|}{|L| + (|L| - 3)} \tag{4.1}$$

This measure is the ratio of the total number of bipartitions (trivial and nontrivial) in the consensus tree derived on the reduced leaf set to the total number of bipartitions in an ideal, fully resolved tree on the original leaf set. By taking trivial bipartitions into account, we automatically penalize a method for removing many leaves, since the number of trivial bipartitions is simply the number of leaves. By adding the number of nontrivial bipartitions, we reward a method for preserving more internal edges, since the denominator is fixed to the number of such edges in an ideal tree. Note that the use of the word 'information' in our definition does not imply information-theoretic foundations.

We can now formulate our main problem, which we call *MISC*, for *Maximum (Relative) Information Subtree Consensus.*

Problem 1 (MISC)**.** *Given a set $\mathcal{T}$ of trees defined on a common leaf set L and a frequency-based consensus method $\mathcal{C}_t$, find a leaf subset L' that maximizes the relative information content $I(\mathcal{C}_t(\mathcal{T}|L'), L, \mathcal{C}_t)$.*

Note that a MAST solution typically maximizes the $|B'|$ term at the expense of the $|L'|$ term—it has no direct penalty for dropping leaves; in contrast, consensus methods typically maximize $|L'|$ (in the case of majority and strict consensus, by forcing $L' = L$) at the expense of $|B'|$. MISC, on the other hand, combines the two aspects into a single formulation.

In defining MIST [14], Bryant seemed to be on a similar trail to ours. He even goes so far as to generalize his algorithm for MIST to introduce parameters α and β which can be used to weight the importance placed on trivial versus nontrivial bipartitions. However, while MIST is a step in the right direction, it is still too restrictive relative to our goals. This is because solutions are required to be agreement subtrees, which we will show in Section 4.5 does not always equate to optimality in MISC-$\mathcal{C}_{m-1}$.

4.2.2 How bipartitions change under leaf deletion

We begin by studying the effect that dropping leaves has on a bipartition profile. For any bipartition in the original profile, there are three cases. We illustrate these cases through a simple example, with an original leaf set of a, b, c, d, e, f and with leaves b and e dropped.

1. **merge:** If two bipartitions differ solely in (a subset of) the leaves being dropped, then those bipartitions get merged in the new profile. For example $ac|bdef$ and $abc|def$ merge into $ac|df$ and the ν set for the merged bipartition consists of the union of the two original bipartitions.
2. **disappear:** If dropping the leaves creates a bipartition with an empty side or makes the bipartition trivial, then the bipartition disappears. For example, both $acdf|be$ and $acd|bef$ disappear.
3. **no change:** Otherwise, a bipartition remains unchanged.

An important observation is that, for all $L'' \subseteq L' \subseteq L$, every nontrivial bipartition in $\mathcal{P}|L''$ and in $\mathcal{C}_t(\mathcal{T}|L'')$ arises as a result of a "no change" of a single bipartition or a "merge" of two or more bipartitions in $\mathcal{P}|L'$. Unfortunately this observation does not suggest an efficient algorithm.

4.2.3 Finding subsets of leaves to drop

Given two bipartitions b_1 and b_2 of L, we can easily identify all leaf subsets L' of minimum cardinality such that dropping L' from L merges b_1 and b_2. If we have $b_1 = A|B$ and $b_2 = C|D$, then the *dropset* L' is the smaller of the two following sets (or either set in case they have the same size):

$$(A\Delta C) \cup (B\Delta D) \text{ or } (A\Delta D) \cup (B\Delta C) \tag{4.2}$$

This concept is exploited in Algorithm 1.

Theorem 5. *Algorithm 1 computes the minimum cardinality dropset for any pair of bipartitions of L.*

Proof. That the dropset causes the two partitions to merge is evident. We establish that the dropset has minimum cardinality by contradiction. Consider that there

Algorithm 1 Find minimum cardinality leaf-dropset that renders $b_1 = b_2$

```
Require: two bipartitions on the same leaf set
Ensure: the dropset (or dropsets if there are two)
 1: function BIPARTITION-PAIR-DROPSET($b_1 = A|B$, $b_2 = C|D$)
 2:     $S_0 \leftarrow A\Delta C \cup B\Delta D$
 3:     $S_1 \leftarrow A\Delta D \cup B\Delta C$
 4:     if $|S_0| < |S_1|$ then
 5:         return $[S_0]$
 6:     else if $|S_1| < |S_0$ then
 7:         return $[S_1]$
 8:     else
 9:         return $[S_0,S_1]$
10:     end if
11: end function
```

exists a smaller dropset merging the two bipartitions. Then there is at least one leaf ℓ in the dropset returned by our algorithm that is not in the smaller dropset. This leaf must be on the same side of the partition in both b_1 and b_2, since otherwise our dropset would not merge the two. But our algorithm uses the symmetric difference of these two sides in computing the dropset, so it could not have chosen ℓ, a contradiction. □

Theorem 6. *The cardinalities of the dropsets returned by Algorithm 1 define a metric on the space of bipartitions of L.*

Proof. Three properties characterize a metric: it must be positive definite and symmetric, and it must obey the triangle inequality. The first two properties are trivial in this case. Suppose we have bipartitions b_1, b_2, and b_3; we want to show that the cardinality of the dropset of b_1 and b_3 cannot exceed the sum of the cardinalities of the dropsets of b_1 and b_2 and of b_2 and b_3. Note that removing both of these dropsets from both b_1 and b_3 merges the two bipartitions, thereby establishing an upper bound on the distance between these two bipartitions in our space; but the distance is the size of the dropset of b_1 and b_3, so that the triangle inequality holds. □

4.3 The Algorithm

We describe the algorithm at a conceptual level, leaving a more formal specification to inset text. First, we build the bipartition profile for the given tree set. Next, we compute the dropset for each pair of bipartitions in the profile such that neither bipartition in the pair appears in the consensus tree, but the pair would appear if merged. For each unique dropset we accumulate the list of bipartition pairs yielding that dropset. These last two parts are formalized in Algorithm 2. We then compute the *impact* of each dropset as the number of bipartition pairs giving rise to that dropset minus the size of the dropset itself. This score corresponds roughly to the difference between the number of edges that will be created and the number of leaves that will be lost should that dropset be used. The dropset of largest impact is then used, the profile updated, the impacts updated, and the process repeated until there does not remain any dropset with a nonnegative impact. This greedy overall framework is formalized in Algorithm 3.

The impact measure ignores disappearing edges and dropsets that are subsets of another—the latter because a superset with deceivingly poor score is likely to get chosen in a subsequent round. The overall algorithm is a greedy heuristic, but does well in practice and on hard instances, as we demonstrate in the next two sections.

There remains the issue, as with all leaf-dropping methods, of what to do with

Algorithm 2 Find potential dropsets by examining all pairs in a profile

```
Require: A bipartition profile $\mathcal{P} = (L, B_{\mathcal{T}}, \nu : B_{\mathcal{T}} \to 2^{\mathcal{T}})$
Require: A frequency-only consensus method $\mathcal{C}_t$ with threshold $t$
Ensure: An object mapping dropsets to lists of bipartition pairs
 1: function POTENTIAL-PROFILE-DROPSETS($\mathcal{P}$, $\mathcal{C}_t$)
 2:     $\Gamma \leftarrow \{b \mid b \in B_{\mathcal{T}}$ and $|\nu(b)| \leq t\}$
 3:     for all pairs of bipartitions $b_1$,$b_2$ in $\Gamma$ do
 4:         if $|\nu(b_1) \cup \nu(b_2)| > t$ then
 5:             $L \leftarrow$ BIPARTITION-PAIR-DROPSET$(b_1, b_2)$
 6:             for $d \in L$ do
 7:                 $\delta[d] \leftarrow \delta[d] \cup \{(b_1, b_2)\}$
 8:             end for
 9:         end if
10:     end for
11:     return $\delta$
12: end function
```

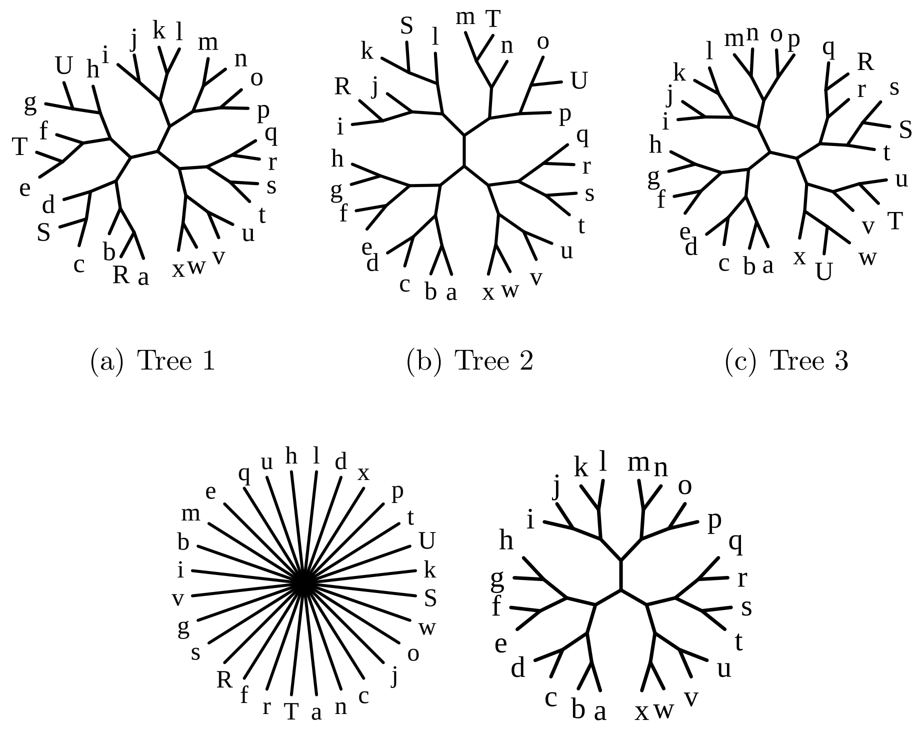
the dropped leaves. The staying power of consensus methods argues for producing a single tree and our method does that. For the rogue taxa, we provide an intriguing strategy that is applicable in some settings in Section 4.4.5.

