2013

Managing and analyzing phylogenetic databases

AKSHAY DEEPAK
Iowa State University

Managing and analyzing phylogenetic databases

by

Akshay Deepak

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Science

Program of Study Committee:

David Fernández-Baca, Major Professor

Oliver Eulenstein

Xiaoqiu Huang

Pavan Aduri

Srikanta Tirthapura

Iowa State University

Ames, Iowa

2013

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I am very grateful to Dr. David Fernández-Baca for being an understanding guide who is always willing to hear the other person and, for supporting me through ups and downs of my research. I thank him for his numerous hours spent on revising the manuscripts to bring them to the highest standards — providing me an invaluable learning experience in the process. He has always managed to give me time for discussions in spite of his busy schedule, yet, our meetings have never been during the late hours or the weekends. This balanced approach towards work and personal life is one of the most admirable things about him.

I am very thankful to my other committee members Dr. Oliver Eulenstein, Dr. Xiaoqiu Huang, Dr. Pavan Aduri and Dr. Srikanta Tirthapura for their valuable time and inputs on my research. A special thanks to Dr. Mike Sanderson and Dr. Michelle M. McMahon for our ongoing collaboration on STBase and other projects. I am very thankful to my lab members for providing me with a friendly environment and for their support in my research. I thank all the departmental staff for their timely help. I gratefully acknowledge the support received from Dr. Fernández-Baca through NSF grants DEB-0829674 and CCF-106029.

I am very grateful to my parents and family members for supporting me to come here for PhD and, for their love and care in my life.

I am very grateful to God for always arranging the best for me and for all the gifts in life. I am very grateful to my friends in Ames, whose unconditional support has helped me in countless ways in my PhD and in leading a wholesome life.

ABSTRACT

The ever growing availability of phylogenomic data makes it increasingly possible to study and analyze phylogenetic relationships across a wide range of species. Indeed, current phylogenetic analyses are now producing enormous collections of trees that vary greatly in size. Our proposed research addresses the challenges posed by storing, querying, and analyzing such phylogenetic databases.

Our first contribution is the further development of STBase, a phylogenetic tree database consisting of a billion trees whose leaf sets range from four to 20000. STBase applies techniques from different areas of computer science for efficient tree storage and retrieval. It also introduces new ideas that are specific to tree databases.

STBase provides a unique opportunity to explore innovative ways to analyze the results from queries on large sets of phylogenetic trees. We propose new ways of extracting consensus information from a collection of phylogenetic trees. Specifically, this involves extending the maximum agreement subtree problem. We greatly improve upon an existing approach based on frequent subtrees and, propose two new approaches based on agreement subtrees and frequent subtrees respectively.

The final part of our proposed work deals with the problem of simplifying multi-labeled trees and handling "rogue" taxa. We propose a novel technique to extract conflict-free information from multi-labeled trees as a much smaller single labeled tree. We show that the inherent problem in identifying rogue taxa is NP-hard and give fixed-parameter tractable and integer linear programming solutions.

CHAPTER 1. INTRODUCTION

Phylogenomic data availability in open access databases such as GenBank (Benson et al. (2008)) and TreeBASE (Piel et al. (2002)) has witnessed an unprecedented growth in the last decade or so. Coupled with the development of sophisticated methods and tools, this has led to large scale phylogenetic analyses covering a diverse range of life. Initiatives such as Tree of Life (`http://www.phylo.org/atol/`, `http://opentreeoflife.org/`) are producing phylogenetic analyses spanning a wide taxonomic coverage and consisting of large number of trees of varying size. The tree repository in STBase ("Species Tree Database") (McMahon et al. (2013)) is one such initiative. It consists a billion phylogenetic trees built from single- and multi-locus datasets assembled from GenBank (Benson et al. (2008)). Interfacing such a database to the online community poses unique challenges with respect to efficient search and retrieval techniques required in querying the database. Existing tree databases (Piel et al. (2002); Sanderson et al. (2008); Finn et al. (2010)) address some of these concerns, however, it is not straight forward to extend them for such a large repository as STBase. Organization of the database and the way user queries are processed further strengthen the need for a separate approach to mine such a database.

The first contribution of this dissertation is a search engine to efficiently query the huge tree repository in STBase (McMahon et al. (2013)). The trees in the repository cover a diverse range of life spanning 41,3628 taxon and more than six million sequences, and vary greatly in size (4-20,000 leaves). A typical query on this database includes a set of related species as per the interest of the user. The result contains a collection of trees restricted to the query set and ranked as per a quality criteria. Through a combination of techniques from information retrieval, notably inverted indexing, and from computational phylogenetics, especially for constructing consensus trees, and use of efficient hash functions, STBase search engine achieves

fast response times — responding to user queries within few seconds.

In the second part of our research we explore alternative ways of mining consensus information in a collection of trees. In this regard, we extend the well known maximum agreement subtree (MAST) problem. Given a collection of input trees on a common leafset, an agreement subtree is a subtree supported by all (100%) the input trees. A MAST is an agreement subtree having the maximum number of leaves among all such agreement subtrees. We relax these two criteria — i.e., 100% support and the maximum cardinality criteria with respect to the leafset — to obtain frequent subtrees (FSTs). Application on real world datasets show that FSTs can provide bigger and/or more informative subtrees than MAST. Moreover, a collection of such FSTs can provide a larger taxon coverage. In certain cases, a FST can be more resolved than the commonly used majority rule tree approach as well as the MAST. We propose efficient algorithms to mine such FSTs. Though very useful, the number of FSTs can become very large as the size of the common leafset is increased. This imposes practical restrictions on the number of FSTs that can be mined and further analyzed. Motivated by this, we further extend our work and propose two new approaches — one based on FSTs and another on agreement subtrees — for identifying common information in a collection of phylogenetic trees.

The third part of our research is on multi-labeled trees (MUL-trees) and the rogue taxon problem. A MUL-tree is species trees with multiple leaves labeled by the same species. Such trees are a common occurrence while inferring species trees from gene trees. With respect to phylogenetic inference, a common problem often encountered with MUL-trees is the presence of conflicting quartets. We introduce the notion of conflict free quartets, and define the information content of a MUL-tree as the set of all conflict-free quartets implied by it. We define the maximally reduced form of a MUL-tree as the smallest tree obtainable from the original tree that retains all the conflict-free information. We propose an efficient algorithm to reduce MUL-trees to their maximally reduced form and evaluate its performance on empirical datasets in terms of both quality of the reduced tree and the degree of data reduction achieved.

A rogue taxon in a collection of phylogenetic trees is one whose position varies drastically from tree to tree. The presence of such taxa can greatly reduce the resolution of the consensus tree (e.g., the majority-rule or strict consensus) for a collection. We show that the underlying

problem in identifying rogue taxon is NP-hard. We give polynomial solutions for a restricted case of strict consensus and integer linear programing solutions for the general case of consensus trees.

1.1 Organization of The Thesis

The rest of the thesis is organized as follows:

Chapter 2: This describes STBase. It has two sub-parts. The first part describes the challenges and techniques in developing the search engine and the second part describes the construction of the billion trees. The chapter is a modified version of the paper submitted to the journal *Systematic Biology*. A preliminary version of the text is also part of my Master's thesis (Deepak (2010)). However, the current version is completely re-written with major additions. I was responsible for designing and implementing the search engine, and developing the online user interface.

Chapter 3: This chapter describes EvoMiner: a new algorithm for enumerating all frequent subtrees in collections of phylogenetic trees, and the improvements EvoMiner achieves over Phylominer — the state of the art algorithm for the same task. This manuscript has been submitted to the journal *Knowledge and Information Systems* and is currently under revision. I was responsible for designing and implementing EvoMiner.

Chapter 4: Extending our work on EvoMiner, in this chapter we propose two novel approaches for identifying common information in collections of phylogenetic trees, one based on agreement subtrees called maximal agreement subtrees, the other on frequent subtrees called maximal frequent subtrees. This is an unpublished manuscript. I was responsible for designing and implementing the algorithms.

Chapter 5: Here we describe a quartet based measure to identify conflict-free information from multi-labeled phylogenetic trees, and give algorithms to extract such conflict-free information. A condensed version of the text appeared in the Proceedings of the 12th *Workshop on Algorithms in Bioinformatics*, 2012 (WABI 2012). The current version

is the paper invited for a special issue of the journal *Algorithms for Molecular Biology* devoted to the selected papers from WABI 2012, and is currently under review. I was responsible for formulating the conflict-free measure and designing the algorithms.

Chapter 6: This chapter explores the complexity of identifying rogue taxon while constructing consensus trees in collections of phylogenetic trees and, gives polynomial solution for the restricted case of strict consensus trees and exact ILP formulations for the general case. A condensed version of this paper appeared in the Proceedings of the 8th *International Symposium on Bioinformatics Research and Applications*, 2012. I was responsible for proving the NP-hardness of the rogue taxon problem and designing the polynomial solution for the restricted case of strict consensus trees.

Chapter 7: Concluding remarks on the thesis appear here.

CHAPTER 2. STBase: ONE BILLION TREES FOR COMPARATIVE BIOLOGY

Michelle M. McMahon, Akshay Deepak, David Fernndez-Baca, Darren Boss and Michael J. Sanderson

Modified from a paper submitted to the journal *Systematic Biology*

Abstract

A new database, STBase, is designed to let comparative biologists quickly retrieve phylogenetic hypotheses pertinent to a query list of species or genus names. The database consists of 1 million single- and multi-locus phylogenetic data sets (each with a confidence set of 1000 trees), computed from GenBank sequence data for 413,000 eukaryotic taxa. Two bodies of theoretical work are leveraged to aid in the assembly of multi-locus concatenated data sets. First, impacts of missing data are ameliorated by assembling only decisive multi-locus data sets (Michael et al. (2010)). Second, a novel "multree" reduction algorithm is used to prune multiply labeled trees to conflict free, conservative, singly-labeled trees that can be combined between loci. A fast inverted indexing scheme permits retrieval of data sets overlapping with the query, which are ranked according to a scoring scheme that weighs tree quality and amount of taxonomic overlap of the tree with the query. Tree quality is assessed by a real-time evaluation of bootstrap support on just the overlapping subtree. Associated sequence alignments, tree files and metadata can be downloaded for subsequent analysis.

2.1 Introduction

Phylogenetic trees have greatly altered comparative biology by rearranging the context for comparison, enhancing statistical power of comparative tests, and broadening taxonomic scope

(Felsenstein (2003); Baum and Smith (2012)). In recent years the demand for phylogenetic trees has been so high that comparative biologists themselves have turned to heuristic or even non-algorithmic methods for assembling trees comprehensive enough to contain the taxa in which they are interested (e.g., Phylomatic Project: Webb and Donoghue (2004); `http://www.phylodiversity.net`; see e.g., Pringle et al. (2007) for an application). This reflects a basic impediment: the mismatch between the set of taxa present in available phylogenetic trees and the set of taxa for which comparative data are available. For example, the Royal Botanic Gardens, Kew, maintains a database of morphological and biochemical data on seeds of angiosperms (Flynn et al. (2006)), which has been used in comparative analyses such as Moles et al.'s (Moles et al. (2005)) study of the correlates of seed size variation. Currently, of the 2572 eudicot species having data for the trait "percent oil content", some 36% have no sequences in GenBank, even though eudicots are arguably one of the best sampled species-rich taxonomic groups in the tree of life (the overall species level sequence coverage across described biodiversity is closer to 10%). Moreover, the 64% that are in GenBank are found patchily among various loci, so that previously reconstructed phylogenetic trees are much more limited in their taxon coverage than this number suggests.

One strategy to overcome this is assembly of ultra-large, dense phylogenies of particular clades (Bininda-Emonds et al. (2007); Nyakatura and Bininda-Emonds (2012); Peters et al. (2011); Smith et al. (2009, 2011); Edwards (2011)), or of particular regions of the world (Lanfear et al. (2011); Saslis-Lagoudakis et al. (2012)), depending on the biological question. However, the distribution of data with respect to taxon coverage is highly uneven across all of biodiversity (Sanderson (2008)), and scaling inference up presents numerous computational challenges (Bader et al. (2006); Goldman and Yang (2008); Liu et al. (2009); Izquierdo-Carrasco et al. (2011)), especially in handling the patchy coverage across multiple sparsely sampled loci (Sanderson et al. (2011); Roure et al. (2012)). An alternative strategy, which should be useful in the near term, is to assemble a very large collection of phylogenetic trees of small to medium scale, and optimize the delivery of these trees via efficient search and retrieval. As larger trees are needed, they can be pieced together by other algorithms (caveat emptor!). One clear advantage of this is that it allows relatively robust estimation of reliability — assessments

in ultra-large trees remaining a challenge — and these estimates of reliability can be returned to the user.

Several tree databases currently provide some aspect of this service, including TreeBASE (Piel et al. (2002)), and our PhyLoTA browser database (Sanderson et al. (2008)), in addition to genomic databases aimed at delivering gene trees instead of species trees (e.g., PFAM; Finn et al. (2010)), but none of these are designed from the ground up to address the taxon mismatch problem. Moreover, our PhyLoTA database computed only data sets from single loci, neglecting the power now so clearly evident in multigene analyses. Here we describe a new database of precomputed phylogenetic trees, STBase, optimized for use by comparative biologists. In it we archive one billion precomputed phylogenies built from single- and multi-locus datasets, the latter guided by recent theory on optimal multigene data set construction and treatment of multrees–that is, trees with more than one leaf having the same name (owing to duplication, repeated sampling, population sampling, etc.). We join this with a search engine that accepts lists of taxon names and efficiently returns a ranked list of trees, the subtrees that overlap with the taxa of interest, and support values. In terms of database size, and retrieval time efficiency, it presents the user with an interactive experience comparable to online BLAST searches against GenBank.

STBase is available online at `http://searchtree.org/`. The contributions in making the billion trees easily accessible online can be divided into two main subheadings. Section 2.2 describes the search engine responsible for processing user queries efficiently and the structural organization of the database. Section 2.3 describes the construction of trees.

2.2 The Search Engine

2.2.1 Overview

The goal of STBase is to provide a tool that accepts a user's query list of taxon names and returns a ranked list of good "hits" to a database of phylogenetic trees. To quantify what "good hit" means (the term meant to be analogous to BLAST searches, Altschul et al. (1990)), we construct a scoring function that increases with the quality of the trees found and the amount

of taxonomic overlap between the tree and the query. We assume that tree quality can be characterized via a confidence set of trees, such as computed by bootstrapping (as here) or by sampling the posterior distribution (Huelsenbeck et al. (2001)). Let A, be the query list, and α be a user supplied weighting parameter indicating the relative importance of tree quality and taxon overlap. For any tree, T, let $\mathcal{L}(T)$ be the taxa in the tree; $T|_A$ be the subtree restricted to just the query taxa, and $\mathcal{L}(T|_A)$ be the taxa shared between the query and the tree. Then define $\omega(\mathcal{L}(T|_A))$ to be some increasing function of the this overlap. Let $\theta(T|A)$ be some increasing function of the quality of the subtree. The score of a "hit" is then

$$S = \alpha \cdot \omega(\mathcal{L}(T|_A)) + \theta(T|A)$$

We construct a normalized score ranging from 0 to 100 (1) using the bootstrap percentages on the subtree as a quality score, (2) using an overlap function given by the number of overlapping taxa divided by the number of query taxa that are in the database (rather than the larger set of query taxa that might include taxa not found in GenBank at all) x 100, and finally, (3) dividing by $1 + \alpha$. Higher values of α make overlap increasingly important relative to quality.

2.2.2 Database Entities

The STBase database consists of five entities namely taxon, genus, sequence, cluster and tree. A taxon consists of a name and an ID. A taxon can have multiple names mapped to the same ID. Each genus consists of name and maps to one or more taxa. Each sequence – represented by an ID – is associated with a taxon and there can be multiple sequences associated with the same taxon. A cluster is a collection of sequences on which trees are constructed. Each cluster consists of a thousand bootstrapped trees, and has a unique ID. Because of their sizes (over 250GB), clusters cannot be kept in main memory. This coupled with the large number of , which poses several challenges to achieving fast query processing. Each leaf of a tree in a cluster corresponds to a sequence. Each leaf of a tree in the result set can correspond to a taxon name, taxon ID, sequence ID or a combination of these three, as indicated by the user. All IDs refer to a numerical value.

2.2.3 Query Processing: Challenges and Techniques

Figure 2.1 shows how a query is processed. Currently the web interface supports queries based on taxon names or genus names or a mix of both, however, keeping future prospects in mind the server implementation also supports queries based on taxon IDs and sequence IDs. When the user inputs a list of taxon names and genus names, genus names are replaced by the corresponding taxon names and the updated query consists of only taxon names. This is followed by five steps: 1) retrieving the corresponding taxon IDs, 2) fetching the sequence IDs associated with each such taxon, 3) identifying the clusters having the desired overlap with the set of query sequences and reading them from disk, 4) processing each cluster to restrict each of the thousand trees in the cluster to the taxa that overlap with the query sequences, 5) summarizing the restricted trees for each cluster as a majority rule tree (MRT), and returning these MRTs to the user. Next, we describe the techniques used to implement these steps efficiently.

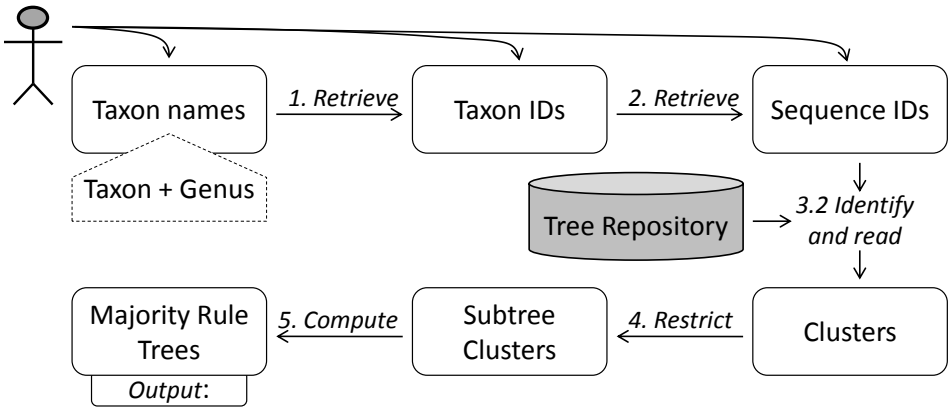


Figure 2.1: The query process

2.2.3.1 Identification of Overlapping Clusters

Given a set of sequence IDs, identifying overlapping clusters and reading them from disk memory (step 3 in Figure 2.1) is the most time consuming part of the query process, as there are 1 million clusters covering over 6 million sequences. STBase accomplishes this task in time that is independent of size of the database through inverted indexing (Zobel and Moffat (2006);

Manning et al. (2008)).

Given a large collection of documents (e.g., web pages), an inverted index allows one to search and retrieve the subset of documents containing one or more words from the query set. It does so by maintaining a mapping from a predefined set of keywords to the documents in the collection that contain them. A document is important if it contains one or more keywords. In STBase, the goal is to find the clusters containing sequences that map to the list of species supplied in the user query (see Figure 2.1). Given a set of query sequences (the keywords), STBase's inverted index tells exactly which clusters (documents) contain those sequences and where those clusters are located on the hard drive. Figure 2.2 gives a high-level picture of STBase's indexing scheme. It consists of two data-structures: Inverted List and the Vocabulary (See Fig. 2.2). Inverted List contains an entry for each keyword, i.e., a sequence. An entry for a sequence contains two things: the number of clusters that have the sequence in their leafset and the list of all such clusters. For each such cluster, this list stores the exact location of the cluster in the disk memory. Vocabulary stores, for each sequence ID, the pointer to the entry in Inverted List corresponding to the sequence. Vocabulary is constructed such that given a query sequence ID, the corresponding pointer to the inverted list is obtained in constant time. STBase achieves this task by maintain Vocabulary as an array; the i^{th}sequence is stored at the i^{th} index.

2.2.3.2 Majority Rule Tree Generation

A query typically results in 100-200 clusters arising from sufficient overlap with the taxon names provided as input. Each of these clusters consists of thousand bootstrapped trees that are then restricted to the query overlap and summarized as an MRT. To generate the MRT efficiently, we used Amenta et al.'s randomized linear time MRT algorithm (Amenta et al. (2003)), which uses hash codes — a constant size object — to represent bipartitions and a clever method to construct the MRT using only these hashed bipartitions. This results in a linear-time (i.e., optimal) algorithm.

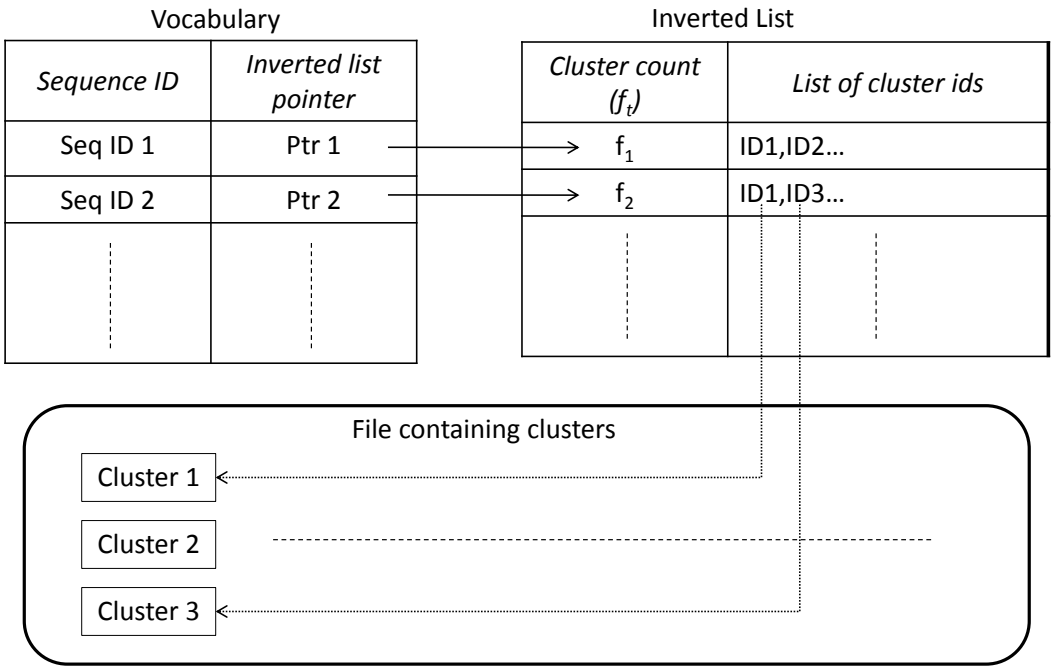


Figure 2.2: Indexing scheme deployed in STBase

2.2.3.3 Mapping between Names and Numbers

Genus names, taxon names, taxon IDs and sequence IDs are provided as part of the tree repository. The user queries on a mix of genus names and taxon names; however, the clusters in the repository are based on sequence IDs. To map efficiently among these entities, STBase employs universal hash functions (Motwani and Raghavan (1995); Cormen (2001)) and string-specific hash functions (Jenkins (1997)), which are capable of inserting and deleting a random element in constant time irrespective of the size of the collection. The current release has 53,849 genus names , 413,627 taxon names, 413,628 taxon IDs and 6,026,845 sequence IDs. Any mapping scheme that is dependent on these numbers will significantly slow down the query processing time. Thus, hashing plays an important role in keeping the query processing time optimal.

As a result of these techniques and some intelligent preprocessing of the data, STBase answers queries in time that is linear in the total size of the query and the output, and independent of the size of the underlying tree repository. For a detailed description of the application of these techniques in STBase, see Deepak (2010).

2.3 Tree Construction

2.3.1 Single Locus Data Sets

A pool of ˜160,000 single locus data sets (Table 2.1) was assembled from GenBank rel. 184 largely according to the PhyLoTA pipeline described earlier (Sanderson et al. (2008)). A set of 517 taxonomic groups in the NCBI hierarchy was selected so as not to exceed 35000 sequences contained in its (nonmodel) organisms. We refer to these as "hub groups". These corresponded in the end very roughly to the rank of Linnean "orders". Within each, clusters of homologous sequences were assembled by all-against-all BLAST search and single-linkage clustering following 50% minimal overlap requirements as described. Multiple sequence alignments were obtained using MUSCLE (Edgar (2004)); ML optimal trees using RAxML (Stamatakis (2006)); and 1000 "fast" parsimony bootstrap trees using PAUP* (Swofford (2003)). Many (111,433; 11.25%) of these clusters had "multrees" (Fellows et al. (2003); Grundt et al. (2004); Huber and Moulton (2006); Popp and Oxelman (2001); Scornavacca et al. (2011)) — that is, they included at least one taxon ID multiple times, owing to sampling of multiple individuals within taxa, multiple alleles or paralogous loci. We exploited a novel multree reduction algorithm (Deepak et al. (2012b)) to reduce each of these multrees to a singly labeled "reduced" tree that is not contradicted by the species level information in the multree. This is a conservative procedure that limits the number of false positive species relationships. The user can view either the multree or reduced tree for a single locus data set. See Figure 2.3.

	Number of data sets	Loci (mean and range)	Taxa (mean and range)	Data set size (mean and range)
Clusters	160,801	1 (1-1)	63.1 (4-8767)	63.1 (4-8767)
Bicliques	762,529	9.8 (2-91)	15.6 (4-510)	142.3 (8-1526)
Decisive quasi-bicliques	67,103	12.4 (2-386)	27.8 (5-1406)	234.7 (10-9516)

Table 2.1: Summary statistics for the three kinds of data sets

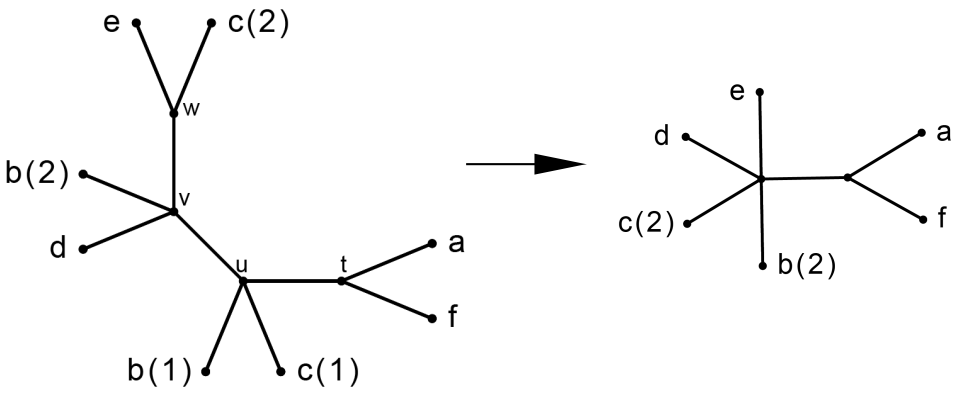


Figure 2.3: An example of MUL-tree reduction. Numbers in parenthesis next to labels indicate the multiplicity of the respective labels and are not part of the labels themselves. The singly-labeled tree on the right is the reduced form of the MUL-tree on the left. It retains only "conflict-free" quartets from the original MUL-tree. A conflict-free quartet is one which is topologically not conflicted by any other quartet displayed by the original tree over the same set of four leaves.

2.3.2 Multi-locus datasets

Assembly of multi-locus concatenated data sets is problematic in cases in which multiple sequences are present in the same taxon — that is, when multrees are inferred for a cluster. We used the reduced set of taxa obtained from the multree reduction as the source of sequence data for assembly of supermatrices. Although this results in a loss of some taxa on average, it also reduces (or at least sidesteps) the conflict between loci arising from biological processes such as gene duplication or incomplete lineage sorting. Although we have not built trees using other methods, this collection of reduced loci/trees could be used as inputs to species tree inference methods as well as simple concatenation.

Two protocols were used to further guide assembly of multi-locus supermatrices from the single locus reduced data sets in each hub group. Both generate multi-locus data sets with the desirable property of decisiveness for all possible trees, which can help limit the impact of missing entries in the supermatrix (Steel and Sanderson (2010); Sanderson et al. (2010, 2011)). A supermatrix, M, is decisive for tree, T, if and only if the subtrees, t_i, for each locus i, obtained by restricting T to only those taxa that have sequence data at locus i, uniquely define T (it is easy to find examples in which many trees are consistent with these subtrees, rather than only

one). A particularly strong form of decisiveness, which holds for some patterns of missing data, is that M may be decisive for all possible trees, T. Our first algorithm assembles maximal complete supermatrices, meaning every taxon is sampled for each locus, by finding so-called maximal bicliques in an associated graph data structure (Sanderson et al. (2003); Driskell et al. (2004)). Since any supermatrix in which one locus includes sequence from all taxa is decisive, these are decisive for all trees. Our second algorithm allows some missing entries in the supermatrix by building incomplete matrices. This algorithm builds a supermatrix using each locus as a reference locus in turn. Taxa restricted only to those in the reference locus, and any other locus with at least 33.3% taxon overlap with the reference locus is allowed to join the supermatrix. Because of the reference locus, this supermatrix is also decisive for all trees, even though it contains missing data, and we refer to it as a so-called decisive quasi-biclique.

An important property of the collection of maximal bicliques or decisive quasi-bicliques is that they can overlap with one another partially. In some cases with relatively dense graphs, there can be a very large number of rather similar multi-locus data sets assembled in this way. We found, for example that within mammals there were a vast number of primate and carnivore bicliques, the two largest nodes, and we therefore took a 2 and 20% sample from these, respectively. Various checks and filters were run on the results. We checked whether there were duplicate data sets within or between nodes in the NCBI hierarchy and whether any decisive quasi-bicliques were actually bicliques (which occurs rarely when the taxon coverage pattern is conducive). In addition we use a BLAST protocol to check that all loci in a data set are independent from each other, sharing no local homologies (these can arise occasionally for a variety of reasons upstream in the pipeline) which might lead to redundant inclusion in the same supermatrix.

Availability

The database can be accessed at http://searchtree.org. Software for the database retrieval engine and the code implementing multree reduction algorithm is available at http://code.google.com/p/search-tree/. All code is distributed according to a GNU GPL v3 license (http://www.gnu.org/licenses/gpl.html).

Author's Contributions

MJS and MMM motivated the problem and constructed the billion trees. AD and DFB designed the search engine. AD implemented the search engine. DB assisted as the System Administrator in deploying the search engine and setting up the Redwood cluster at MJS's lab. AD, MJS and DB contributed towards the development of the online interface. MJS and DFB coordinated the project.

CHAPTER 3. EvoMiner: FREQUENT SUBTREE MINING IN PHYLOGENETIC DATABASES

Akshay Deepak, David Fernndez-Baca, Srikanta Tirthapura, Michael J. Sanderson and Michelle M. McMahon

A paper submitted (under revision) to the journal *Knowledge and Information Systems*

Abstract

The problem of mining collections of trees to identify common patterns, called frequent subtrees (FSTs), arises often when trying to interpret the results of phylogenetic analysis. FST mining generalizes the well-known maximum agreement subtree problem in phylogenetics. Here we present EvoMiner, a new algorithm for mining frequent subtrees in collections of phylogenetic trees. EvoMiner is an Apriori-like level-wise method, which uses a novel phylogeny-specific constant-time candidate generation scheme, an efficient fingerprinting-based technique for downward closure, and a lowest common ancestor based support counting step that requires neither costly subtree operations nor database traversal. Our algorithm achieves speed-ups of up to 100 times or more over Phylominer, the current state-of-the-art algorithm for mining phylogenetic trees. EvoMiner can also work in depth first enumeration mode, to use less memory at the expense of speed. We also demonstrate the utility of FST mining as a way to extract more meaningful phylogenetic information from collections of trees when compared to maximum agreement subtrees and majority rule trees — two commonly used approaches in phylogenetic analysis for extracting consensus information from a collection of trees over a common leaf set.

3.1 Introduction

A phylogeny (or phylogenetic tree or evolutionary tree) depicts the evolutionary relationships among a set of species (Baum (2008)). Aside from their intrinsic scientific interest, phylogenies have diverse applications, which include characterizing cryptic biological diversity (Barns et al. (1996)), crop improvement (Flint-Garcia et al. (2005)), identifying snakebite antivenins (Slowinski and Keogh (2000)), tracking the spread of epidemic diseases (Smith and Patton (1999)), political science (Currie et al. (2010)), and historical linguistics (Gray et al. (2009)).

Here we consider the problem of identifying common patterns —specifically, frequent subtrees— in collections of phylogenetic trees. A frequent subtree (FST) is a tree t that is "supported" by at least some given fraction of the input trees. That is, t is embedded as a subtree in at least the given fraction of the input trees[1]. The need to mine collections of phylogenies for frequent patterns arises in different contexts. For example, when building evolutionary trees from single-gene data sets, Bayesian inference (Huelsenbeck and Ronquist (2001)) produces a posterior distribution of trees, while the bootstrap method (Felsenstein (1985)) yields a set of trees, from which confidence intervals are obtained for various groupings of the species. In both cases, frequent subtrees can reveal common patterns overlooked by conventional techniques, such as majority-rule trees or maximum agreement subtrees. Collections of phylogenetic trees also arise in multi-gene (or even whole-genome) studies. Large sets of such trees have been assembled into databases such as TreeBASE (Piel et al. (2002)) and PhyLoTA (Sanderson et al. (2008)). In general, these phylogenies overlap only partially in the sets of species they cover. Further, these trees often disagree with respect to the placement of the species they share in common, due to either error or to complex biological processes (e.g., horizontal gene transfer, gene duplication, convergent evolution, and varying evolutionary rates) (Rannala and Yang (2008)). Here again, FSTs can be valuable, pointing to different histories for parts of the genome (Zou et al. (2008)).

In this paper, we present EvoMiner, a fast algorithm for enumerating FSTs. Further, we

[1]Strictly speaking, we required that t be *displayed by* at least some given fraction of the input trees. Formal definitions of all the concepts used in this Introduction are given in Sect. 3.2.

demonstrate experimentally the effectiveness of the FST approach, as compared to other widely-used methods for identifying common phylogenetic information. To put our contributions in context, we first review the existing work.

3.1.1 Related Work

3.1.1.1 Tree Mining

Tree mining has been an active area of research in the past decade; for a survey, see references Chi et al. (2004a); Jimenez et al. (2010a). Table 3.1 summarizes some of the proposed approaches. Based on the type of tree, tree mining tasks can be classified as ordered (TreeMiner (Zaki (2005)), FREQT (Asai et al. (2002)), Chopper and XSpanner (Wang et al. (2004))) or unordered (Unot (Asai et al. (2003)), uFreqt (Nijssen and Kok (2003)), Bei et al. (2009)), rooted (Bei et al. (2009)) or unrooted (CMTreeMiner (Chi et al. (2005)), DRYADE (Termier et al. (2004))). Based on the type of subtree, they can be classified as induced (Asai et al. (2002)), embedded (Zaki (2005); Liu and Liu (2011)), or bottom-up. Based on the relationship among the frequent subtrees, they can be classified as maximal (PathJoin (Xiao and Yao (2003)), Hadzic et al. (2010)) and closed (Chi et al. (2005); Feng et al. (2010); Wang et al. (2012); Nguyen and Yamamoto (2010)).

A number of approaches have been proposed for mining ordered trees. Asai et al.'s Freqt method (Asai et al. (2002)) uses rightmost expansion scheme for candidate generation and occurrence lists for frequency counting. Zaki's TreeMiner algorithm (Zaki (2005)) introduced a scope list based representation of the occurrence list of a frequent subtree. Coupled with a clever string encoding, this results in efficient candidate generation and frequency counting. The enumeration proceeds in a depth-first manner. Wang et al. (2004) proposed the Chopper and Xspanner algorithms to mine subtrees without candidate generation. These scale well for large sets of trees because they use the pattern-growth method (Pei and Han (2002); Han et al. (2004, 2000)) for enumeration, which represents the database in a compact form and avoids generating an exponential number of candidates (Wang et al. (2004)). Yang et al. (2003) gave algorithms for ordered tree mining in the context of mining frequent XML query patterns.