4.4 Experimental Results

We have implemented our approach as a standalone Python-based prototype. Our current implementation is suitable for datasets of up to a thousand trees on a thousand leaves. Scaling up to 10,000 trees on 10,000 leaves is simply a matter of reimplementing our approach as part of RAxML [76] so as to leverage the efficient bipartition manipulation routines therein. In the following, we present results on artificial datasets constructed to cause difficulties to various consensus methods, followed by results on biological datasets that we used in previous chapter on bootstrapping (Chapter 3). We then discuss implications of our results on the interpretation of phylogenetic reconstruction. We conclude by a smaller study on biological datasets using a slight modification of our algorithm to maximize the number of nontrivial bipartitions in the result.

4.4.1 Difficult instances

Our algorithm is particularly well suited to the so-called “pathological” instances used in the literature to critique the strict or majority consensus. In this section be cover a number of specific instances and instance families which exhibit the inherent limitations of frequency-based consensus methods, the effectiveness of our approach, as well as limitations with our approach.



(a) Tree 1 (b) Tree 2 (c) Tree 3

(d) $\mathcal{C}_{m-1}(\mathcal{T})$ (e) $\mathcal{C}_{m-1}(\mathcal{T}|\{a,\ldots,x\})$

Figure 4.1: A simple, yet starkly contrasting, example (top) for which the strict consensus returns a star tree, but for which our algorithm correctly identifies the rogue taxa and produces a fully resolved tree (bottom).

A First Example

A classic example is an instance where the trees share a common subtree of $n-k$ leaves, but where the remaining k leaves destroy resolution in the consensus.

This example uses the strict consensus. An instance consists of just three trees, defined on the 28-leaf set $\{a,b,\ldots,x,R,S,T,U\}$. The common backbone consists of the 24 taxa $\{a,b,\ldots,x\}$, as illustrated in Figure 4.1(e)). The rogue taxa form the set $\{R,S,T,U\}$; they vary in position on the backbone as indicated in Figures 4.1(a), 4.1(b), and 4.1(c). The strict consensus tree of the three trees is shown in Figure 4.1(d): it is just a star, with no nontrivial bipartition (no internal tree edge) and its relative information content is $I(\mathcal{T}, L, \mathcal{C}_{m-1}) = \frac{28+0}{28+25} = \frac{28}{53} \approx 0.53$. Our algorithm correctly identifies the rogue set, however, so that its strict consensus tree on the remaining set of leaves is the backbone, with an relative information content of $I(\mathcal{T}|\{a, \ldots, x\}, L, \mathcal{C}_{m-1}) = \frac{24+21}{28+25} = \frac{45}{53} \approx 0.85$.

The 1-Cherry Trees

The behavior exhibited in the previous example is not limited to small trees. In this section we introduce a simple, but infinitely sized, family of instances exhibiting similar behavior. We continue, as in the previous section, with the strict consensus method. An instance in this family is fully specified by a parameter k. An instance has $m = 3 \cdot 2^k$ trees, and $n = 3 \cdot 2^{k+1} + 1$ leaves. The common backbone that exists in each tree consists of all but one of the leaves and structurally is a fully balanced binary tree. The remaining leaf, name it R wanders and occurs together with every second leaf X in a bipartition of the form $RX|rest$ (which is colloquially referred to as a *cherry* in phylogenetics, hence the name of the family). See Figure 4.2 for the cherry tree with $k = 1$.

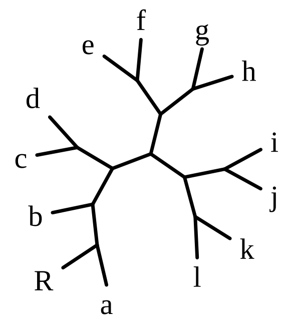
Our algorithm correctly identifies the rogue taxon in all cherry trees. The relative information content of cherry tree k is

$$\frac{3 \cdot 2^{k+1} + 1}{(3 \cdot 2^{k+1} + 1) + (3 \cdot 2^{k+1} + 1 - 3)} = \frac{3 \cdot 2^{k+1} + 1}{2 \cdot 3 \cdot 2^{k+1} - 1}$$

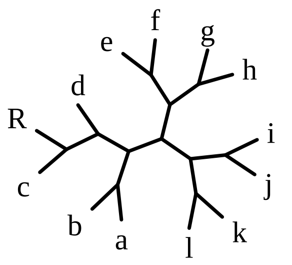
which in the limit $k, n, m \to \infty$ tends toward $\frac{1}{2}$. After rogue identification and elimination by our algorithm, which in this case optimally solves MISC-$\mathcal{C}_{m-1}$, the relative information content is

$$\frac{3 \cdot 2^{k+1} + 3 \cdot 2^{k+1} - 3}{(3 \cdot 2^{k+1} + 1) + (3 \cdot 2^{k+1} + 1 - 3)} = \frac{2 \cdot 3 \cdot 2^{k+1} - 3}{2 \cdot 3 \cdot 2^{k+1} - 1}$$

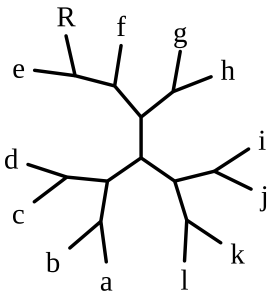
which in the limit $k, n, m \to \infty$ tends toward 1.



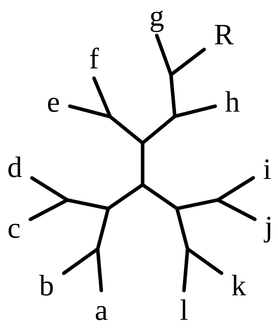
(a) Tree 1



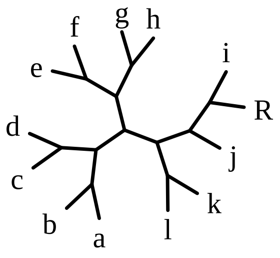
(b) Tree 2



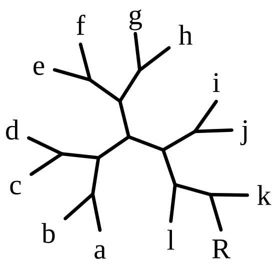
(c) Tree 3



(d) Tree 4



(e) Tree 5



(f) Tree 6

Figure 4.2: The second instance (the first is $k = 0$) of our family of cherry trees. There are $3 \cdot 2^k = 6$ trees on $3 \cdot 2^{k+1} + 1 = 13$ leaves.

4.4.2 The r-Cherry Trees

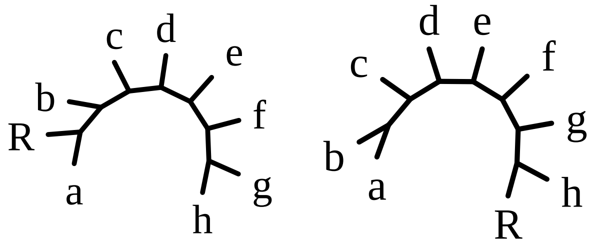
The 1-cherry tree instances are somewhat unsatisfying as pathological instances because a much simpler strategy than our algorithm is sufficient for finding the single rogue taxon (e.g., the polynomial time procedure of dropping each leaf in turn and applying the strict consensus method to assess relative information content of the result). However, it is straightforward to adapt the basic principle behind 1-cherry trees to induce an effectively arbitrary number of rogue taxa. Given k and a desired number of rogue taxa r (where r divides $\frac{n}{2}$ evenly), we take $n = 3 \cdot 2^{k+1} + r$ and $m = \frac{3 \cdot 2^k}{r}$. In the first tree, the rogue taxa are attached as cherries to every second taxon, for a total of $\frac{n}{r}$ taxa. In the second tree, the rogue taxa are attached to the next $\frac{n}{r}$ (while attaching as a cherry to every second) taxa. This pattern continues in each subsequent tree. An instance of this family is actually what we used as our first motivational example for our algorithm (Figure 4.1), which is the $k = 2$'th 4-cherry tree.