Algorithm	Type of input trees	Type of subtrees
Chopper (Wang et al. (2004))	Ordered	Embedded
CMTreeMiner (Chi et al. (2005))	Ordered /Unrooted	Embedded, Maximal/Closed
DRYADE (Termier et al. (2004))	Unrooted	Maximal/Closed
FreeTreeMiner (Chi et al. (2003))	Unordered/Unrooted	Induced
FREQT (Asai et al. (2002))	Ordered	Induced
HybridTreeMiner (Chi et al. (2004b))	Unordered/Unrooted	Induced
PathJoin (Xiao and Yao (2003))	Unrooted	Embedded, Maximal/Closed
Phylominer (Zhang and Wang (2008))	Unordered/Unrooted	Phylogenetic
POTMiner (Jimenez et al. (2010b))	Ordered/Unordered/Partially-ordered	Induced/Embedded
SLEUTH (Zaki (2004))	Unordered	Embedded
TreeMiner (Zaki (2005))	Ordered	Embedded
uFreqt (Nijssen and Kok (2003))	Unordered	Induced
Unot (Asai et al. (2003))	Unordered	Induced
Xspanner Wang et al. (2004)	Ordered	Embedded
EvoMiner	Unordered/Unrooted	Phylogenetic

Table 3.1: Some frequent subtree-mining approaches

They adopted a rightmost expansion approach for candidate generation.

In the field of unordered tree mining, Xiao and Yao (2003) proposed efficient algorithms for discovering frequent subtrees under the assumption that no two siblings have the same label. Unot, by Asai et al. (2003), and uFreqt, by Nijssen and Kok (2003) — proposed independently — use very similar techniques for unordered frequent subtree mining. Both extend the ordered tree mining approach of Asai et al. (2002) by enumerating subtrees in a canonical form. Zaki (Zaki (2004)) extended the efficiency of TreeMiner (Zaki (2005)) for mining unordered embedded subtrees. Chi et al (Chi et al. (2003, 2004b)) studied the unordered tree mining problem for rooted and unrooted trees.

Loosely speaking, phylogenetic tree mining is a kind of unordered embedded subtree mining. However, phylogenetic trees possess a special structure — only leaves are labeled and non-leaf nodes must be of degree two or more (Sect. 3.2.1) — which affects the definition of the subtree operation itself. This demands a specialized data mining approach. To our knowledge Zhang et al's Phylominer (Zhang and Wang (2008)) is the only published algorithm for mining phylogenetic trees; thus, it is the reference point for evaluating our work. Phylominer is an Apriori-like approach that uses rightmost path extension for candidate generation and an occurrence list for frequency counting. It introduces a novel phylogeny-specific canonical form for evolutionary trees, which we also use in our approach. The candidate generation strategy highlights structural properties of phylogenetic trees that are useful for efficient candidate generation.

3.1.1.2 Graph Mining and Itemset Mining

Graph mining (Aggarwal and Wang (2010); Zou et al. (2010); Jia et al. (2011)) is closely related to subtree mining, in fact, tree mining can be viewed as a special case of graph mining. Reference Aggarwal and Wang (2010) gives a comprehensive survey on the topic. Itemset mining (or mining of association rules) (Agrawal et al. (1996); Bhaskar et al. (2010); Liu et al. (2011)) is closely related to both graph mining and tree mining. Graphs and trees can be seen as itemsets with additional constraints resulting from their respective topologies. The classical Apriori (Agrawal et al. (1996)) algorithm has been the most influential in this field — as duly noted in Wu et al. (2008) — and has influenced a lot of subtree mining approaches (Chi et al.

(2004a); Jimenez et al. (2010a)). It is not surprising that it is also the basis of our work.

3.1.1.3 Maximum Agreement Subtrees and Majority Rule Trees

Maximum agreement subtrees (MASTs) (Finden and Gordon (1985); Kao et al. (2001); Amir and Keselman (1994)) and consensus trees (Bryant (2003a); Scornavacca (2009)) are perhaps the approaches most often-used by evolutionary biologists to identify common phylogenetic information in collections of trees. An MAST of a collection of input trees is a common embedded subtree with the largest number of leaves (see Fig. 3.1). Consensus methods aim to find one tree that includes all the species, and captures the common information in the collection of trees. Among the most popular such methods is the majority-rule tree (MRT) (Margush and McMorris (1981a)), defined as follows. A *cluster* in a tree is the set of all leaf descendants of some node in the tree. The MRT of a collection of trees is the tree that exhibits all clusters present in the majority of the input trees (see Fig. 3.2).

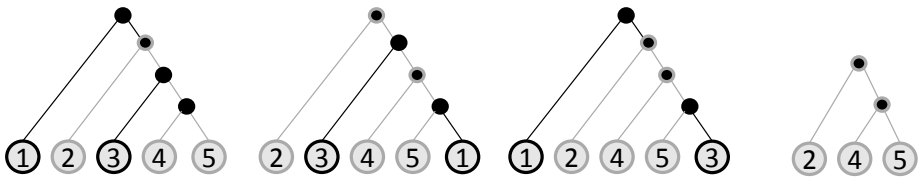
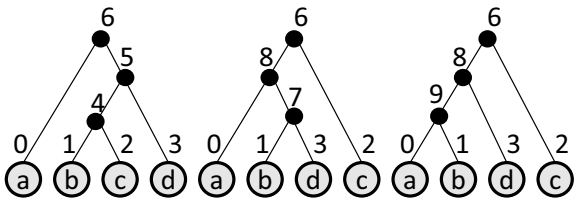


Figure 3.1: (a) A collection of input trees. The lightly shaded part indicates embedding of the common MAST — shown separately in (b).

The MRT and MAST problems have notably different complexities (see Table 3.2). The MRT can be constructed in time that is linear in the total size of the input trees (Amenta et al. (2003)). In contrast, while an MAST of two trees can be computed in polynomial time (Amir and Keselman (1994); Kao et al. (2001)), finding an MAST of three or more trees is NP-hard if the trees have unbounded degree (Amir and Keselman (1994)). The problem is polynomially-solvable when the maximum node-degree is bounded by a constant (Farach et al. (1995a)).

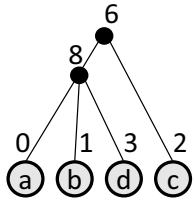
Aside from speed, an important consideration when choosing a method to extract common substructures from collections of trees is the number of leaves in the subtree extracted; the



(a)

Cluster	Frequency
0 - (a)	3
1 - (b)	3
2 - (c)	3
3 - (d)	3
4 - (bc)	1
5 - (bcd)	1
6 - $(abcd)$	3
7 - (bd)	1
8 - (abd)	2
9 - (ab)	1

(b)



(c)

Figure 3.2: (a) A collection of input trees. (b) The clusters in the input trees along with their frequencies. Clusters $0-3, 6$ and 8 occur in the majority of the input trees. (c) The MRT.

larger the better. Here, MRTs have an advantage over MASTs, since the latter often have fewer leaves. MASTs, on the other hand, are in a sense stronger representatives of the common substructures, because each MAST is embedded as a subtree in each tree in the input collection. This cannot be said about the MRT. In fact, the MRT may not be embedded as a subtree in any of the input trees.

Another factor in selecting a method is the degree of resolution of the trees it produces. We say a phylogeny is well-resolved if its number of internal edges is high, relative to its number of leaves. More formally, we define the resolution of a tree T as

$$\text{resol}(T) = \frac{|\text{internal edges in } T|}{|\text{leaves in } T| - 2} \times 100\%. \tag{3.1}$$

(A similar measure was used before in Pattengale et al. (2011).) The denominator in (3.1) is simply the maximum number of internal edges a tree can have, and is used to normalize the degree of resolution across trees of different sizes. The more resolved a phylogeny is, the more informative it is considered to be. The least informative tree we can have is a *fan*, i.e., a star-like tree with zero internal edges. A fan states the trivial fact that the species at its leaves descend from a common ancestor. It has been observed that MRTs tend to be poorly resolved (Ganapathysaravanabavan and Warnow (2001)); MASTs typically perform better than MRTs in this regard.

Reference	Consensus Approach	Max. input trees	Max. node-degree	Complexity
Amenta et al. (2003)	MRT	Any	Any	$O(tn)$
Cole et al. (2000)	MAST	2	2	$O(n \log n)$
Steel and Warnow (1993)	MAST	2	Any	$O(n^{4.5} \log n)$
(Bryant, 1997, pp. 174–182)	MAST	Any	2	$O(tn^3)$
Farach et al. (1995a)	MAST	Any	d	$O(tn^3 + n^d)$
Amir and Keselman (1994)	MAST	Any	Any	NP-hard

Table 3.2: Overview of consensus approaches. t denotes the number of input trees and n the size of the common leaf set.

FSTs offer several advantages over MASTs and MRTs:

- An FST is usually more resolved than the MRT and always has at least as many leaves as an MAST. This is because an MAST is, by definition, also an FST, because it is supported by every input tree. (Note that the MRT is not guaranteed to be an FST.) Indeed, the requirement that an MAST be supported by all input trees rules out subtrees that might be more informative than any MAST but still quite frequent and thus equally important for phylogenetic inference (Zhang and Wang (2008)). An example of this is shown in Fig. 3.3. Here, the FSTs are better resolved than both the MAST and the MRT, which are fans. For instance, the first FST in Fig. 3.3 indicates that $s1$ and $s2$ are closer to each other than each is to any of the species among $s3 - s5$.

- FSTs can reveal a more complete phylogenetic picture than an MAST or an MRT. MASTs and MRTs tend to depict phylogenetic relationships among only a limited subset of the species (Ganapathysaravanabavan and Warnow (2001)). For example, in Fig. 3.4 neither the MRT nor any of the MASTs gives information about relationships among species $s1 - s4$, but the FST does. In fact, by itself, an MAST can be misleading (Swenson et al. (2011)).

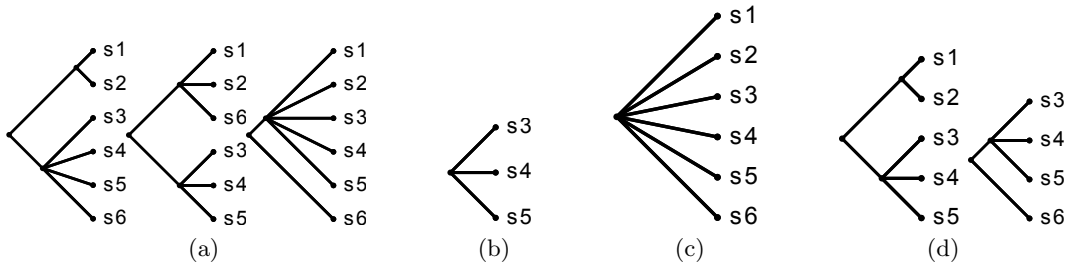


Figure 3.3: (a) A collection of input trees, along with (b) their MAST, (c) their MRT, and (d) two FSTs with 67% support (i.e., two out of the three input trees support them).

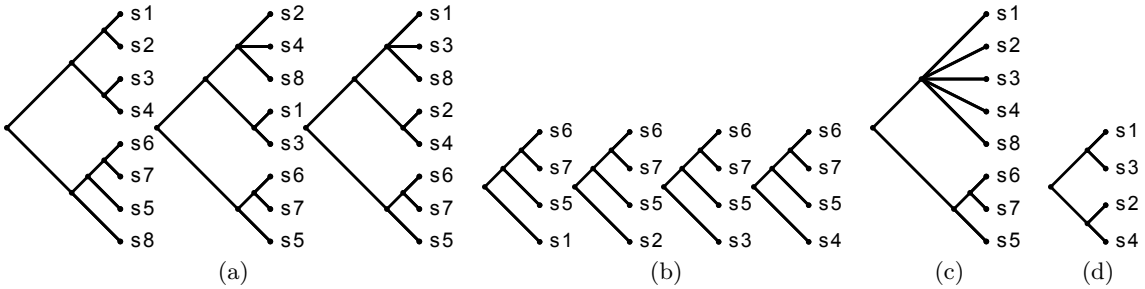


Figure 3.4: (a) A collection of input trees, together with (b) their MASTs, (c) their MRT, and (d) an FST with support 67%.

- FSTs are useful for solving MAST-related problems. These include finding a maximum compatible subtree (Ganapathysaravanabavan and Warnow (2001)), finding a maximum agreement supertree (Guillemot and Berry (2010)), and computing the kernel of maximum agreement subtrees (Swenson et al. (2011)).

- FSTs can be used to mine subtree patterns on collections of trees having leaf sets that are partially overlapping but not identical. This ability is particularly useful in mining phylogenetic databases such as TreeBASE (Piel et al. (2002)) and PhyLoTA (Sanderson et al. (2008)), which contain trees with different leaf sets, and where the number of common leaves is relatively small. Neither MASTs or MRTs are suitable for this purpose. For instance, while we could apply MASTs by simply restricting all the trees to the common leaf set, this would fail to give any results when the common overlap among the leaf sets is low; see Fig. 3.5. Indeed, just a couple of trees with no or very little overlap can result in such a scenario in a collection with any number of trees.

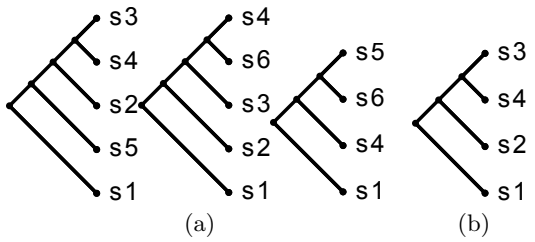


Figure 3.5: (a) A collection of input trees on partially overlapping leafsets. Restricting the input trees to the common leafset (i.e. species $s1$ and $s4$) and then applying MAST or MRT will not yield any useful information. However, (b) the FST contains species $s1 - s4$, which is clearly informative.

3.1.2 Our Contributions

We introduce EvoMiner, an efficient algorithm to mine FSTs in phylogenetic databases. Over a broad range of inputs, EvoMiner is 100 times faster (and often more) than Phylominer (Zhang and Wang (2008)) — the current state-of-the-art algorithm for the same task. The features that enable EvoMiner to achieve this speedup are:

1. An efficient phylogeny-specific constant-time candidate generation scheme, which exploits structural properties to produce fewer potential candidates.

2. A novel fingerprinting based scheme for the downward-closure operation, which in linear time checks support for all k of the $(k-1)$-leaf subtrees.

3. An efficient lowest common ancestor (LCA) based scheme to count support, which neither requires the subtree operation nor a traversal through the database.

EvoMiner works in both a breadth-first candidate enumeration mode (like Apriori (Agrawal et al. (1996))) as well as in depth-first enumeration (Zaki (2005)) mode. The first is quicker, while the second uses less memory, enabling it to handle larger trees. An implementation of EvoMiner, which also works on collections of trees with partially overlapping leaf sets, is available by request from the authors.

Finally, we demonstrate experimentally that, for a wide collection of biological datasets, FST mining (whether or not it is done with EvoMiner) has significant advantages over the

use of MASTs and MRTs. That is, FST mining can produce agreement subtrees that are significantly larger than MASTs and more resolved than the MRT (see Sect. 3.4.2).

3.2 Preliminaries

3.2.1 Phylogenies

As usual, a ***rooted tree*** is a connected acyclic graph, which contains a special node called the ***root***. The ***depth*** of a node u in a rooted tree T, denoted $\text{depth}^T(u)$, is the number of edges from the root to that node; thus the root node is at depth 0. We denote the lowest common ancestor (LCA) of two nodes u and v in T by $\text{LCA}^T(u, v)$. When the tree T is clear from the context, we drop the superscripts. A ***k-leaf tree*** is a tree with k leaves. An edge is called ***internal*** if neither of its end points is a leaf node. Two nodes are called ***siblings*** if they have a common parent node.

A ***phylogenetic tree*** (or, for brevity, simply a ***tree*** or a ***phylogeny***) is a rooted[2] tree where every internal (i.e., non-leaf) node has at least two children, whose leaves are in a bijection with a set of labels. The labels in a phylogenetic tree represent a set of taxonomic units — species — under consideration, and the branching structure of the tree represents the history of the evolution of the species from a common ancestor (Baum (2008)). Indeed, each internal node in a phylogenetic tree represents a hypothetical ancestor or an event (e.g., emergence of a new species) in the evolutionary history that separates the ancestor and descendants at that node.

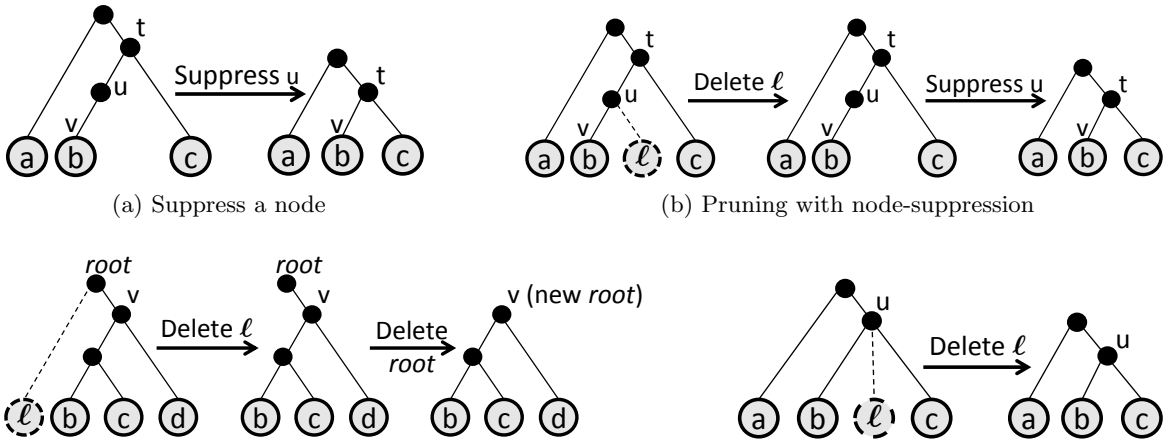
Let $\mathcal{L}_T$ denote the leaf label set of tree T, and ψ_T denote the bijection that maps the leaf nodes to their unique labels. For convenience, we refer to the set of leaf nodes by their labels in $\mathcal{L}_T$. From this point forward, unless the context requires making a distinction, we will drop the subscripts in $\mathcal{L}_T$ and ψ_T, and write $\mathcal{L}$ and ψ respectively. For the rest of the paper, we assume without loss of generality that the leaf label set $\mathcal{L}$ consists of distinct integers in the range $[1, |\mathcal{L}|]$; thus, the labels are ordered.

Let u be an internal non-root node in some tree (not necessarily a phylogenetic tree), such

[2]Phylogenies can also be unrooted (NCBI (2002)), but here we deal exclusively with rooted phylogenies.

that u has only one child v. Then, *suppressing* u means contracting the edge (u, v); i.e., deleting u and adding an edge from the parent of u to v. For example, in Fig. 3.6a, to suppress u, it is deleted and an edge is added from t to v. Node u is a ***neighbor*** of node v if edge (u, v) exists. To *prune* a leaf ℓ, we first delete it. Let u be ℓ's neighbor. If u is not the root, and the deletion of ℓ makes u a degree-two node, we suppress u (see Fig. 3.6b). If u is the root and deleting ℓ makes it a degree one node, u is deleted and its neighbor becomes the new root (see Fig. 3.6c). Otherwise, u remains as it is (see Fig. 3.6d).

Consider a tree T and a set $\mathcal{L}' \subseteq \mathcal{L}_T$. The *restriction* of T to $\mathcal{L}'$, denoted by $T|_{\mathcal{L}'}$, is the tree obtained by pruning leaves $\mathcal{L}_T - \mathcal{L}'$ from T. We denote the fact that two trees T_1 and T_2 are isomorphic by writing $T_1 \equiv T_2$. Given two phylogenies T and T', we say that T ***displays*** T' —or, equivalently, that T' is ***displayed by*** T— if $\mathcal{L}_{T'} \subseteq \mathcal{L}_T$ and $T' \equiv T|_{\mathcal{L}_{T'}}$ (Semple and Steel (2003a)). For example, each of Fig. 3.7a and Fig. 3.7b shows two trees such that the tree on the left displays the tree on the right. For phylogenetic trees, the notion of "displayed by" replaces the usual notion of "embedded subtree" that is commonly used in the data mining literature (e.g., as in Zaki (2005)).



(a) Suppress a node

(b) Pruning with node-suppression

(c) Pruning a leaf attached to a degree-two root

(d) Pruning without node-suppression

Figure 3.6: Pruning in phylogenetic trees

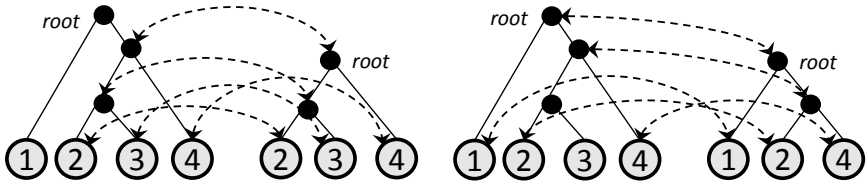
3.2.2 Frequent Subtree Mining in Phylogenetics

Let $D = \{T_1, T_2 \ldots T_n\}$ be a database of n trees on a common label set $\mathcal{L}$. Let minSup $\in [1, n]$ be an input parameter. A tree T with $\mathcal{L}_T \subseteq \mathcal{L}$ is said to be a (phylogenetic) ***frequent subtree*** (FST) in D, if there exists $D' \subseteq D$ with $|D'| >$ minSup such that for all $T' \in D'$, T' displays T. $|D'|$ is called the ***support*** of T in D, and is denoted by sup(T). The (phylogenetic) ***FST mining problem*** is to identify all the FSTs in D.

There are significant differences between phylogenetic FST mining and unordered tree mining. These dissimilarities stem from the peculiarities of phylogenetic trees — only the leaf nodes are labeled and all internal nodes must have at least two children — and from the fact that we are looking for displayed trees rather than embedded subtrees. When applied to collections of phylogenetic trees, ordinary methods for unordered tree mining can yield trees with redundant internal nodes; i.e., nodes with just one child. Figures 3.7a and 3.7b show the same tree with two different displayed trees. Of these two displayed trees, the one in Fig. 3.7b is not an embedded subtree according to the standard definition, since, to obtain it, we not only need to delete a leaf, but we also have to suppress a node. Mining for displayed trees, rather than embedded subtrees makes sense in phylogenetics, where we are primarily interested in the groupings among species. Thus, for the tree in Fig. 3.7b species '2' and '4' are evolutionarily closer to each other than either is to species '1'. The same relative evolutionary information is preserved even if we prune species '3'. This characteristic of phylogenies offers some advantages for FST mining, since it enables us to use efficient LCA-based techniques. At same time it limits the application of existing methods in data mining literature for enumerating frequent subtrees.

We should note that the edges of phylogenetic trees sometimes have numerical values associated to them, called branch lengths, representing the time elapsed or other measure of divergence separating ancestor and descendant during evolution. Dealing with branch lengths is beyond the scope of this paper.

Next, we describe several concepts that are essential to EvoMiner, including canonical form, equivalence classes, and prefix trees. Most of these notions are illustrated in Fig. 3.8.



(a) Without node-suppression (b) With node-suppression

Figure 3.7: Phylogenetic subtree. Arrow mappings indicate the nodes that were retained form the original tree. The topmost node represents the root.

3.2.2.1 Canonical Form

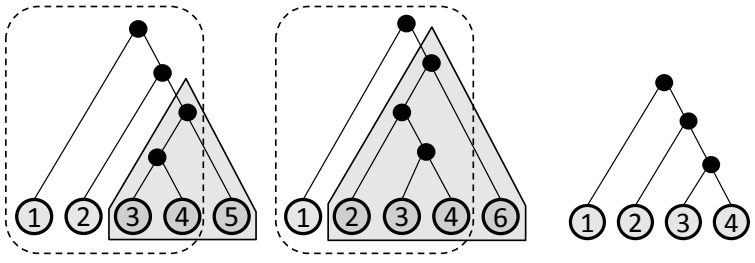
The ***virtual label*** of an internal node v is the minimum label among all leaf descendants of v. The children of an internal node are ordered from left to right based on the sequence in which they are encountered in an inorder depth-first traversal (IDFT), the leftmost child being encountered first. A tree T is in ***canonical form*** (Zhang and Wang (2008)) if, for every internal node, its children are ordered from left to right by their virtual labels. It can be seen that two trees are isomorphic if and only if they have the same canonical forms. By generating all trees in canonical form, it is straightforward to test if two trees are isomorphic and prevent duplicate enumeration. EvoMiner relies on this property to ensure that each FST is enumerated exactly once.

3.2.2.2 Rightmost Leaf, Prefix Tree, and Heaviest Subtree

The ***rightmost leaf*** of tree T is the last leaf encountered in the IDFT of T. A useful property of the canonical form is that pruning either the last leaf or the second-to-last leaf encountered in the IDFT yields a subtree that is also canonical (Zhang and Wang (2008)). The subtree resulting from pruning the rightmost leaf is called the ***prefix tree*** or simply the ***prefix***. The ***heaviest subtree*** (Zhang and Wang (2008)) is the subtree rooted at the parent of the rightmost leaf.

3.2.2.3 Equivalence Class

Let R denote the relation: 'sharing a common prefix' between two canonical trees. Note that R is an equivalence relation. Two canonical trees T and T' are said to be equivalent with respect to R (or simply equivalent), denoted $T \sim T'$ if they share a common prefix. An ***equivalence class*** E is a set of canonical trees that are pair-wise equivalent; i.e., all trees in E share a common prefix tree; that is, for every pair $(T, T') \in E, T \sim T'$. Thus, all trees in E share a common prefix tree. The shared $(k-1)$-leaf prefix tree, called the ***core tree***, uniquely identifies the members of an equivalence class. Any two trees in an equivalence class differ with respect to their rightmost leaf and (topologically) with respect to their heaviest subtrees. The partition of the set of frequent k-leaf trees into equivalence classes is the basis for our enumeration approach, which generates larger frequent subtrees by extending the core tree (Sect. 3.3.1).



(a) Equivalence class (b) Prefix tree

Figure 3.8: (a) Two trees belonging to the same equivalence class. The common prefix tree (shown separately in (b)) is encircled by a dotted line; the respective rightmost leaves are the ones outside the dotted line. The shaded part represents the respective heaviest subtrees.

3.3 The EvoMiner Algorithm

Figure 3.9 gives a high-level description of EvoMiner. The algorithm uses an Apriori-like (Agrawal et al. (1996)) candidate generation scheme, and uses breadth-first search to enumerate frequent subtrees. EvoMiner begins by computing the LCA mappings for every tree in the input database D. That is, for each tree T in D, and every pair $\{u, v\}$ of leaves of T, it computes $\text{LCA}(u, v)$ and stores it in a three-dimensional array indexed by triplet (T, u, v). In

our implementation, these LCA values are computed in quadratic time and space per tree by traversing the tree in a depth-first manner and computing the LCA values of the leaf-descendants at a node. For a given database D on the leaf set $\mathcal{L}$, this one-time task takes $O(|D||\mathcal{L}|^2)$ time. We should point out that it is well-known that one can pre-process a tree in linear time and space to produce a data structure that can answer any LCA query on that tree in constant time (Harel and Tarjan (1984); Schieber and Vishkin (1988); Bender and Farach-Colton (2000)). Such algorithms are quite useful when the number of LCA queries is limited and the pre-processing dominates the total time. That is not the case in our application. Indeed, EvoMiner queries all possible LCA values while enumerating all FSTs on three leaves, and then does a constant number of LCA queries for every FST generated thereafter. Although both our three-dimensional array and the specialized LCA data structures (Harel and Tarjan (1984); Schieber and Vishkin (1988); Bender and Farach-Colton (2000)) offer constant-time access to LCA-values, the former's constant factor is smaller than the latter's, which makes a significant difference in practice.

After the LCA mappings are computed, EvoMiner repeatedly alternates between two steps until all FSTs are enumerated. The first step is ***candidate generation***, which provides a set of potential frequent candidate trees. This is done so that each frequent subtree is enumerated only once. The second step is ***frequency counting***, which examines the candidate trees, identifying the frequent subtrees among them. This is a potentially time-consuming operation, since it can involve frequent traversals through the input database and subsequent subtree operations. Next, we describe how each of these steps is implemented in EvoMiner.

3.3.1 Candidate Generation

We denote the set of all equivalence classes on frequent k-leaf trees by EC_k. The input for the candidate generation step is an equivalence class in EC_k. The output is a set of potential candidate subtrees on $k+1$ leaves extending the k-leaf trees in the equivalence class. The candidate generation strategy has two parts. The first is pairwise joining of frequent subtrees within an equivalence class to produce larger candidate trees (sometimes referred to as equivalence class based extension (Zaki (2005))). This is achieved in constant time per

EvoMiner(D, minSup)

computeLCA_Mappings(D)
$Ft \leftarrow$enumerateFrequentTriplets(D, minSup)
$EC_3 \leftarrow$ computeEquivalenceClasses(Ft)
$E_{\text{next}} \leftarrow EC_3$, Result $\leftarrow \emptyset$
while $E_{\text{next}} \neq \emptyset$ **do**
  $E_{\text{next}} \leftarrow$enumerateNextLevel($E_{\text{next}}$, minSup)
  Result $\leftarrow$ Result $\cup E_{\text{next}}$
return Result

enumerateNextLevel(EC_k, minSup)

$EC_{k+1} \leftarrow \emptyset$
for all $e \in EC_k$ **do**
  for all $T_x \in e$ **do**
    $e_{\text{next}} \leftarrow \emptyset$
    for all $T_y \in e$ such that $T_x \neq T_y$ **do**
      candidates $\leftarrow$ join(T_x, T_y)
      for all $T^{\text{join}} \in$ candidates **do**
        if downwardClosure(T^{join}) **then**
          if sup(T^{join}) $>$ minSup **then**
            $e_{\text{next}} \leftarrow e_{\text{next}} \cup T^{\text{join}}$
    $EC_{k+1} \leftarrow EC_{k+1} \cup e_{\text{next}}$
return EC_{k+1}

Figure 3.9: The EvoMiner algorithm

pair. The second part is a linear-time pruning of generated candidate trees through a downward closure operation, which tests whether all k-leaf subtrees of a given $(k+1)$-leaf tree are frequent (Agrawal et al. (1996); Chi et al. (2003)). The enumeration of FSTs starts with *triplets* (trees on three leaves), as triplets are the smallest trees that offer meaningful evolutionary information.

3.3.1.1 Pairwise Extension

Let $\langle T_x, T_y \rangle$ be an ordered pair of distinct frequent k-leaf trees from the same equivalence class e of EC_k. The pairwise extension operation "joins" $\langle T_x, T_y \rangle$ in all possible ways so as to generate a set join(T_x, T_y) of $(k+1)$-leaf candidate trees, called ***joined trees***, such that every $T^{\text{join}} \in$ join(T_x, T_y) satisfies the following:

$$T^{\text{join}} \text{ is in canonical form, } T_x \text{ is the prefix of } T^{\text{join}}, \text{ and } T_y \equiv T^{\text{join}}|_{\mathcal{L}_{T_y}}. \tag{3.2}$$

That is, every such tree T^{join} is a $(k+1)$-leaf tree having T_x as its prefix and T_y as its subtree. Thus, T^{join} belongs to the equivalence class with T_x as its core tree. Since T_x and T_y are in the same equivalence class, they differ only with respect to their heaviest subtrees. This fact restricts the possible ways in which they can be joined.

Condition (3.2) implies that T^{join} is obtained by attaching the rightmost leaf of T_y to the rightmost path of T_x (the path from the root to the rightmost leaf); this is known as a *rightmost path extension* (Asai et al. (2003)). Let x and y denote the rightmost leaf of T_x and T_y respectively, p_x and p_y denote the parents of x and y respectively, and T^{core} represent the core of the equivalence class e. For an internal node u, let numChild(u) denote its number of children. To satisfy (3.2), T^{join} can have one of four possible topologies, which we refer to as type 1–4 joins. These are described next.

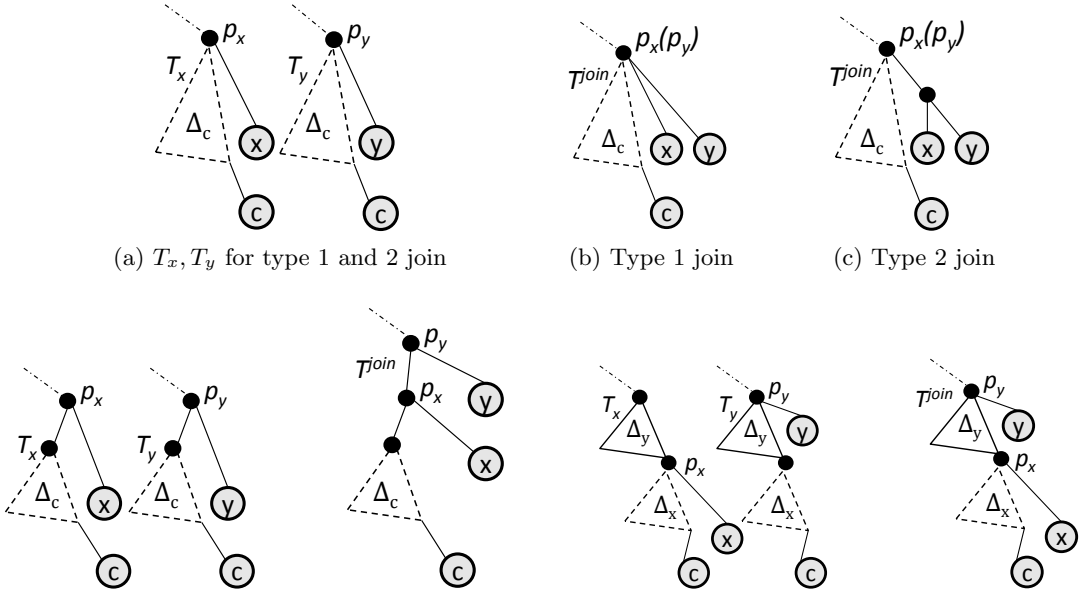
Type 1: In this case, as shown in Fig. 3.10a, the leaves x and y of the participating trees are attached at the same depth on the rightmost path of T^{core}; i.e., depth(p_y) = depth(p_x). In T^{join}, x and y are siblings, and their depths are the same as in T_x and T_y; see Fig. 3.10b. For T^{join} to be canonical, we must have $\psi(x) < \psi(y)$ (recall that we assume that the labels are distinct numbers). Figures 3.11a and 3.11b exemplify type 1 joins.

Type 2: As in type 1 joins, depth(p_y) = depth(p_x) and x and y are siblings in T^{join} (Fig. 3.10a). In T^{join}, however, each of x and y is one level deeper than it was in T_x and T_y; see Figure 3.10c. Thus, pruning either x or y in T^{join} leaves the parent with only one child, so the parent is suppressed. (This suppression does not occur in Type 1 joins, because the common parent of x and y in T^{join} must have degree greater than two; see Figure 3.10b.) Figures 3.11a and 3.11c exemplify type 2 joins.

Type 3: Fig. 3.10d shows the participating trees. As in type 1 and 2 joins, depth(p_y) = depth(p_x). In this case, however, p_y is the parent of p_x in T^{join}; see Figure 3.10e. Since pruning x in T^{join} causes its parent node to be suppressed (see Fig. 3.10e), the only way we can get this T^{join} is if numChild(p_y) = numChild(p_x) = 2. Figures 3.11d and 3.11e exemplify type 3 joins.

Type 4: In this case, as shown in Fig. 3.10f, depth(p_y) < depth(p_x). Thus, the depth of y on the rightmost path of T^{core} is lower than that of x. As a result there is only one way to join T_x and T_y so as to satisfy condition (3.2). See Fig. 3.10g. Note that p_y becomes an ancestor of p_x in T^{join}. Figures 3.11f and 3.11g exemplify type 4 joins.

Observe that if depth(p_y) > depth(p_x), we cannot join T_x and T_y while satisfying condition (3.2), since T_x cannot be the prefix tree in this case. FSTs from such joins are enumerated when considering the ordered pair $\langle T_y, T_x \rangle$.