4.4.3 The Comb/Caterpillar

Another instance (family) that exhibits extremal behavior with respect to relative information content arises when subjecting the *comb* or *caterpillar* tree (so named by their appearance when drawn) to rogue taxa. The simplest case consists of two trees, on an arbitrary number of taxa, where the rogue taxon occurs at each end of the backbone tree. See Figure 4.3 for the instance of this family with nine taxa. This family of caterpillar instances is also often used to demonstrate a brittleness of the Robinson-Foulds metric. Namely, the RF distance between the two trees is maximum (for the given number of taxa), whereas the trees are clearly nearly identical. Our algorithm performs particularly well on caterpillar (and related) instances because hidden edges in such instances are almost always revealed by merging *pairs* of bipartitions.

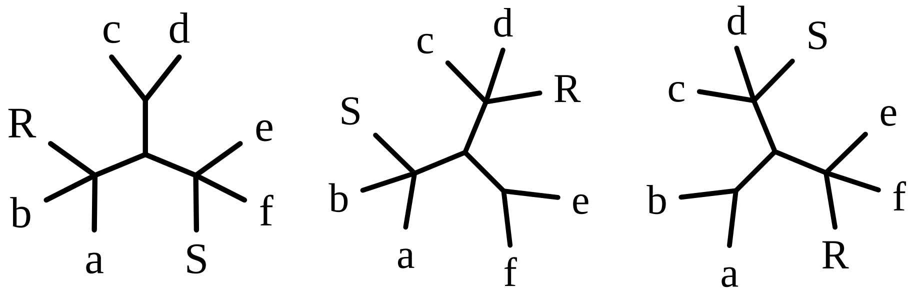
Instance Requiring a 3-way Merge

As was discussed earlier, bipartitions in a restricted consensus tree can be uncovered as the result of merging three or more bipartitions. Since our algorithm only considers bipartition pairs as merging candidates, an instance which only admits improvement via a 3 (or more) way merge will fail to be solved by our algorithm. Figure. 4.4 illustrates such an instance. The relative information content of the non-restricted instance is $\frac{8}{13}$ whereas removing leaves R and S yields a situation where the relative



(a) Tree 1 (b) Tree 2

Figure 4.3: An instance where all of the internal structure of the strict consensus tree is destroyed by a single rogue taxon, even with only two trees in the set.



(a) Tree 1 (b) Tree 2 (c) Tree 3

Figure 4.4: This instance will not be solved correctly by our algorithm. This is because identifying R and S as rogue requires merging three bipartitions.

information content is $\frac{9}{13}$. However, our algorithm will recommend to not drop any leaves.

A note about $\mathcal{C}_{\frac{m}{2}}$

All of the instance presented in this section considered relative information content in light of the strict consensus method ($\mathcal{C}_{m-1}$). However, it is trivial to adapt these instance families into easy/difficult cases for our algorithm when operating under the majority rule ($\mathcal{C}_{\frac{m}{2}}$) consensus method. Namely, for any instance discussed so far with m trees, add an additional $m - 1$ *star* trees (recall that the star tree contains zero nontrivial bipartitions). This simple transformation yields instances that perform identically with respect to relative information content (with $\mathcal{C}_{\frac{m}{2}}$ substituted for $\mathcal{C}_{m-1}$).

4.4.4 Results on biological data

We applied our method to half of the datasets used in the last chapter on bootstrapping methods (Chapter 3 and available at `http://lcbb.epfl.ch/BS.tar.bz2`.) There are 10 datasets of single-gene and multi-gene DNA sequences, with anywhere from 125 to 994 taxa. For each dataset we generated 1,000 bootstrap replicates and applied our algorithm to the resulting trees using both $\mathcal{C}_{\frac{m}{2}}$ and $\mathcal{C}_{m-1}$. Our algorithm found rather diverse dropset sizes across the 10 datasets. The results are depicted in Figure 4.5, where a quartet of histogram bars are shown for each dataset with a nonempty dropset. The first histogram bar (a negative quantity) denotes how many leaves were dropped, while the second bar (a positive quantity) denotes how many nontrivial bipartitions were uncovered. The third bar is the sum of the first two, simply depicting the net (non-normalized) contribution to relative information content. The final bar is discussed in Section 4.4.5.

4.4.5 Biological interpretation

Maximum likelihood phylogenetic analyses are typically conducted in two steps. First the reconstruction proper is performed, yielding a "best tree." Then a number of bootstrap replicate trees are generated, say 500 of them; for each bipartition b in the best tree, its support value is calculated as a normalized count of the number of

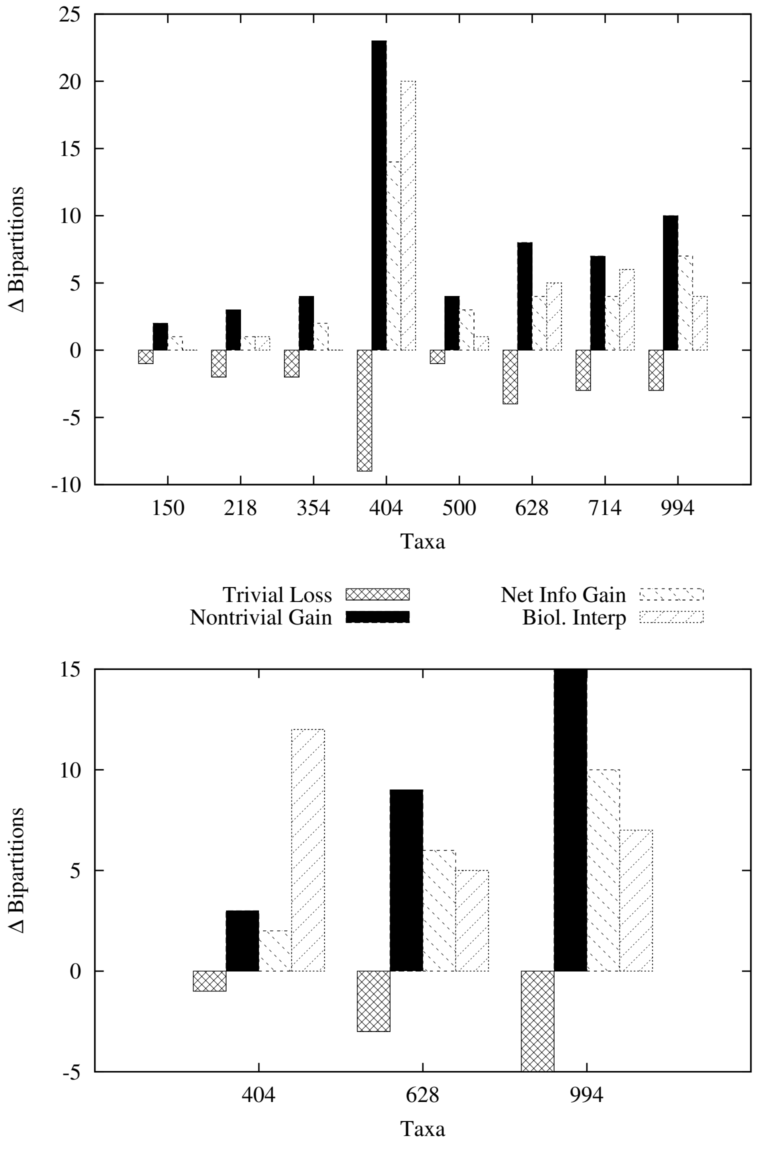


Figure 4.5: The performance of Algorithm 3 in terms of how much "hidden" consensus is uncovered in biological data sets. The top plot is for majority consensus, the bottom for strict consensus. The tree sets each consist of 1,000 bootstrap replicates generated by the RAxML 7.2.6 Rapid Bootstrap Algorithm.

replicates in which b appears. Researchers tend to consider edges with support lower than 75% as unreliable [30].

If, however, rogue taxa are at work in the replicate set, the support values for certain bipartitions can be deceivingly depressed. To remedy this problem, we propose that Algorithm 3 be applied to the replicate set in order to identify rogue taxa. If a dropset of nonzero size is found, this dropset is then removed from each tree in the replicate set. Finally, the (modified) support value is calculated as a normalized count of the replicates in which b' appears such that, if we have $b = A|B$, then, without loss of generality, we have $b' = A' \subseteq A|B' \subseteq B$. In this way, support values in the "best tree" are less susceptible to the deceiving influence of rogue taxa. This approach offers one possible solution to the data display problem of leaf-dropping methods. We still return a single tree on the original leaf set (the "best tree" as reconstructed by an ML method), but support values for individual bipartitions more accurately reflect the underlying replicate data.

In our datasets, recomputing support values as suggested above yields very intriguing and promising results. All but two of the identified dropsets succeeded in pushing at least one previously hidden edge in the "best tree" over the 75% threshold. The number of edges uncovered by this application of our technique is displayed in the fourth histogram bar in Figures 4.5(a) and 4.5(b). In the dataset with 404 taxa, 20 edges were uncovered in this manner, pointing to a need for reevaluation of the phylogeny.

4.4.6 Increasing resolution

Our algorithm can easily be modified to maximize nontrivial bipartitions, that is, to remove taxa so as to increase resolution. With such a setting, our algorithm loosely matches the goal of Cranston and Rannala [20], so we analyzed the same dataset with our technique to compare our results to theirs. The data set consists of 85 species of Canformia Carnivora [33]. We obtained the sequence data from TreeBASE (`http://www.treebase.org`, Study Accession # S1532) and reconstructed a tree using RAxML-7.2.6 [76] under the GTRCAT approximation. Additionally, RAxML was used to generate 350 bootstrap replicates (the number chosen by RAxML's bootstopping algorithm). Analyzing these 350 trees with our modified Algorithm 3 and using majority consensus generated fully resolved trees with 50 to 55 taxa, a value consistent with the size of the agreement subtrees observed by Cranston and Rannala [20].

4.5 Complexity Theoretic Aspects of MISC-$\mathcal{C}$

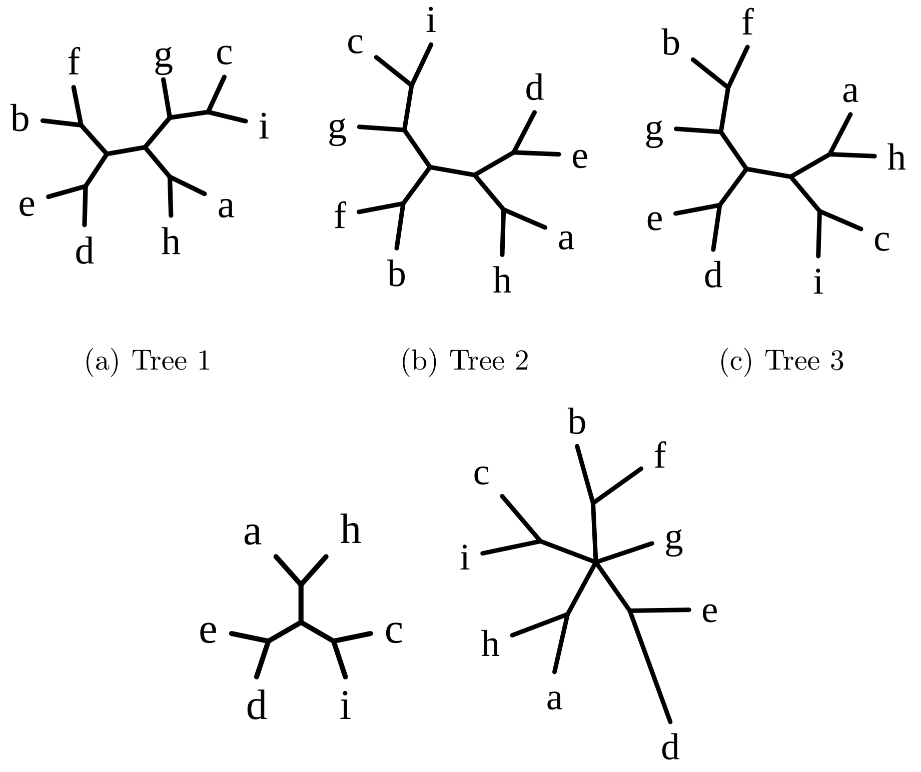
Rather than addressing the complexity of MISC directly, our line of inquiry to this end began with an observation that for the strict consensus method ($\mathcal{C}_{m-1}$) it is the case that the Maximum Agreement Subtree (MAST) yields an interesting lower bound for MISC-$\mathcal{C}_{m-1}$. We explore this observation more rigorously in the subsequent section, but intuitively MAST is an interesting lower bound because it often maximizes the $|B'|$ term of relative information content at the expense of the $|L'|$ term. This perspective yields another way to view consensus methods, namely that they too represent lower bounds for relative information content which instead maximize $|L'|$ at the expense of $|B'|$. Indeed, this is another part of the motivation for our relative information content measure – precisely to address this tradeoff.

Since in this chapter we have focused on two of the most popular consensus methods, strict and majority rules, we also investigate whether the MAST lower bound for MISC-$\mathcal{C}_{m-1}$ has an analogue in the MISC-$\mathcal{C}_{\frac{m}{2}}$ case. This subject comprises the work presented in the latter half of the subsequent section and most of Section 4.5.2, where we show that the problem seems considerably more difficult (in the theoretic sense) than in the strict setting.

While the lower bounds mentioned thus far are interesting in that they often maximize nontrivial bipartitions, also observe that *any* leaf subset yields a lower bound on relative information content. Simply restrict the original tree set to the leaf subset, run the consensus method, and count the bipartitions. Other lower bounds, which really just amount to polynomial time heuristics for MISC, that are valid and may merit consideration are: a) removing leaf subsets of size less than or equal to some constant and b) removing leaf subsets that appear as a clade in the original leaf set. However the greedy heuristic presented in the previous sections likely outperforms these proposals in most cases. On the other hand, (b) may have strong applicability in situations where a wandering clade is causing problems, as it is highly likely that the wandering clade appears together, on its own, it at least one of the input trees.

4.5.1 MAST, MIST, and MR-MIST?

While MAST often yields solutions that maximize the $|B'|$ term in relative information content, this is not always the case. In Figure 4.6 we present an instance where dropping no leaves yields more nontrivial bipartitions (in the strict consensus tree) than can be found in a MAST. Interestingly, this counterexample establishes the same outcome for MIST. This is counterintuitive because MIST explicitly maximizes the number of nontrivial bipartitions in the agreement subtree. The problem lies in the constraint that a solution to MIST must be an agreement subtree, which turns out to be overly restrictive even for MISC-$\mathcal{C}_{m-1}$. To see that the example in Figure 4.6 represents a counterexample (to our original conjecture that MAST always maximizes nontrivial bipartitions in MISC-$\mathcal{C}_{m-1}$) for both MAST and MIST, observe in this particular instance that any MAST is also a MIST. This is because all of the input trees are fully resolved, and thus all agreement subtrees will be fully resolved. Thus, if there were a solution containing an extra trivial (resp. nontrivial) bipartition, then such a solution would be required to have an extra nontrivial (resp.



(a) Tree 1 (b) Tree 2 (c) Tree 3

(d) MAST (e) Strict Consensus

Figure 4.6: MAST as a lower bound (to maximize $|B'|$) counterexample

trivial) bipartition.

A related example (though on rooted trees) was given by Swofford [84], which we reproduce in Figure 4.7. This instance is cited by Bryant as the reason he investigated the MIST problem. In this example there is a leaf subset a, b, c, d, e, f which occurs as a *star*, i.e., all six leaves hang off of the same internal node, such that there exists no nontrivial bipartition separating any of the leaves in this subset. As this pattern is shared by both trees, it is forced into the MAST (due to its cardinality), which in turn masks another common feature between the two trees $(g, (h, (i, (j, k))))$ which has only one fewer leaf but much more informative internal structure. Since MAST is not discriminating regarding the internal structure that it throws away, it performs

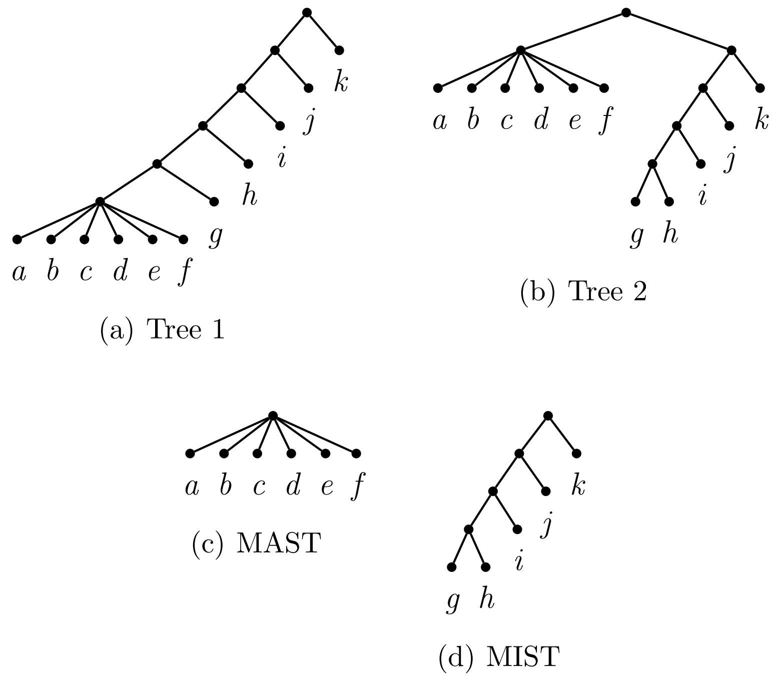


Figure 4.7: Swofford's instance where an informationless but large feature (a, b, c, d, e, f) that is shared between the two trees masks another slightly smaller but more informative feature $(g, (h, (i, (j, k))))$ in the MAST, whereas MIST returns the preferred feature.

poorly on instances of this variety.

To better understand the actual relationship between MAST, MIST, and MISC-$\mathcal{C}_{m-1}$ we offer the following lemma.

Lemma 1. *All trees in $\mathcal{T}$ are equal to each other (i.e., agreement (sub)trees) if and only if $C_{m-1}(\mathcal{T})$ is equal to a tree $T^* = (L, B) \in \mathcal{T}$ where $|B|$ is maximum (over $\mathcal{T}$).*

Proof. We prove each direction in the iff separately.