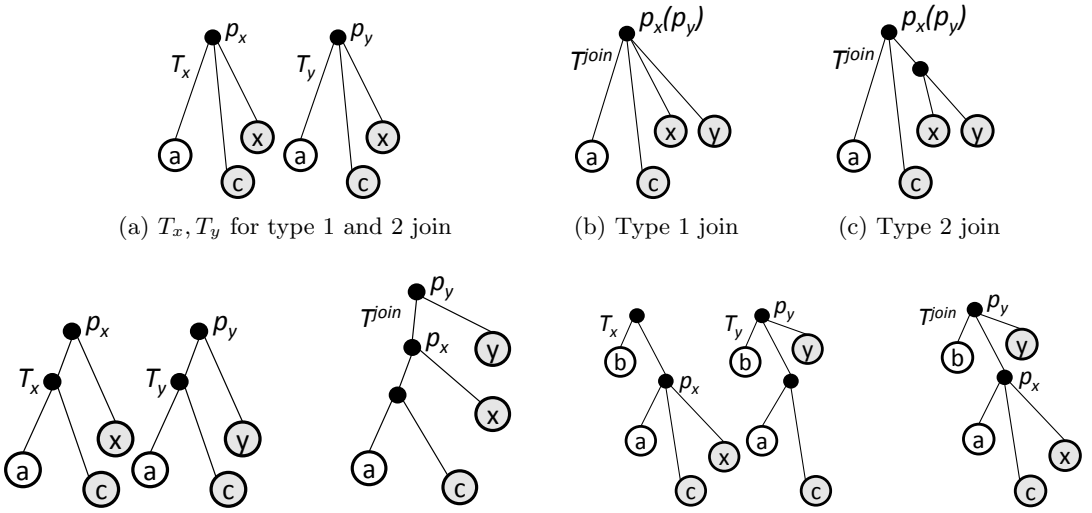


(a) T_x, T_y for type 1 and 2 join (b) Type 1 join (c) Type 2 join

(d) T_x, T_y for type 3 join (e) Type 3 join (f) T_x, T_y for type 4 join (g) Type 4 join

Figure 3.10: Different types of pairwise join. A dotted triangle represents a part of the tree that may be empty, while a solid triangle represents a non-empty part of the tree. Δ reflects the topologies of the heaviest subtrees. 'c' denotes the rightmost leaf of the common core tree.

Clearly, we can identify in constant time the type(s) of join resulting from an ordered pair of trees in an equivalence class and the tree resulting from each case is canonical. Hence, a join can be done in constant time for a pair of input trees. This is an important difference with respect to Phylominer, where the candidate generation scheme requires comparing the respective topologies of the heaviest subtrees of the input trees, which takes $O(k)$ time. Another difference is that by comparing depth(p_y) with depth(p_x), we generate fewer candidate trees, because fewer cases are considered for each possibility. This means that fewer trees go to the



(a) T_x, T_y for type 1 and 2 join (b) Type 1 join (c) Type 2 join

(d) T_x, T_y for type 3 join (e) Type 3 join (f) T_x, T_y for type 4 join (g) Type 4 join

Figure 3.11: An example for each of the join type shown in Fig. 3.10.

frequency counting step, which saves further time.

Theorem 1. *Pairwise extension enumerates all FSTs and each FST is enumerated only once. Moreover the enumerated FSTs are in canonical form.*

Proof. We use induction on k, the number of leaves in an FST. If $k = 3$, then all such FSTs are uniquely generated during triplet enumeration, the starting step of Algorithm 3.9. Consider $k > 3$. Assume all FSTs on $k - 1$ leaves have been uniquely enumerated. Let T be a k-leaf FST in canonical form. Let T_x and T_y respectively be the subtrees obtained by pruning the last leaf and the second last leaf in the IDFT of T. Since T is in canonical form, so are T_x and T_y. Clearly T_x and T_y are FSTs on $k - 1$ leaves and share the same prefix tree. Thus, they must have been uniquely enumerated and must belong to the same equivalence class e. Since T satisfies condition (3.2) with respect to T_x and T_y, we have $T \in \text{join}(T_x, T_y)$. Thus pairwise extension must enumerate T. Further, if the members of e are considered in an ordered fashion for pairwise extension so that $\langle T_x, T_y \rangle$ is considered only once, then T will also be enumerated only once. □

3.3.1.2 Downward-Closure Operation

The downward closure operation takes a k-leaf candidate tree T (generated by pairwise extension) and checks whether all k of its $(k-1)$-leaf subtrees are frequent. Thus, it requires all $(k-1)$-leaf frequent subtrees to have been enumerated beforehand. EvoMiner uses an Apriori-like level-wise approach. That is, EC_{k+1} is enumerated only after EC_k has been enumerated. A common approach for the downward-closure operation is to first generate all k of the $(k-1)$-leaf subtrees and then check if each subtree is frequent by indexing it into an efficient data structure (Agrawal et al. (1996); Zhang and Wang (2008)). This requires at least $O(k^2)$ time, as indexing into any data structure will take at least $O(k)$ time and there are $O(k)$ subtrees to check. Things can get more complex if the subtrees themselves need to be checked for isomorphism (Zhang and Wang (2008)). We next present an efficient fingerprinting-based scheme that checks in $O(k)$ time whether all k of the $(k-1)$-leaf subtrees are frequent.

Fingerprint generation. Fingerprinting is a mechanism that maps a large data item to a much shorter bit string. A good fingerprint function has a very small probability of mapping two different data items to the same bit string. This probability is inversely affected by the size of the bit string, which is generally a constant for a given application. Our approach involves generating fingerprints for all frequent subtrees in EC_{k-1} and storing them in a hash table. Given a k-leaf candidate tree, to check if one of its $(k-1)$-leaf subtree is frequent, we check if its fingerprint is present in the hash table. In our computational experiments, the fingerprinting based pruning technique enables us to achieve speed-ups up to 100% as compared to an alternative depth-first mining approach (see Sect. 3.3.5).

We employ the fingerprint function used in the Rabin–Karp pattern-matching algorithm (Karp and Rabin (1987)). For this, we represent each tree in the database by the sequence of nodes encountered in the IDFT of the tree (this traversal sequence is called an Euler tour). For example, the string representation of the tree shown in Fig. 3.8b is 'D1UDD2UDD3UD4UUU', where 'D'/'U' respectively represent downward/upward traversal in the tree, and are selected such that $\mathrm{D}, \mathrm{U} \notin \mathcal{L}$. Clearly, the size of the string representation is of the same order as the size of the tree and it uniquely identifies an ordered tree. Since frequent subtrees are enumerated

in canonical form, they are always ordered. Thus any frequent subtree is uniquely identified by its string representation. The fingerprint function interprets a b-bit string as a b-bit integer. The fingerprint of a b-bit string $s = (s_1 s_2 \dots s_b)$, denoted $F_p(s)$, is computed as:

$$F_p(s) = \left(\sum_{i=1}^{b} 2^{i-1} s_i \right) \mod p,$$

where p is a random prime. While the cost of computing the original fingerprint for a b-bit integer is $\Theta(b)$, a fingerprint can be updated in constant time after deletion of a substring (Motwani and Raghavan (1995)). For example given the fingerprint of the string '123456', we can compute the fingerprint of the result of deleting substring '34' by observing that $F_p(\text{'1256'}) = (F_p(\text{'123456'}) - F_p(\text{'1234'}) \times 2^{\text{length}(\text{'56'})} + F_p(\text{'12'}) \times 2^{\text{length}(\text{'56'})}) \mod p$, which is easy if the fingerprints for the prefix strings '1234' and '12' have been pre-computed.

We note that in many applications of fingerprinting, the prime is chosen randomly so as to avoid an attempt by an adversary to select strings s_1 and s_2 such that $s_1 \neq s_2$ but $F_p(s_1) = F_p(s_2)$. In our case, however, it is reasonable to assume that the input distribution is oblivious to p. Hence, we use a fixed prime p. This allows us to select a large p. This is helpful because for two random input strings and a fixed prime p, the probability that the two strings have the same fingerprint is $\frac{1}{p}$. By selecting a large p, we keep this probability low. The size of the fingerprint is clearly constant for a chosen prime p.

Given a k-leaf tree, calculating the fingerprint of one of its $(k-1)$-leaf subtrees involves deleting the corresponding leaf and extracting the fingerprint from the fingerprint of the original tree in constant time. As shown in Fig. 3.12, there are two possible cases for pruning a leaf ℓ. In the first case (Fig. 3.12a), no node is suppressed and the string representation changes from '...DDaU$D\ell U$DbUU...' to '...DDaUDbUU...', where a and b are the siblings of ℓ. This involves deletion of substring $D\ell U$ (emphasized in *italics* in the original string). The fingerprint is updated as:

$$\begin{aligned} F_p(\text{'...DDaUDbUU...'}) = (&F_p(\text{'...DDaU}D\ell U\text{DbUU...'}) \\ &- F_p(\text{'...DDaU}D\ell U\text{'}) \times 2^{\text{length}(\text{'DbUU'})} \\ &+ F_p(\text{'...DDaU'}) \times 2^{\text{length}(\text{'DbUU'})}) \mod p. \end{aligned}$$

In the second case (Fig. 3.12b), a node is suppressed and the string representation changes from '...DDaUD$D\ell UD$bUUU...' to '...DDaUDbUU...'. This involves deletion of two substrings: '$D\ell UD$' and 'U' (emphasized in *italics* in the original string). Each substring is deleted one at a time. On deleting '$D\ell UD$' from '...DDaUD$D\ell UD$bUUU...', the fingerprint is updated as:

$$\begin{aligned} F_p(\text{'...DDaUDbUUU...'}) = (&F_p(\text{'...DDaUD}D\ell UD\text{bUUU...'}) \\ &- F_p(\text{'...DDaUD}D\ell UD\text{'}) \times 2^{\text{length('bUUU')}} \\ &+ F_p(\text{'...DDaUD'}) \times 2^{\text{length('bUUU')}}) \bmod p. \end{aligned}$$

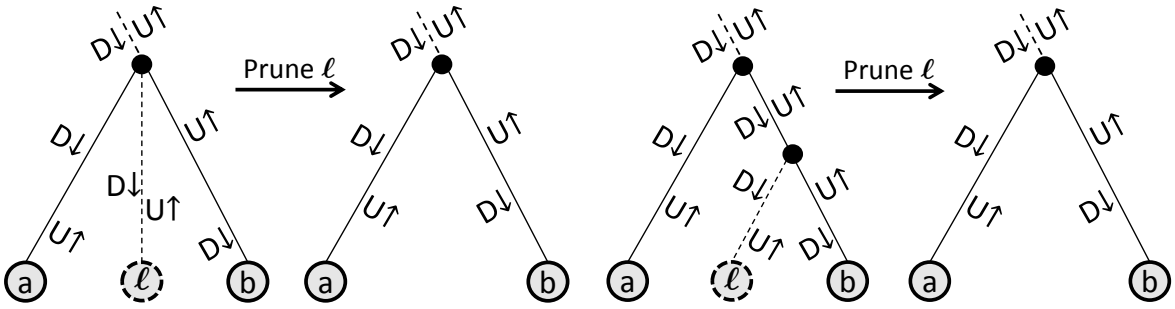
Next, on deleting the second substring 'U' from '...DDaUDbUUU...', the fingerprint is updated as:

$$\begin{aligned} F_p(\text{'...DDaUDbUU...'}) = (&F_p(\text{'...DDaUDb}U\text{UU...'}) \\ &- F_p(\text{'...DDaUDb}U\text{'}) \times 2^{\text{length('UU')}} \\ &+ F_p(\text{'...DDaUDb'}) \times 2^{\text{length('UU')}}) \bmod p. \end{aligned}$$

Each of the above updates can be achieved in constant time if the fingerprints of all the prefixes of the string representation of the original tree is pre-computed. Computing all prefixes takes $O(k)$ time for a k-leaf tree. With this information, computing the fingerprint corresponding to the deletion of a leaf takes constant time. Thus, the fingerprints for all $(k-1)$-leaf subtrees can be computed in $O(k)$ time. What remains is a lookup in the hash table to check if the subtree is frequent. If any of the $(k-1)$-leaf subtrees is not frequent, then the candidate tree is pruned away from enumeration.

When using fingerprints as just described, there is a small probability of a false match[3]; i.e., two different subtrees having the same fingerprint. Since the number of frequent subtree patterns is often large, we want to further strengthen the fingerprinting scheme. To do so, we hash the leaf sets of the subtrees and store the hash codes along with the fingerprints in the hash table. Thus we only compare two subtrees when they share the same leaf set. In our experiments, we never encountered duplicate values in the hash table when using this strengthened scheme. While theoretically there is still a chance of a false match, these errors

[3]Recall that this probability is $\frac{1}{p}$ for a chosen prime p (Sect. 3.3.1.2, page 37).



(a) Without node-suppression (b) With node-suppression

Figure 3.12: Fingerprint update in pruning a leaf

are eventually detected in the frequency counting step. Further, when amortized, false matches do not affect the overall complexity. Note that false matches do not affect the correctness of the algorithm since a false match never prunes away a frequent subtree from enumeration.

Note that any subtree obtained by deleting a leaf that is the leftmost child of its parent may not be canonical. This is because in canonical form the virtual label of a node is the same as its leftmost child. Deleting such a leaf results in a new leftmost child for its parent, and hence, a new virtual label, which is greater than the previous one. This new virtual label of the node may be greater than the virtual label of its next sibling in the IDFT of the tree — requiring repositioning of the node among its siblings to restore the canonical form. This repositioning effect can cascade all the way to the root if each node on the path from the leaf being deleted to the root is the leftmost child of its parent. Thus our fingerprinting scheme does not consider any such subtree. In the worst case, only half of the subtrees are considered. This leads to the possibility of a *false positive* with respect to the downward closure operation; i.e., referring a k-leaf candidate to the frequency counting step even though it has an infrequent $(k-1)$-leaf subtree. However, in our experiments we found a low false positive rate (.01 to 4%) when compared to a complete downward-closure check, which considers all $(k-1)$-leaf subtrees (see Sect. 3.4.1). The explanation for the low rate of false positives is that our problem empirically exhibits an *abundance of witnesses* property (Hromkovič, 2005, chapter 6). The goal here is to identify a 'guilty' infrequent candidate k-leaf tree posing as a frequent one. A witness here is an infrequent $(k-1)$-leaf subtree of the candidate tree that witnesses against the latter being

frequent. However, every $(k-1)$-leaf subtree is not infrequent, i.e., a witness. Thus, the more $(k-1)$-leaf subtrees are checked for being infrequent, the higher are the chances of coming across a witness. Our experiments show that the witness count, even when only considering half of the subtrees, remains abundant. Thus, a complete downward-closure check, while thorough, increases the running time without significantly improving the amount of pruning achieved. Note that this is only a pruning step. A false positive here only means that some candidates that could have been pruned by the exhaustive check were not pruned in this step. This in no way affects the correctness of the algorithm because these false positives will be eventually detected in the frequency counting step.

3.3.2 Frequency Counting

For each frequent subtree t, we maintain an occurrence list containing all the trees in the database that have t as a subtree. While considering a tree $T^{\text{join}} \in \text{join}(T_x, T_y)$, we first compute the intersection of the occurrence lists of T_x and T_y. We then count how many trees in the intersection display T^{join}. Let $\mathcal{L}^{\cup} = \{\mathcal{L}_{T^{\text{core}}} \cup x \cup y\}$ denote the leaf set of T^{join}, where T^{core} is the common prefix tree of T_x and T_y. For each tree T in the intersection list, EvoMiner uses an LCA-based scheme to determine in constant time whether T^{join} is displayed by T —i.e., if $T^{\text{join}} \equiv T|_{\mathcal{L}^{\cup}}$— without actually computing $T|_{\mathcal{L}^{\cup}}$. In contrast, Phylominer takes linear time for the corresponding step, because for every tree T in the intersection list, it explicitly constructs $T|_{\mathcal{L}^{\cup}}$ and then checks if this tree is isomorphic to T^{join} (using the linear-time algorithm of Wang et al. (2005)).

For a given pair $\langle T_x, T_y \rangle$ of trees in an equivalence class, and a tree T in the intersection of the occurrence lists of T_x and T_y, the subtree $T|_{\mathcal{L}^{\cup}}$ must be the result of one of the four types of joins on $\langle T_x, T_y \rangle$ as described in Sect. 3.3.1.1. We next give precise conditions based on the LCA values for each of the four cases. The meaning of the symbols c, x, y, p_x and p_y is the same as in Sect. 3.3.1.1.

Lemma 2. *$T|_{\mathcal{L}^{\cup}}$ is the result of a type 1 join if and only if*

1. $\text{depth}(\text{LCA}^T(c, x)) = \text{depth}(\text{LCA}^T(c, y))$,

2. $\text{depth}(\text{LCA}^T(c, x)) = \text{depth}(\text{LCA}^T(x, y))$, *and*

3. $\psi(x) < \psi(y)$.

Proof. Clearly if $T|_{\mathcal{L}^\cup}$ is the result of a type 1 join, it satisfies conditions 1–3. To prove the *only if* part, let conditions 1–3 be satisfied. Since T_x and T_y are obtained by attaching x and y respectively to the rightmost path of T^{core}, condition 1 implies that $\text{depth}(p_y) = \text{depth}(p_x)$. Thus, $T|_{\mathcal{L}^\cup}$ must be a join of type 1, 2 or 3. Now, a type 3 join requires the parent of y to be an ancestor of the parent of x in $T|_{\mathcal{L}^\cup}$ — a case ruled out by condition 1. Further, conditions 2 and 3 imply that the join must be of type 1. □

Lemma 3. *$T|_{\mathcal{L}^\cup}$ is the result of a type 2 join if and only if*

1. $\text{depth}(\text{LCA}^T(c, x)) = \text{depth}(\text{LCA}^T(c, y))$,

2. $\text{depth}(\text{LCA}^T(c, x)) < \text{depth}(\text{LCA}^T(x, y))$, *and*

3. $\psi(x) < \psi(y)$.

Proof. The proof is similar to that of Lemma 2. Again $T|_{\mathcal{L}^\cup}$ can be the result of either a type 1 or a type 2 join. Conditions 2 and 3 imply that the join must be of type 2. □

Lemma 4. *$T|_{\mathcal{L}^\cup}$ is the result of a type 3 join if and only if*

1. $\text{depth}(\text{LCA}^T(c, x)) > \text{depth}(\text{LCA}^T(c, y))$, *and*

2. $\text{depth}^{T_x}(p_x) = \text{depth}^{T_y}(p_y)$.

Proof. Condition 2 implies that $T|_{\mathcal{L}^\cup}$ must be the result of a join of type 1, 2 or 3. Condition 1 rules out type 1 and 2 joins. Thus, the join must be of type 3. □

Lemma 5. *$T|_{\mathcal{L}^\cup}$ is the result of a type 4 join if and only if* $\text{depth}^{T_x}(p_x) > \text{depth}^{T_y}(p_y)$.

Proof. As per the given condition, $T|_{\mathcal{L}^\cup}$ can only be the result of a type 4 join. □

Theorem 6. *The frequency counting scheme correctly identifies $T|_{\mathcal{L}^\cup}$ as a result of a join of type 1, 2, 3 or 4 in constant time.*

Proof. Clearly, Lemmas 2–5 are mutually exclusive and correctly identify $T|_{\mathcal{L}^\cup}$ as a result of a join of type 1, 2, 3 or 4. Further, each condition in the lemmas can be evaluated in constant time. The claim follows. □

3.3.3 Correctness and Complexity Analysis

Theorem 1 proves that pairwise extension uniquely enumerates all potential FSTs. Theorem 6 proves that the frequency counting step correctly identifies all true FSTs out of the potential candidates. Thus, EvoMiner identifies all FSTs correctly and non-redundantly. We next discuss the time complexities of the different steps of EvoMiner (EM) and compare the total time with that of Phylominer (PM). In the process, we explain why EM is faster than PM. As before, let the input database be D, consisting of n trees on a common leafset $\mathcal{L}$. Let $\mathcal{F}$ denote the set of all FSTs.

Initialization. This one-time task involves (1) computing LCA mappings for all pairs of leaves for all the input trees and (2) enumerating all frequent triplets. Step 1 takes $O(n|\mathcal{L}|^2)$ time. Though this step is not done by PM, it takes only a small fraction of the total time, as $|\mathcal{L}|^2$ is negligible compared with $|\mathcal{F}|$, the total number of frequent patterns. Step 2 takes $O(n|\mathcal{L}|^3)$ time. PM starts by evaluating all frequent pairs of leaves instead of triplets. However, because it enumerates all FSTs, PM also enumerates all frequent triplets. Thus the net complexity for evaluating all frequent triplets is the same in PM as in EM.

Candidate Generation. Both EM and PM use pairwise extension. In EM candidates are generated in constant time. In PM it takes $O(k)$ time to join two k-leaf candidate trees as PM compares the topologies of the heaviest subtrees of the two candidates. Further, by exploiting the structural properties of the candidates, EM generates fewer potential candidates than PM.

Downward Closure. Both EM and PM use downward closure to prune infrequent candidates. For a k-leaf candidate tree, PM checks whether all k of its $(k-1)$-leaf subtrees are frequent by referring to a hash table that stores the previously enumerated frequent $(k-1)$-leaf

subtrees. This takes $O(k^2)$ time, as there are $O(k)$ subtrees and indexing each takes $O(k)$ time. EM does the same operation in $O(k)$ time through its fingerprinting scheme.

Frequency Counting. Both EM and PM use occurrence lists. For a potential k-leaf frequent candidate, PM examines every tree T in the intersection of the occurrence lists, computing the restriction of T to the leaf set of the candidate and then comparing this restricted tree to the candidate using a linear-time isomorphism check. PM takes $O(k)$ time for each of these steps. EM avoids computing the restriction, as well as the isomorphism test. Instead, it performs frequency counting in constant time via its LCA-based scheme. Overall, for each potential candidate, frequency counting in PM takes $O(nk)$ time while it takes $O(n)$ time in EM.

Theorem 7. *The time complexity of* EvoMiner *is* $O(n|\mathcal{L}|^3 + n|\mathcal{F}| + |\mathcal{L}||\mathcal{F}|)$ *where* n *is the number of trees in the database,* $\mathcal{F}$ *is the set of FSTs, and* $\mathcal{L}$ *is the common leaf set. When, as typically happens in practice,* $|\mathcal{F}| \gg |\mathcal{L}|^3$*, the time is* $O(n|\mathcal{F}| + |\mathcal{L}||\mathcal{F}|)$.

Proof. As discussed, the time complexity for initialization step is $O(n|\mathcal{L}|^2) + O(n|\mathcal{L}|^3) = O(n|\mathcal{L}|^3)$. Each k-leaf candidate generation takes $O(k)$ time, which is $O(|\mathcal{L}|)$. The downward closure operation for each k-leaf candidate takes $O(k)$ time, which is $O(|\mathcal{L}|)$. The frequency counting step takes $O(n)$ time for each candidate. Totaling these estimates, we obtain the claimed bound. □

3.3.4 Extension to Partially Overlapping Leaf Sets.

So far, we have assumed that all trees in the database have the same leaf set. The sets of trees considered by evolutionary biologists often have only partially overlapping leaf sets. We can easily extend EvoMiner to mine such collections of trees, without any loss in efficiency. We do so as follows. While computing the LCA for all pairs of leaves for all the input trees during the initialization phase, we flag an LCA value if one of the leaves involved in the pair is not present in the input tree. These flagged LCA values are not considered during frequency counting. Note that such an extension is not as direct in Phylominer, as its frequency counting step is not based on LCA values.

3.3.5 Depth-first Mining

We can use many of the ideas behind EvoMiner in a depth-first mining scheme, along the lines of Zaki (2005), which uses depth first search in the enumeration graph (Chi et al. (2004a)). This alternative scheme does not require candidate generation (similar to Wang et al. (2004); Han et al. (2000, 2004)), since the frequency counting step gives sufficient conditions to distinguish among all possible candidates from pairwise-extension. Fig. 3.13 gives a high-level description of the depth-first enumeration scheme. The existence of a joined-subtree in line 5 can be checked in constant time per tree using the conditions given in Lemmas 2-5. We note that while depth-first mining does not benefit from efficient pruning through downward closure (which can be very effective as the number of trees in the database becomes large), it uses less memory than the breadth-first approach, allowing very large trees to be mined. This is because to extend a k-leaf tree, we only need to store its ancestor equivalence classes, unlike the breadth-first enumeration approach where all the equivalence classes of the previous level must be stored. In the next section, we give a bound on the memory required by the depth-first mining scheme. In Sect. 3.4, we give the results of an experimental evaluation of the trade-offs between depth-first and breadth-first mining.

EvoMinerDF(D, minSup)
1: computeLCA_Mappings(D)
2: $Ft \leftarrow$enumerateFrequentTriplets(D, minSup)
3: $EC_3 \leftarrow$ computeEquivalenceClasses(Ft)
4: **for all** $e \in EC_3$ **do**
5: depthFirstRecursive(e, minSup)

depthFirstRecursive(e, minSup)
1: **for all** $T_x \in e$ **do**
2: **print** T_x
3: $e^{T_x} \leftarrow \emptyset$
4: **for all** $T_y \in e$ such that $T_x \neq T_y$ **do**
5: **if** greater than minSup trees in D exhibit a common subtree T_{xy} over $\mathcal{L}_{T_x} \cup \mathcal{L}_{T_y}$ with T_x as its prefix **then**
6: $e^{T_x} \leftarrow e^{T_x} \cup T_{xy}$
7: **if** $e^{T_x} \neq \emptyset$ **then**
8: depthFirstRecursive(e^{T_x}, minSup)

Figure 3.13: EvoMinerDF — depth-first enumeration

3.3.6 Discussion

Apriori-based methods for mining frequent patterns — such as association rules, frequent closed itemsets, max-patterns, sequential patterns, or constraint-based mining of frequent patterns — can generate an exponential number of candidates for a frequent pattern (see Han and Pei (2000); Geerts et al. (2005)). It turns out, however, that EvoMiner does not suffer from a similar combinatorial explosion. In fact, as we show next, the number of candidates generated in the enumeration of an FST is polynomially-bounded.

Theorem 8. *The number of candidates generated in the enumeration of an FST is $O(|\mathcal{L}|^3)$, where $\mathcal{L}$ is the common leaf set of the input trees. The number of candidates generated per FST is $O(|\mathcal{L}|)$.*

Proof. An FST can have at most $|\mathcal{L}|$ leaves. Each such leaf is added during a pair-wise join within an equivalence class. Within an equivalence class, the maximum number of pairs that can participate in a join operation is $\binom{|\mathcal{L}|}{2}$, which is less than $|\mathcal{L}|^2$. Each such pair can result in a maximum of three types of joined candidates (Sect. 3.3.1.1)[4]. Thus, the maximum number of candidates generated in the enumeration of an FST is $3|\mathcal{L}|^2|\mathcal{L}|$, which is $O(|\mathcal{L}|^3)$ as claimed. Note that this is a worst-case estimate, as it includes both frequent and infrequent candidates. If we are to consider the number of candidates generated per FST, we only need to consider the maximum number of pair-wise joins in which an FST can participate within an equivalence class, i.e., $O(|\mathcal{L}|)$, which gives the second part of the result. □

The breadth-first enumeration scheme in EvoMiner requires all the FSTs to be kept in memory. This limits the size of the input trees, because the number of FSTs grow exponentially with the size of the trees (Sect. 3.4). However, the depth-first enumeration scheme in EvoMinerDFonly stores the FSTs of the ancestor equivalence classes while enumerating a k-leaf FST. The next result shows that the memory required by the depth-first enumeration scheme scales polynomially with the number of input trees and the size of the common leaf set.

[4]This will happen when the pair being considered for join operation has the same topology as the input trees for type 3 join. Such a pair will produce candidates of join types 1, 2, and 3.

Theorem 9. EvoMinerDF *requires* $O(n|\mathcal{L}|^3)$ *space, where* n *is the number of input trees and* $\mathcal{L}$ *is the common leaf set.*

Proof. The enumeration tree has depth at most $\mathcal{L}$. Enumerating an FST at this depth will require storing $O(|\mathcal{L}|)$ ancestor equivalences classes, each of which can have at most $|\mathcal{L}|$ FSTs. Thus, the maximum number of FSTs to be stored is $O(|\mathcal{L}|^2)$. Storing each such FST requires $O(|\mathcal{L}|)$ space for the subtree and $O(n)$ space for the occurrence-list. Thus, the maximum space required to store all FSTs is $O(n|\mathcal{L}|^3)$. Adding to this the space required to store LCA mappings, which is $O(n|\mathcal{L}|^3)$, we get the claimed figure. □

To close this section, we note that pattern-growth methods (Pei et al. (2007, 2004); Han et al. (2001)) are a potential alternative to EvoMiner's Apriori approach. While the pattern-growth approach has been shown to scale well for large databases (Han et al. (2000)) and has been used to mine FSTs in collections of ordered trees (Wang et al. (2004)), it is not obvious how to extend the technique to unordered trees. Indeed, although our string encoding of phylogenetic trees is basically an ordered tree representation, our encoding does not satisfy an essential property on which the known pattern-growth methods for unordered tree mining rely. That is, in our approach, the string encoding of a subtree is *not* a prefix of the string encoding of the original tree. In fact, deleting just one leaf from a tree can drastically change its string encoding.

3.4 Experiments and Results

To evaluate the performance of EvoMiner, as well as to test the effectiveness of the FST approach compared to MASTs and MRTs, we conducted experiments on real and simulated data. All experiments were performed on an Intel Core2 Duo E8500 @ 3.16 GHz machine running Windows 7 Professional 64 bit edition with 8GB of RAM. Algorithms were implemented in C++ and compiled using Microsoft Visual C++ 2008 (part of Microsoft Visual Studio 2008, Version 9.0.21022.8 RTM).

We note that in many of our experiments, we used support values of 99% and 50%. While these values may appear high compared to those commonly used in the data-mining literature

(e.g., Wang et al. (2004); Chi et al. (2005)), they reflect standard practice in phylogenetics, which typically demands a strong consensus among the trees in the input collection. The stronger the consensus, the higher is the confidence that can be placed in the common representative tree. For example, MASTs (Sect. 3.1.1.3) require 100% support. Strict-consensus trees (Bryant (2003a)) are another example of 100% support, as they are built from the clusters present in all the input trees. Majority-rule consensus trees (Sect. 3.1.1.3) are a more relaxed version strict-consensus trees, where one only requires that a cluster be present in a majority of the trees —at least 50% support— for it to be included.

3.4.1 Performance of EvoMiner

We compare the performance of our algorithm with that of Phylominer (Zhang and Wang (2008)) using the original C++ implementation of its authors. Our experiments involve four datasets, indexed as $D1 - D4$. $D1$ and $D2$ consist of synthetic phylogenetic trees. Data set $D1$ closely resembles the one used to evaluate the performance of Phylominer. Each tree in $D1$ was produced by first generating a random binary tree based on the Yule model (Yule (1925)); for each such tree, we randomly chose a set of 30% of the internal edges, and contracted all edges in the set. To produce dataset $D2$, we first generated one random tree, which was then replicated to get the required number of trees. Each replicated tree was then perturbed by randomly contracting 10% of its internal edges and randomly swapping 10% of its leaf labels with another random leaf. This resulted in a set of trees having high commonality, which aligns with one of the utilities of EvoMiner as a consensus tree algorithm. Datasets $D3$ and $D4$ were taken from published phylogenetic analyses. $D3$ is from the Bayesian analysis of Lewis and Lewis (2005), while $D4$ consists of bootstrap trees from Pattengale et al. (2011). Bayesian analyses and bootstrap trees are typical candidates for consensus tree algorithms (Sul and Williams (2009); Pattengale et al. (2011)); the trees in these datasets have a very high commonality. We extracted datasets of different sizes (in terms of the number of leaves and the number of trees) from $D3$ and $D4$ by randomly selecting the required number of trees and restricting them on a random set of leaves of the required size.

Fig. 3.14 compares the performance of EvoMiner with Phylominer on datasets $D1 - D4$.

For each comparison, three different leaf sets of sizes 15, 25 and 35 were considered. For $D1$ and $D2$, the minSup value was 50%. Since, $D3$ and $D4$ have highly similar trees, the minSup value was 99% to single out the highly frequent subtrees among the frequent subtrees. The range of leaf set sizes and the number of trees reflects the typical inputs on which phylogenetic analyses involving MAST and related problems are done; see, e.g., Swenson et al. (2011); Zhang and Wang (2008). In each of the datasets, EvoMiner is faster than Phylominer by a factor of up to 100 (sometimes more).

For comparison purposes, the physical memory was capped at 4GB. This explains the missing entries in the graphs. EvoMiner is able to handle larger datasets — both in terms of the number of trees and the number of leaves — because it uses a vertical bitmap representation of the database (Ayres et al. (2002)). In this representation, in the occurrence list of a FST, a bit is reserved for every tree in the database. If the minimum support value is very small, there is a risk of under-utilizing memory, because the number of unset bits would far exceed the number of set bits for the occurrence list of an FST. In our studies, however, the minimum support value is always greater than 50%. Thus, memory utilization was high.

While breadth-first enumeration has the advantage of downward-closure operation and vertical bitmap representation of the database results in a smart utilization of memory, the fact remains that the number of FSTs that can be enumerated is limited by the available memory. However, in depth-first mining mode memory is not a limitation because the enumeration tree is explored in a depth-first manner. Thus, if the run time is not a consideration, users can mine up to 10000 trees on 254 leaves with the current implementation.

As shown in Fig. 3.14, the difference in runtimes of EvoMiner and Phylominer frequently reaches 1000 seconds in our experiments. This improved speed has a practical impact. As mentioned in the Introduction, phylogenetic analysis typically yields a collection of trees rather than a single tree. However, many of the generated trees are not part of the final output. For example, Markov chain Monte Carlo (MCMC) simulations used for Bayesian phylogenetic inference (Mau et al. (1999)) involve multiple runs until convergence is reached. In such cases, it is essential to identify the common information in each run quickly. Speed is also important in summarizing the commonalities among phylogenetic trees during interactive visualizations

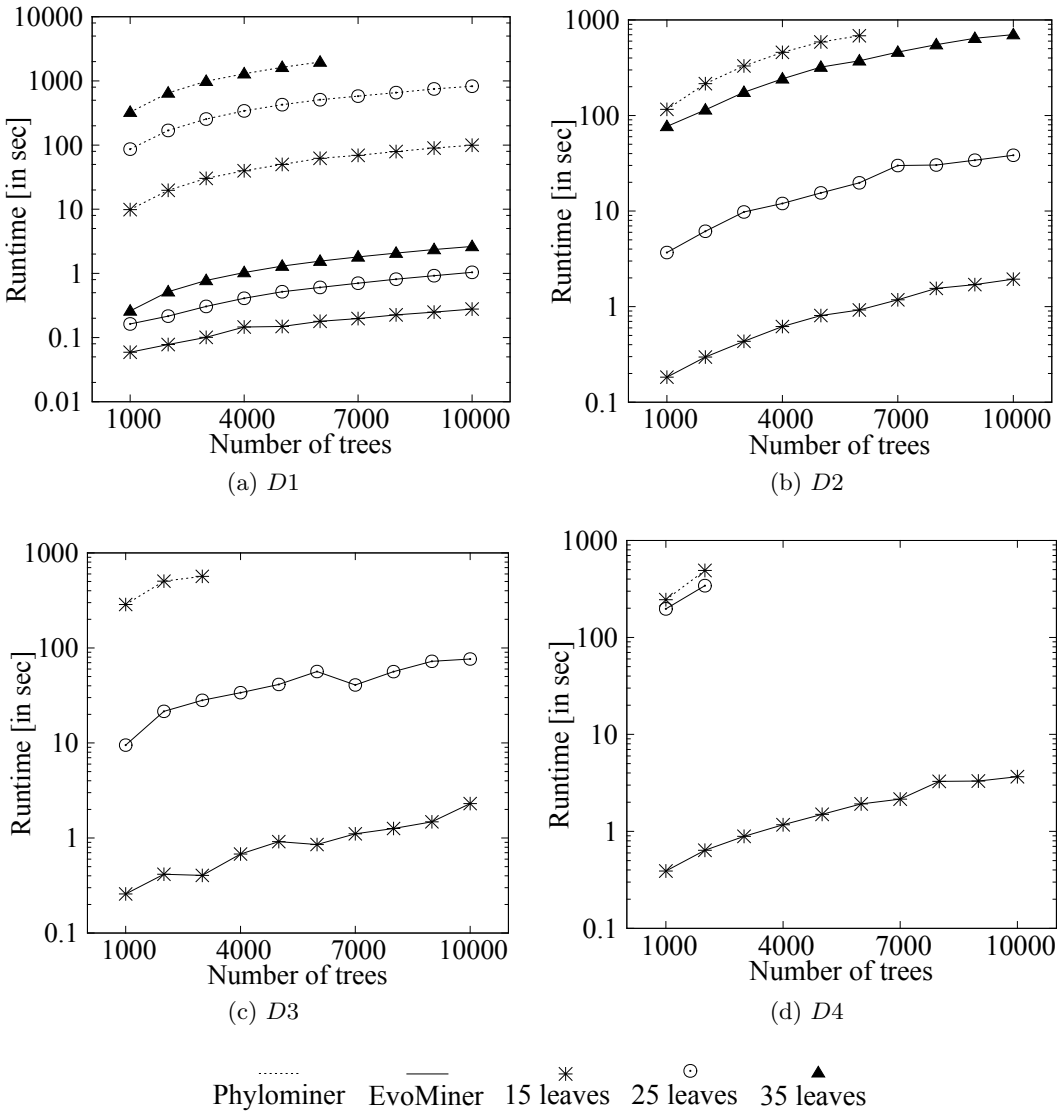


Figure 3.14: Performance comparison

(Amenta et al. (2003)).

Fig. 3.15a confirms the exponential growth of the number of frequent subtrees with an increase in the size of the leaf set. These experiments were done on the Bayesian analysis dataset $D3$ with 100 trees and minSup as 99%. Figure 3.15b shows the corresponding growth of the run time. The slope of this graph is steeper than that of Fig. 3.15a because, as indicated by Theorem 7, the run time depends not just on the the size of the leafset, $|\mathcal{L}|$, but also on $|\mathcal{F}|$, the number of frequent subtrees and on n, the number of trees. Theorem 7 also helps to explain the anomaly in the run time graph for $|\mathcal{L}|$ between 10 and 15: When $|\mathcal{F}|$ is small, we cannot assume that $|\mathcal{F}| \gg |\mathcal{L}|^3$, which makes the $n|\mathcal{L}|^3$ term in the running time non-negligible.

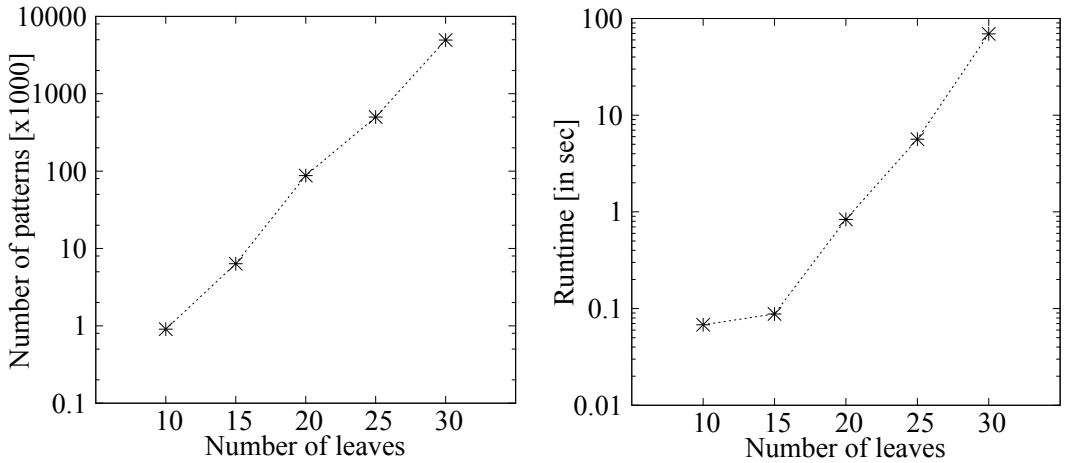
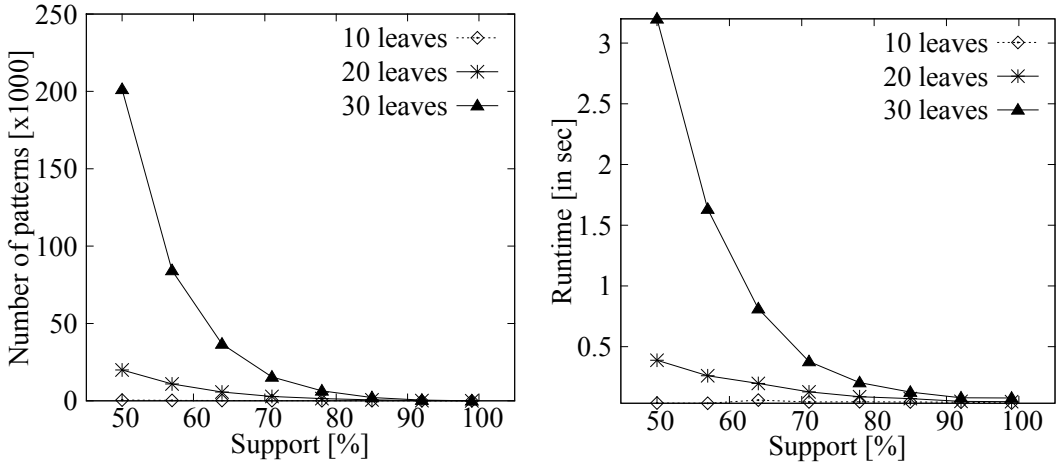


Figure 3.15: Exponential growth of the number of frequent patterns and its effect on the run time.

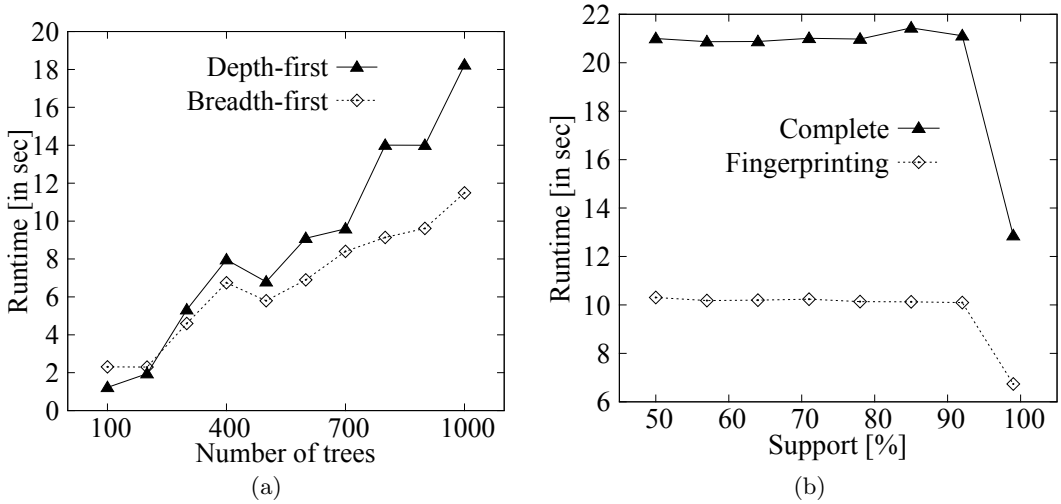
Figure 3.16a shows how the number of frequent subtree patterns varies with respect to minSup on dataset $D2$ (500 trees). The exponential decrease in the number of frequent subtrees as minSup increases is to be expected: The pruning of a frequent tree has a cascading effect on all of its frequent supertrees. The run time behaves in a similar fashion (see Fig. 3.16b). The correlation between the number of frequent subtrees and the run time with respect to minSup confirms that the run time depends directly on the number of frequent subtrees generated.

Figure 3.17a compares the performance of depth-first mining with the candidate generation based (breadth first enumeration) approach with respect to the size of the database. When



(a) (b)

Figure 3.16: Effect of minSup on the number of frequent subtrees and the run time.



(a) (b)

Figure 3.17: Depth-first mining and fingerprinting based pruning technique

the number of trees is small, the frequency counting step takes about the same time as the downward-closure operation and hence the latter becomes an overhead. As the number of trees grows, the frequency counting step becomes costlier and it saves time to use downward closure. The cross-over point comes around 200-300 trees for Bayesian analysis dataset $D3$, with minSup set to 99% and leaf set size of 30. Fig. 3.17b compares our fingerprinting based pruning technique with a complete downward closure operation. The latter additionally considers any remaining $(k-1)$-leaf subtrees for a k-leaf candidate tree, which are not considered by the former. For reasons discussed in Sect. 3.3.1.2, the fingerprinting technique is clearly more efficient than the complete operation. This experiment was done on bootstrapped dataset $D4$ with 1000 trees on 20 leaves. The false positive ratio hovered around 3.2% for this experiment.

3.4.2 Experiments on Biological Data

To study the effectiveness of FST mining, we compared the results of applying the FST, MAST, and MRT approaches to biological data. As mentioned in Sect. 3.1.1.3, both MASTs and MRT are frequently used in phylogenetics. Indeed, a search on "maximum agreement subtree" and "majority-rule tree" on Google Scholar[5] (`http://scholar.google.com/`) returned about 365 and 1140 results respectively. Among other things, MASTs are used as a metric to compare phylogenies (Goddard et al. (1994); Dong and Kraemer (2004); Farach and Thorup (1994)), to compute the congruence index of phylogenetic trees (De Vienne et al. (2007); Lapointe and Rissler (2005)), to identify horizontal gene transfer events (Daubin et al. (2002)), to resolve ambiguity in terraces in phylogenetic tree space (Sanderson et al. (2011)), and as a consensus approach (Bryant (2003a)). MRTs are used, among other things, to analyze phylogenetic trees from Bayesian analysis (Huelsenbeck and Ronquist (2001)) and bootstrapping (Felsenstein (1985)), two widely-used phylogenetic analysis techniques. As we shall see, FST mining can identify representative trees for a collection that are superior to the MAST(s) or the MRT with respect to both size and resolution.

[5]Both terms were searched in double quotes, i.e., all words in the query appear together in all returned documents.

Datasets. Our datasets were derived from bootstrapped trees used in a previous study (Pattengale et al. (2011)) on majority rule trees. Trees were constructed using 17 DNA alignments containing 125 up to 2,554 sequences. The data spans a diverse range of sequences including rbcL genes, mammalian sequences, bacterial and archaeal sequences, ITS sequences, fungal sequences, and grasses. We ordered the trees based on the number of sequences and refer to the datasets as A–Q. For each dataset we randomly selected a set of 15 leaves and a set of 100 trees. We then restricted each of the trees on these 15 leaves to obtain a collection of 100 trees on a common leaf set of size 15. We generated 100 such random collections for each of the dataset in A–Q. The experimental results were averaged over these 100 collections.

Frequent Subtree versus Maximum Agreement Subtree. Note that the set of all MASTs is a subset of the set of FSTs. Thus, for every MAST T there is always an FST that has the same or higher resolution as T. Thus, in comparing MASTs with FSTs. we focus on the gain in the number of leaves.

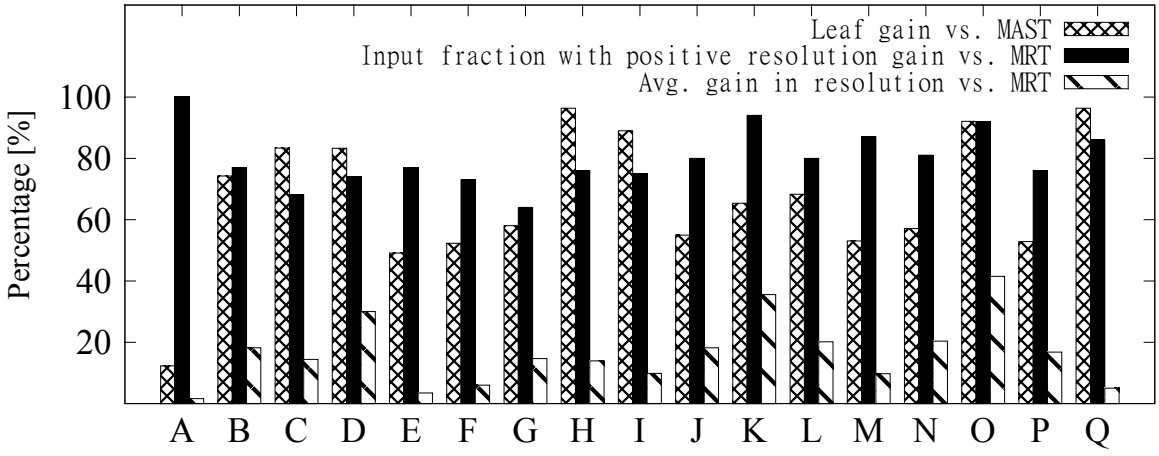


Figure 3.18: FST vs. MAST and MRT

For each collection of 100 trees, we generated all MASTs. We then mined all FSTs in the collection with minSup = 50%. For each MAST T we selected the FST T' with the maximum number of leaves such that T is a subtree of T'. Note that T' is a maximal subtree. We then compared the size of T' (i.e., the number of leaves) with that of T. The first histogram bar in

Fig. 3.18 shows this leaf gain, which is defined as

$$\text{gain}(T, T') = \frac{|\text{leaves in } T'| - |\text{ leaves in } T|}{|\text{leaves in } T|} \times 100\%.$$

In all datasets, we found FSTs larger than any MAST. In some of the cases the leaf gain was as high as 100%.

Frequent Subtree versus Majority Rule Tree. As mentioned in the Introduction, the MRT can be poorly resolved. To explore this issue further, we compared the degrees of resolution of the MRT with that of the FSTs by considering what we call FST profiles. An FST T is *maximal* if there is no other FST T' such that T is a subtree of T'. An *FST-profile* is a collection of maximal FSTs such that the combined leaf set contains all the species present in the input trees. For example, the FSTs in Fig. 3.3d form an FST-profile for the input collection of trees shown in Fig. 3.3a. We measure the resolution of a tree T using equation (3.1) from Sec. 3.1.1.

For each collection of 100 trees, we computed the MRT using HashCS (Sul and Williams (2009)) and generated the corresponding FST-profile from the collection of all FSTs mined using EvoMiner. For each tree T^F in the FST-profile, we restricted the MRT to the same leaf set as that of T^F to obtain T^M, computed the difference in tree-resolution values of T^F and T^M, and then averaged the difference over all the trees in the FST-profile. The second histogram bar in Fig. 3.18, denotes the fraction of the input collections that observed a positive resolution gain in the FST-profile over the MRT. In all cases, a high fraction of the 100 collections resulted in FSTs that were better resolved than the MRTs. The third histogram bar shows the average resolution gain, which in some cases exceeds 30%.

3.5 Conclusion

We introduced EvoMiner, a new algorithm for mining frequent subtrees in phylogenetic databases. We compared our work with Phylominer, another algorithm for the same problem, and showed speed-ups of up to 100 times, and sometimes more. We also demonstrated the utility of FST mining as a way to extract meaningful phylogenetic information from collections

of trees, making it a valuable alternative to MASTs and MRTs in various settings. We now mention some directions for future work.

The sheer number of FSTs can overwhelm a user. One could instead mine for maximal and closed subtrees (Chi et al. (2004a)). While there has been significant work on this subject (Xiao and Yao (2003); Hadzic et al. (2010); Termier et al. (2004); Chi et al. (2005); Feng et al. (2010); Wang et al. (2012)), for reasons mentioned in Sections 3.1.1.1 and 3.2.2, mining maximal phylogenetic subtrees demands a separate approach. We intend to investigate whether we can extend EvoMiner for this purpose. Note that the number of MASTs can grow exponentially with the number of leaves (Kubicka et al. (1992)); the number of maximal and closed subtrees will grow at least as fast. Thus, we intend to develop an approach that will mine a 'representative' set of FSTs that will be pair-wise distinct and well sampled from the entire set (along the lines of Zhang et al. (2009)). Another way to handle large datasets can be to use a parallel implementation (Do et al. (2010)) or to explore approximate solutions (Ke et al. (2009)).

An important open problem is to derive bounds on the number of FSTs. Such bounds would be useful for EvoMiner and EvoMinerDF, as they would allow us to estimate the progress of those algorithms, giving the end-user the option to mine only some desired fraction of the total. Bounds of this sort have been derived for frequent sequential patterns (Raissi and Pei (2011)), but it is not obvious to extend these results to frequent subtrees.

In addition to pure topology, phylogenetic trees typically have other attributes, such as support values for nodes and branch lengths. To our knowledge, the problem of mining phylogenies with such attributes has not been studied. Finally, it would be interesting to see if one can apply the fingerprinting technique for the downward closure operation for mining arbitrary trees, not just phylogenies.

Acknowledgments

This work was supported in part by National Science Foundation grant DEB-0829674. The authors thank Drs. Sen Zhang and Jason T. L. Wang for sharing the source code of Phylominer and discussions on their work. They also thank Drs. Seung-Jin Sul and Tiffani L. Williams

for sharing the datasets from Bayesian analyses, and Dr. Nicholas D. Pattengale for sharing the datasets consisting of bootstrapped trees. A special thanks to the anonymous reviewers at KAIS whose detailed comments helped greatly in improving the paper.

CHAPTER 4. ENUMERATING ALL MAXIMAL FREQUENT SUBTREES

Akshay Deepak and David Fernndez-Baca

An unpublished paper.

A poster on the paper was presented in the 8th *International Symposium on Bioinformatics Research and Applications*, 2012

Abstract

Background: A common problem in phylogenetic analysis is to identify frequent patterns in a collection of phylogenetic trees. Roughly speaking, the goal is to find a subset of the species (taxa) on which all or some significant subset of the trees agree. One of the popular methods to do so is maximum agreement subtrees. These are also used, among other things, as a metric for comparing phylogenetic trees, computing congruence indices and to identify horizontal gene transfer events.

Results: We give algorithms and experimental results for two approaches to identify common patterns in a collection of phylogenetic trees, one based on agreement subtrees, called maximal agreement subtrees, the other on frequent subtrees, called maximal frequent subtrees. These approaches can return subtrees on a larger set of taxa than maximum agreement subtrees, and are capable of revealing new common phylogenetic relationships not present in either maximum agreement subtrees or the majority rule tree (a popular consensus method). Our current implementation is available on the web.

Conclusions: Our approach is the first to enumerate maximal agreement subtrees and maximal frequent subtrees in collections of phylogenetic trees. They can reveal a more complete phylogenetic picture in applications using maximum agreement subtrees and, at times, return more resolved subtrees than the majority rule tree.

4.1 Background

Phylogenetic trees are unordered trees whose leaves are in one-to-one correspondence with a set of species. An *agreement subtree* for a collection of phylogenetic trees on a common leaf set is a subtree homeomorphically included in all of the input trees. A *maximal agreement subtree* (MXST) is an agreement subtree that is not a subtree of any other agreement subtree. An MXST is a *maximum agreement subtree* (MAST) if it has the largest number of leaves (Finden and Gordon (1985)). MASTs are used, among other things, as a metric for comparing phylogenetic trees (Goddard et al. (1994); Dong and Kraemer (2004); Farach and Thorup (1994)), computing their congruence index (De Vienne et al. (2007); Lapointe and Rissler (2005)), to identify horizontal gene transfer events (Daubin et al. (2002)), for resolving ambiguity in terraces in phylogenetic tree space (Sanderson et al. (2011)) and as a consensus approach (Bryant (2003a)). The MAST problem is polynomially-solvable for two trees, but is NP-hard for three or more input trees, if their degree is unbounded (Amir and Keselman (1994)).

An MXST can reveal shared phylogenetic information not displayed by any of the MASTs (see Figure 4.1). Even more common substructure can be uncovered if we relax the requirement that the subtree returned has to be supported by all the input trees. Let f be a number in the interval $\left(\frac{1}{2}, 1\right]$. An *$f$-frequent subtree*, or simply a *frequent subtree* (FST), for a collection of m leaf-labeled trees on a common leaf set, is a subtree homeomorphically included in at least $f \cdot m$ of the input trees. A *maximal FST* (MFST) is an FST that is not a subtree of any other FST. Thus, an MXST is an MFST with $f = 1$. The set of all MFSTs is a compact non-redundant summary of the set of all FSTs: Every FST is a subtree to some MFST but every MFST is not a subtree to any other FST. Thus, every MFST reveals some unique phylogenetic information that is not displayed by any other FST.

A well-supported MFST can have more leaves and be more resolved than an MAST (see

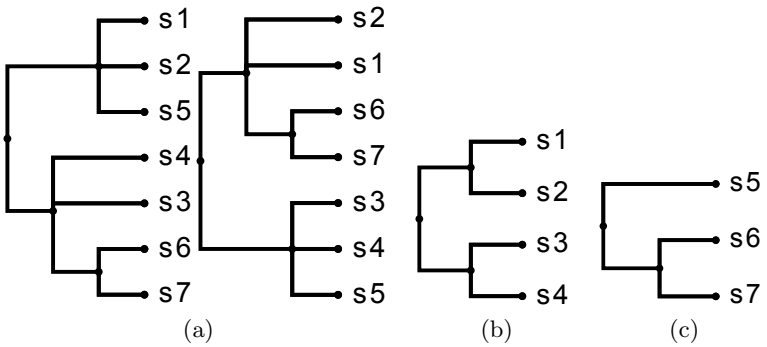


Figure 4.1: (a) A collection of two trees and their (b) MAST. (c) An MXST that has fewer leaves than the MAST but is not displayed by it.

Sect. 79). In the more general setting where the leaf sets of the input trees have little overlap, the gap between the size of an MFST and that of an MAST can be even wider. Indeed, in this case any agreement tree would tend to be quite small — see Figure 4.2. Further, an MFST can be more resolved than the majority rule tree, which can be greatly affected by "rogue" taxa; that is, taxa whose positions vary widely within the input collection (Swenson et al. (2011); Pattengale et al. (2011)) (see Figure 4.3).

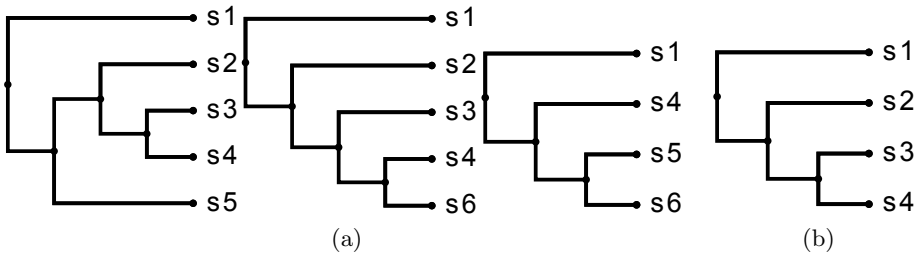


Figure 4.2: (a) A collection of three trees and (b) an MFST with $f = \frac{2}{3}$. MAST or MRT cannot be applied effectively as the common overlap consists of only two leaves.

Motivated by the above, we introduce a new algorithm MfstMiner for enumerating MXSTs and MFSTs. MfstMiner enumerates MFTSs over partially overlapping leafsets as well. We compare MfstMiner with Phylominer (Zhang and Wang (2008)), an algorithm for enumerating all FSTs and show that enumerating MFSTs can be orders of magnitude faster than enumerating all FSTs. Our experiments show that well-supported MFSTs can have many more leaves than a MAST, and can reveal new phylogenetic information not displayed by any

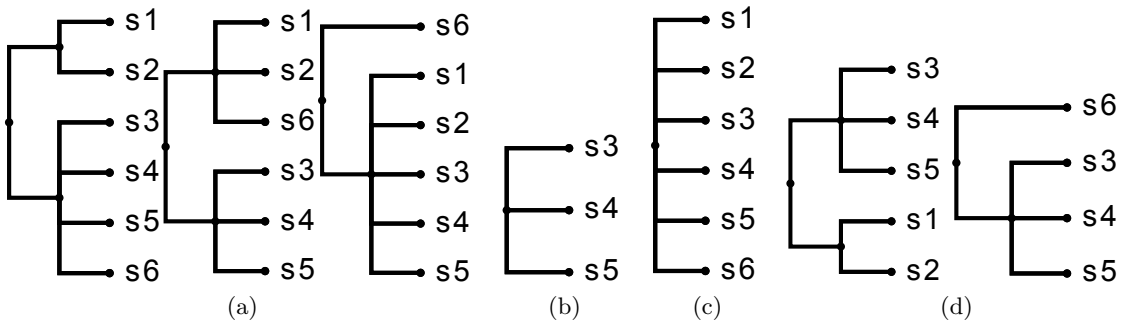


Figure 4.3: (a) Three input trees. (b) Their MAST, which is star-like. (c) Two MFSTs with $f = \frac{2}{3}$, each fully resolved and larger than the MAST. (d) The majority rule tree, which is also star-like.

of the MASTs.

To our knowledge the enumeration of MFSTs has not been studied before. Our current implementation can be downloaded from `http://www.cs.iastate.edu/akshayd/mfstMiner/`; it works for up to 250 leaves and 10,000 trees.

4.1.1 Related Work

Due its utility and inherent complexity, the MAST problem attracted computational biologists and mathematicians alike. It was first studied by Finden and Gordon (Finden and Gordon (1985)). It is polynomially solvable for two trees and over the years its complexity has progressively improved (Amir and Keselman (1994); Steel and Warnow (1993); Kao et al. (2001)). However, for more than two trees with unbounded degrees it becomes NP-hard (Amir and Keselman (1994)). For trees with bounded degree, it again becomes solvable in polynomial time (Farach et al. (1995b); Bryant (1997)).

Maximal subtree mining (Wang and Liu (1998); Xiao and Yao (2003); Chi et al. (2005)) and maximal subgraph mining (Huan et al. (2004); Thomas et al. (2006)) have received prominent attention in data mining literature. However, the algorithms deployed in these fields cannot be applied here as phylogenetic trees possess a special structure — only leaves are labeled and the non-leaf nodes must be of degree two or more. The problem of mining frequent phylogenetic subtrees (subtrees included in majority of the input trees) has been studied by Zhang and Wang

(2008), who proposed a polynomial time algorithm (Phylominer) to mine all frequent subtrees in a collection of phylogenetic trees. By definition the set of all frequent subtrees include the set of all MFSTs; however, the number of frequent subtrees can be exponentially more than the number of MFSTs. Thus, mining MFSTs exclusively, instead of all MFSTs, saves a lot of time and the result set is much smaller to analyze. To our knowledge, our work is the first one to deal with the problem of mining MFSTs for phylogenetic trees.

4.1.2 Preliminaries

A ***phylogenetic tree*** is an unordered rooted[1] leaf-labeled tree. Leaf labels represent the taxonomic units (species) under study. A node is ***internal*** if it is not a leaf node. If a phylogeny T is rooted, each internal node must have at least two children. Let $\mathcal{L}_T$ denote the leaf label set of tree T, and ψ_T denote the bijection that maps the leaf nodes to their unique labels. For convenience, we refer to the set of leaf nodes by their labels in $\mathcal{L}_T$. From this point forward, unless the context requires making a distinction, we will drop the subscripts in $\mathcal{L}_T$ and ψ_T, and write $\mathcal{L}$ and ψ respectively. For the rest of the paper, we assume without loss of generality that the leaf label set $\mathcal{L}$ consists of distinct integers in the range $[1, |\mathcal{L}|]$; thus, the labels are ordered.

Let u be an internal non-root node in some tree (not necessarily a phylogenetic tree), such that u has only one child v. Then, *suppressing* u means contracting the edge (u, v); i.e., deleting u and the edges incident on it and adding an edge from the parent of u to v. To *prune* a leaf ℓ, we first delete it. Let u be ℓ's neighbor. If u is not the root, and the deletion of ℓ makes u a degree-two node, we suppress u. If u is the root and deleting ℓ makes it a degree one node, u is deleted and its neighbor becomes the new root. Consider a tree T and a set $\mathcal{L}' \subseteq \mathcal{L}_T$. The *restriction* of T to $\mathcal{L}'$, denoted by $T|_{\mathcal{L}'}$, is the minimal homeomorphic subtree of T connecting the leaves with labels in $\mathcal{L}'$ (that is, we start with the minimal subtree of T connecting $\mathcal{L}'$, and repeatedly suppress non-root nodes with at most one child until no such nodes remain). We denote the fact that two trees T_1 and T_2 are isomorphic by writing $T_1 \equiv T_2$. A tree T' is an

[1]Phylogenetic trees can also be unrooted (Felsenstein (2004)), but here we deal exclusively with rooted phylogenetic trees.

subtree of another tree T if $\mathcal{L}_{T'} \subseteq \mathcal{L}_T$ and $T' \equiv T|_{\mathcal{L}_{T'}}$.

The ***depth*** of a node u in a tree T, denoted $\text{depth}^T(u)$, is the number of edges from the root to that node; thus the root node is at depth 0. We denote the lowest common ancestor (LCA) of two nodes u and v in T by $\text{LCA}^T(u, v)$. When the tree T is clear from the context, we drop the superscripts. A ***k-leaf tree*** is a tree with k leaves.

4.2 Algorithmic Framework

We first discuss the algorithm for ASTs/MXSTs because it is simpler (since $f = 1$). We then extend it for FSTs/MFSTs.

We enumerate all MXSTs from the solution space of all ASTs. We show that any k-leaf AST can be enumerated by combining two unique $(k-1)$-leaf ASTs having certain properties. We call the k-leaf tree a *join* on the two smaller $(k-1)$-leaf trees. To efficiently enumerate all ASTs by joining smaller ASTs this way three issues must be addressed. The first is to avoid redundant enumeration of the MXSTs, which can happen by either enumerating multiple isomorphic representations of the same MXST or multiple copies of the same isomorphic representation. The second potential complication is that while a k-leaf AST is enumerated by joining two unique $(k-1)$-leaf ASTs, the opposite is not true, i.e., these two $(k-1)$-leaf ASTs can potentially combine in more than one topology over k leaves. Thus, given two $(k-1)$-leaf ASTs, the topologies these combine into across the trees in the input collection needs to be accounted. A k-leaf AST exists as a result of joining these two $(k-1)$-leaf ASTs if only one topology is supported across all input trees. Third is that the total number ASTs can be exponentially larger than the total number of MXSTs. Thus, while enumerating all MXSTs from the solution space of all ASTs, we must contain the combinatorial explosion due to the number of ASTs. We next describe how we address these three issues.

4.2.1 Non-redundant Enumeration

To avoid generating multiple isomorphic copies of the same tree, we enumerate subtrees in "canonical form" (Zhang and Wang (2008)) (an ordered representation for phylogenetic trees). To enumerate every canonical representation once, we define a parent-child relationship over

the space of all ASTs. This induces an enumeration tree over the solution space, where each node represents a collection of ASTs grouped together via an equivalence relation. Leaf nodes represent potential MXSTs and each MXST belongs to a unique leaf node. This scheme is motivated by the reverse search technique for enumeration (Avis and Fukuda (1996)).

4.2.1.1 Canonical Form

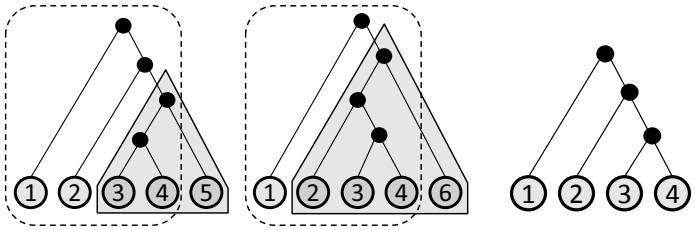
The ***virtual label*** of an internal node v is the minimum label among all leaf descendants of v. The children of an internal node are ordered from left to right based on the sequence in which they are encountered in an inorder depth-first traversal (IDFT), the leftmost child being encountered first. A tree T is in ***canonical form*** (Zhang and Wang (2008)) if, for every internal node, its children are ordered from left to right by their virtual labels. It can be seen that two trees are isomorphic if and only if they have the same canonical forms. By generating all trees in canonical form, it is straightforward to test if two trees are isomorphic and prevent duplicate enumeration. MFSTMINER relies on this property to ensure that each FST is enumerated exactly once. Henceforth, a tree is assumed to be in its canonical form unless mentioned otherwise.

4.2.1.2 Enumeration Tree

The key notion for defining the enumeration tree is that of an *equivalence class*. To explain this notion, we first need some definitions. The ***rightmost leaf*** of tree T is the last leaf encountered in the IDFT of T. The subtree that results from pruning the rightmost leaf is called the ***prefix tree*** or simply ***prefix***. It is so called because the IDFT of the prefix tree is the largest prefix of the IDFT of the original tree that is not the original tree. The ***heaviest subtree*** (Zhang and Wang (2008)) is the subtree rooted at the parent of the rightmost leaf.

An ***equivalence class*** is a set of canonical trees that share a common prefix. We call this common prefix tree the ***core tree***. Thus, an equivalence class of k-leaf trees will have a $(k-1)$-leaf core tree. For an equivalence class E, let E^c denote its core tree, and, for an AST T, let $\mathcal{E}_T$ denote the equivalence class that has T as its core tree. Any two trees in an equivalence class differ only with respect to their rightmost leaf, therefore, topologically their

difference is restricted to their heaviest subtrees. Figure 4.4 illustrates the defined concepts. The equivalence relation "sharing a common prefix" partitions any set of canonical trees into disjoint subsets. Each such subset is an equivalence class identified by its unique core tree.



(a) Equivalence class (b) Prefix tree

Figure 4.4: (a) Two trees belonging to the same equivalence class. The common prefix tree (shown separately in (b)) is encircled by the dotted lines; the respective rightmost leaves are the ones outside the dotted lines. The shaded part represents the respective heaviest subtrees.

Each node in the enumeration tree represents a unique equivalence class. An equivalence class E is the *parent* of equivalence class F if $F^c \in E$. Clearly each node has a unique parent. Note that as we traverse from an internal node towards a leaf, the core tree of each node in the path corresponds to a new leaf being added as a suffix to the IDFT of the core tree of its parent node. Thus, if equivalence class E is an ancestor of equivalence class F, the IDFT of E^c is a prefix of the IDFT of F^c, and vice versa.

A node in the enumeration tree is a leaf if its core tree is not a prefix to any other AST. Thus, its corresponding equivalence class is empty. Note that every MXST is the core tree of some leaf node. The converse is not true because a $(k-1)$-leaf tree may not be the prefix of a given k-leaf tree, yet can be a subtree of it. The root of the enumeration tree is an empty node that has all the equivalence classes containing 3-leaf ASTs as its children. This is because three is the minimum number of leaves on which phylogenetic inference can be meaningful. For an equivalence class E, the *branch* at E represents the subtree induced in the enumeration tree by all the leaf descendants of E, or simply E if it is a leaf. ASTs X and Y are considered to be of a *common descent* if neither is a descendant of the other. Figure 4.5 gives an example of an enumeration tree.

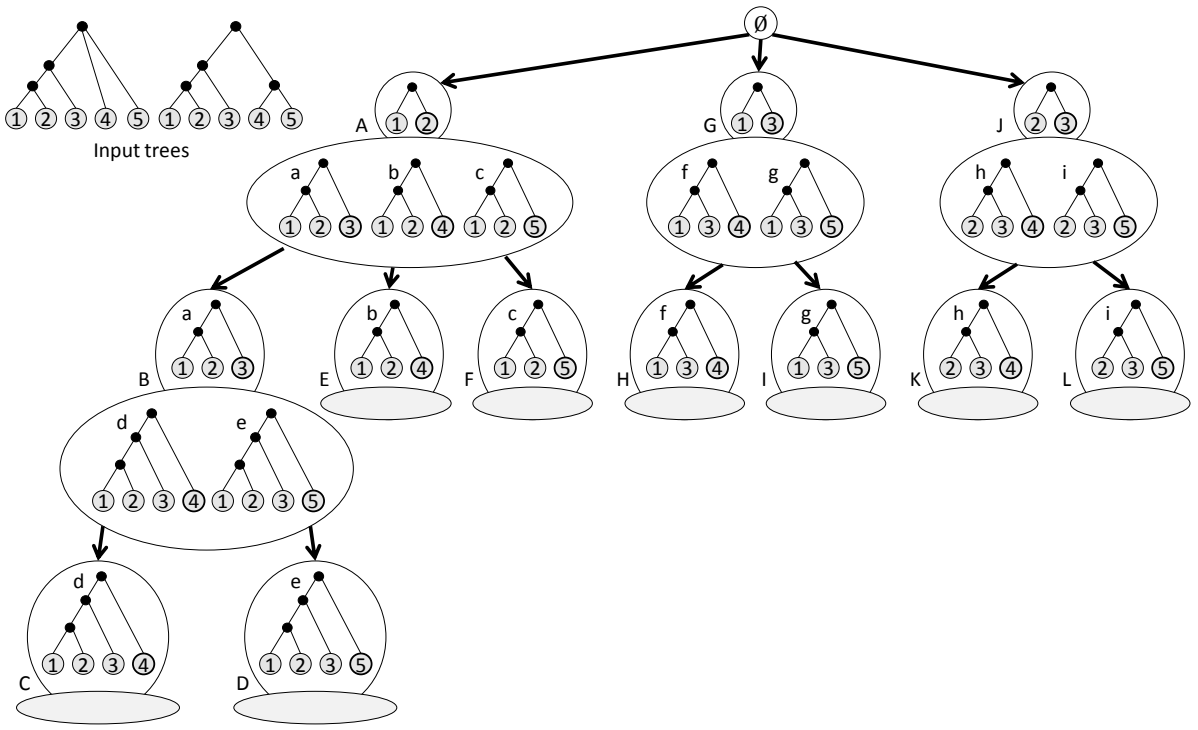


Figure 4.5: Each node in the tree represents an equivalence class. Trees in an equivalence class differ only with respect to their right most leaves (circled in bold for each tree). The bubble at the top of a node contains the core tree of the corresponding equivalence class. An equivalence class contains all ASTs that have its core tree as their common prefix. The core tree of an equivalence class belongs to its parent equivalence class. For example, the core tree of equivalence class B is a, which belongs to A — the parent of B. All 3-leaf ASTs have been partitioned into equivalence classes A, G and J (children of the root node). The leaf nodes (indicated by shaded ellipses) are empty equivalence classes and their core trees represent potential MXSTs. Here, d and e, the respective core trees of leaf nodes C and D, are the only MXSTs. They also happen to be the MASTs for the input trees.