$\rightarrow$ Since the strict consensus tree is equal to T^*, then all of the other trees in the set must have at least have all of the bipartitions in T^*. Since $|B|$ is maximum, then no other trees have more nontrivial bipartitions, such that all of the trees in the set are forced to have the same set of bipartitions.

$\leftarrow$ This direction is trivial. Specifically, since all trees are equal, it follows that the strict consensus tree of the set is equal to each tree in the set, and that the number of bipartitions is constant across the set (and thus T^* can legitimately be any tree in the set).

□

Corollary 3. *In instances of MISC-$\mathcal{C}_{m-1}$ having optimal solution L^* – if the consensus tree of $\mathcal{T}|L^*$ is equal to the most resolved tree in $\mathcal{T}|L^*$, then the instance can be optimally solved by taking the MIST.*

This relationship between MISC-$\mathcal{C}_{m-1}$ and MAST/MIST piqued our curiosity to investigate whether there is a naturally phrased MAST-like problem that lower bounds MISC-$\mathcal{C}_{\frac{m}{2}}$ in a similar manner. Unfortunately, this seems not to be the case.

In fact, a MAST-like phrasing seems lost as soon as the strictness of strict consensus is relaxed at all. To illustrate this we consider the relationship between trees in $\mathcal{T}|L'$ where L' is a leaf subset maximizing the number of nontrivial bipartitions in $\mathcal{C}_{m-2}$ (i.e., one tree less–strict than strict). Consider the bipartitions in $\mathcal{C}(\mathcal{T}|L')$, and the trees which support them. For the purpose of illustration, there are two extreme cases. In the first, each bipartition is missing in a distinct tree. This case has a seemingly desirable property (P1) that for any pair of trees in $\mathcal{T}|L$, each tree has at most one bipartition that is not in the other. In the other extreme case, all trees but one are identical and the final tree contains no bipartitions in common with the other $m-1$ trees. This case also has a seemingly desirable property (P2) that L is an agreement subtree when ignoring the single discordant tree.

P1 would suggest phrasing a MAST-like optimization problem where no two trees have *Robinson-Foulds distance* greater than some threshold. P2 would suggest phrasing a MAST-like problem where one attempts to add leaves to a *skeleton* tree consisting of a MAST obtained by ignoring some subset of trees in $\mathcal{T}|L$.

However consider P2 with respect to the first extreme case. Specifically, observe that there are no two trees in $\mathcal{T}|L'$ that agree on all bipartitions, and thus no notable skeleton MAST. Also consider P1 with respect to the second extreme case. Specifically, observe that the *Robinson-Foulds distance* between the discordant tree and any other tree in the set is maximum. We conclude then, anecdotally at least, that there is not an obvious MAST-like phrasing of the problem that tends to maximize the $|B'|$ term (of relative information content) in a setting more relaxed than strict consensus.

4.5.2 MFRC, MFRC-$\mathcal{C}_{\frac{m}{2}}$ and MMDS

The tractability of MAST (on three or more trees) arises when bounding the *degree* of one of the input trees. This observation along with the conclusion reached in the previous section prompted us to pose the following (more restrictive) problem, in hopes of perhaps finding suitably constrained subsets of MISC-$\mathcal{C}_{\frac{m}{2}}$ instances that are amenable to a MAST/MIST-like formulation/approach.

MAXIMUM FULLY RESOLVED CONSENSUS (MFRC-$\mathcal{C}$)

INSTANCE: A set of phylogenetic trees $\mathcal{T}$ defined on a common leaf set L, and a consensus method $\mathcal{C}$.

SOLUTION: A leaf subset L' such that the tree $\mathcal{C}(\mathcal{T}|L')$ is fully resolved.

MEASURE: The cardinality of L'.

This problem is related to MAST, at the very least, in the manner indicated by the following lemma. Also observe that as noted earlier, when there is at least one fully resolved tree in the input, then MAST and MIST are equivalent problems. As such, the following lemma applies to MIST as well. To our knowledge, we are the first to view MAST as a strict consensus problem.

Lemma 2. *If $\mathcal{T}$ contains at least one tree with maximum degree 2, then MFRC-$\mathcal{C}_{m-1}$ is equivalent to MAST.*

Proof. Denote an optimal solution to MAST as L^*_{MAST} and an optimal solution to MFRC-$\mathcal{C}_{m-1}$ as $L^*_{MFRC-\mathcal{C}_{m-1}}$. We show in turn that $|L^*_{MAST}| \geq |L^*_{MFRC-\mathcal{C}_{m-1}}|$ and $|L^*_{MAST}| \leq |L^*_{MFRC-\mathcal{C}_{m-1}}|$ thereby implying $|L^*_{MFRC-\mathcal{C}_{m-1}}| = |L^*_{MAST}|$.

- $\geq$ – Since the consensus tree $\mathcal{C}_{m-1}(\mathcal{T}|L^*_{MFRC-\mathcal{C}_{m-1}})$ is fully resolved, it must be the case that $\mathcal{T}|L^*_{MFRC-\mathcal{C}_{m-1}}$ represents a set of agreement subtrees. Since MAST maximizes the size of agreement subtrees, $|L^*_{MAST}|$ will always meet or exceed $|L^*_{MFRC-\mathcal{C}_{m-1}}|$
- $\leq$ – Since at least one of the trees in $\mathcal{T}$ is fully resolved, all agreement subtrees will be fully resolved. Further, the strict consensus tree of a set of agreement subtrees is equal to the agreement subtree itself. Thus the strict consensus of the MAST will be fully resolved. Since MFRC maximizes the size of the fully resolved consensus tree, $|L^*_{MFRC-\mathcal{C}_{m-1}}|$ will always meet or exceed $|L^*_{MAST}|$.

□

When there does not exist a fully resolved tree in the input the complexity of MFRC-$\mathcal{C}_{m-1}$ is open, although we suspect that its complexity resembles that of MAST, and likely tractable when limited to bounded degree trees.

However, we now move on to MFRC-$\mathcal{C}_{\frac{m}{2}}$, as generalization of *some* MAST-like problem into sub–strict setting has been our goal from the beginning of this section.

Our most fruitful attempt in solving MFRC-$\mathcal{C}_{\frac{m}{2}}$ has come in trying to adapt the MAST algorithm of Amir and Keselman [5]. Note that their algorithm operates on a set of rooted trees, and we follow suit by generalizing in the rooted setting. This is not an obstacle, however, as rooted MAST algorithms apply to unrooted instances without much complication, as the unrooted MAST is simply the maximum sized rooted MAST, over $|L|$ possible rootings (at a common leaf) of the rooted case.

Because we have not used rooted trees thus far in this document, a bit of terminology is in order. Whereas bipartitions (internal edges) are the key structural property of unrooted trees, *internal nodes* are the pertinent structure in rooted trees. Moreover, the set of leaves descendant from an internal node is referred to as its *clade* or *cluster*. Whereas an unrooted tree can be unambiguously represented by its bipartitions, a rooted tree can be unambiguously represented by its clades. We denote the clade descendant from an internal node ℓ as $L(\ell)$. We also use the well–known concept of a *least (or lowest) common ancestor* (LCA) of two leaves a and b, which is the internal node ℓ satisfying two properties: P1) ℓ has both a and b as descendants and P2) ℓ has no other descendants that satisfy P1. In a tree T, the least common ancestor of leaves a and b will be denoted $lca^T(a, b)$.

One of the keys to the approach of Amir and Keselman is the concept of *maximal decomposable sets* (MDS). These are leaf (sub)sets $A = A_L \cup A_R$ that satisfy two conditions. One, there is an internal node in each of the original (but restricted) trees that have child clades A_L and A_R, this is the decomposable part. Two, there isn't a larger $A' = A'_L \cup A'_R$ where $A_L \subseteq A'_L$, $A_R \subseteq A'_R$, and $A \subseteq A'$, this is the maximality. In Amir and Keselman's paper, MMDS are defined as $A = A_1 \cup A_2 \cup \ldots \cup A_k$. This is because they define things in terms of trees having 'degree bounded by k.' We can collapse to $k = 2$ because of our fully-resolved solution constraint.

We view the first condition (decomposable) in a slightly different, though equivalent, light. Specifically, when restricting the tree set to A, the strict consensus tree (of the restricted set) has as sibling top-level clades A_L and A_R, and this view motivates the following definition. We define *Maximal Majority Decomposable Sets (MMDS)* $A = A_L \cup A_R$ such that restricting the input tree set to A yields a majority rules consensus tree with top-level clades A_L and A_R, and there is not an $A' = A'_L \cup A'_R$ where $A_L \subseteq A'_L$, $A_R \subseteq A'_R$, and $A \subset A'$.

We observe that MMDS' do indeed seem relevant in MFRC-$\mathcal{C}_{\frac{m}{2}}$, as is captured by the following observation.