4.2.1.3 Pairwise Join

The canonical form has the property that pruning either the last leaf or the second-to-last leaf encountered in the IDFT results in a subtree that is also canonical (Zhang and Wang (2008)). Thus, every k-leaf AST T corresponds to a unique ordered pair (T_x, T_y) of $(k-1)$-leaf ASTs where T_x and T_y are obtained by pruning the last leaf and the second-to-last leaf respectively in the IDFT of T. Note that T_x and T_y share a common prefix. Vice versa, T can be obtained by 'joining' this unique pair (T_x, T_y). Based on this, we define tree T to be a ***join***

on an ordered pair (T_x, T_y) of $(k-1)$-leaf ASTs such that T_x and T_y share a common prefix, if:

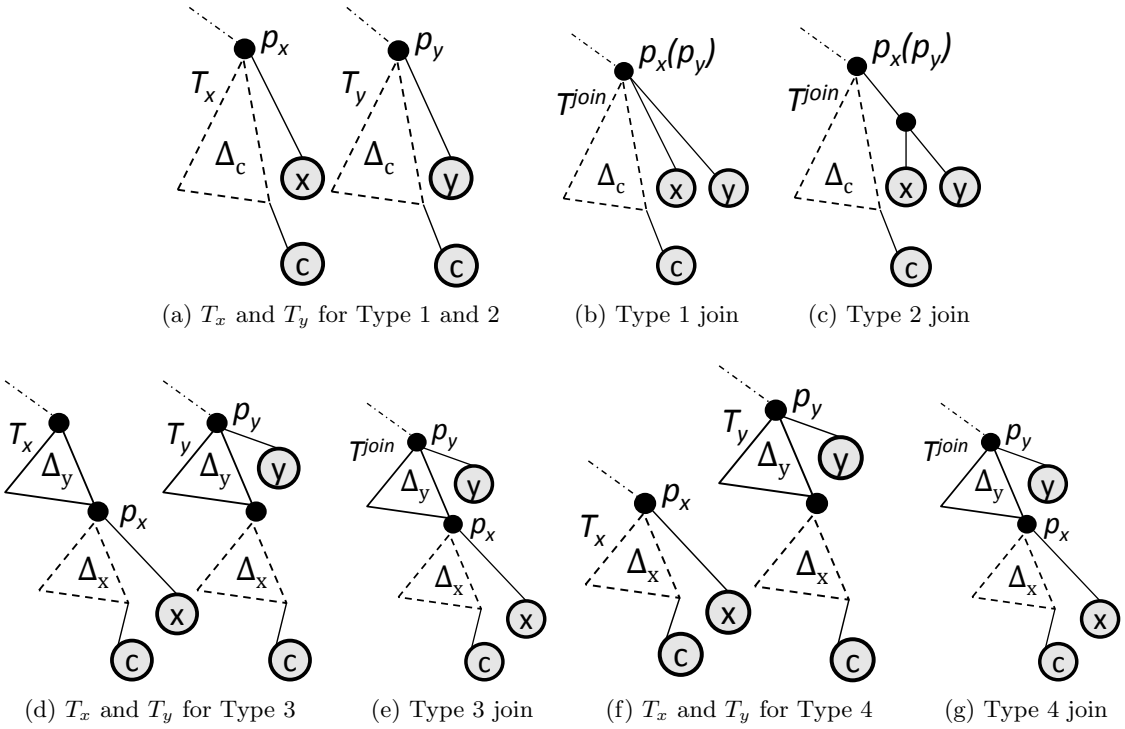
$$T \text{is in canonical form, has } T_x \text{ as its prefix and has } T_y \text{ as its subtree.} \tag{4.1}$$

Our scheme exploits condition 4.1 heavily. Consider equivalence classes E and F, where E is the parent of F and E consists of $(k-1)$-leaf ASTs. We claim that any k-leaf tree $T \in F$ is the result of joining two $(k-1)$-leaf trees in E. Specifically, T is the result of joining a unique ordered pair of trees (T_x, T_y) in E such that condition 4.1 is satisfied. Observe that for any ordered pair (T_x, T_y) satisfying condition 4.1 with respect to T, T_x is the core tree of F and belongs to E. Further, T_x and T_y share a common prefix; thus, T_y also belongs to E. The claim follows.

While every tree in F can be obtained by joining a unique ordered pair of trees in E, for a given ordered pair (T_x, T_y) in E there may be multiple Ts satisfying condition 4.1. The way in which (T_x, T_y) join to produce T depends on the topology of the subtree displayed by an input tree over the leafset $\mathcal{L}_{T_x} \cup \mathcal{L}_{T_y}$. We next describe the four possible ways, denoted type 1-4, in which an ordered pair (T_x, T_y) can join as per condition 4.1. In the subsequent discussion, let x and y denote the rightmost leaf of T_x and T_y respectively, p_x and p_y denote the parents of x and y respectively, and T^{core} represent the core tree of equivalence class E. Note that T^{core} is also the common prefix of T_x and T_y. For an internal node u, let numChild(u) denote its number of children. Different type of joins arise due to the relative values of depth(p_y) and depth(p_x).

Type 1: Figure 4.6a shows the participating trees. Leaves x and y are attached at the same depth on the rightmost path of T^{core}, i.e., depth(p_y) = depth(p_x). Figure 4.6b shows the resulting join. Here, x and y are attached as siblings to the same parent node in the joined tree. Thus, for the resulting joined tree to be canonical, we must have $\psi(x) < \psi(y)$ (recall that we assume that the labels are distinct numbers). Further, x and y are attached at the same depth in the joined tree as in T_x and T_y, respectively.

An example: Figure 4.7a shows the input trees and Fig. 4.7b shows the corresponding joined tree.



(a) T_x and T_y for Type 1 and 2 (b) Type 1 join (c) Type 2 join

(d) T_x and T_y for Type 3 (e) Type 3 join (f) T_x and T_y for Type 4 (g) Type 4 join

Figure 4.6: Different types of pairwise join. A dotted triangle represents part of the tree which may be empty while a solid triangle represents a non-empty part of the tree. Δ reflects topologies of the heaviest subtrees. 'c' denotes the rightmost leaf of the common core tree.

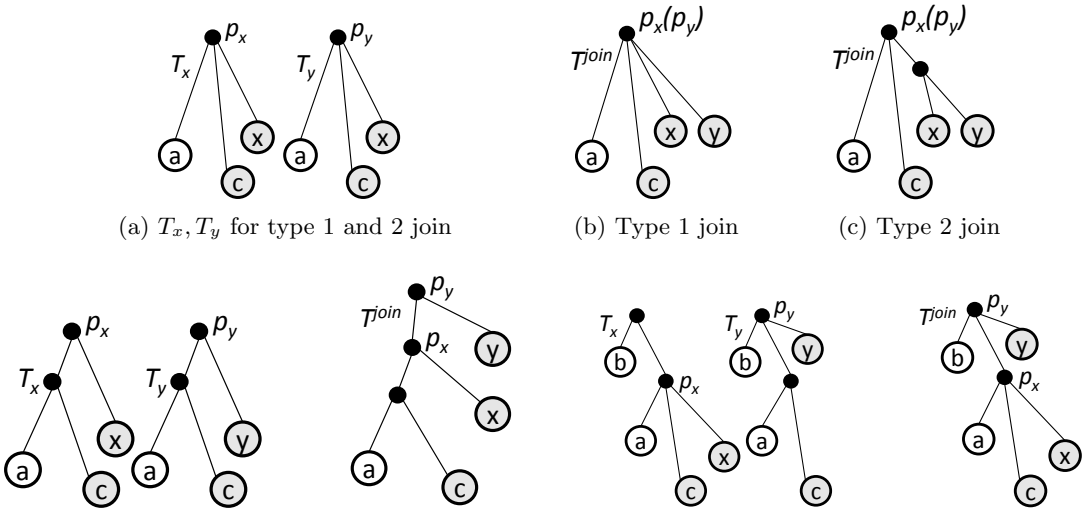
Type 2: The input trees have the same structure as in type 1 join (Fig.4.6a); however, x and y are attached to the same parent in the joined tree at one level deeper than their respective depths in the participating trees as shown in Fig.4.6c.

An example: Figure 4.7a shows the input trees and Fig. 4.7c shows the corresponding joined tree.

Type 3: Figure 4.6d shows the participating trees. Note that the participating trees are a special case of type 1 and 2 join; i.e., $\text{depth}(p_y) = \text{depth}(p_x)$ holds here as well. However, in the resulting join p_y becomes the parent of p_x as shown in Fig. 4.6e. For this to be possible, we must have $\text{numChild}(p_y) = \text{numChild}(p_x) = 2$.

An example: Figure 4.7d shows the input trees and Fig. 4.7e shows the corresponding joined tree.

Type 4: Figure 4.6f shows the participating trees. Here $\text{depth}(p_y) < \text{depth}(p_x)$; i.e., on the



(a) T_x, T_y for type 1 and 2 join (b) Type 1 join (c) Type 2 join

(d) T_x, T_y for type 3 join (e) Type 3 join (f) T_x, T_y for type 4 join (g) Type 4 join

Figure 4.7: An example for each of the join type shown in Fig. 4.6.

rightmost path of T^{core}, leaf y is attached at a lesser depth than leaf x. As a result there is only one way to join T_x and T_y so as to satisfy condition 4.1. See Fig. 4.6g. Here, p_y becomes an ancestor of p_x in the joined tree.

An example: Figure 4.7f shows the input trees and Fig. 4.7g shows the corresponding joined tree.

The above scheme leads to a natural formulation for generating all members of children of E. For every ordered pair $(T_x, T_y) \in E$ such that the pair joins only in one way in the all the trees in the input collection, add the joined tree to $\mathcal{E}_{T_x}$. The ordering indicates that the joined tree has the first tree of the ordered pair as its prefix.

Observe that for $\text{depth}(p_y) > \text{depth}(p_x)$, a join satisfying condition 4.1 is not possible because T_x cannot be the prefix of the joined tree. ASTs from 'such' joins are enumerated when considering the ordered pair (T_y, T_x).

4.2.2 Support Estimation

An AST is enumerated by combining two smaller ASTs. However, an AST can arise out of their combination only if the two ASTs exhibit a common type of join (topology) in all the

input trees. Determining this involves identifying the types of joins the smaller ASTs exhibit across the input trees, and if a particular join is supported across all the input trees. For this we deploy a one-time least common ancestor based preprocessing step, after which the join type in each input tree can be identified in constant time.

Consider an ordered pair of trees (T_x, T_y) in an equivalence class and let $\mathcal{L}^{\cup} = \mathcal{L}_{T_x} \cup \mathcal{L}_{T_y}$. For an input tree T, we say the join induced by (T_x, T_y) in T is of type A if $T|_{\mathcal{L}^{\cup}}$ in canonical form corresponds to type A join with respect to ordered pair (T_x, T_y). Let T^{join} denote $T|_{\mathcal{L}^{\cup}}$ in canonical form. This step classifies T^{join} as one of the four join types. If a particular join is supported by all the input trees, i.e. $f = 1$, the corresponding joined tree is an AST. A nave way to classify the join type could be to restrict T over $\mathcal{L}^{\cup}$, canonicalize the restriction and identify the canonicalized tree as one of the four join types. However, this will require time at least linear in the size of T. In Theorem 10 we describe a least common ancestor (LCA) based scheme that in constant time identifies T^{join} as a result of one of the four join types. The LCA values are computed as a preprocessing step. The meaning of the symbols c, x, y, p_x and p_y is the same as in Sect. 4.2.1.3. Superscripts indicate the reference tree.

Theorem 10. *1. T^{join} is a result of type 1 join if and only if*

(a) $\text{depth}(\text{LCA}^T(c, x)) = \text{depth}(\text{LCA}^T(c, y))$,

(b) $\text{depth}(\text{LCA}^T(c, x)) = \text{depth}(\text{LCA}^T(x, y))$ *and*

(c) $\Phi(x) < \Phi(y)$.

2. T^{join} is a result of type 2 join if and only if

(a) $\text{depth}(\text{LCA}^T(c, x)) = \text{depth}(\text{LCA}^T(c, y))$,

(b) $\text{depth}(\text{LCA}^T(c, x)) < \text{depth}(\text{LCA}^T(x, y))$ *and*

(c) $\Phi(x) < \Phi(y)$.

3. T^{join} is a result of type 3 join if and only if

(a) $\text{depth}^{T_x}(p_x) = \text{depth}^{T_y}(p_y)$ *and*

(b) $\text{depth}(\text{LCA}^T(c,x)) > \text{depth}(\text{LCA}^T(c,y))$.

4. *T^{join} is a result of type 4 join if and only if* $\text{depth}^{T_x}(p_x) > \text{depth}^{T_y}(p_y)$.

Proof. Let us consider each of the cases separately.

1. Clearly if T^{join} is a result of type 1 join, it satisfies 1a-1c. To prove the *only if* part, let 1a-1c be satisfied. Since T_x and T_y are obtained by attaching x and y respectively to the rightmost path of T^{core}, and each is a subtree of T, 1a implies that $\text{depth}(p_y) = \text{depth}(p_x)$. Thus, T^{join} is a result of either of type 1, 2 or 3 join. Type 3 join requires p_y to be the parent of p_x in T^{join}, which is ruled out by 1a. Further, 1b and 1c imply that the join must be of type 1.

2. The proof proceeds in a similar fashion as that for part 1. Again T^{join} can be a result of either type 1 or 2 join. Conditions 2b and 2c imply that the join must be of type 2.

3. Condition 3a implies that T^{join} must be a result of type 1, 2 or 3 join. Condition 3b rules out type 1 and 2 joins. Thus the join must be of type 3.

4. As per given, T^{join} can be a result of type 4 join only.

□

Clearly cases 1—4 are mutually exclusive and each can be evaluated in constant time. Observe that (a) all the cases in Theorem 10 involve comparison of the depth of the LCAs of two pairs of leaf nodes in T and not the actual values, and (b) every such two pairs have one leaf in common. Using this, instead of identifying the join type in all the trees in the input collection individually, we do it in constant time across all the input trees at once, i.e., in the case of ASTs for a given ordered pair (T_x, T_y) we answer in constant time, if they join to result in an AST. To achieve this, we create a map that marks every ordered set of three leaves $\{i,j,k\}$ as '¿', '¡' or '=' if for every input tree T, $\text{depth}(\text{LCA}^T(i,j)) > \text{depth}(\text{LCA}^T(i,k))$, $\text{depth}(\text{LCA}^T(i,j)) < \text{depth}(\text{LCA}^T(i,k))$ or $\text{depth}(\text{LCA}^T(i,j)) = \text{depth}(\text{LCA}^T(i,k))$ respectively. If this comparison is not the same for all the input trees, the corresponding entry is flagged indicating that a

common type of join is not exhibited across all the input trees. This map is created as a preprocessing step.

4.2.3 Containing the Combinatorial Explosion

One way to enumerate all MXSTs is to visit all the leaf nodes in the enumeration tree representing potential MXSTs. However, this requires traversing the complete enumeration tree and can lead to a combinatorial explosion due to the number of ASTs. To limit this we use a heuristic that given a node in the enumeration tree determines if none of its leaf descendants contain a MXST without traversing subtree below it, i.e., if so determined, the branch at the node is pruned away from enumeration.

4.2.3.1 Pruning Heuristic

Let X and Y be two equivalence classes. The branch at Y is ***pruned*** by X, or simply Y is pruned by X, if X and Y are of a common descent, and for every descendant A of Y (including Y), there exists at least one descendant B of X such that B^c displays A^c. In such a case none of the leaf descendants of Y can be an MXST. Thus, the branch at Y need not be enumerated. For example, in Fig.5, node A prunes node G because, including itself, G has 3 descendants as G, H and I, whose respective core trees are displayed by the respective core trees of nodes B, C and D, which are descendants of A. If this information can be know when G is first visited, the branch at G need not be enumerated further, saving time. In other words, the branch at G can be pruned. Similarly, A also prunes J because the respective core trees of nodes J, K and L are displayed by the respective core trees of nodes B, C and D. Further, among the descendants of A, nodes E and F are respectively pruned by nodes C and D.

For the next set of results, let E denote an equivalence class with r as the rightmost leaf of its core tree. Let X, Y, Z be children of E in the enumeration tree. Let x, y, z be the rightmost leaves of X^c, Y^c, Z^c respectively. Clearly $\{X^c, Y^c, Z^c\} \in E$. We say $[i, j, k]$ is an *agreement triplet* if all the input trees display the same triplet over the leafset $\{i, j, k\}$. For an ordered pair of trees (A, B) in an equivalence class, having a and b as their respective rightmost leaves, we say tree T_{ab} *exists* if (A, B) join as T_{ab} across all the input trees. Theorem 11 characterizes

pruning among children and among grandchildren of a node in the enumeration tree.

Theorem 11. *1. X prunes Y if either of the following holds:*

(a) T_{xy} exists and is not of join type 2.

(b) T_{xy} exists as join type 2 and for every child Z of E such that T_{yz} exists, $[x, y, z]$ is an agreement triplet.

2. If T_{xy} and T_{yz} exist, and T_{xy} is of join type 2, then $\mathcal{E}_{T_{xy}}$ prunes $\mathcal{E}_{T_{yz}}$ if T_{yz} is not of join type 2.

Fact 12 and Lemma 13 are instrumental in proving Theorem 11. We present these first.

Fact 12. *For a given tree on four leaves, any three of the four possible triplets define the tree.*

Lemma 13. *Let T_{xy} and T_{yz} exist, and $[x, y, z]$ be an agreement triplet. Then, T_{xz} exists and there exists an AST T on leafset $\mathcal{L}_{E^c} \cup \{x, y, z\}$ such that $\mathcal{E}_T$ is a descendant of X.*

Proof. From the given, $[x, y, z]$ is an agreement triplet and for any $\ell \in E^c$, $[\ell, x, y]$ and $[\ell, y, z]$ are agreement triplets. Thus by Fact 12, $[\ell, x, z]$ is also an agreement triplet. Thus, there exists an AST T on leafset $\mathcal{L}_{E^c} \cup \{x, y, z\}$. Since $\{X^c, Y^c, Z^c\} \in E$, T is a descendant of E. Further, joins T_{xy} and T_{yz} imply that T can only be enumerated by joining T_{xy} and T_{xz} in one of the two possible ways. Thus, T_{xz} exists and $\mathcal{E}_T$ is a descendant of X. □

Proof of Theorem 11. 1. (a) Let T_{xy} be of join type 1 with triplet $[r, x, y]$ of type (r, x, y). Consider any $T_{yz} \in Y$. Let T_{yz} be of join type 1 with triplet $[r, y, z]$ of type (r, y, z). Observe that any tree that displays both (r, x, y) and (r, y, z) must display (r, x, y, z) as well. Thus, $[x, y, z]$ is an agreement triplet. Thus, by Lemma 13, T_{xz} exists. Similarly, for T_{yz} of join type 2, 3 and 4, it can be shown that $T_{xz} \in X$ exists and $[x, y, z]$ is an agreement triplet. Thus:

Remark 14. If T_{xy} exists as join type 1, for any $T_{yz} \in Y$, T_{xz} exists and $[x, y, z]$ is an agreement triplet.

Let A be a descendant of Y. Let $\mathcal{L}_d = \mathcal{L}_{A^c} - \mathcal{L}_{Y^c}$. Consider potential AST T on leafset $\{\mathcal{L}_{A^c} \cup x\}$ such that $\mathcal{E}_T$ is a descendant of X. For T to exist, for every $\{a, b\} \in \mathcal{L}_d$, joins $\{T_{xa}, T_{xb}\} \in X$ must exist, and, $[x, y, a]$, $[x, y, b]$, $[x, a, b]$ must be agreement triplets. Consider any $\{a, b\} \in \mathcal{L}_d$. Let T_{xy} be of join type 1. Clearly, $T_{ya} \in Y$ exists. Thus, Remark 14 implies that T_{xa} exists and $[x, y, a]$ is an agreement triplet. Similarly, $T_{xb} \in X$ exists and $[x, y, b]$ is an agreement triplet. Since, $[x, y, a]$, $[x, y, b]$ and $[y, a, b]$ are agreement triplets, by Fact 12, $[x, a, b]$ is also an agreement triplet. Thus, there exists an AST T on leafset $\{\mathcal{L}_{A^c} \cup x\}$. Further, for every $\ell \in \{\mathcal{L}_d \cup y\}$ the existence of $T_{x\ell} \in X$ implies that $\mathcal{E}_T$ is a descendant of X. Thus, for every descendant A of Y, there exists at least one descendant $\mathcal{E}_T$ of X such that T displays A^c. Thus, X prunes Y if T_{xy} is of join type 1. Using similar arguments, it can be shown that for T_{xy} of join type 3 and 4, X prunes Y.

(b) It is given that T_{xy} and T_{yz} exist, and $[x, y, z]$ is an agreement triplet. Thus, by Lemma 13, T_{xz} exists. Again, using arguments similar to the proof of part 1a, it can be shown that AST T on leafset $\{\mathcal{L}_{A^c} \cup x\}$ exists and it is a descendant of X. Thus, for every descendant A of Y, there exists at least one descendant $\mathcal{E}_T$ of X such that T displays A^c. Thus, X prunes Y.

2. Let T_{xy} be of join type 2 with triplet $[r, x, y]$ of type $(r, (x, y))$. Let T_{yz} be of join type 1 with triplet $[r, y, z]$ of type (r, y, z). Observe that any tree that displays both $(r, (x, y))$ and (r, y, z) must display $(r, (x, y), z)$ as well. Thus, $[r, x, z]$ and $[x, y, z]$ are also agreement triplets. Thus by Lemma 13, T_{xz} and AST T on leafset $\mathcal{L}_{E^c} \cup \{x, y, z\}$ exist such that $\mathcal{E}_T$ is a descendant of X. Further, $(r, (x, y), z)$ implies that z is the rightmost leaf of T. Thus, T is enumerated by joining ordered pair (T_{xy}, T_{xz}), i.e., $\mathcal{E}_T$ is a child of $\mathcal{E}_{T_{xy}}$. The same can be shown for T_{yz} of join type 3 and 4. Thus:

Remark 15. If T_{xy} exists as join type 2, for any $T_{yz} \in Y$ such that T_{yz} is not of join type 2, the join $T_{x-yz} \in \mathcal{E}_{T_{xy}}$ on ordered pair (T_{xy}, T_{xz}) exists and $[x, y, z]$ is an agreement triplet.

Let A be a descendant of $\mathcal{E}_{T_{yz}}$. Let $\mathcal{L}_d = \mathcal{L}_{A^c} - \mathcal{L}_{T_{yz}}$. Consider potential AST T on leafset

$\{\mathcal{L}_{A^c} \cup x\}$ such that $\mathcal{E}_T$ is a descendant of $\mathcal{E}_{T_{xy}}$. For T to exist, for every $\{a, b\} \in \mathcal{L}_d$, joins $T_{x-ya} \in \mathcal{E}_{T_{xy}}$ on ordered pair (T_{xy}, T_{xa}) and $T_{x-yb} \in \mathcal{E}_{T_{xy}}$ on ordered pair (T_{xy}, T_{xb}) must exist, and $[x, y, a]$, $[x, y, b]$, $[x, z, a]$, $[x, z, b]$, $[x, a, b]$ must be agreement triplets. Let join T_{ya} be of type 2 with triplet $[r, y, a]$ of type $(r, (y, a))$. Since A is a descendant of $\mathcal{E}_{T_{yz}}$, the join $T_{y-za} \in \mathcal{E}_{T_{yz}}$ on ordered pair (T_{yz}, T_{ya}) exists. Let T_{yz} be of type 1 join with triplet $[r, y, z]$ of type (r, y, z). Note that any tree that displays both $(r, (y, a))$ and (r, y, z) must also display $(r, (y, a), z))$. However, $(r, (y, a), z))$ cannot exist in T_{y-za} as a cannot be the rightmost leaf in it. Similarly for T_{yz} of join type 3 or 4, and for T_{ya} of join type 2, it can be shown that a cannot be the rightmost leaf in T_{y-za}. Thus, T_{ya} is not of join type 2. Thus, by Remark 15, the join $T_{x-ya} \in \mathcal{E}_{T_{xy}}$ on ordered pair (T_{xy}, T_{xa}) exists and $[x, y, a]$ is an agreement triplet. Since $[x, y, z]$, $[x, y, a]$, $[y, z, a]$ are agreement triplets, by Fact 12, $[x, z, a]$ is also an agreement triplet. Similarly, $T_{x-yb} \in \mathcal{E}_{T_{xy}}$ exists and, $[x, y, b]$ and $[x, z, b]$ are agreement triplets. Since, $[y, a, b]$, $[x, y, a]$ and $[x, y, b]$ are agreement triplets, by Fact 12, $[x, a, b]$ is also an agreement triplet. Thus, there exists an AST T on leafset $\{\mathcal{L}_{A^c} \cup x\}$. Further, for every $\ell \in \{\mathcal{L}_d \cup z\}$, existence of join $T_{x-y\ell} \in \mathcal{E}_{T_{xy}}$ on ordered pair $(T_{xy}, T_{x\ell})$ implies that T is descendant of $\mathcal{E}_{T_{xy}}$. Thus, for every descendant A of $\mathcal{E}_{T_{yz}}$, there exists at least one descendant $\mathcal{E}_T$ of $\mathcal{E}_{T_{xy}}$ such that T displays A^c. Thus, T_{xy} prunes T_{yz}.

□

Pruner-list. Note that conditions in part (1a) and part (2) of Theorem 11 can be evaluated in constant time while condition in Theorem 11(1b) will take linear time. Neither of these require enumeration of the pruned branch at Y. However, there can be cases when Y or a descendant of Y is pruned by X but it cannot be identified using Theorem 11. For such a case, the branch at Y needs to be enumerated and potential pruning by X verified. For this, we maintain a pruner-list for every child of Y. Let X, Y, Z be children of an equivalence class E in the enumeration tree. Let x, y, z be the right most leaves of X^c, Y^c, Z^c respectively. Let joins T_{xy}, T_{yz} exist such that none of the cases of Theorem 11 hold. Then, *pruner-list* of $\mathcal{E}_{T_{yz}}$ contains x if $[x, y, z]$ is an agreement triplet. The pruner list of $\mathcal{E}_{T_{yz}}$ also inherits members from

the intersection of the pruner lists of $\mathcal{E}_{T_y}$ and $\mathcal{E}_{T_z}$. Now, Y is pruned by an equivalence class A with a as the rightmost leaf of A^c, if all children of Y have a in their pruner-list. This can be shown by using arguments similar to the proof of Theorem 11(1a).

4.2.4 MfstMiner Algorithm

Algorithm 1 gives a high-level description of the MfstMiner algorithm for the special case of enumerating all MXSTs. Here, for AST T_x, x denotes its rightmost leaf. C is the collection of input trees. First, all AST triplets are enumerated and partitioned as the set of all equivalence classes consisting of ASTs on three leaves, denoted by EC_3. Each equivalence class in EC_3 represents a child of the root of the enumeration tree. Subroutine enumerateNode accepts an equivalence class E as input and enumerates the branch at E in the enumeration tree. Lines 7-10 represents the pair-wise joining among members of an equivalence class in the enumeration tree. Pruning as per different cases of Theorem 11 and as per the pruner-list scheme is indicated at respective places in the algorithm. In line 12, an empty equivalence class — representing a leaf node — is queued for potential output as an MXST. This leaf node is produced as output in line 16 if it is found to be not pruned. A non-empty equivalence class is queued for further enumeration in line 14. Such an equivalence class is recursively enumerated in line 21 if it is found to be not pruned.

Algorithm 2 gives a detailed description of the subroutine enumerateNode in Algorithm 1. Children of E that need to be further enumerated are stored in enumList, while those that are potential MXSTs, their core trees are stored in outputList. For a label y, y.joinList stores a label x if join T_{xy} exists and is of join type 2. This is used in identifying pruning as per conditions in part (1b) and part (2) of Theorem 11, and through the pruner-list scheme. Line 10 corresponds to $\mathcal{E}_{T_y}$ being pruned by $\mathcal{E}_{T_x}$ as per Theorem 11(1a). In line 13, join $\mathcal{E}_{T_{xy}}$ inherits members from the intersection of pruner-list of the participating trees $\mathcal{E}_{T_x}$ and $\mathcal{E}_{T_y}$. In line 16, the core tree of the empty equivalence class $\mathcal{E}_{T_x}$ (a leaf node in the enumeration tree) corresponds to a potential MXST and is added to the outputList. In line 20, it is produced as output if it is pruned neither by its siblings, i.e., children of its parent E, nor by the children of any of the ancestor equivalence classes of E (indicated by condition $\mathcal{E}_T$.prunerList $= \emptyset$). If $\mathcal{E}_{T_x}$

Algorithm 1 Enumerating all MXSTs — a high-level description.

MFSTMINER(C)

1: $EC_3 \leftarrow$ enumerateAST_Triplets(C)
2: **for all** $E \in EC_3$ **do**
3: enumerateNode(E)

enumerateNode(E)
4: **for all** $T_x \in E$ **do**
5: $\mathcal{E}_{T_x} \leftarrow \emptyset$
6: **if** $\mathcal{E}_{T_x}$ is not marked pruned by other children of E as per Theorem 11(1a) **then**
7: **for all** $T_y \in E$ such that $T_x \neq T_y$ **do**
8: **if** join T_{xy} exists **then**
9: Mark for potential pruning of other children of E by $\mathcal{E}_{T_x}$ as per Theorem 11(1a)
10: $\mathcal{E}_{T_x} \leftarrow \mathcal{E}_{T_x} \cup T_{xy}$
11: **if** $\mathcal{E}_{T_x} = \emptyset$ **then**
12: queue T_x to be produced as output
13: **else**
14: queue $\mathcal{E}_{T_x}$ for further enumeration
15: **for all** T queued for output such that T is neither pruned as per Theorem 11(1a) nor by the pruner-list scheme **do**
16: **print** T
17: **for all** $\mathcal{E}_{T_y}$ queued for further enumeration **do**
18: **if** any $T_{yz} \in \mathcal{E}_{T_y}$ is pruned as per Theorem 11(2) **then**
19: delete T_{yz} from $\mathcal{E}_{T_y}$
20: **if** $\mathcal{E}_{T_y}$ is neither pruned as per Theorem 11(1b) nor by the pruner-list scheme **then**
21: enumerateNode($\mathcal{E}_{T_y}$)

is not empty, it is added to enumList for potential enumeration of the branch at $\mathcal{E}_{T_x}$. In line 24, $\mathcal{E}_{T_{yz}}$ is pruned by T_{xy} (the join corresponding to $x \in y$.joinList) as per Theorem 11(2). In line 28, label $x \in y$.joinList (corresponding to the type 2 join T_{xy}) is added to pruner-list of $\mathcal{E}_{T_{yz}}$ if $[x, y, z,]$ is an agreement triplet. In line 30, if $\forall T_{yz} \in \mathcal{E}_{T_y}$, $x \in \mathcal{E}_{T_{yz}}$.prunerList comes from a y.joinList in the current iteration of subroutine enumerateNode, this corresponds to pruning as per Theorem 11(1b); else pruning is a result of the pruner-list scheme. If such a $\mathcal{E}_{T_y}$ is not pruned by any of the cases, then it is recursively enumerated (line 32).

4.2.4.1 The General Case of Enumerating MFSTs

Having outlined the algorithmic framework of MFSTMINER for the special case of enumerating all MXSTs, here we discuss the general case of mining MFSTs. The main difference is that

Algorithm 2 Enumerating all MXSTs.

enumerateNode(E)

outputList $\leftarrow \emptyset$, enumList $\leftarrow \emptyset$
for all $T_x \in E$ **do**
  x.joinList $\leftarrow \emptyset$
for all $T_x \in E$ **do**
  $\mathcal{E}_{T_x} \leftarrow \emptyset$
  if $\mathcal{E}_{T_x}$ is not marked pruned **then**
    for all $T_y \in E$ such that $T_x \neq T_y$ **do**
      if join T_{xy} exists **then**
        if T_{xy} is not of join type 2 **then**
          mark $\mathcal{E}_{T_y}$ as pruned { pruning as per Theorem 11(1a)}
        else
          y.joinList $\leftarrow y$.joinList $\cup\, x$
        $\mathcal{E}_{T_{xy}}$.prunerList $\leftarrow \mathcal{E}_{T_x}$.prunerList $\cap\, \mathcal{E}_{T_y}$.prunerList
        $\mathcal{E}_{T_x} \leftarrow \mathcal{E}_{T_x} \cup T_{xy}$
    if $\mathcal{E}_{T_x} = \emptyset$ **then**
      outputList $\leftarrow$ outputList $\cup\, T_x$
    else
      enumList $\leftarrow$ enumList $\cup\, \mathcal{E}_{T_x}$
for all $T \in$ outputList such that $\mathcal{E}_T$ is not pruned **and** $\mathcal{E}_T$.prunerList $= \emptyset$ **do**
  print T
for all $\mathcal{E}_{T_y} \in$ enumList such that $\mathcal{E}_{T_y}$ is not pruned **do**
  for all $T_{yz} \in \mathcal{E}_{T_y}$ **do**
    if T_{yz} is not of join type 2 **and** y.joinList $\neq \emptyset$ **then**
      delete T_{yz} from $\mathcal{E}_{T_y}$ {pruning as per Theorem 11(2)}
    else
      for all $x \in y$.joinList **do**
        if $[x, y, z]$ is an agreement triplet **then**
          $\mathcal{E}_{T_{yz}}$.prunerList $\leftarrow \mathcal{E}_{T_{yz}}$.prunerList $\cup\, x$
  if $\exists x$ such that $\forall T_{yz} \in \mathcal{E}_{T_y}$, $x \in \mathcal{E}_{T_{yz}}$.prunerList **then**
    mark $\mathcal{E}_{T_y}$ as pruned {pruning as per Theorem 11(1b) or due to the pruner-list scheme}
  if $\mathcal{E}_{T_y}$ is not pruned **and** $\mathcal{E}_{T_y} \neq \emptyset$ **then**
    enumerateNode($\mathcal{E}_{T_y}$)

the support for ASTs is always $f = 1$, while FSTs can additionally have any $f \in \left(\frac{1}{2}, 1\right)$. This does not affect the enumeration tree and the pairwise join, however, the support estimation and the pruning strategy need to incorporate the general case when $f \in \left(\frac{1}{2}, 1\right)$. We discuss these steps next.

Support Estimation. Given T_x and T_y in an equivalence class, when $f < 1$, a join T^{xy} on ordered pair (T_x, T_y) can be an FST if it is supported by at least fraction f of the input trees, i.e., these input trees have T^{xy} as a subtree. Note that any such T^{xy} will be supported only by those trees that support both T_x and T_y. Motivated by this, for each FST T_x we maintain the list of all trees in the input collection that support T_x. We call this list the *support-list* of T_x. For a FST T_x, let T_x.supList denote its support-list. Thus, to estimate if the join on ordered pair (T_x, T_y) results in an FST, we apply Theorem 10 only on trees in T_x.supList $\cap$ T_y.supList. We store the the support list as a bitmap representation (Ayres et al. (2002)) for efficient memory utilization and fast computation of intersection of two support-lists using logical operators.