Observation 1. *For all internal nodes ℓ of a (fully resolved) majority rules tree of $\mathcal{T}|L'$ having immediate children ℓ_L and ℓ_R – $\forall x \in L(\ell_L)$ and $\forall y \in L(\ell_R)$, there is an MMDS $A = A_L \cup A_R$ where $L(\ell_L) \subseteq A_L$ and $L(\ell_R) \subseteq A_R$ which can be found by considering only subtrees rooted at $lca^T(x, y)$ (in each tree of the original instance).*

This observation implies that solutions to MFRC-$\mathcal{C}_{\frac{m}{2}}$ will be composed of (subsets) of MMDS'. Unfortunately, we have discovered a reduction to MMDS that is very suspicious in the sense that the problem that reduces to MMDS is likely $\mathcal{NP}$-hard.

The reduction is from a problem on p-intersection graphs, and is rather straightforward. A p-intersection graph $G = (V, E)$ associates each node with a subset of a finite set S, and an edge exists between any pair of nodes whose intersection has size exceeding p (for which here we consider $p = \frac{|S|}{2}$). The p-intersection graph problem from which we reduce is the following: what is the maximum cardinality vertex subset such that the intersection of overlaps between all pairs in the subset exceeds a given threshold? Although any feasible solution will appear in the p-intersection graph as a clique, not all cliques represent feasible solutions. In the reduction, each element in S gives rise to a distinct tree. Every vertex $v \in V$ gives rise to a unique leaf. Go through each $s \in S$ in turn, and accumulate the leaves corresponding to any vertex containing s. That set of leaves becomes a top-level clade in the tree corresponding to s. The rest of the leaves become the other top-level clade in the

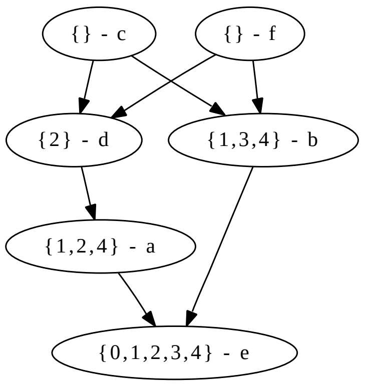


Figure 4.8: The original instance

tree for s. Figures 4.8, 4.9 and 4.10 illustrate the transformation. Figure 4.8 shows the original instance. Figure 4.9 shows one possible (multifurcating) transformed instance, whereas figure 4.10 shows another possible (bifurcating) transformed instance. The reason for the two transformed instances is to, on one hand, illustrate the essence of the transformation (Figure 4.9, where each tree has two clades), but also to show that resolving the two top-level clades (called for in the reduction) arbitrarily such that they are fully resolved doesn't invalidate the reduction. As such, the reduction establishes that bounding the degree of input trees would not invalidate the reduction.

It is straightforward to verify that for any MMDS $A = A_L \cup A_R$ (with the additional constraint that a leaf having S as its set in the original instance is in A_L and a leaf having the empty set as its set in the original instance is in A_R), the sets corresponding to the leaves of A_L are a feasible solution for the original instance. Any such MMDS with maximum cardinality A_L corresponds to an optimal solution to the original instance. In the example instance here, the MMDS $\{b, e\} \cup \{c, d, f\}$ gives rise to the solution in the original instance of $\{1, 3, 4\} \cap \{0, 1, 2, 3, 4\}$.

.

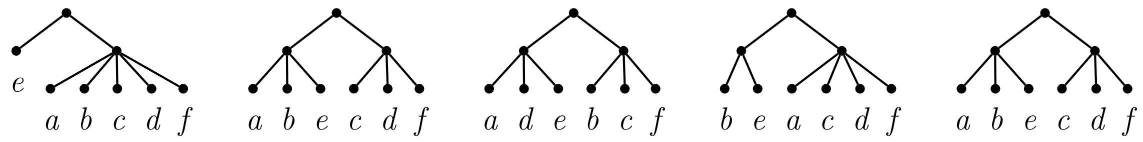


Figure 4.9: A (multifurcating) transformed instance

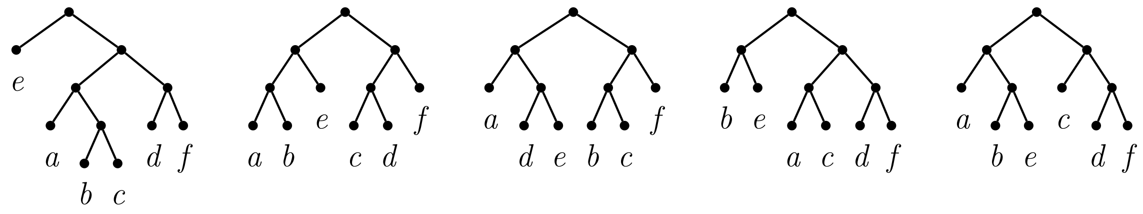


Figure 4.10: An alternative (bifurcating) transformed instance

Algorithm 3 Our top level iterative heuristic for finding dropsets

```
Require: A tree set $\mathcal{T}$
Require: A frequency-only consensus method $C$ with threshold $t$
Ensure: A set of leaves to drop, composed of the union of dropsets
 1: function SELECT-AND-REMOVE-DROPSETS($\mathcal{T}$)
 2:     $d^* \leftarrow d_{greedy} \leftarrow \emptyset$
 3:     repeat
 4:         $\mathcal{P} \leftarrow$ BUILD-BIPARTITION-PROFILE($\mathcal{T}|(L - d^*)$)
 5:         $\delta \leftarrow$ POTENTIAL-PROFILE-DROPSETS($\mathcal{P}, \mathcal{C}_t$)
 6:         $maximpact = 0$
 7:         $d_{greedy} = \emptyset$
 8:         for all $d \in \delta$'s domain do
 9:             if $|d| - |\delta[d]| \geq maximpact$ then
10:                 $d_{greedy} = d$
11:                 $maximpact = |d| - |\delta[d]|$
12:             end if
13:         end for
14:         $d^* = d^* \cup d_{greedy}$
15:     until $d_{greedy} = \emptyset$
16:     return $d^*$
17: end function
```

Chapter 5

Conclusions

Like every other creature on the face of the earth, Godfrey was, by birthright, a stupendous badass, albeit in the somewhat narrow technical sense that he could trace his ancestry back up a long line of slightly less highly evolved stupendous badasses to that first self-replicating gizmo—which, given the number and variety of its descendants, might justifiably be described as the most stupendous badass of all time. Everyone and everything that wasn't a stupendous badass was dead.

- Neal Stephenson, *Cryptonomicon*

5.1 Robinson-Foulds Computations

As computational biologists everywhere increasingly turn to phylogenetic computations to further their understanding of genomic, proteomic, and metabolomic data, and do so on larger and larger datasets, a fast computational method to compare large collections of trees will be required to support interactive analyses.

We used an embedding in high-dimensional space and techniques for computing vector norms from high-dimensional geometry to design the first sublinear-time approximation scheme to compute Robinson-Foulds distances between pairs of trees. We implemented our algorithm and provided experimental support for its computational advantages. We also resolved an open issue from the preliminary version of the paper upon which this material is based [65] by presenting an efficient procedure for embedding trees. Thus our algorithm not only outperforms repeated applications of Day's algorithm for large collections of trees, it also achieves similarly spectacular speedups for smaller collections of very large trees. In the process, we presented a unified view of algorithms that rely on lists of vertices and bipartitions, a view that allowed us to improve the speed of Day's algorithm as well.

The new implementation of our algorithm, `FastRF`, used to run all of our experiments is open-source and available for download from `compbio.unm.edu`

5.2 Bootstopping

We have conducted the first large-scale empirical ML-based study of the convergence properties of bootstrapping, using biological datasets that cover a wide range of input alignment sizes and a broad variety of organisms and genes. In addition, we have developed and assessed two bootstopping criteria that can be computed at run time and do not rely on externally provided reference trees to determine convergence. The criteria have been designed so as to capture a stopping point that provides sufficient accuracy for an unambiguous biological interpretation of the resulting consensus trees or best-known ML trees with support values. The correlation between bootstopped support values and support values from 10,000 reference trees exceeds 99.5% in all cases, while the relative weighted tree distance (used with the WC criterion) is smaller than the specified threshold value in all cases. We conclude that the WC criterion yields better performance and higher accuracy than FC while it correlates very well with the mean error of support values on the best-scoring tree. We advocate the use of WC over FC because it only takes into account the BS support of "important" bipartitions which are subject to biological interpretation. We have also shown that the number of replicates required to achieve a certain level of accuracy is highly dataset-dependent for real data, so that, by using our criteria, an investigator need only compute as many replicates as necessary, thus avoiding the waste of scarce computational resources, in particular for future large-scale phylogenomic analyses. Finally, we have fully integrated the criteria into the current release of RAxML and provided a detailed description and study of implementation issues associated to the stopping functions. Our production level implementation yields speedups of the stopping function up to a factor of 7 on datasets with thousands of taxa.

Since the preliminary version of the paper upon which this material is based, we have completed the full integration of the advanced hashing techniques into RAxML 7.2.6. We have also parallelized the hash table operations using Pthreads and vectorized operations on bit vectors by using SSE3 instructions [1]. Finally, we plan to devise ways to dynamically adapt the spacing of FC/WC criteria (which is currently fixed at 50) to the convergence speed of the BS replicates, i.e., use a more sparse spacing for the initial phase and a denser spacing for the later phase of the BS search.

5.3 Uncovering Hidden Consensus

We have presented a novel framework to define rogue taxa so as to maximize the relative information present in a consensus tree computed after removing these rogue taxa. This framework defines a bicriterion problem, MISC, that is the first to balance explicitly loss of taxa with gain in resolution in a setting other than agreement subtrees. We have also provided an effective greedy heuristic to find a good set of such rogue taxa. This algorithm was tested on both pathological cases from the literature and a variety of biological data. The changes in the consensus tree can be parlayed into more accurate bootstrap scores, which in turn can lead to the reevaluation of phylogenetic trees, as we showed on our biological datasets.

Further algorithmic work includes a characterization of the computational complexity of the MISC problem, as well as improved algorithms for it, including approximation algorithms with known performance guarantees. Generalizing our approach to support consensus methods other than frequency-based methods is another algorithmic problem worth investigating. Finally, there is certainly room to extend and apply our techniques in different domains, most notably in Bayesian phylogenetics (as suggested in Section 4.4.6) and for the subtree mergers used in the Disk-Covering Methods (as suggested in [67]). On the bioinformatics side, our preliminary findings indicate that existing phylogenies can be significantly refined by applying our approach to the recomputation of bootstrap support.

References

[1] A. J. Aberer, N. D. Pattengale, and A. Stamatakis. Parallel computation of phylogenetic consensus trees. In *Proc. International Conference on Computational Science (ICCS) 2010*, pages Accepted, To Appear, 2010.

[2] D. Achlioptas. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *J. Comput. Syst. Sci.*, 66:671–687, 2003.

[3] B. Allen and M. Steel. Subtree transfer operations and their induced metrics on evolutionary trees. *Annals of Combinatorics*, 5(1):1–15, 2001.

[4] N. Amenta, F. Clarke, and K. St. John. A linear-time majority tree algorithm. *Lecture Notes in Computer Science*, pages 216–227, 2003.

[5] A. Amir and D. Keselman. Maximum agreement subtree in a set of evolutionary trees. *SIAM Journal on Computing*, 26:758–769, 1994.

[6] D. Andrews and M. Buchinsky. On the Number of Bootstrap Repetitions for Bootstrap Standard Errors, Confidence Intervals, and Tests. *Cowles Foundation Paper 1141R*, 1997.

[7] D. Andrews and M. Buchinsky. A Three-Step Method for Choosing the Number of Bootstrap Repetitions. *Econometrica*, 68(1):23–51, 2000.

[8] D. Andrews and M. Buchinsky. Evaluation of a three-step method for choosing the number of bootstrap repetitions. *J. of Econometrics*, 103(1-2):345–386, 2001.

[9] D. Andrews and M. Buchinsky. On The Number of Bootstrap Repetitions for BCa Confidence Intervals. *Econometric Theory*, 18(4):962–984, 2002.

[10] J. Archie, W. H. Day, W. Maddison, C. Meacham, F. J. Rohlf, D. Swofford, and J. Felsenstein. The newick tree format. http://evolution.genetics.washington.edu/phylip/newicktree.html.

[11] H. Bandelt and A. Dress. Split decomposition: A new and useful approach to phylogenetic analysis of distance data. *Molecular Phylogenetics and Evolution*, 1(3):242–252, September 1992.

[12] O. Bininda-Edmonds, editor. *Phylogenetic Supertrees: Combining information to reveal the Tree of Life*. Kluwer Academic Publishers, 2004.

[13] S. Brooks and A. Gelman. General Methods for Monitoring Convergence of Iterative Simulations. *J. of Computational and Graphical Statistics*, 7(4):434–455, 1998.

[14] D. Bryant. *Hunting for trees, building trees and comparing trees: theory and method in phylogenetic analysis*. PhD thesis, University of Canterbury, 1997.

[15] D. Bryant. A classification of consensus methods for phylogenetics. In *Bioconsensus*, volume 61 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 163–184. American Math. Soc. Press, 2002.

[16] D. Bryant. The splits in the neighborhood of a tree. *Annals of Combinatorics*, 8(1):1–11, 2004.

[17] D. Bryant and V. Moulton. Neighbor-Net: An Agglomerative Method for the Construction of Phylogenetic Networks. *Mol Biol Evol*, 21(2):255–265, 2004.

[18] J. Carter and M. Wegman. Universal classes of hash functions (Extended Abstract). In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 106–112. ACM New York, NY, USA, 1977.

[19] B. Chor and T. Tuller. Maximum likelihood of evolutionary trees: hardness and approximation. *Bioinformatics*, 21(1):97–106, 2005.

[20] K. A. Cranston and B. Rannala. Summarizing a Posterior Distribution of Trees Using Agreement Subtrees. *Syst Biol*, 56(4):578–590, 2007.

[21] B. DasGupta, X. He, T. Jiang, M. Li, J. Tromp, and L. Zhang. On computing the nearest neighbor interchange distance. In *Proc. DIMACS Workshop on Discrete Problems with Medical Applications*, volume 55 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 125–143. American Math. Soc. Press, 2000.

[22] A. Davidson and D. Hinkley. *Bootstrap Methods and Their Application.* Cambridge University, 2003.

[23] R. Davidson and J. MacKinnon. Bootstrap tests: how many bootstraps? *Econometric Reviews*, 19(1):55–68, 2000.

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

[25] B. Efron and R. Tibshirani. *An introduction to the bootstrap.* Chapman & Hall New York, 1993.

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

[27] J. Felsenstein. Evolutionary trees from DNA sequences: a maximum likelihood approach. *J. Mol. Evol.*, 17:368–376, 1981.

[28] J. Felsenstein. Confidence Limits on Phylogenies: An Approach Using the Bootstrap. *Evolution*, 39(4):783–791, 1985.

[29] J. Felsenstein. Phylip - phylogeny inference package (version 3.2). *Cladistics*, 5:164–166, 1989.

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

[31] W. Fitch and E. Margoliash. Construction of phylogenetic trees. *Science*, 155(3760):279–284, 1967.

[32] L. Foulds and R. Graham. The Steiner problem in phylogeny is NP-complete. *Advances in Applied Mathematics*, 3(43-49):299, 1982.

[33] T. L. Fulton and C. Strobeck. Molecular phylogeny of the arctoidea (carnivora): Effect of missing data on supertree and supermatrix analyses of multiple gene data sets. *Molecular Phylogenetics and Evolution*, 41(1):165–181, October 2006.

[34] O. Gauthier and F.-J. Lapointe. Seeing the Trees for the Network: Consensus, Information Content, and Superphylogenies. *Syst Biol*, 56(2):345–355, 2007.

[35] A. Gelman and D. Rubin. Inference from iterative simulation using multiple sequences. *Stat. Sci.*, 7:457–511, 1992.

[36] N. Goldman, J. P. Anderson, and A. G. Rodrigo. Likelihood-Based Tests of Topologies in Phylogenetics. *Syst Biol*, 49(4):652–670, 2000.

[37] P. Goloboff. Analyzing large data sets in reasonable times: solution for composite optima. *Cladistics*, 15:415–428, 1999.

[38] P. A. Goloboff, S. A. Catalano, J. M. Mirande, C. A. Szumik, J. S. Arias, M. Källersjö, and J. S. Farris. Phylogenetic analysis of 73060 taxa corroborates major eukaryotic groups. *Cladistics*, 25:1–20, 2009.

[39] G. Grimm, S. Renner, A. Stamatakis, and V. Hemleben. A Nuclear Ribosomal DNA Phylogeny of Acer Inferred with Maximum Likelihood, Splits Graphs, and Motif Analyses of 606 Sequences. *Evol. Bioinf. Online*, 2:279–294, 2006.

[40] S. Guindon and O. Gascuel. A Simple, Fast, and Accurate Algorithm to Estimate Large Phylogenies by Maximum Likelihood. *Syst. Biol.*, 52(5):696–704, 2003.

[41] W. Guo and S. Peddada. Adaptive Choice of the Number of Bootstrap Samples in Large Scale Multiple Testing. *Stat. Appls. in Genetics and Mol. Biol.*, 7(1), 2008.

[42] D. Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, 21(1):19–28, 1991.

[43] P. Hall. On the Number of Bootstrap Simulations Required to Construct a Confidence Interval. *The Annals of Statistics*, 14(4):1453–1462, 1986.

[44] S. Hedges. The number of replications needed for accurate estimation of the bootstrap P value in phylogenetic studies. *Mol. Biol. Evol.*, 9(2):366–369, 1992.

[45] A. Hejnol, M. Obst, A. Stamatakis, M. Ott, G. W. Rouse, G. D. Edgecombe, P. Martinez, J. Bagu, X. Bailly, U. Jondelius, M. Wiens, W. E. G. Mller, E. Seaver, W. C. Wheeler, M. Q. Martindale, G. Giribet, and C. W. Dunn. Assessing the root of bilaterian animals with scalable phylogenomic methods. *Proceedings of the Royal Society B: Biological Sciences*, 276(1677):4261–4270, 2009.