Pruning Strategy. Given equivalence classes X and Y, while verifying if Y is pruned by X, we also need to consider the support-list of X^c and Y^c. We say $[x, y, z]$ is a *frequent triplet* if at least fraction f of the input trees display the same triplet over the leafset $\{x, y, z\}$. Let $[x, y, z]$.supList denote the support-list of such a frequent triplet. Based on this, for the case of enumerating MFSTs, Theorem 11 can be restated as:

Theorem 16. *1. X prunes Y if either of the following holds:*

- *(a) T_{xy} exists, Y^c.supList $\subseteq$ X^c.supList and T_{xy} is not of join type 2.*
- *(b) T_{xy} exists as join type 2, Y^c.supList $\subseteq$ X^c.supList and for every $Z \in E$ such that T_{yz} exists, $[x, y, z]$ is a frequent triplet with Y^c.supList $\subseteq$ $[x, y, z]$.supList.*

2. If T_{xy} and T_{yz}, exist and T_{xy} is of join type 2, then $\mathcal{E}_{T_{xy}}$ prunes $\mathcal{E}_{T_{yz}}$ if T_{yz} is not of join type 2 and T_{yz}.supList $\subseteq$ T_{xy}.supList.

Pruner-list. Pruning cases not identified by Theorem 16, require the use of pruner-list. In the case of MFSTs, along with leaf label the pruner-list also contains the support-list of the

core tree of the equivalence class that is claiming to prune. To explain further, let T_x, T_y, T_z be FSTs in an equivalence class E with x, y, z as their respective rightmost leaves, such that joins T_{xy} and T_{yz} exist, and none of the cases of Theorem 16 hold. Then, the next set of conditions describe pruner-lists for enumerating MFSTs.

1. $\mathcal{E}_{T_{yz}}$.prunerList contains the entry $(x, T_{xy}.\text{supList})$ if T_{xy} is not of join type 2 and $|T_{xy}.\text{supList} \cap T_{yz}.\text{supList}| \geq f$.

2. $\mathcal{E}_{T_{yz}}$.prunerList contains the entry $(x, S_\cap = T_{xy}.\text{supList} \cap [x, y, z].\text{supList})$ if T_{xy} is of join type 2, $[x, y, z]$ is a frequent triplet and $|S_\cap \cap T_{yz}.\text{supList}| \geq f$.

3. For every leaf label w such that $(w, S^y_\cap) \in \mathcal{E}_{T_y}$.prunerList and $(w, S^z_\cap) \in \mathcal{E}_{T_z}$.prunerList exist, $\mathcal{E}_{T_{yz}}$.prunerList contains entry $(w, S_\cap = S^y_\cap \cap S^z_\cap)$ if $|S_\cap \cap T_{yz}.\text{supList}| \geq f$.

Condition 1 and 2 describe addition of new labels to the pruner-list $\mathcal{E}_{T_{yz}}$, while condition 3 describes inheritance of labels from the pruner-list of $\mathcal{E}_{T_y}$ and $\mathcal{E}_{T_z}$. Now, $\mathcal{E}_{T_{yz}}$ is considered pruned by an equivalence class A with a as the rightmost leaf of A^c if for every $T_{yb} \in \mathcal{E}_{T_y}$, $\mathcal{E}_{T_{yb}}$.prunerList contains an entry $(a, S_b(a))$, and, for $S_\cap = \bigcap_{T_{yb} \in \mathcal{E}_{T_y}} S_b(a)$, $\mathcal{E}_{T_{yz}}$.supList $= S_\cap$.

This completes the description of MfstMiner for the general case of mining MFSTs. The overall framework is the same as the special case of mining all MXSTs. The difference lies in the finer details of incorporating support-list in the support estimation and the pruning step. These details were discussed in this section and can be easily incorporated in Algorithm 2.

4.3 Results and Discussion

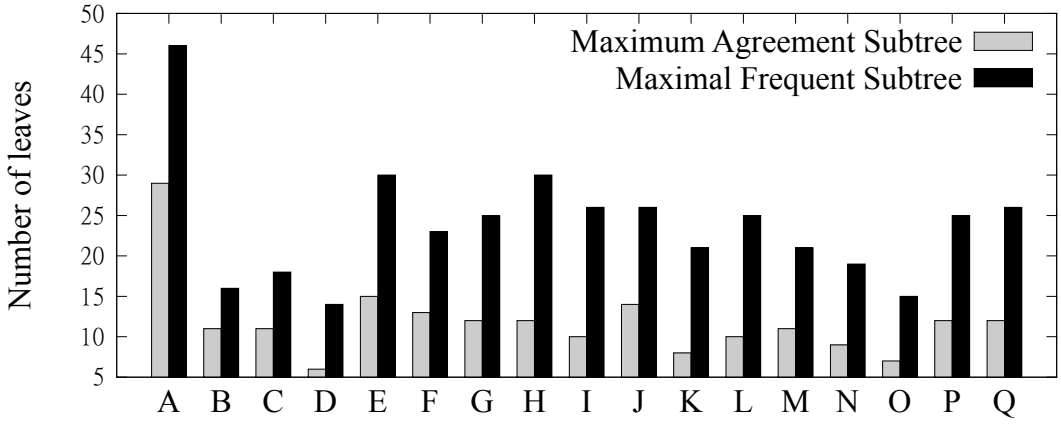
To study the effectiveness of MfstMiner, we conducted three kinds of experiments. The first category compares MFSTs with MAST. The second category compares MfstMiner with Phylominer (Zhang and Wang (2008)) — an algorithm that enumerates all FSTs. The third category evaluates the scalability of MfstMiner with respect to the number of trees and the size of the leafset. Our dataset consists of bootstrapped trees used in a previous study (Pattengale et al. (2011)) on majority rule trees. There are seventeen sets of trees, referred to as $A - Q$, constructed from a diverse range of sequences including rbcL genes, mammalian

sequences, bacterial and archaeal sequences, ITS sequences, fungal sequences , and grasses. To extract datasets of different sizes (in terms of the number of leaves and the number of trees), we randomly selected the required number of trees and restricted them on a random set of leaves of the required size.

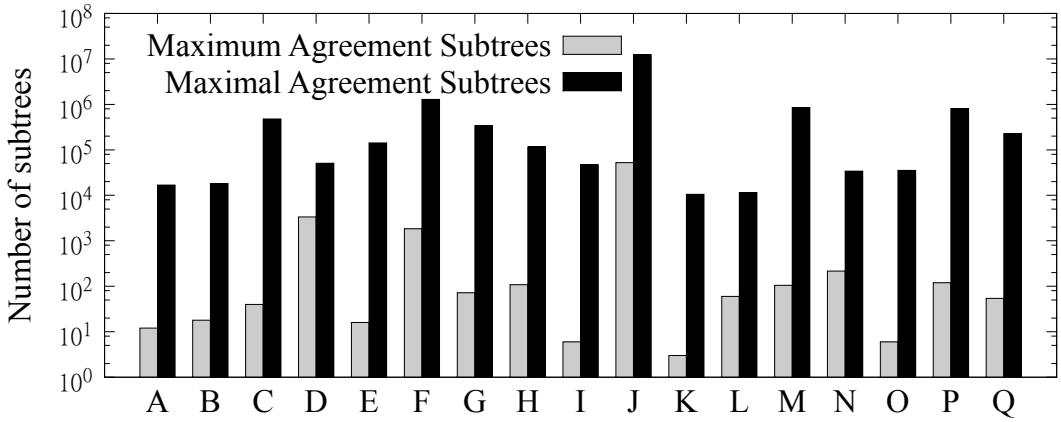
MFSTs vs. MASTs. Figure 4.8a compares the size of the MAST with the size of the largest MFST. This experiment was conducted on a set of 100 trees on 50 leaves from each of the datasets. MFSTs were enumerated for $f = 0.51$. In some cases the size of the largest MFST is more than twice the size of the corresponding MAST. Figure 4.8b compares the number of MASTs with the number of MFSTs for $f = 1$. There are significantly more MFSTs. This is notable because any MFST that is not a MAST is also not displayed by any of the MASTs. Thus, such a MFST reveals unique agreement information among the input trees. This experiment was conducted on a set of 100 trees on 100 leaves from each of the datasets.

Comparison with Phylominer. This experiment was done on dataset A with 100 trees for each of the leafset. Figure 4.9a compares MfstMiner with Phylominer (Zhang and Wang (2008)) for $f = 1$, showing enumerating MFSTs is orders of magnitude faster than enumerating all FSTs. Figure 4.9b shows the corresponding counts for FSTs and, MFSTs and FSTs checked by MfstMiner while enumerating all MFSTs. While the number of MFSTs and the number of FSTs checked by MfstMiner exhibit near-polynomial behavior, the number of FSTs grow exponentially. Figure 4.9c and Figure 4.9d show the corresponding numbers for $f = .95$. For these experiments the physical memory was capped at the 4GB limit. Phylominer uses a scheme that requires all the enumerated FSTs to be kept in memory. This explains the missing entries in the case of Phylominer. MfstMiner uses a depth-first scheme to traverse the enumeration tree and only needs to keep FSTs along a branch. Thus, it is not affected by the memory limit.

Scalability of MfstMiner. Figure 4.10 shows the scalability of MfstMiner with respect to the number of leaves. The first experiment (see Figure 4.10a) was done on dataset P for $f = 1$. The second experiment (see Figure 4.10b) was done on dataset Q for $f = .95$. For each leafset, 100 trees were extracted for the corresponding dataset.



(a) MFSTs have more number of leaves than MASTs, thus, they reveal common agreement over a larger set of taxa.

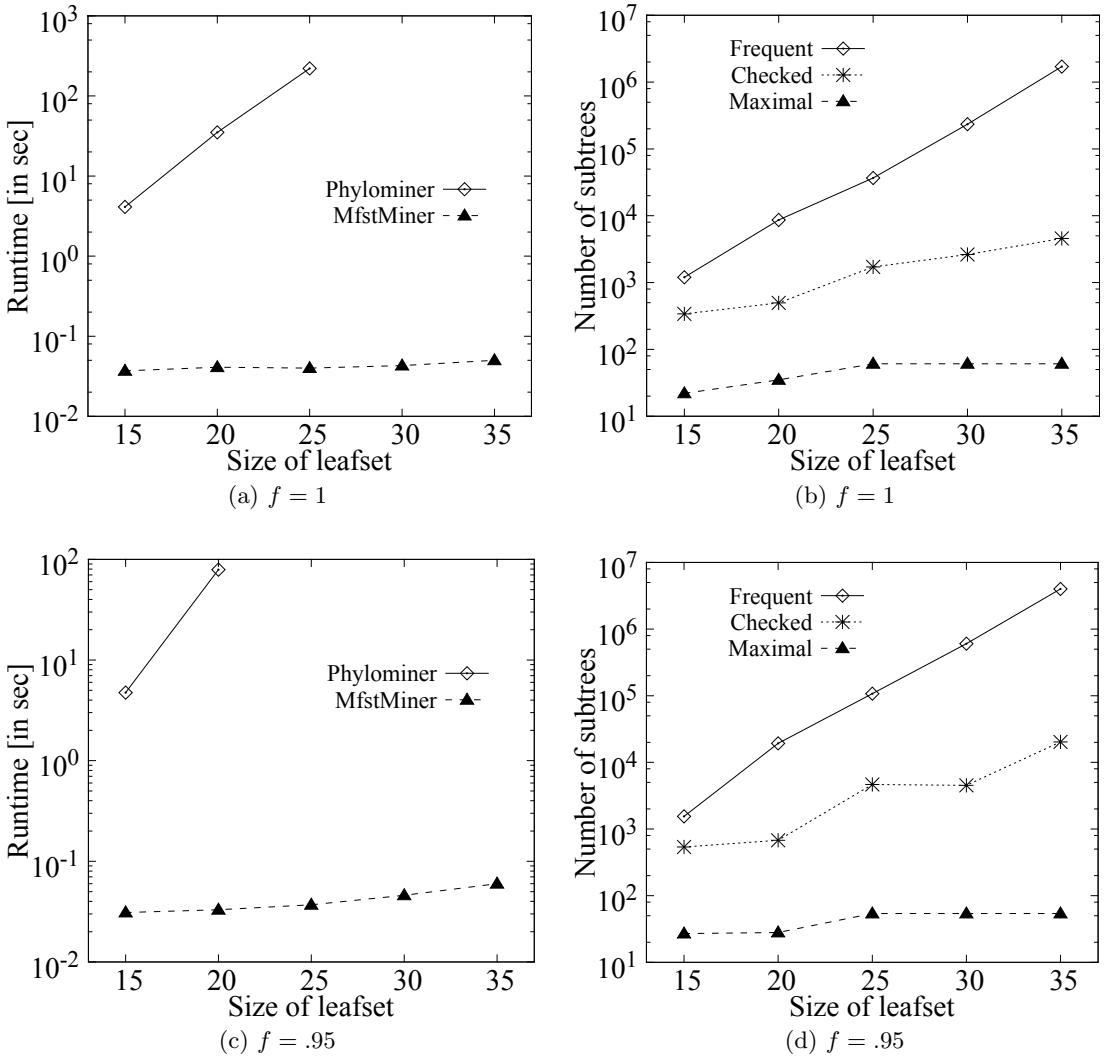


(b) MXSTs are more in number than MASTs, thus, they reveal new agreement trees.

Figure 4.8: Utility of MFSTs over MASTs.

4.4 Conclusions

We introduced a new algorithm to mine MFSTs in collections of phylogenetic trees. Our experiments show that MFSTs can be significantly larger and, more numerous and diverse than the MASTs. The set of all MFSTs is a compact and non-redundant summary of the set of all FSTs. We demonstrated this through experiments on biological datasets and compared the efficiency of our approach with Phylominer (Zhang and Wang (2008)) – an algorithm to enumerate all FSTs. We also showed the scalability of our approach for larger leafsets. The current implementation of MfstMiner can be downloaded from `http://www.cs.iastate.`



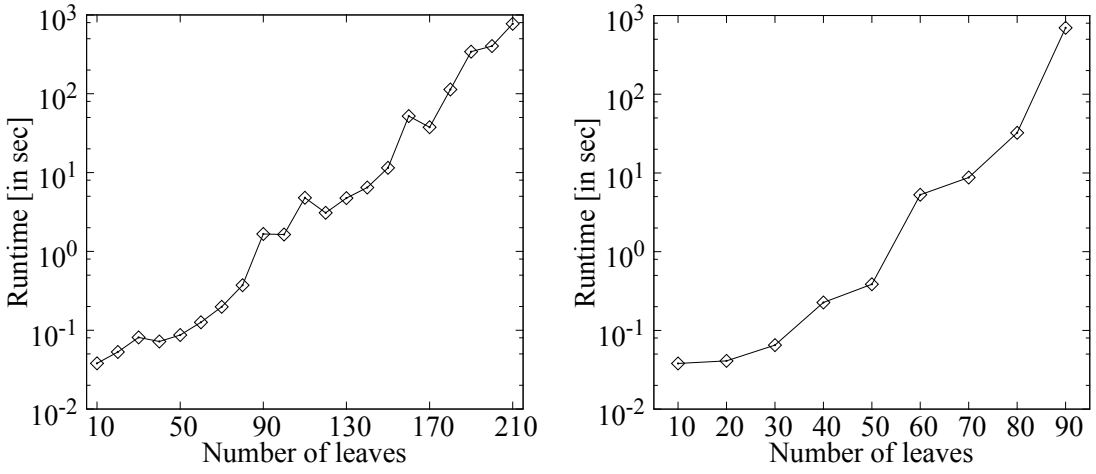
(a) $f = 1$ (b) $f = 1$

(c) $f = .95$ (d) $f = .95$

Figure 4.9: Comparison with Phylominer

edu/akshayd/mfstMiner/. The current implementation works for up to 250 leaves and 10,000 trees.

As a future work, we intend to use MFSTs in practical applications that involve MASTs (Goddard et al. (1994); De Vienne et al. (2007); Daubin et al. (2002)). We note that the enumeration of MFSTs can take long for larger leafsets. In this regard, we intend to develop a scheme that can randomly sample MFSTs – giving a smaller result set that is randomly sampled from the entire solution set. We note that while enumeration of FSTs is possible for $f \in \left(0, \frac{1}{2}\right]$ as well, this can potentially lead to two different FSTs over the same leafset.



(a) $f = 1$ (b) $f = .95$

Figure 4.10: Scalability of MfstMiner

With little modification in the pruning strategy, the proposed algorithms can be extended to enumerate all FSTs for $f \in \left(0, \frac{1}{2}\right]$.

Author's Contributions

AD and DFB conceived the problem, designed the experiments and drafted the manuscript. AD designed and implemented the algorithms, and implemented the experiments. DFB coordinated the project. All authors read and approved the final manuscript.

Acknowledgments

This work was supported in part by National Science Foundation grant DEB-0829674. The authors thank Drs. Sen Zhang and Jason T. L. Wang for sharing the source code of Phylominer and discussions on their work. They also thank Dr. Nicholas D. Pattengale for sharing the set of bootstrapped trees.

CHAPTER 5. EXTRACTING CONFLICT-FREE INFORMATION FROM MULTI-LABELED TREES

Akshay Deepak, David Fernndez-Baca and Michelle M. McMahon

A paper invited for a special issue of the journal *Algorithms for Molecular Biology* devoted to the selected papers from the *Workshop on Algorithms in Bioinformatics*, 2012

A condensed version appeared in the Proceedings of the 12th *Workshop on Algorithms in Bioinformatics*, 2012

Abstract

Background: A multi-labeled tree, or MUL-tree, is a phylogenetic tree where two or more leaves share a label, e.g., a species name. A MUL-tree can imply multiple conflicting phylogenetic relationships for the same set of taxa, but can also contain conflict-free information that is of interest and yet is not obvious.

Results: We define the information content of a MUL-tree T as the set of all conflict-free quartet topologies implied by T, and define the maximal reduced form of T as the smallest tree that can be obtained from T by pruning leaves and contracting edges while retaining the same information content. We show that any two MUL-trees with the same information content exhibit the same reduced form. This introduces an equivalence relation among MUL-trees with potential applications to comparing MUL-trees. We present an efficient algorithm to reduce a MUL-tree to its maximally reduced form and evaluate its performance on empirical datasets in terms of both quality of the reduced tree and the degree of data reduction achieved.

Conclusions: Our measure of conflict-free information content based on quartets is simple and topologically appealing. In the experiments, the maximally reduced form is often much smaller than the original tree, yet retains most of the taxa. The reduction algorithm is quadratic in the number of leaves and its complexity is unaffected by the multiplicity of leaf labels or the degree of the nodes.

5.1 Background

Multi-labeled trees, also known as MUL-trees, are phylogenetic trees that can have more than one leaf with the same label (Fellows et al. (2003); Grundt et al. (2004); Huber and Moulton (2006); Popp and Oxelman (2001); Scornavacca et al. (2011)) (Figure 5.1). MUL-trees arise naturally and frequently in data sets containing multiple gene sequences for the same species (Sanderson et al. (2008)), but they can also arise in biogeographical studies or cospeciation studies where leaves represent individual taxa yet are labeled with their areas (Ganapathy et al. (2006)) or hosts (Johnson et al. (2003)).

MUL-trees, unlike singly-labeled trees, can contain conflicting species-level phylogenetic information due to biological processes such as whole genome duplications (Lott et al. (2009)) or incomplete lineage sorting (Rasmussen and Kellis (2012)), to artifactual processes such as inferential error, or, frequently, an unknown combination of several factors. However, they can also contain substantial amounts of conflict-free information. Here we provide a way to extract this information; specifically, we have the following results.

- We introduce a new quartet-based measure of the information content of a MUL-tree, defined as the set of conflict-free quartets that the tree displays (see Sect. 89).
- We introduce the concept of the maximally-reduced form (MRF) of a MUL-tree T, the smallest tree with the same information content as T (see Sect. 91), and show that any two MUL-trees with the same information content have the same MRF (Theorem 22).
- We present a simple algorithm to construct the MRF of a MUL-tree (see Sect. 98). Its running time is quadratic in the number of leaves and does not depend on the multiplicity of the leaf labels or the degrees of the internal nodes.

- We present computational experience with an implementation of our MRF algorithm (see Sect. 101). In our test data, the MRF is often significantly smaller than the original tree, while retaining most of the taxa.

We now give the intuition behind our notion of information content, deferring the formal definitions of this and other concepts to the next section. Quartets (i.e., sets of four species) are a natural starting point, since they are the smallest subsets from which we can draw meaningful topological information. A singly-labeled tree implies exactly one topology on any quartet. More precisely, each edge e in a singly-labeled tree implies a bipartition (A, B) of the leaf set, where each part is the set of leaves on one of the two sides of e. From (A, B), we derive a collection of bipartitions $ab|cd$ of quartets, such that $\{a, b\} \subseteq A$ and $\{c, d\} \subseteq B$. Clearly, if one edge in a singly-labeled tree implies some bipartition $q = ab|cd$ of $\{a, b, c, d\}$, then there can be no other edge that implies a bipartition, such as $ac|bd$, that is in conflict with q. Indeed, the quartet topologies implied by a singly-labeled tree uniquely identify it (Steel (1992)).

The situation for MUL-trees is more complicated, as illustrated in Figure 5.1. Here, the presence of two copies of labels b and c — $b(1)$ and $b(2)$, and, $c(1)$ and $c(2)$ — leads to two conflicting topologies on the quartet $\{b, c, d, e\}$. Edge (u, v) implies the bipartition $bc|de$, corresponding to the labels $\{b(1), c(1), d, e\}$, while edge (v, w) implies $bd|ce$ corresponding to the leaves $\{b(2), c(2), d, e\}$. On the other hand, the quartet topology $af|bc$, implied by edge (t, u), has no conflict with any other topology that the tree exhibits on $\{a, b, c, f\}$. We show that the set of all such conflict-free quartet topologies is compatible (Theorem 17). That is, for every MUL-tree T there exists at least one singly-labeled tree that displays all the conflict-free quartets of T — and possibly some other quartets as well. Motivated by this, we only view conflict-free quartet topologies as informative, and define the information content of a MUL-tree as the set of all conflict-free quartet topologies it implies.

We should note that conflicting quartets may well provide valuable information, whether about paralogy, deep coalescence, or mistaken annotations. In some cases, species-level phylogenetic information can be recovered from conflicted quartets through application of, e.g., gene-tree species-tree reconciliation (generally an NP-hard problem (Stolzer et al. (2012))). However, this is not feasible when the underlying cause of multiplicity is unknown or when

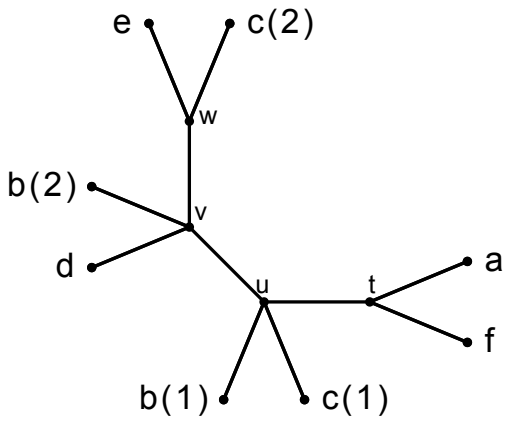


Figure 5.1: A MUL-tree. Numbers in parenthesis next to labels indicate the multiplicity of the respective labels and are not part of the labels themselves.

conducting large-scale analyses. Our definition of information content is deliberately designed to make no assumptions about the cause of conflict. It is also conservative with respect to species relationships, i.e., it does not introduce quartets not originally supported by the data. Further, knowing the information content of a MUL-tree allows us to easily identify its conflicting quartets as well.

A MUL-tree may have leaves that can be pruned and edges that can be contracted without altering the tree's information content, i.e., without adding or removing conflict-free quartets. For example, in Figure 5.1, every quartet topology that edge (v, w) implies is either in conflict with some other topology (e.g., for set $\{b, c, d, e\}$) or is already implied by some other edge (e.g., $af|ce$ is also implied by (t, u)). Thus, (v, w) can be contracted without altering the information content. In fact, the information content remains unchanged if we also contract (u, v) and remove the leaves labeled $b(1)$ and $c(1)$. We define the MRF of a MUL-tree T as the tree that results from applying information-preserving edge contraction and leaf pruning operations repeatedly to T, until it is no longer possible to do so. The MRF of the tree in Figure 5.1 is shown in Fig. 5.2. In this case, the MRF is singly-labeled; however, this is not true in general (see Sect. 100). If the MRF is itself a MUL-tree, it is not possible to reduce the original to a singly-labeled tree without either adding at least one quartet that did not exist conflict-free in T or by losing one or more conflict-free quartets.

Since any two MUL-trees with the same information content have the same MRF, rather

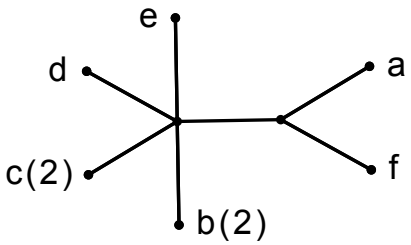


Figure 5.2: The MRF for the MUL-tree in Fig. 5.1.

than comparing MUL-trees directly, we can instead compare their MRFs. This is appealing mathematically, because it focuses on conflict-free information content, and also computationally, since an MRF can be much smaller than the original MUL-tree. Indeed, on our test data, the MRF was frequently singly-labeled. This reduction in input size is especially significant if the MUL-tree is an input to an algorithm whose running time is exponential in the label multiplicity, such as Ganapathy et al.'s algorithm to compute the contract-and-refine distance between two area cladograms (Ganapathy et al. (2006)) or Huber et al.'s algorithm to determine if a collection of "multi-splits" can be displayed by a MUL-tree (Huber et al. (2008)).

For our experiments, we also implemented a post-processing step, which converts the MRF to a singly-labeled tree, rendering it available for analyses that require singly-labeled trees, including supermatrix (de Queiroz and Gatesy (2007); Wiens and Reeder (1995)) and supertree methods (Baum (1992); Ragan (1992); Bansal et al. (2010); Swenson et al. (2012)). On the trees in our data set, the combined taxon loss between the MRF computation and the post processing was much lower than it would have been had we simply removed all duplicate taxa from the original trees.

Previous work on MUL-trees has concentrated on finding ways to reduce MUL-trees to singly-labeled trees (typically in order to provide inputs to supertree methods) (Scornavacca et al. (2011)), and to develop metrics and algorithms to compare MUL-trees (Ganapathy et al. (2006); Puigbò et al. (2007); Marcet-Houben and Gabaldón (2011); Huber et al. (2011)). In contrast to our approach — which is purely topology-based and is agnostic with respect to the cause of label multiplicity — the assumption underlying much of the literature on MUL-trees is that taxon multiplicity results from gene duplication. Thus, methods to obtain singly-

labeled trees from MUL-trees usually work by pruning subtrees at putative duplication nodes. Although the proposed algorithms are polynomial, they are unsatisfactory in various ways. For example, in Scornavacca et al. (2011) if the subtrees are neither identical nor compatible, then the subtree with smaller information content is pruned, which seems to discard too much information. Further, the algorithm is only efficient for binary rooted trees. In Puigbò et al. (2007) subtrees are pruned arbitrarily, while in Marcet-Houben and Gabaldón (2011) at each putative duplication node a separate analysis is done for each possible pruned subtree. Although the latter approach is better than pruning arbitrarily, in the worst case it can end up analyzing exponentially many subtrees.

5.2 MUL-trees and Information Content

A *MUL-tree* is a triple (T, M, ψ), where (i) T is an unrooted tree[1] with leaf set $\mathcal{L}(T)$ all of whose internal nodes have degree at least three, (ii) M is a set of labels, and (iii) $\psi_T : \mathcal{L}(T) \to M$ is a surjective map that assigns each leaf of T a label from M. (Note that if ψ is a bijection, T is singly labeled; that is, singly-labeled trees are a special case of MUL-trees.) For brevity we often refer to a MUL-tree by its underlying tree T. In what follows, unless stated otherwise, by a tree we mean a MUL-tree.

An edge (u, v) in T is *internal* if neither u nor v belong to $\mathcal{L}(T)$, and is *pendant* otherwise. A *pendant node* is an internal node that has a leaf as its neighbor.

Let (u, v) be an edge in T and T' be the result of deleting (u, v) from T. Then T_u^{uv} (T_v^{uv}) denotes the subtree of T' that contains u (v). M_u^{uv} (M_v^{uv}) denotes the set of labels in T_u^{uv} (T_v^{uv}) but not in T_v^{uv} (T_u^{uv}). C^{uv} is the set of labels common to both T_u^{uv} and T_v^{uv}. Observe that M_u^{uv}, M_v^{uv} and C^{uv} partition M. For example, in Figure 5.1, $M_u^{uv} = \{a, f\}$, $M_v^{uv} = \{e, d\}$, $C^{uv} = \{b, c\}$.

A (resolved) *quartet* in a MUL-tree T is a bipartition $ab|cd$ of a set of labels $\{a, b, c, d\}$ such that there is an edge (u, v) in T with $\{a, b\} \in M_u^{uv}$ and $\{c, d\} \in M_v^{uv}$. We say that (u, v) *resolves* $ab|cd$. For example, in Figure 5.1, edge (t, u) resolves $af|bc$.

[1]The results presented here can be extended to rooted trees, using triplets instead of quartets, exploiting the well-known bijection between rooted and unrooted trees (Semple and Steel, 2003b, p. 20).

The *information content of an edge* (u, v) of a MUL-tree T, denoted $\Delta(u, v)$, is the set of quartets resolved by (u, v). An edge (u, v) in tree T is *informative* if $|\Delta(u, v)| > 0$; (u, v) is *maximally informative* if there is no other edge (u', v') in T with $\Delta(u, v) \subset \Delta(u', v')$. The *information content* of T, denoted $\mathcal{I}(T)$, is the combined information content of all edges in the tree; that is $\mathcal{I}(T) = \bigcup_{(u,v)\in E} \Delta(u, v)$, where E denotes the set of edges in T.

The next result shows that the quartets in $\mathcal{I}(T)$ are conflict-free.

Theorem 17. *For every MUL-tree T, there is a singly labeled tree T' such that $\mathcal{I}(T) \subseteq \mathcal{I}(T')$.*

Proof. Repeat the following step until T has no multiply-occurring labels. Pick any multiply-occurring label ℓ in T, select an arbitrary leaf labeled by ℓ, and relabel every other leaf labeled by ℓ, by a new, unique, label. The resulting tree T' is singly labeled, and all labels of T are also present in T'. Consider a quartet $ab|cd$ in T, that is resolved by edge (u, v). Assume that $\{a, b\} \in M_u^{uv}$ and $\{c, d\} \in M_v^{uv}$. Thus, T_u^{uv} contains all the occurrences of label a. Clearly, this also holds for the only occurrence of a in T'. Similar statements can be made about labels b, c, and d. Thus, the quartet $ab|cd$ is resolved by edge (u, v) in T', and, hence, T' displays all quartets of T. □

Note that there are examples where the containment indicated by the above result is proper.

To conclude this section, we give some results that are useful for the MUL-tree reduction algorithm (see Sect. [sec:Algorithm]). In the next lemmas, (u, v) and (w, x) denote two edges in tree T that lie on the path $P_{u,x} = (u, v, \ldots, w, x)$ as shown in Fig. 5.3.

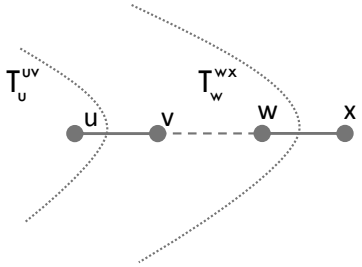


Figure 5.3: Supportive illustration for the proof of Lemma 18

Lemma 18. *If $|M_u^{uv}| = |M_w^{wx}|$ then $M_u^{uv} = M_w^{wx}$. Otherwise, $M_u^{uv} \subset M_w^{wx}$.*

Proof. Refer to Fig. 5.3. Since T_u^{uv} is a subtree of T_w^{wx}, $M_u^{uv} \subseteq M_w^{wx}$ by definition of M_u^{uv}. Thus, if $|M_u^{uv}| = |M_w^{wx}|$, we must have $M_w^{wx} = M_u^{uv}$ and, if $|M_u^{uv}| \neq |M_w^{wx}|$, we must have $M_u^{uv} \subset M_w^{wx}$. □

Together with Lemma 18, the next result allows us to check whether the information content of an edge is a subset of that of another based solely on the cardinalities of the M_u^{uv}s.

Lemma 19. *$\Delta(u, v) \subseteq \Delta(w, x)$ if and only if $M_v^{uv} = M_x^{wx}$.*

Proof. (Only if) Suppose $\Delta(u, v) \subseteq \Delta(w, x)$; therefore, $M_v^{uv} \subseteq M_x^{wx}$. By definition, $M_v^{uv} \supseteq M_x^{wx}$; hence, $M_v^{uv} = M_x^{wx}$.

(If) Suppose $M_v^{uv} = M_x^{wx}$. By definition, $M_u^{uv} \subseteq M_w^{wx}$, which implies that $\Delta(u, v) \subseteq \Delta(w, x)$. □

Lemma 20. *Suppose $\Delta(u, v) \subseteq \Delta(w, x)$. Then, for any edge (y, z) on $P_{u,x}$ such that v is closer to y than to z, $\Delta(u, v) \subseteq \Delta(y, z) \subseteq \Delta(w, x)$.*

Proof. By Lemma 19, since $\Delta(u, v) \subseteq \Delta(w, x)$, we have $M_v^{uv} = M_x^{wx}$. Now consider an edge (y, z) on $P_{u,x}$. By definition $M_v^{uv} \supseteq M_z^{yz} \supseteq M_x^{wx}$. But $M_v^{uv} = M_x^{wx}$, therefore $M_v^{uv} = M_z^{yz} = M_x^{wx}$. By definition $M_u^{uv} \subseteq M_y^{yz} \subseteq M_w^{wx}$. Hence, by Lemma 19, $\Delta(u, v) \subseteq \Delta(y, z) \subseteq \Delta(w, x)$. □

5.3 Maximally Reduced MUL-trees

Our goal is to provide a way to reduce a MUL-tree T as much as possible, while preserving its information content. Our reduction algorithm uses the following operations.

Prune(v)**:** Delete leaf v from T. If, as a result, v's neighbor u becomes a degree-two node, connect the former two neighbors of u by an edge and delete u.

Contract(e)**:** Delete an internal edge e and identify its endpoints.

A leaf v in T is *prunable* if the tree that results from pruning v has the same information content as T. An internal edge e in T is *contractible* if the tree that results from contracting e

has the same information content as T. T is *maximally reduced* if it has no prunable leaf and no contractible internal edge.

Theorem 21. *Every internal edge in a maximally reduced tree T resolves a quartet that is resolved by no other edge.*

Proof. We rely on two facts. First, every internal node in the tree has degree at least three. Second, every internal edge in the tree resolves a quartet; otherwise, the edge would be contractible and the tree would not be maximally reduced.

Consider any edge (u, v) in the tree. To prove that (u, v) resolves a quartet not resolved by any other edge, we need to show that there exists a quartet $ab|cd$ of the form shown in Fig. 5.4. First, we describe how to select leaves a and b. Consider the following cases:

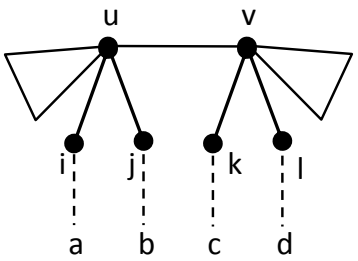


Figure 5.4: Supportive illustration for the proof of Theorem 21. Quartet $ab|cd$ is resolved only by edge (u, v). Here, $a \in M_i^{ui}$, $b \in M_j^{uj}$, $c \in M_k^{vk}$ and $d \in M_l^{vl}$.

u has at least two neighbors i and j, apart from v, that are internal nodes. Then, we select any $a \in M_i^{ui}$ and any $b \in M_j^{uj}$.

u has only one neighbor $i \neq v$ that is an internal node. Then, at least one of u's neighboring leaves must participate in a quartet that (u, v) resolves. Without such a leaf, (u, v) would resolve the same set of quartets as (u, i), so one of these two edges would be contractible, contradicting the assumption that the tree is maximally reduced. We select this leaf as b and we select any $a \in M_i^{ui}$.

All neighbors of u, except v, are leaves. Then, at least two of its neighbors must participate in a quartet, because (u, v) must resolve a quartet. We select the two neighbors as a and b.

In every case, we can select the desired leaves a and b. By a similar argument, we can also select the desired c and d. This proves the existence of the desired quartet $ab|cd$. Therefore,

each internal edge of T uniquely resolve a quartet. □

The next result shows that the set of quartets resolved by a maximally reduced tree uniquely identifies the tree.