[46] D. Hillis and J. Bull. An empirical test of bootstrapping as a method for assessing confidence in phylogenetic analysis. *Systematic Biology*, 42(2):182, 1993.

[47] S. Holmes. Bootstrapping phylogenetic trees: Theory and methods. *Statistical Science*, 18(2):241–255, 2003.

[48] D. Huson. SplitsTree: analyzing and visualizing evolutionary data. *Bioinformatics*, 14(1):68–73, 1998.

[49] P. Indyk. Algorithmic applications of low-distortion geometric embeddings. In *Proc. 42nd IEEE Symp. Foundations of Comput. Sci. (FOCS'01)*, pages 10–33. IEEE Computer Society, 2001.

[50] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. 30th ACM Symp. Theory of Comput. (STOC'98)*, pages 604–613. ACM Press, 1998.

[51] L. Jermiin, G. Olsen, K. Mengerson, and S. Easteal. Majority-rule consensus of phylogenetic trees obtained by maximum-likelihood analysis. *Molecular Biology and Evolution*, 14(12):1296, 1997.

[52] W. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Cont. Math.*, 26:189–206, 1984.

[53] D. M. Lambert and C. D. Millar. Ancient genomics is born. *Nature*, 444:275–276, 2006.

[54] N. Lartillot, S. Blanquart, and T. Lepage. PhyloBayes. v2. 3, 2007.

[55] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15:215–245, 1995.

[56] D. Maddison. The discovery and importance of multiple islands of most-parsimonious trees. *Syst. Zool.*, 40(3):315–328, 1991.

[57] B. Manly. *Randomization, Bootstrap and Monte Carlo Methods in Biology.* CRC Press, 1997.

[58] T. Margush and F. McMorris. Consensus n-trees. *Bulletin of Mathematical Biology*, 43:239–244, 1981.

[59] B. Moret. Towards a discipline of experimental algorithmics. *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges: Papers Related to the DIMACS Challenge on Dictionaries and Priority Queues (1995-1996) and the DIMACS Challenge on Near Neighbor Searches (1998-1999)*, page 197, 2002.

[60] E. Mossel and E. Vigoda. Limitations of Markov chain Monte Carlo algorithms for Bayesian inference of phylogeny. *Ann. Appl. Probab.*, 16(4):2215–2234, 2006.

[61] J. Nylander, J. Wilgenbusch, D. Warren, and D. Swofford. Awty (are we there yet?): a system for graphical exploration of mcmc convergence in bayesian phylogenetics. *Bioinformatics*, 2007. advance access, published August 30.

[62] N. D. Pattengale, M. Alipour, O. R. P. Bininda-Emonds, B. M. E. Moret, and A. Stamatakis. How many bootstrap replicates are necessary? In S. Batzoglou, editor, *Research in Computational Molecular Biology, 13th Annual International Conference, RECOMB 2009, Tucson, AZ, USA, May 18-21, 2009. Proceedings*, volume 5541 of *Lecture Notes in Computer Science*, pages 184–200, 2009.

[63] N. D. Pattengale, M. Alipour, O. R. P. Bininda-Emonds, B. M. E. Moret, and A. Stamatakis. How many bootstrap replicates are necessary? *Journal of Computational Biology*, 17(3):xxx–yyyy, 2010.

[64] N. D. Pattengale, E. J. Gottlieb, and B. M. Moret. Efficiently computing the robinson-foulds metric. *Journal of Computational Biology*, 14(6):724–735, 2007. PMID: 17691890.

[65] N. D. Pattengale and B. M. E. Moret. A sublinear-time randomized approximation scheme for the robinson-foulds metric. In A. Apostolico, C. Guerra, S. Istrail, P. A. Pevzner, and M. S. Waterman, editors, *RECOMB*, volume 3909 of *Lecture Notes in Computer Science*, pages 221–230. Springer, 2006.

[66] N. D. Pattengale, K. M. Swenson, and B. M. Moret. Uncovering hidden phylogenetic consensus, 2010. Submitted.

[67] N. D. Pattengale, K. M. Swenson, M. M. Morin, and B. M. Moret. Higher fidelity subtree merging for disk-covering methods. Poster, Algorithmic Biology, 2006. http://www.calit2.net/events/algorithmicbio/files/PattengaleAlgoBio2006.pdf.

[68] A. Rambaut and A. Drummond. Tracer MCMC trace analysis tool version 1.3, 2004.

[69] W. Rand. Objective criteria for the evaluation of clustering methods. *J. American Stat. Assoc.*, 66:846–850, 1971.

[70] B. Redelings. Bayesian phylogenies unplugged: Majority consensus trees with wandering taxa. http://www4.ncsu.edu/∼bdredeli/wandering.pdf.

[71] D. Robinson and L. Foulds. Comparison of phylogenetic trees. *Math. Biosciences*, 53:131–147, 1981.

[72] F. Ronquist and J. Huelsenbeck. MrBayes 3: Bayesian phylogenetic inference under mixed models. *Bioinformatics*, 19(12):1572–1574, 2003.

[73] S. Smith and M. Donoghue. Rates of Molecular Evolution Are Linked to Life History in Flowering Plants. *Science*, 322(5898):86–89, 2008.

[74] D. Soltis, M. Gitzendanner, and P. Soltis. A 567-taxon data set for angiosperms: The challenges posed by bayesian analyses of large data sets. *Int. J. of Plant Sci.*, 168(2):137–157, 2007.

[75] D. Soltis and P. Soltis. Applying the Bootstrap in Phylogeny Reconstruction. *Statist. Sci.*, 18(2):256–267, 2003.

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

[77] A. Stamatakis, P. Hoover, and J. Rougemont. A Rapid Bootstrap Algorithm for the RAxML Web Servers. *Sys. Biol.*, 2008. in press.

[78] A. Stamatakis, H. Meier, and T. Ludwig. New Fast and Accurate Heuristics for Inference of Large Phylogenetic Trees. In *Proc. of IPDPS2004*, HICOMB Workshop, Proceedings on CD, Santa Fe, New Mexico, 2004.

[79] A. Stamatakis and M. Ott. Efficient computation of the phylogenetic likelihood function on multi-gene alignments and multi-core architectures. *Phil. Trans. R. Soc. series B, Biol. Sci.*, 363:3977–3984, 2008.

[80] S. Sul, G. Brammer, and T. Williams. Efficiently Computing Arbitrarily-Sized Robinson-Foulds Distance Matrices. In *Proceedings of the 8th international workshop on Algorithms in Bioinformatics*, pages 123–134. Springer, 2008.

[81] S. Sul and T. Williams. A randomized algorithm for comparing sets of phylogenetic trees. In *Proceedings of the 5th Asia-Pacific bioinformatics conference: Hong Kong, 15-17 January 2007*, page 121. Imperial College Pr, 2007.

[82] S. J. Sul and T. L. Williams. An experimental analysis of consensus tree algorithms for large-scale tree collections. In *Proceedings of 5th Intl. Symposium on Bioinformatics Research and Applications (ISBRA'09)*, Springer LNBI 5542, pages 100–111, 2009.

[83] D. Swofford. *PAUP*: Phylogenetic analysis using parsimony (* and other methods), version 4.0b10.* Sinauer Associates, 2002.

[84] D. L. Swofford. When are phylogeny estimates from molecular and morphological data incongrugent? In *Miyamoto M. M., Cracraft J., eds. Phylogenetic analysis of DNA sequences*, pages 295–333. Oxford Univ. Press, 1991.

[85] J. L. Thorley. *Cladistic Information, Leaf Stability And Supertree Construction.* PhD thesis, University of Bristol, 2000.

[86] J. L. Thorley, M. Wilkinson, and M. Charleston. The information content of consensus trees. In A. Rizzi, M. Vichi, and H. Bock, editors, *Studies in Classification, Data Analysis, and Knowledge Organization*, Advances in Data Science and Classification, pages 91–98. Springer, 1998.

[87] G. Valiente. *Combinatorial Pattern Matching Algorithms in Computational Biology Using Perl and R.* Chapman & Hall/CRC, 2009.

[88] T. Warnow. Large–scale phylogenetic reconstruction. In S. Aluru, editor, *Handbook of Computational Biology*. Chapman & Hall, CRC Computer and Information Science Series, 2005.

[89] S. Whelan. New Approaches to Phylogenetic Tree Search and Their Application to Large Numbers of Protein Alignments. *Syst. Biol.*, 56(5):727–740, 2007.

[90] M. Wilkinson. Common Cladistic Information and its Consensus Representation: Reduced Adams and Reduced Cladistic Consensus Trees and Profiles. *Syst Biol*, 43(3):343–368, 1994.

[91] M. Wilkinson. More on reduced consensus methods. *Syst. Biol.*, 44:435–439, 1995.

[92] M. Wilkinson. Majority-rule reduced consensus trees and their use in bootstrapping. *Mol Biol Evol*, 13(3):437–444, 1996.

[93] D. Zwickl. *Genetic Algorithm Approaches for the Phylogenetic Analysis of Large Biological Sequence Datasets under the Maximum Likelihood Criterion.* PhD thesis, University of Texas at Austin, April 2006.