Theorem 22. *Let T and T' be two maximally reduced trees such that $\mathcal{I}(T) = \mathcal{I}(T')$. Then, T and T' are isomorphic.*

The *maximally reduced form* (MRF) of a MUL-tree T is the tree that results from repeatedly pruning prunable leaves and contracting contractible edges from T until this is no longer possible. Theorem 22 shows that we can indeed talk about "the" MRF of T. Before proving Theorem 22, we mention some of its consequences.

Corollary 23. *Every MUL-tree has a unique MRF.*

Corollary 24. *Any two MUL-trees with the same information content have the same MRF.*

Corollary 25. *If a maximally reduced MUL-tree T is not singly-labeled, there does not exist a singly-labeled tree T' such that $\mathcal{I}(T) = \mathcal{I}(T')$.*

Note that Corollary 25 does not contradict Theorem 17. If the MUL-tree in Theorem 17 is maximally reduced and not singly-labeled, the containment is proper; i.e., $\mathcal{I}(T) \neq \mathcal{I}(T')$, which is the claim of Corollary 25. Figure 5.5 illustrates this. Any singly-labeled tree resolving the same set of quartets must be obtained by removing one of the leaves labeled with f. However, doing so will also introduce quartets that are not resolved by the maximally reduced MUL-tree.

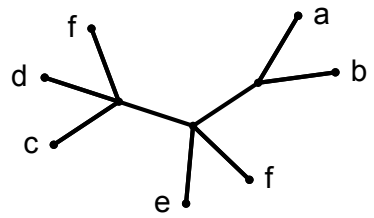


Figure 5.5: A maximally reduced MUL-tree

Corollary 26. *The relation "sharing a common MRF" is an equivalence relation on the set of MUL-trees.*

The last result implies that MUL-trees can be partitioned into equivalence classes, where each class consists of the set of all trees with the same information content. Thus, instead of comparing MUL-trees directly, we can compare their maximally reduced forms.

We now proceed to the proof of Theorem 22. We need two lemmas.

Lemma 27. *There is a bijection ϕ between the respective sets of internal edges of T and T' with the following property. Let (u, v) be an internal edge in T and let $(u', v') = \phi(u, v)$. Then, $M_u^{uv} = M_{u'}^{u'v'}$ and $M_v^{uv} = M_{v'}^{u'v'}$. Therefore, $\Delta(u, v) = \Delta(u', v')$.*

Proof. Consider an edge (u, v) in T. By Theorem 21, (u, v) must resolve a quartet $ab|cd$ not resolved by any other edge as shown in Fig. 5.4. We claim that this quartet must be resolved uniquely by an edge (u', v') in T'. Suppose not. Using arguments similar to those in the proof of Lemma 20, we can show that all edges that resolve $ab|cd$ in T' form a path $(u', x', \ldots, w', v')$, where possibly $x' = w'$, as shown in Fig. 5.6. Here, $\{a, b\} \subseteq M_{u'}^{u'x'}$ and $\{c, d\} \subseteq M_{v'}^{w'v'}$.

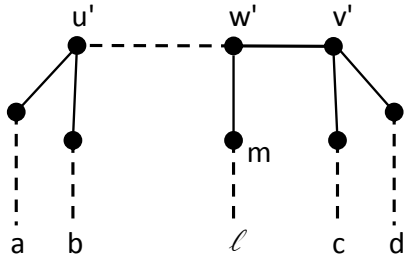


Figure 5.6: Supportive illustration for the proof of Lemma 27

Since (w', v') resolves a quartet not resolved by any other edge, by Theorem 21 there exists a label ℓ as shown, where $\ell \in M_m^{w'm}$. Since $ab|\ell d$ is a quartet in T' and $\mathcal{I}(T) = \mathcal{I}(T)$, it must be true that $\ell \in M_v^{uv}$ in T. Clearly, T does not resolve the quartet on $\{a, \ell, d, c\}$ in the same way, $a\ell|cd$, as T'. This contradicts the assumption that $\mathcal{I}(T) = \mathcal{I}(T')$. Thus, (u', v') must be an edge. Moreover, only one such edge exists in T' as it uniquely resolves the quartet $ab|cd$.

Now consider any label $f \in M_u^{uv}$ such that $f \notin \{a, b, c, d\}$. Label f must be in $M_{u'}^{u'v'}$; otherwise, T and T' would resolve the quartet $\{a, f, c, d\}$ differently. Similarly, any such $f \in M_{u'}^{u'v'}$ must be in M_u^{uv} as well. Thus $M_u^{uv} = M_{u'}^{u'v'}$. In the same way, we can prove that $M_v^{uv} = M_{v'}^{u'v'}$. Thus, $\Delta(u, v) = \Delta(u', v')$.

We have shown that there is a one-to-one mapping ϕ from edges in T to edges of T' such that $\Delta(e) = \Delta(\phi(e))$. To complete the proof, we show that ϕ is onto. Suppose that for some edge e' in T' there is no edge e in T such that $\phi(e) = e'$. But then e' must resolve a quartet not resolved by any other edge in T'. This quartet cannot be in $\mathcal{I}(T)$, contradicting the assumption that $\mathcal{I}(T) = \mathcal{I}(T')$. □

Let ϕ be the bijection between the edge sets of T and T' from the preceding lemma.

Lemma 28. *Let (u, v) and (v, x) be any two neighboring internal edges in T, and let $(p, q) = \phi(u, v)$ and $(r, s) = \phi(v, x)$ be the corresponding edges in T' such that $M_u^{uv} = M_p^{pq}$ and $M_v^{vx} = M_r^{rs}$. Then, (p, q) and (r, s) are neighbors in T' with $q = r$.*

Proof. Since (u, v) and (v, x) are neighbors, and each resolves a quartet that is not resolved by the other, $M_u^{uv} \subset M_v^{vx}$ and $M_v^{uv} \supset M_x^{vx}$. By Lemma 27, this implies that $M_p^{pq} \subset M_r^{rs}$ and $M_q^{pq} \supset M_s^{rs}$. Thus, the only way (p, q) and (r, s) can exist in T' is as part of the path $P_{p,s} = (p, q, \ldots, r, s)$. If $q \neq r$, then consider the edge (t, r) on P_{ps} such that p is closer to t than to r. Then, the following must hold:

$$M_p^{pq} \subset M_t^{tr} \subset M_r^{rs} \tag{5.1}$$

and

$$M_q^{pq} \supset M_r^{tr} \supset M_s^{rs} \tag{5.2}$$

Let $(z, w) = \phi^{-1}(t, r)$ be the edge in T corresponding to (t, r). Irrespective of the position of (z, w) in T, (5.1) and (5.2) cannot be simultaneously true with respect to edges (u, v), (v, x) and (z, w) in T. Therefore, $q = r$, which proves the desired result. □

Proof of Theorem 22. Lemmas 27 and 28 show that T and T' are isomorphic with respect to their internal edges. It remains to show a one-to-one correspondence between their leaf sets. For this, we match up the leaves attached at every pendant node in T and T'. We start with pendant nodes to which only one internal edge is attached. For example, consider an internal edge (u, v) in T such that v is a pendant node and T_v^{uv} has only leaves. Let $(u', v') = \phi(u, v)$ be the corresponding edge in T' such that $M_u^{uv} = M_{u'}^{uv}$. By Lemma 27, $C^{uv} = C^{u'v'}$. Moreover,

neither T nor T' have prunable leaves. Thus, the same set of leaves must be attached at v and v' respectively. In subsequent steps, we select an internal edge (u, v) in T such that v is a pendant node and all the other pendant nodes in T_v^{uv} have already been matched up in previous iterations. Again, let $(u', v') = \phi(u, v)$ such that $M_u^{uv} = M_{u'}^{uv}$. Using similar arguments, the same set of leaves must be attached at v and v' respectively. Proceeding this way, each pendant node in T can be paired with the corresponding pendant node in T', and be shown to have the same set of leaves attached to them. This shows that T and T' are isomorphic, as claimed. □

5.4 Identifying Contractible Edges and Prunable Leaves

In preparation for the MUL-tree reduction algorithm of the next section, we give some results that help to identify contractible edges and prunable leaves.

The setting for the next result is the same as for Lemmas 19 and 20: (u, v) and (w, x) are two edges in tree T that lie on the path $P_{u,x} = (u, v, \ldots, w, x)$ (see Fig. 5.3). We say that subtree T_z^{yz} *branches out* from the path $P_{u,x}$ if $y \in P_{u,x} - \{u, x\}$, and $z \notin P_{u,x}$.

Lemma 29. *Suppose* $\Delta(u, v) \subseteq \Delta(w, x)$ *then*

1. every internal edge on a subtree branching out from $P_{u,x}$ is contractible, and
2. if $\Delta(u, v) = \Delta(w, x)$, every leaf on a subtree branching out from $P_{u,x}$ is prunable. Thus, the entire subtree can be deleted without changing the information content of the tree.

Proof. Refer to Fig. 5.7.

1. Consider any edge (a, b) in a subtree branching out of $P_{u,x}$, as shown. We claim that $M_a^{ab} \cup C^{ab} = M$; i.e., all the labels in M appear in T_a^{ab}. This means that $M_b^{ab} = \emptyset$, so (a, b) is uninformative.

   To prove the claim, observe first that, by definition, $M_u^{uv} \cup C^{uv} \cup M_v^{uv} = M$. By Lemma 19, since $\Delta(u, v) \subseteq \Delta(w, x)$, we have $M_x^{wx} = M_v^{uv}$, so

$$M_u^{uv} \cup C^{uv} \cup M_x^{wx} = M. \tag{5.3}$$

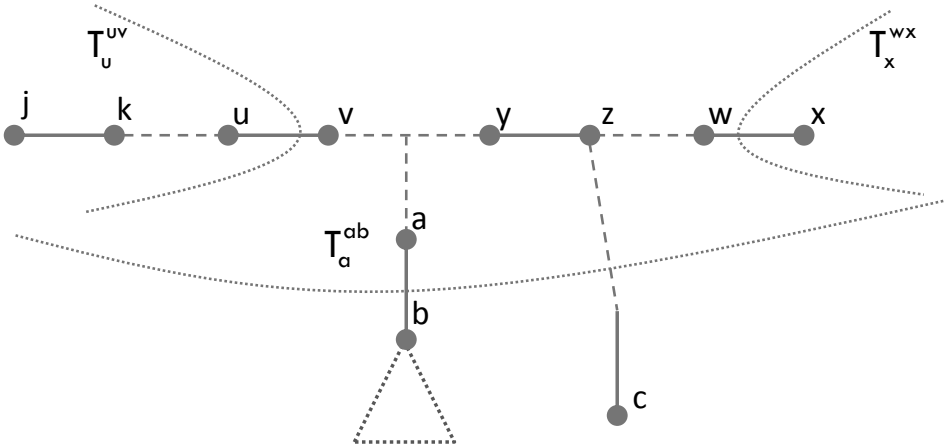


Figure 5.7: Supportive illustration for the proof of Lemma 29

Now, $M_u^{uv} \cup C^{uv}$ is the set of labels on the leaves of T_u^{uv}, while every label in M_x^{wx} appears in T_x^{wx}. Hence, T_u^{uv} and T_x^{wx} jointly contain every label in M. Since T_u^{uv} and T_x^{wx} are subtrees of T_a^{ab}, this completes the proof of the claim.

2. Suppose $\Delta(u, v) = \Delta(w, x)$. By an argument similar to the one used in the proof of Lemma 20, we can show that any edge (y, z) on the path $P_{v,w} = (v \dots w)$ (see Fig. 5.7) satisfies $M_v^{uv} = M_z^{yz} = M_x^{wx}$ and $M_u^{uv} = M_y^{yz} = M_w^{wx}$. Consider a leaf c as shown; let ℓ be its label. Then, ℓ appears in T_x^{wx}, for else $M_y^{yz} \neq M_w^{wx}$, a contradiction. Similarly, ℓ appears in T_u^{uv}. Now, let S be the tree obtained after pruning leaf c.

   (a) $\mathcal{I}(T) \subseteq \mathcal{I}(S)$: Suppose pruning c removes a quartet from $\mathcal{I}(T)$. If such a quartet exists in T, it must be resolved by an edge $(j, k) \in T_u^{uv}$ (say). But then (j, k) still resolves the same quartet in S because $\ell \in M_x^{wx}$, and the labels in T_x^{wx} are a subset of those in T_k^{jk}. This is a contradiction.

   (b) $\mathcal{I}(S) \subseteq \mathcal{I}(T)$: Suppose pruning c adds a quartet to $\mathcal{I}(S)$ that is not in $\mathcal{I}(T)$. Such a quartet in S must be resolved by an edge (j, k) in S_u^{uv} (say), that before pruning satisfied $\ell \in C^{jk}$, but now has $\ell \notin M_k^{jk}$. However $\ell \in M_x^{wx}$; therefore we still have $\ell \in C^{jk}$ and the edge still cannot resolve the quartet, a contradiction.

Hence, c is prunable.

□

Lemma 30. *Suppose that T is a MUL-tree where no pendant node is adjacent to two or more leaves with the same label. Let ℓ be any multiply-occurring label in T and let T' be the minimal subtree of T that spans all the leaves labeled by ℓ. Then, any leaf in T labeled ℓ attached to a pendant node of degree at least three in T' is prunable.*

Proof. Refer to Fig. 5.8. Consider any pendant node v of degree at least three in T' attached to a leaf labeled ℓ. Clearly deleting the leaf does not change the information content of any edge in T_u or T_y. Now consider an edge (w, x) in T' as shown. Note that $\ell \in C^{wx}$, so ℓ does not contribute to $\Delta(w, x)$. After deleting the leaf, we still have $\ell \in C^{wx}$, so $\Delta(w, x)$ remains unchanged. Therefore, the leaf is prunable.

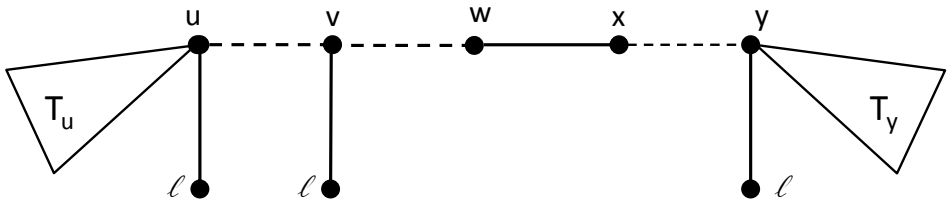


Figure 5.8: Supportive illustration for the proof of Lemma 30. The leaves attached to pendant nodes u, v, and y are labeled by ℓ, and the subtrees indicated by T_u and T_y do not contain a leaf labeled with ℓ. Nodes u and y have degree two in T', while v has degree three.

□

5.5 The Reduction Algorithm

We now describe a $O(n^2)$ algorithm to compute the MRF of an n-leaf MUL-tree T. In the previous section, the MRF was defined as the tree obtained by applying information-preserving pruning and contraction operations to T, in any order, until it is no longer possible. For efficiency, however, the sequence in which these steps are performed is important. Our algorithm has three distinct phases: a preprocessing step, redundant edge contraction, and pruning of redundant leaves. We describe these next and then give an example.

5.5.1 Preprocessing

For every edge (u, v) in T, we compute $|M_u^{uv}|$ and $|M_v^{uv}|$. This can be done in $O(n^2)$ time as follows. First, traverse subtrees T_u^{uv} and T_v^{uv} to count number of distinct labels n_u^{uv} and n_v^{uv} in each subtree. Then, $|M_u^{uv}| = |M| - n_v^{uv}$ and $|M_v^{uv}| = |M| - n_u^{uv}$. We then contract non-informative edges; i.e., edges (u, v) where $|M_u^{uv}|$ or $|M_v^{uv}|$ is at most one.

5.5.2 Edge Contraction and Subtree Pruning

Next, we repeatedly find pairs of adjacent edges (u, v) and (v, w) such that $\Delta(u, v) \subseteq \Delta(v, w)$ or vice versa, and contract the less informative of the two. By Lemmas 18 and 19, we can compare $\Delta(u, v)$ and $\Delta(v, w)$ in constant time using the precomputed values of $|M_u^{uv}|$ and $|M_v^{uv}|$. Lemma 29(1) implies that we should also contract all internal edges incident on v or in the subtrees branching out of v. Further, by Lemma 29(2), if $\Delta(u, v) = \Delta(v, w)$, we can in fact delete these subtrees entirely, since their leaves are prunable. Lemma 20 implies that all such edges must lie on a path, and hence can be identified in linear time. The total time for all these operations is linear, since at worst we traverse every edge twice.

5.5.3 Pruning Redundant Leaves

The tree that is left at this point has no contractible edges; however, it can still have prunable leaves. We first prune any leaf with a label ℓ that does not participate in any resolved quartet. Such an ℓ has the property that for every edge (u, v), $\ell \notin M_u^{uv}$ and $\ell \notin M_v^{uv}$. All such leaves can be found in $O(n^2)$ time and $O(n)$ space.

Next, we consider sets of leaves with the same label ℓ that share a common neighboring pendant node. Such leaves can be found in linear time. For each such set, we delete all but one element. Let T be the tree that results from removing such leaves. Now, the only prunable leaves with a given label ℓ that might remain are leaves attached to different pendant nodes. By Lemma 30, we can identify and prune such leaves by performing the following steps.

1. For each label ℓ, consider the subgraph on the leaves labeled by ℓ.

2. In this subgraph, delete any leaf not attached to a degree 2 pendant node as it is a redundant leaf.

This takes $O(n)$ time per label and $O(n^2)$ time total. The space used is $O(n)$. Hence, the overall time and space complexities are $O(n^2)$ time and $O(n)$, respectively.

The resulting tree has no contractible edges nor prunable leaves. Therefore, it is the MRF of the original MUL-tree.

5.5.4 An Example

We illustrate the reduction of the unrooted MUL-tree shown in Fig. 5.9 to its MRF.

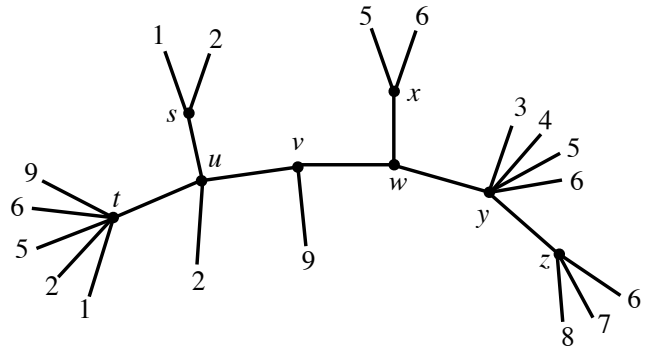


Figure 5.9

1. In the preprocessing step, we find that $M_t^{tu} = \emptyset$, $M_s^{su} = \emptyset$ and $M_x^{wx} = \emptyset$, so edges (t, u), (s, u) and (w, x) are uninformative. They are therefore contracted, resulting in the tree shown in Fig. 5.10.

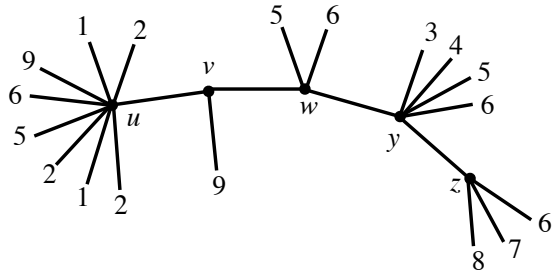


Figure 5.10

2. Since $\Delta(u, v) \subset \Delta(v, w)$, contract (u, v). The result is shown in Fig. 5.11.

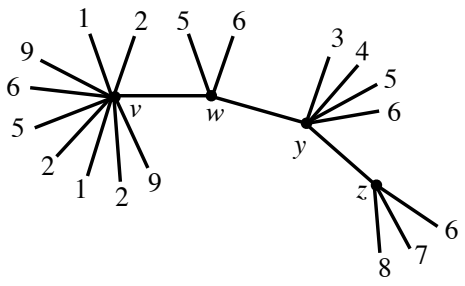


Figure 5.11

3. Since $\Delta(v, w) = \Delta(w, y)$, delete the subtree branching out at w from the path from v to y and contract (v, w). The result is shown in Fig. 5.12.

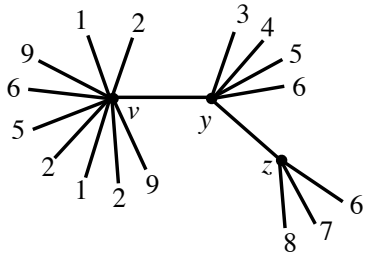


Figure 5.12

4. Prune taxon 6, which does not participate in any quartet, and all duplicate taxa at the pendant nodes. The result, shown in Fig. 5.13, is the MRF of the original tree.

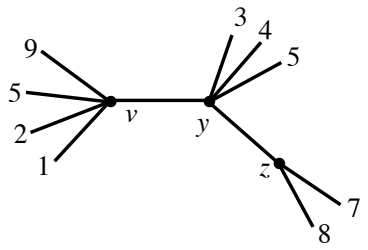


Figure 5.13

5.6 Results and Discussion

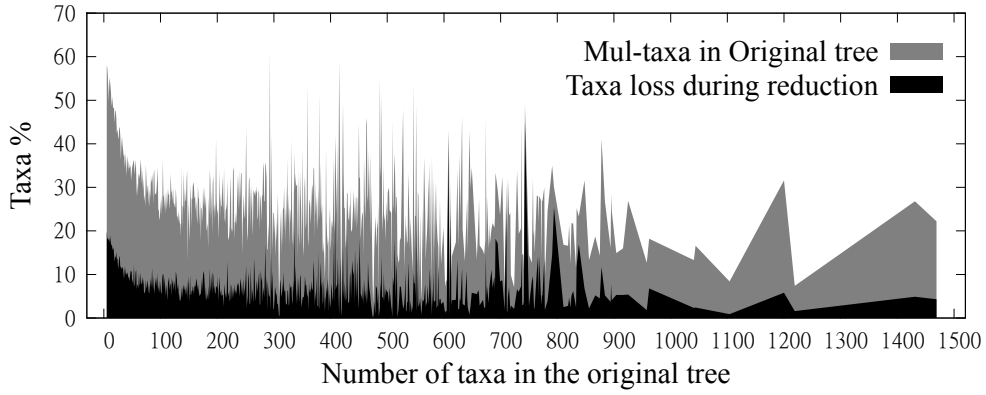
We implemented our MUL-tree reduction algorithm, as well as a second step that restricts the MRF to the set of labels that appear only once, which yields a singly-labeled tree. We tested

our two-step program on a set of 110,842 MUL-trees obtained from the PhyLoTA database (Sanderson et al. (2008)) (http://phylota.net/; GenBank eukaryotic nucleotide sequences, release 184, June 2011), which included a broad range of label-set sizes, from 4 to 1500 taxa.

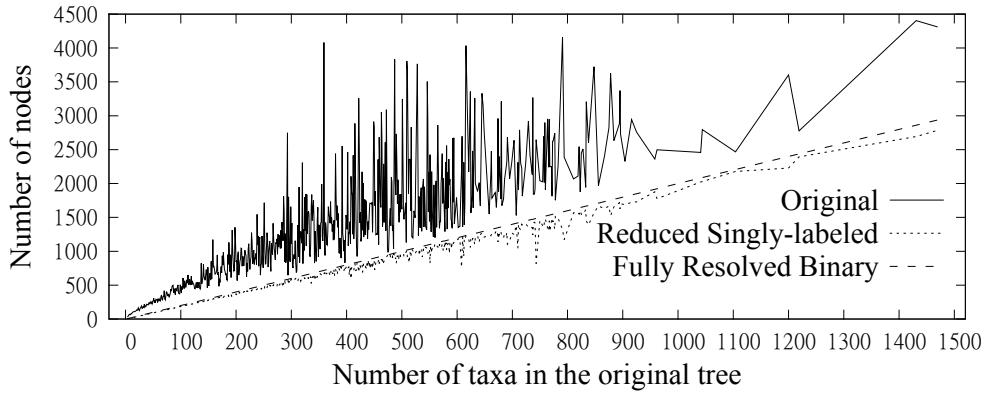
There were 8,741 trees (7.8%) with essentially no information content; these lost all resolution either when reduced to their MRFs, or in the second step. The remaining trees fell into two categories. Trees in set A had a singly-labeled MRF; 65,709 trees (59.3%) were of this kind. Trees in set B were reduced to singly-labeled trees in the second step; 36,392 trees (32.8%) were of this kind. Reducing a tree to its MRF (step 1), led to an average taxon loss of 0.83% of the taxa in the input MUL-tree. The total taxon loss after the second step (reducing the MRFs in set B to singly-labeled trees), averaged 12.81%. This taxon loss is not trivial, but it is far less than the 41.27% average loss from the alternative, nave, approach in which all MUL-taxa (taxa that label more than one leaf) are removed at the outset. Note that, by the definition of MRFs, taxa removed in the first step do not contribute to the information content, since all non-conflicting quartets are preserved. On the other hand, taxa removed in the second step do alter the information content, because each such taxon participated in some non-conflicting quartet. Information content, in this case, will be lost but new information is never introduced, so the algorithm can be considered conservative.

Taxon loss is sensitive to the number of total taxa and, especially, MUL-taxa, as demonstrated in Fig. 5.14a. The gray function shows the percentage of MUL-taxa in the original input trees, which is the taxon loss if we had restricted the input MUL-trees to the set of singly-labeled leaves. The black function shows the percentage of MUL-taxa lost after steps 1 and 2 of our reduction procedure.

In addition to the issue of taxon loss, we investigated the effect of our reduction on edge loss, i.e., the level of resolution within the resulting singly-labeled tree. Input MUL-trees were binary and therefore had more nodes than twice the number of taxa (Fig. 5.14b, solid line), whereas a binary tree on singly labeled taxa would have approximately as many nodes as twice the number of taxa (Fig. 5.14b, dashed line). We found that, although there was some edge loss, the number of nodes in the reduced singly-labeled trees (Fig. 5.14b, dotted line) corresponded well to the total possible, indicating low levels of edge loss. Note that each point



(a)



(b)

Figure 5.14: Experimental results

on the dotted or solid lines represents an average over all trees with the same number of taxa.

We have integrated our reduction algorithm into STBase (available at http://searchtree.org/), a phylogenetic tree search engine that takes a user-provided list of species names and finds matches with a precomputed collection of phylogenetic trees, more than half of which are MUL-trees, assembled from GenBank sequence data. The trees returned are ranked by a tree quality criterion that takes into account overlap with the query set, support values for the branches, and degree of resolution. We have added functionality to provide reduced singly-labeled trees as well as the MUL-trees based on the full leaf set and the label sets from the reduced singly-labeled trees are used in downstream supermatrix construction.

5.7 Conclusions

We introduced an efficient algorithm to reduce a multi-labeled MUL-tree to a maximally reduced form with the same information content, defined as the set of non-conflicting quartets it resolves. We showed that the information content of a MUL-tree uniquely identifies the MUL-tree's maximally reduced form. This has potential application in comparing MUL-trees by significantly reducing the number of comparisons as well as in extracting species-level information efficiently and conservatively from large sets of trees, irrespective of the underlying cause of multiple labels. Our algorithm can easily be adapted to work for rooted trees.

Further work investigating the relationship of the MRF to the original tree under various biological circumstances is also underway. We might expect, for example, that well-sampled nuclear gene families reduce to very small MRF trees, and that annotation errors in chloroplast gene sequences (in which we expect little gene duplication), result in relatively large MRF trees. Comparing the MRF to the original MUL-tree may well provide a method for efficiently assessing and segregating data sets with respect to the causes of multiple labels.

It would be interesting to compare our results with some of the other approaches for reducing MUL-trees to singly-labeled trees (e.g., Scornavacca et al. (2011)) or, indeed, to evaluate if our method can benefit from being used in conjunction with such approaches.

Author's Contributions

MMM and DFB conceived the problem. AD, DFB and MMM designed the experiments and drafted the manuscript. AD designed and implemented the algorithms, and implemented the experiments. DFB coordinated the project. All authors read and approved the final manuscript.

Acknowledgments

This work was supported in part by National Science Foundation grant DEB-0829674. We thank Mike Sanderson for helping to motivate this work, for many discussions about the prob-

lem formulation, and for our ongoing collaboration in the STBase project. Sylvain Guillemot listened to numerous early versions of our proofs and offered many insightful comments.

CHAPTER 6. IDENTIFYING ROGUE TAXA THROUGH REDUCED CONSENSUS: NP-HARDNESS AND EXACT ALGORITHMS

Akshay Deepak, Jianrong Dong and David Fernndez-Baca

A condensed version of this paper appeared in the Proceedings of the 8th *International Symposium on Bioinformatics Research and Applications*, 2012

Abstract

Background: A rogue taxon in a collection of phylogenetic trees is one whose position varies drastically from tree to tree. The presence of such taxa can greatly reduce the resolution of the consensus tree (e.g., the majority-rule or strict consensus) for a collection. The reduced consensus approach aims to identify and eliminate rogue taxa to produce more informative consensus trees. Given a collection of phylogenetic trees over the same leaf set, the goal is to find a set of taxa whose removal maximizes the number of internal edges in the consensus tree of the collection. Quite a few heuristic approaches have been proposed to solve the above problem. However, its complexity characterization and exact solutions remain open problems.

Results: We show that the reduced consensus problem is NP-hard for strict and majority-rule consensus. We give a polynomial-time algorithm for reduced strict consensus when the maximum degree of the strict consensus of the original trees is bounded. We describe exact integer linear programming formulations for computing reduced strict, majority and loose consensus trees. In experimental tests, our exact solutions improved over heuristic methods on several problem instances.

Conclusions: Our results are the first one to prove that identification of rogue taxon through reduced consensus approach is an NP-hard problem. Our exact solutions — both constrained polynomial-time algorithm and integer linear programming formulations — are also the first of their kind. Results show that they can be a useful benchmark for evaluating heuristic approaches and, identify interesting characteristics and limitations of the various consensus methods.

6.1 Background

Consensus methods such as majority rule and strict consensus trees (Margush and McMorris (1981b)) are often used to summarize in a single tree, a *consensus tree*, the common information in a collection of trees over a common leaf set. Such collections are routinely produced by phylogenetic analyses by parsimony, maximum likelihood or Bayesian methods. Consensus trees can be greatly affected by *rogue taxa* (sometimes called *wandering taxa*); that is taxa whose positions can vary dramatically without having a strong effect on a tree's overall score. The presence of just a few such taxa can lead to poorly-resolved consensus trees, even when there is substantial agreement relative to the remaining taxa (Redelings (2009); Wilkinson (1994); Thomson and Shaffer (2010); Sullivan and Swofford (1997); Nadler et al. (2007)). Figure 6.1 illustrates this for the case of strict consensus.

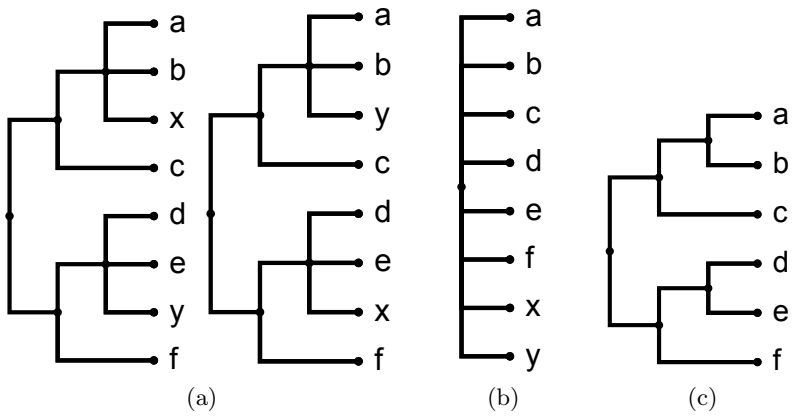


Figure 6.1: The effect of rogue taxon on consensus trees. (a) Two input trees and (b) their strict consensus — a star-like tree. (c) The strict consensus of the trees, after removing the rogue taxon x and y from the input trees.

Here we consider the problem of finding a set of leaves (i.e., possible rogue taxa) to be removed so as to maximize the number of internal edges in the consensus tree on the reduced leaf set. Our main results are the following[1].

- Proofs that the underlying decision problem is NP-complete for three trees in the strict consensus case and for four trees in the case of majority rule trees.
- A polynomial-time algorithm for reduced strict consensus when the maximum degree of the strict consensus tree of the original collection of trees is fixed.
- An integer linear programming (ILP) formulation for obtaining exact solutions for reduced strict, majority and loose consensus.
- An experimental comparison with the heuristic proposed in Pattengale et al. (2011), showing that in several cases our reduced consensus formulations allow us to uncover increased common phylogenetic information without discarding many more leaves.

6.1.0.1 Relationship to previous work

There is a sizable literature on identifying rogue taxa (see, e.g., Wilkinson (1994); Thomson and Shaffer (2010); Pattengale et al. (2011); Cranston and Rannala (2007); Wilkinson (1996, 1995)). Agreement and frequent subtree approaches — where one seeks induced subtree common to all or some fraction of the input trees — have been suggested by some. For example, Cranston and Rannala use it to summarize the posterior distribution of a collection of trees resulting from Bayesian analysis (Cranston and Rannala (2007)). Intuitively, since the placement of rogue taxa varies widely from tree to tree, these taxa will be excluded from the agreement subtrees. There are several papers on computing agreement subtrees (e.g., Finden and Gordon (1985); Amir and Keselman (1994); Farach et al. (1995c); Lee et al. (2005); Swenson et al. (2011)) and frequent subtrees (e.g., Chi et al. (2004a); Xiao and Yao (2003); Zaki (2005); Zhang and Wang (2008)); however, agreement-based approaches can tend to be more conservative than consensus trees. Further, the underlying optimization problems are NP-hard (Amir and Keselman (1994); Pattengale et al. (2011)). Wilkinson (Wilkinson (1994, 1996,

[1]Work on ILP formulations and the following experimental comparison is contributed by Jianrong Dong

1995)) appears to have been the first to propose the use of reduced majority rule and strict consensus as a means to identify rogue taxa. While the underlying principle behind all proposed reduced consensus methods is to gain internal edges at the expense of dropping leaves (Redelings (2009)), the precise cost measure can vary. The criterion we use in this paper is to maximize the resolution of the consensus tree, without taking into account how many leaves are eliminated. In contrast, Pattengale et al.'s objective function is a weighted sum of the total number of leaves retained and the number of clusters (actually, bipartitions) obtained (Pattengale et al. (2011)). Our NP-completeness proof shows that this problem is hard for strict and majority-rule consensus, when the weight assigned to the leaves is zero. To our knowledge, the complexity of the problem in the general case has not been determined. We conjecture that this case is also hard. Indeed, Pattengale et al. only offer a heuristic for it.

The use of integer linear programming in phylogenetics appears to be relatively new (Gusfield et al. (2007); Sridhar et al. (2008)). The formulations presented here are related to the author's previous work on constructing majority-rule supertrees and conservative supertrees (Dong et al. (2010); Dong and Fernández-Baca (2011)). Nevertheless, there are some important differences from that work. While our ILP formulations can only be used for relatively small data sets, they offer a valuable benchmark against which to compare heuristics. They have also allowed us to identify certain interesting characteristics and limitations of the various consensus methods. In particular, our experiments suggest that assigning equal weight to the number of clusters and the number of taxa in the consensus tree might be too conservative an approach.

6.2 Methods

6.2.1 Preliminaries

A *phylogenetic tree* is an unordered rooted or unrooted tree whose leaves are in a bijection with a set of *labels* or *taxa*. We sometimes refer to a phylogenetic tree simply as a *phylogeny* or as a *tree*. A node is *internal* if it is not a leaf. An edge is *internal* if its two endpoints are internal. If a phylogeny T is rooted, we require that each internal node have at least two

children; if T is unrooted, every internal node is required to have degree at least three. Here, we limit our discussion to collections of trees over the same set of taxa. Without loss of generality, we can view such trees as rooted, since, when the trees are unrooted, we can arbitrarily pick any taxon as an out-group, which is equivalent to fixing a common root (Semple and Steel (2003a); Amenta et al. (2003)).

Let T be a phylogenetic tree. We write V_T, E_T, and $\mathcal{L}_T$ to denote the set of vertices, the set of edges, and the set of leaf-labels of T respectively. For convenience, we refer to the set of leaf nodes by their labels in $\mathcal{L}_T$. For an internal node u, let $\mathrm{Ch}(u)$ denote its set of children and let $d(u) = |\mathrm{Ch}(u)|$. Two leaves form a *cherry* if they have a common parent. Let $\varepsilon(T)$ denote the number of internal edges in T.

To *contract* an internal edge is to delete the edge and identify its endpoints. To *suppress* a degree-two node is to delete the node and identify its neighbors. The *restriction* of T to set $X \subseteq \mathcal{L}_T$, denoted by $T|_X$, is the tree on leaf set X obtained from the minimal subtree of T spanning X by suppressing all degree-two nodes except the root. The root of $T|_X$ is the node that was originally closest to the root of T.

The *cluster* induced by a node u in tree T, denoted $\mathrm{Clus}(u)$, is the set of all the leaf descendants of u. A cluster is *trivial* if it is induced by a leaf or the root and is *non-trivial* otherwise. Let A be a subset of $\mathcal{L}_T$. Tree T *contains* cluster A if there is a node u of T such that $A = \mathrm{Clus}(u)$. We say that $A \subseteq \mathcal{L}_T$ is *compatible with* T if there exists a tree T' such that T' contains all the clusters of T, as well as cluster A.

Throughout the rest of the paper, $P = (T_1, \ldots, T_m)$ is a tuple of trees[2] over a common leaf set, which is denoted by $\mathcal{L}_P$. Let X be a subset of $\mathcal{L}_P$. Then, the *restriction* of P to X, denoted $P|_X$, is the tuple of trees $(T_1|_X, \ldots, T_m|_X)$.

6.2.1.1 Consensus methods

A *phylogenetic consensus method* (or simply a *consensus method*) is a function C that maps a tuple of trees P to a tree $C(P)$ with leaf set $\mathcal{L}_P$. A variety of consensus methods have been

[2]We refer to a *tuple* of trees, rather than a set or collection, because, for the case of majority-rule consensus, it is necessary to allow repetitions.

defined (Bryant (2003b)). Here we focus on the following three.

- The *majority-rule tree* of P, denoted $\text{Maj}(P)$ is the tree containing precisely those clusters that occur in more than half of the trees in P.
- The *strict consensus tree* of P, denoted $\text{Str}(P)$ is the tree containing precisely those clusters that occur in every tree in P.
- The *loose consensus tree* of P, denoted $\text{Loose}(P)$, contains exactly those clusters that appear in some tree in P and that are compatible with every tree in P.

6.2.1.2 Reduced consensus

Let C denote some consensus method. Given a collection of trees P, the *reduced C consensus problem* is to find a set $X \subseteq \mathcal{L}_P$ that maximizes $\varepsilon(C(P|_X))$. For example, the reduced majority-rule consensus problem aims at restricting the input trees to a set X such that $\text{Maj}(P|_X)$ — the majority-rule tree on the reduced leafset X — has the maximum number of internal edges among all such possible Xs. Observe that $\varepsilon(C(P|_X)) \leq |\mathcal{L}_P| - 2$, regardless of the consensus method C.

We also consider a *decision version* of the reduced consensus problem. Given a tuple of trees P, a consensus method C, and an integer k between 1 and $|\mathcal{L}_P| - 2$, the question is whether there exists a set $X \subseteq \mathcal{L}$ such that $\varepsilon(C(P|_X)) = k$.

6.2.2 The Reduced Consensus Problem is NP-complete

Theorem 31. *The reduced strict consensus problem is NP-complete even for three trees.*

Proof. Clearly the problem is in NP. We use reduction from the *3-dimensional matching problem* (3DM), which is known to be NP-complete (Karp (1972)). The input to 3DM consists of finite disjoint sets A, B and C, where $|A| = |B| = |C| = k$, and a set $M \subseteq A \times B \times C$. The question is whether there exists a set $M' \subseteq M$ of size k such that for any two distinct triplets (a_i, b_i, c_i) and (a_j, b_j, c_j) in M', $a_i \neq a_j$, $b_i \neq b_j$ and $c_i \neq c_j$. Such an M', if it exists, is called a *witness* for the 3DM instance.

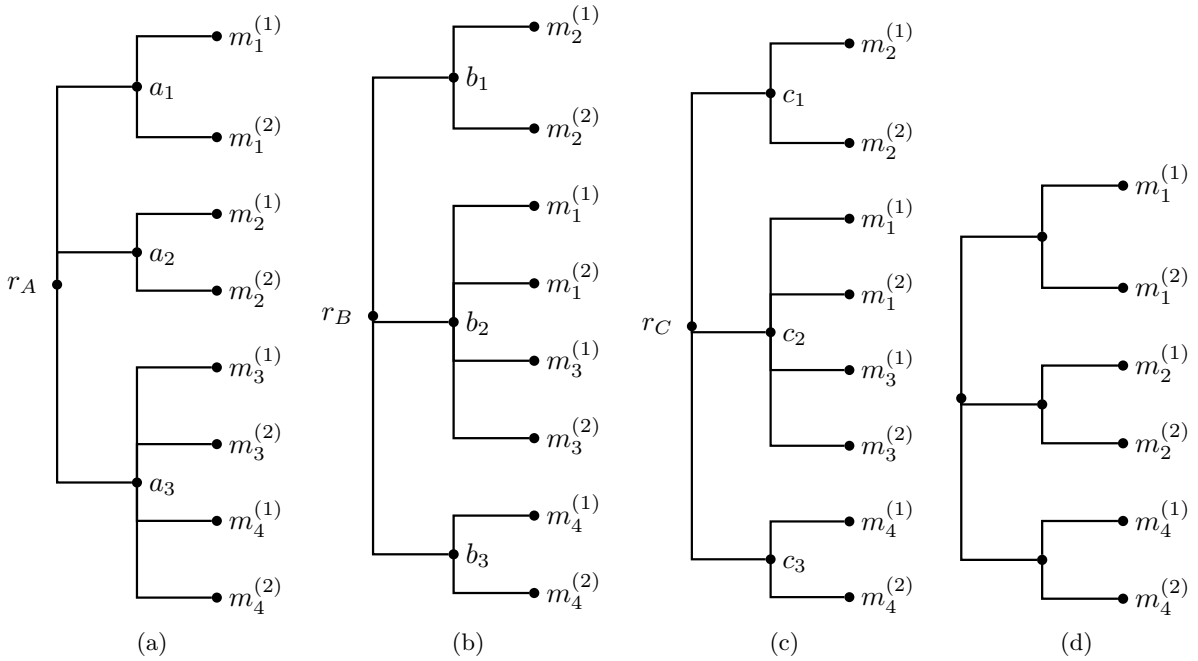


Figure 6.2: Reduction from 3DM problem. Consider an instance of 3DM with $A = \{a_1, a_2, a_3\}$, $B = \{b_1, b_2, b_3\}$, $C = \{c_1, c_2, c_3\}$ and $M = \{m_1 : (a_1, b_2, c_2),\ m_2 : (a_2, b_1, c_1),\ m_3 : (a_3, b_2, c_2),\ m_4 : (a_3, b_3, c_3)\}$. The solution consists of triplets $\{m_1, m_2, m_4\}$. (a) ,(b) and (c) show T_A, T_B and T_C respectively. (d) shows the strict consensus tree on the reduced leafset corresponding to the solution of the given instance.

Given an instance (A, B, C, M) of 3DM, construct a tuple of trees (T_A, T_B, T_C) as follows. Assume without loss of generality that every element of A, B, C is present in at least one triplet in M else a witness is not possible. For each $m \in M$, create two taxa $m^{(1)}$ and $m^{(2)}$, and let $\mathcal{L}_P = \bigcup_{m \in M} \{m^{(1)}, m^{(2)}\}$. Let T_A initially be an empty tree with only r_A as its root node. For each $a_i \in A$, add a child a_i to r_A. Now T_A is a star-like tree with k leaves. For each triplet $m = (a_i, b_i, c_i)$ in M, attach leaves $m^{(1)}, m^{(2)}$ at node a_i in T_A. Similarly, construct trees T_B and T_C corresponding to the input sets Y and Z. Clearly, $\varepsilon(T_A) = \varepsilon(T_B) = \varepsilon(T_C) = k$. This reduction is clearly polynomial. See Fig. 6.2 for an example.

We claim that instance (A, B, C, M) has a witness if and only if there exists an $X \subseteq \mathcal{L}_P$ for $P = (T_A, T_B, T_C)$ such that $\varepsilon(\mathrm{Str}(P|_X)) = k$.

For the forward direction, let $M' \subseteq M$ be a witness for (A, B, C, M). Let $X = \bigcup_{m \in M'} \{m^1, m^{(2)}\}$ be the reduced leaf set. Let $S = \mathrm{Str}(P|_X)$. Since for every two distinct triplets $m_i = (a_i, b_i, c_i)$

and $m_j = (a_j, b_j, c_j)$ in M', $a_i \neq a_j$, $b_i \neq b_j$ and $c_i \neq c_j$, the corresponding cherries $(m_i^{(1)}, m_i^{(2)})$ and $(m_j^{(1)}, m_j^{(2)})$ are not attached at the same node in any of the trees in (T_A, T_B, T_C). Thus, each such cherry corresponding to an $m \in M'$ is a distinct two-element cluster in $\text{Str}(P|_X)$. Therefore, there is an internal edge corresponding to each such size-two cluster in $\text{Str}(P|_X)$. Further $|M'| = k$ implies $\varepsilon(\text{Str}(P|_X)) = k$.

Now we prove the reverse direction. Suppose there exists a subset $X \subseteq \mathcal{L}_P$ such that $\varepsilon(\text{Str}(P|_X)) = k$. Note that $\text{Str}(P|_X)$ cannot have a cherry with leaves corresponding to two distinct triplets m_i and m_j in M as this will lead to $m_i = m_j$, a contradiction. Thus, the endpoints of each internal edge are the root and the parent of a cherry, and each such cherry corresponds to a unique element in M. Let $M' \subseteq M$ be the set of triplets such that for each $m_i = (a_i, b_i, c_i)$ in M', the corresponding cherry with leaves $m_i^{(1)}, m_i^{(2)}$ is a cluster in $\text{Str}(P|_X)$. Clearly $|M'| = k$. Since $\varepsilon(\text{Str}(P|_X)) = \varepsilon(T_A) = k$, each internal edge — whose endpoints are the root and the parent of the cherry consisting of $m_i^{(1)}, m_i^{(2)}$ — in $\text{Str}(P|_X)$ corresponds to a unique internal edge attached to a_i in T_A. Thus, triplet m_i covers a unique a_i in A. The same applies with respect to B and C. Thus, for every two distinct triplets $m_i = (a_i, b_i, c_i)$ and $m_j = (a_j, b_j, c_j)$ in M', $a_i \neq a_j$, $b_i \neq b_j$ and $c_i \neq c_j$ holds. Thus M' is a witness for the given 3DM instance. □

Theorem 32. *The reduced majority rule consensus problem is NP-complete, even for four trees.*

Proof. Modify the construction used in the proof of Theorem 31 by adding a dummy star-like tree T_D with $\mathcal{L}_{T_D} = \mathcal{L}_{T_A} = \mathcal{L}_{T_B} = \mathcal{L}_{T_C}$ to the tuple (T_A, T_B, T_C). The reduced majority rule tree of this new collection of trees is the strict consensus of the original tuple, because the dummy tree contains no non-trivial clusters, so the clusters from the first three trees constitute a majority. □

6.2.3 Fixed Parameter Tractability

We now prove that the reduced consensus problem relative to strict consensus is fixed parameter tractable in the maximum out-degree of $\text{Str}(P)$. Formally, we claim the following.

Theorem 33. *Let $P = (T_1, \ldots, T_m)$ be a tuple of phylogenetic trees over the same leaf set and let d denote the maximum out-degree of a node in* $\mathrm{Str}(P)$*. Then, the reduced strict consensus problem for P can be solved in $O(m \cdot |\mathcal{L}_P| \cdot 2^d)$ time.*

Note that the above result is practical only when the strict consensus of P does not contain high-degree nodes. Nevertheless, even if this is not the case, the algorithm can still be used in combination with heuristic approaches such as (Pattengale et al. (2011); Cranston and Rannala (2007)). Through these methods, one can bring the maximum out-degree to within reasonable bounds, after which one can invoke Theorem 33.

Suppose X is a subset of $\mathcal{L}_P$. Let $S = \mathrm{Str}(P)|_X$ and $R = \mathrm{Str}(P|_X)$. Observe that R is a *refinement* of S. That is, S can be obtained from R by contracting zero or more internal edges of R. For example, see Fig. 6.3. Here S (Fig. 6.3d) can be obtained from R (Fig. 6.3c) by contracting edge (r, u). Thus, for every $u \in V_S$, there exists a mapping $\vartheta_u : 2^{\mathcal{L}_S} \to 2^{E_R}$ such that $\vartheta_u(X)$ is the set of all edges in R that are contracted into u. For example, for the root node r in S (Fig. 6.3d), edge (r, u) in R (Fig. 6.3c) is contracted into r. Thus, for $X = \{a, b, c\}$, $\vartheta_r(X) = \{(r, u)\}$. If $\varepsilon(R) > \varepsilon(S)$, there must exist at least one internal node u in S such that $|\vartheta_u(X)| > 0$. Node u is said to be *refined* in R. For example, in Fig. 6.3, the root node r of S is refined in R.

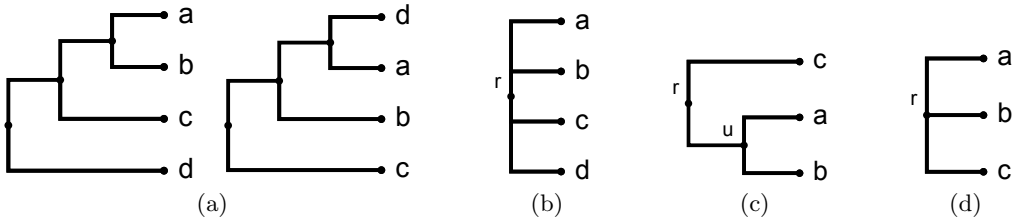


Figure 6.3: (a) A pair P of trees on leaf set $\mathcal{L}_P = \{a, b, c, d\}$. (b)$\mathrm{Str}(P)$. (c) $R = \mathrm{Str}(P|_X)$ for $X = \{a, b, c\}$. (d) $S = \mathrm{Str}(P)|_X$.

The proof of Theorem 33 relies on the following result, which is proved at the end of this section.

Lemma 34. *Let u be a node in* $\mathrm{Str}(P)$*. Let X be a maximal set of taxa such that u is refined in* $\mathrm{Str}(P|_X)$ *and let $Y = \mathcal{L}_P - X$. Then the following hold.*

1. *For every* $a \in \mathrm{Ch}(u)$ *if* $\mathrm{Clus}(a) \cap Y \neq \emptyset$ *then* $\mathrm{Clus}(a) \subseteq Y$

2. $Y \subseteq \bigcup_{a \in \mathrm{Ch}(u)} \mathrm{Clus}(a)$

Lemma 34 shows that (1) the refinement of node u must occur at the cost of removing one or more clusters corresponding to $\mathrm{Ch}(u)$ and (2) only clusters corresponding to $\mathrm{Ch}(u)$ need to be removed. Therefore, to refine any node u in $\mathrm{Str}(P)$ we only need to consider the clusters corresponding to $\mathrm{Ch}(u)$ in their entirety; i.e., either a cluster corresponding to $\mathrm{Ch}(u)$ is completely retained in $\mathrm{Str}(P|_X)$ or is completely absent from $\mathrm{Str}(P|_X)$.

For a given node u in $\mathrm{Str}(P)$, define $\mathrm{opt}(u)$ as follows

$$\mathrm{opt}(u) = \max_{X \subseteq \mathrm{Clus}(u)} \{\varepsilon\left(\mathrm{Str}(P|_X)\right)\}. \tag{6.1}$$

When u is a leaf, $\mathrm{opt}(u) = 0$. When u is the root node, $\mathrm{opt}(u)$ gives the value of the optimum solution to the reduced strict consensus problem. Note that when considering the optimum solution in (6.1), an $X \subset \mathrm{Clus}(u)$ needs to be considered only if it refines u. By Lemma 34, such an X must contain one or more clusters corresponding to $\mathrm{Ch}(u)$ in entirety. Thus, $\mathrm{opt}(u)$ can be expressed as:

$$\mathrm{opt}(u) = \max_{U \subseteq \mathrm{Ch}(u)} \left\{ \sum_{v \in U} \mathrm{opt}(v) + \iota(U) + |\vartheta_u\left(\mathrm{Clus}(U)\right)| \right\}, \tag{6.2}$$

where $\iota(U)$ denotes the number of nodes in U that remain as internal in $\mathrm{Str}(P|_X)$ and $\mathrm{Clus}(U) = \bigcup_{v \in U} \mathrm{Clus}(v)$. Here, $\sum_{v \in U} \mathrm{opt}(v)$ corresponds to the contribution of internal edges by the subtrees rooted at each of the children of u in U, $\iota(U)$ is the number of internal edges contributed by the internal nodes in U and $|\vartheta_u\left(\mathrm{Clus}(U)\right)|$ refers to the internal edges gained due to the refinement of u. For example, consider Fig. 6.3. Here $\mathrm{Str}(P|_X)$ (Figure 3-c) is the optimum solution for $X = \{a, b, c\}$. This happens when the root node r is refined in $\mathrm{Str}(P|_X)$. Thus, $\mathrm{opt}(r)$ occurs at $U = \{a, b, c\}$ as per (6.2). Here, $\sum_{v \in U} \mathrm{opt}(v) = 0$ as every node in U is a leaf. For the same reason $\iota(U) = 0$. $\vartheta_r\left(\mathrm{Clus}(U)\right) = \vartheta_r(X) = \{(u, v)\}$. Thus, $\mathrm{opt}(r) = 1$ i.e. the reduced leafset $X = \{a, b, c\}$ gives the maximum number of internal edges as 1.

Note that while (6.1) requires iteration over all subsets of $\mathrm{Clus}(u)$, (6.2) only requires iteration over subsets of $\mathrm{Ch}(u)$. For a given $U \subseteq \mathrm{Ch}(u)$, $\vartheta_u\left(\mathrm{Clus}(U)\right)$ can be computed in

$O(m|U|)$ time. This can be done in the following way. For every node u in $\mathrm{Str}(P)$, we maintain an array of size m whose i-th entry points to the node v in the i-th tree in P such that $\mathrm{Clus}(u) = \mathrm{Clus}(v)$. Thus, for a given $U \subseteq \mathrm{Ch}(u)$ and a tree T in P, we can find the subtree induced by $U + u$ in T in $O(|U|)$ time[3]. Observe that here U corresponds to the leaves of this subtree. For all the trees in P, this takes $O(m|U|)$ time. Computing the strict consensus of these subtrees takes $O(m|U|)$ time (Amenta et al. (2003)). The number of internal edges in this strict consensus tree corresponds to $|\vartheta_u(\mathrm{Clus}(U))|$.

Assuming $\mathrm{opt}(v)$ is known for all $v \in \mathrm{Ch}(u)$, expression (6.2) can be evaluated in $O(m|U|)$ time. Thus, $\mathrm{opt}(u)$ can be computed in $O(m \cdot d(u) \cdot 2^{d(u)})$ time. Thus, in an inorder depth first traversal of $\mathrm{Str}(P)$, opt values for all nodes can be calculated in $O(m \cdot |\mathcal{L}_P| \cdot 2^d)$. Theorem 33 follows.

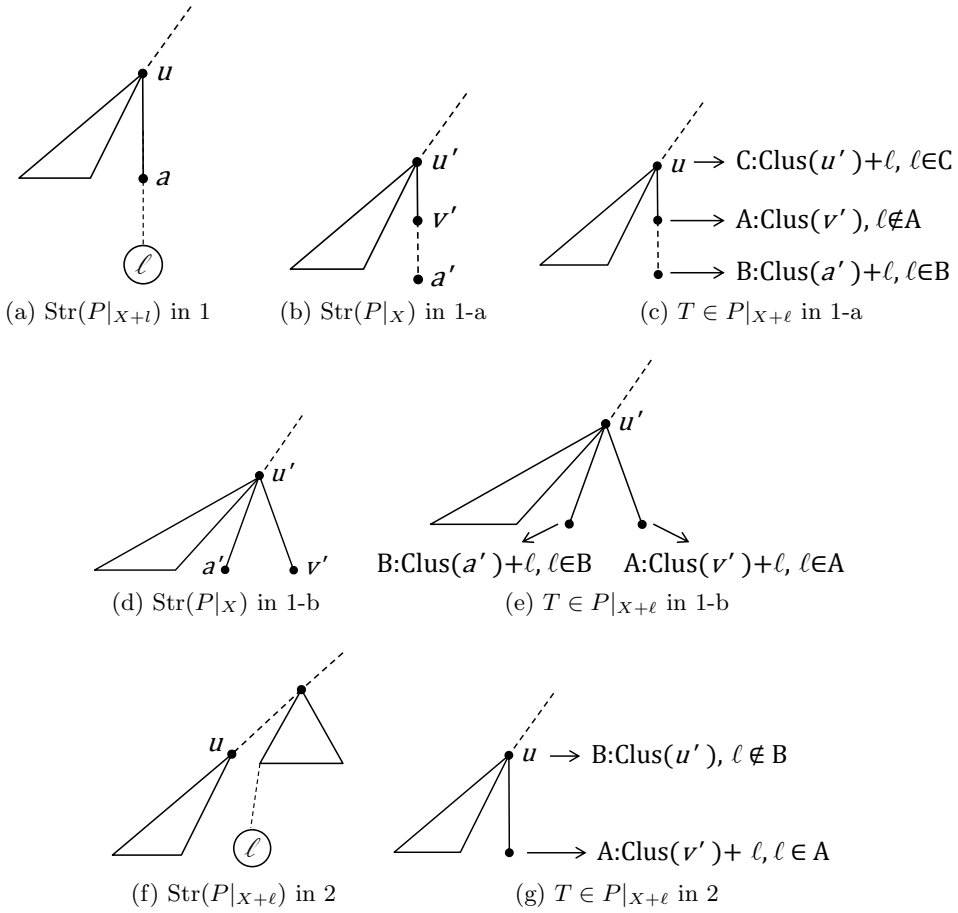
One appealing feature of the algorithm just described is that it can easily be implemented so that, among all subsets X that maximize $\varepsilon(\mathrm{Str}(P|_X))$, it returns a maximal one. That is, no proper superset of X yields an optimal solution or, equivalently, the set of dropped taxa is minimal.

Proof of Lemma 34. Since $R = \mathrm{Str}(P|_X)$ is a refinement of $S = \mathrm{Str}(P)|_X$, $V_S \subseteq V_R$. To differentiate, for every $u \in V_S$, let u' be the corresponding node in V_R. Let ℓ be any leaf from $Y = \mathcal{L}_P - X$. Since X is maximal, u is not refined in $\mathrm{Str}(P|_{X+\ell})$. Let (u', v') be an edge in $\vartheta_u(X)$, where u' is the parent of v' i.e. this edge does not exist in $\mathrm{Str}(P|_{X+\ell})$ but exists in the refined strict consensus tree $\mathrm{Str}(P|_X)$.

1. We use proof by contradiction. Suppose there exists an $a \in \mathrm{Ch}(u)$ such that $\mathrm{Clus}(a) \cap Y \neq \emptyset$ but $\mathrm{Clus}(a) \nsubseteq Y$. Thus, the corresponding node a' exists in $\mathrm{Str}(P|_X)$. Without loss of generality, assume $\ell \in \mathrm{Clus}(a) \cap Y$. Figure 6.4a shows $\mathrm{Str}(P|_{X+\ell})$ for this case. In the refined strict consensus tree $\mathrm{Str}(P|_X)$, both a' and v' are children of u'. Based on the relative position of a' with respect to v', there are two possible cases.

2. a' is a descendant of v'. Figure 6.4b shows $\mathrm{Str}(P|_X)$ for this case. Since the edge (u', v') does not exists in $\mathrm{Str}(P|_{X+\ell})$ but exists in $\mathrm{Str}(P|_X)$, cluster $\mathrm{Clus}(v') + \ell$ does not exist

[3]Given a set A and an element x, we write $A + x$ to denote $A \cup \{x\}$.



(a) Str($P|_{X+l}$) in 1 (b) Str($P|_X$) in 1-a (c) $T \in P|_{X+\ell}$ in 1-a

(d) Str($P|_X$) in 1-b (e) $T \in P|_{X+\ell}$ in 1-b

(f) Str($P|_{X+\ell}$) in 2 (g) $T \in P|_{X+\ell}$ in 2

Figure 6.4: Supportive illustrations for Proof of Lemma 34

in all the trees in $P|_{X+\ell}$ but exists (as Clus(v')) in all the trees in $P|_X$. Thus, at least one tree $T \in P|_{X+\ell}$ (see Fig. 6.4c), must contain cluster $A = \text{Clus}(v')$. However, T also contains clusters $B = \text{Clus}(a') + \ell$ and $C = \text{Clus}(u') + \ell$. Given that v' is a child of u' and a' is a descendant of v', the existence of clusters A, B, and C in a tree is not possible. Hence this case leads to a contradiction.

(a) a' is not a descendant of v'. Figure 6.4d shows Str($P|_X$) for this case. Again, since the edge (u', v') does not exist in Str($P|_{X+\ell}$) but exists in Str($P|_X$), cluster Clus(v') does not exist in all the trees in $P|_{X+\ell}$ but exists in all the trees $P|_X$. Thus, at least one tree $T \in P|_{X+\ell}$ (see Fig. 6.4e), must contain cluster $A = \text{Clus}(v') + \ell$. T also contains cluster $B = \text{Clus}(a') + \ell$. Given that neither of v', a' is a descendant of the

other, the existence of clusters A and B in a tree is not possible. Hence this case leads to a contradiction.

Since each case leads to a contradiction, we have $\text{Clus}(a) \subseteq Y$.

3. Suppose $Y \not\subseteq \bigcup_{a \in \text{Ch}(u)} \text{Clus}(a)$. We can assume without loss of generality that $\ell \notin \bigcup_{a \in \text{Ch}(u)} \text{Clus}(a)$. Thus, $\ell \notin \text{Clus}(u')$. Figure 6.4f shows $\text{Str}(P|_{X+\ell})$ for this case. Again, at least one tree $T \in P|_{X+\ell}$ (see Fig. 6.4g), must contain cluster $A = \text{Clus}(v') + \ell$. This tree also contains cluster $B = \text{Clus}(u')$. Given that v' is the child of u' and $\ell \notin \text{Clus}(u')$, the existence of clusters A and B in a tree is not possible, so we have a contradiction. Thus, $Y \subseteq \bigcup_{a \in \text{Ch}(u)} \text{Clus}(a)$.

□

6.2.4 ILP Formulations

We present ILP formulations for computing reduced strict, loose, and majority-rule consensus trees. The formulations share some characteristics with those developed elsewhere for supertrees (Dong et al. (2010); Dong and Fernández-Baca (2011)); however, they differ in important aspects. Unlike the supertree case, there is no need to model the fill-in process, where each input tree is augmented with the taxa that it is missing. Ensuring that this is done legally for supertree computation necessitates many constraints. In contrast, for reduced consensus, reduced trees are completely determined by the selection of rogue taxa. Moreover, clusters that are identical or compatible prior to taxon reduction remain identical or compatible.

We now describe the main constants, variables, and constraints of the ILP formulations. Unless mentioned otherwise, variables are binary. For brevity, we will simply express the constraints as logical expressions involving set operations. These can easily be transformed into linear inequalities — see, e.g., Dong et al. (2010). As before, let $P = (T_1, \ldots, T_m)$ be a tuple of rooted trees over the same leaf set; let $n = |\mathcal{L}_P|$. Note that n includes the special root taxon.

For $j = 1, \ldots, m$, we represent T_j by a $n \times g_j$ matrix $M(T_j)$ whose columns correspond to the nontrivial clusters of T_j that do not appear in T_i, for any $i < j$. Thus we only represent

distinct splits in the profile. Let $g = \sum_{j=1}^{m} g_j$. Suppose column i of $M(T_j)$ corresponds to cluster A in T_j. Then, $M_{xi}(T_j) = 0$ if taxon x is in A, and $M_{xi}(T_j) = 1$ otherwise. Tuple P is represented by the matrix $M(P)$ obtained by concatenating matrices $M(T_1), \ldots, M(T_m)$.

The first tree is special for reduced strict consensus tree because all reduced splits or clusters can be found in the first reduced tree. Thus, the relationship is between a column in the first tree and a column in any tree. This is not true for loose or majority-rule reduced consensus trees, because the reduced split/cluster might come from another tree. Hence, the relationship is between two columns the profile.

For strict and majority-rule reduced consensus trees, we need to know which reduced split is in which tree. Hence, for these two trees, define an $g \times m$ binary matrix $Intree$ for bookkeeping purposes. $Intree(i, j) = 1$ if and only if the i^{th} reduced split is in the j^{th} reduced tree. If the i^{th} original split is in the j^{th} original tree, so are their reduced forms. Hence, $Intree(i, j) = 1$ for those original splits found in original trees. However, even if the i^{th} original split is not in the original j^{th} tree, when reduced, the reduced i^{th} split could still be in the reduced j^{th} tree because

- the reduced i^{th} split might be trivial, or
- it becomes identical with a reduced split (either trivial or nontrivial) in the reduced j^{th} tree

Therefore, if $Intree(i, j) \neq 1$ for original split/original tree, we put a question mark to $Intree(i, j)$. We will use the unknown $Intree(i, j)$ to define U_{ij} variables later.

Note that it is unnecessary to define $Intree(i, j)$ for loose reduced consensus trees, since compatibility is required instead of presence in certain tree.

To represent the subset $X \subseteq \mathcal{L}(P)$ from which we obtain the reduced collection of trees $P|_X$, define $n - 1$ variables $S_1, \ldots, S_{n-1}$, where $S_r = 1$ precisely if the r^{th} taxon is included in X. For every pair i, j of columns of $M(P)$, and each taxon r such that $S_r = 1$, we can encounter one of four patterns: 00, 01, 10, or 11. The pattern 11 always appears since every split has the root filling as 1. The presence or absence of these patterns is indicated by the settings of variables $B_{ij}^{(ab)}$, where $a, b \in \{0, 1\}$, $i = 1, \ldots, g_1$ in the strict reduced consensus tree

case, or $i = 1, \ldots, g$ in the loose and majority-rule reduced consensus tree cases, $j = 1, \ldots, g$, and $i < j$. $B_{ij}^{(ab)} = 1$ precisely when there is a taxon r such that $S_r = 1$, $M_{ri}(P) = a$ and $M_{rj}(P) = b$. We have that $B_{ij}^{(ab)} \Leftrightarrow \bigcup_{q=1}^{p} S_{r_q}$ when ab are 01 and 10 for strict and majority-rule reduced consensus trees, ab are 00, 01 and 10 for loose reduced consensus trees.

Since M is known, $B_{ij}^{(ab)}$s are determined by the related S_rs. We have

$$B_{ij}^{(ab)} \Leftrightarrow \bigcup_{q=1}^{p} S_{r_q} \tag{6.3}$$

that is,

$$\begin{aligned} -S_{r_1} - S_{r_2} - \cdots - S_{r_p} + B_{ij}^{(ab)} &\leq 0 \\ S_{r_1} + S_{r_2} + \cdots + S_{r_p} - n \cdot B_{ij}^{(ab)} &\leq 0 \end{aligned} \tag{6.4}$$

Note that some $B_{ij}^{(ab)} \equiv 0$ if the pattern ab does not exist in all the rows of splits i and j. Thus in implementation we do not define them and their related constraints 6.4.

Define binary variables $\delta_1^0, \ldots, \delta_{g_1}^0$ and $\delta_1^1, \ldots, \delta_{g_1}^1$ to represent the nontrivial clusters in the reduced strict consensus tree. For reduced loose and majority-rule consensus trees, replace g_1 with g. Nontrivial clusters must have at least two 0s and two 1s. Hence, for cluster j, the sum of the S_r's corresponding to $M_{rj}(P) = 0$ must be at least 2, and the sum of S_r corresponding to $M_{rj}(P) = 1$ must be at least 1, since the root contributes the other 1. For $1 \leq j \leq g_1$ (or g), we have $\delta_j^0 \Leftrightarrow \sum_{q=1}^{p} S_{r_q} \geq 2$, or equivalently

$$\begin{aligned} 2 \cdot \delta_j^0 - \sum_{q=1}^{p} S_{r_q} &\leq 0 \\ \sum_{q=1}^{p} S_{r_q} - n \cdot \delta_j^0 &\leq 1 \end{aligned} \tag{6.5}$$

Similarly, we have $\delta_j^1 \Leftrightarrow \sum_{q=1}^{p} S_{r_q} \geq 1$ where $1 \leq r_q \leq n-1$, or equivalently

$$\begin{aligned} \delta_j^1 - \sum_{q=1}^{p} S_{r_q} &\leq 0 \\ \sum_{q=1}^{p} S_{r_q} - n \cdot \delta_j^1 &\leq 0 \end{aligned} \tag{6.6}$$

Note that for reduced strict consensus trees we can use fewer δ^0 and δ^1 variables. This is because tree T_1 (in fact, any input tree) must contain every cluster of the reduced strict

consensus tree. Thus, we only need g_1 cluster variables, and correspondingly fewer constraints. As seen below, similar savings can be achieved for other variables and constraints.

Define binary variables E_{ij}, where $i = 1, \ldots, g_1$ for reduced strict consensus tree, or $i = 1, \ldots, g$ for reduced majority-rule and loose consensus trees, $j = 1, \ldots, g$ and $i < j$, to indicate identity between distinct reduced clusters. Observe that $E_{ij} \Leftrightarrow \neg B_{ij}^{(01)} \wedge \neg B_{ij}^{(10)}$, or

$$\begin{aligned} B_{ij}^{(01)} + B_{ij}^{(10)} + 2 \cdot E_{ij} &\leq 2, \\ B_{ij}^{(01)} + B_{ij}^{(10)} + E_{ij} &\geq 1. \end{aligned} \tag{6.7}$$

For reduced strict consensus trees, we define $g_1 - 1$ binary D variables for reduced strict consensus trees to indicate duplicate clusters. For reduced majority-rule and loose consensus trees, replace g_1 with g. Observe that $D_1 \equiv 0$ and thus is not a variable. For $2 \leq p \leq g_1$ (or g), $D_p \Leftrightarrow \bigcup_{i=1}^{p-1} E_{ip}$, or

$$\begin{aligned} D_p - \sum_{i=1}^{p} E_{ip} &\leq 0 \\ \sum_{i=1}^{p} E_{ip} - p \cdot D_p &\leq 0 \end{aligned} \tag{6.8}$$

For reduced strict consensus trees, based on the unknown elements of $Intree$, we define at most $g_1 \cdot (m-1)$ binary variables U_{ij} where $i = 1, \ldots, g_1$ and $j = 2, \ldots, m$. $U_{ij} = 1$ precisely if the i^{th} reduced cluster is in the j^{th} reduced tree. That is, $U_{ij} \Leftrightarrow \bigcup_{\ell \in M(T_j)} E_{i\ell} \cup \neg\delta_i^0 \cup \neg\delta_i^1$ (if $i > \ell$, $E_{\ell i}$ is replaced by $E_{i\ell}$).

Equivalently we have

$$\begin{aligned} U_{ij} + \delta_i^0 + \delta_i^1 - E_{i\ell_1} - \cdots - E_{i\ell_{g_j}} &\leq 2 \\ E_{i\ell_1} + \cdots + E_{i\ell_{g_j}} - \delta_i^0 - \delta_i^1 - (g_j + 2) \cdot U_{ij} &\leq -2 \end{aligned} \tag{6.9}$$

Note that $U_{i1} \equiv 1$ and thus are constants.

For reduced majority-rule consensus trees, based on the unknown elements of $Intree$, we define at most $g \cdot (m-1)$ binary variables U_{ij} where $i = 1, \ldots, g$ and $j = 1, \ldots, m$. $U_{ij} = 1$ precisely if the i^{th} reduced cluster is in the j^{th} reduced tree. That is, $U_{ij} \Leftrightarrow \bigcup_{\ell \in M(T_j)} E_{i\ell} \cup \neg\delta_i^0 \cup \neg\delta_i^1$ (if $i > \ell$, $E_{\ell i}$ is replaced by $E_{i\ell}$).

Equivalently we have

$$\begin{aligned} U_{ij} + \delta_i^0 + \delta_i^1 - E_{i\ell_1} - \cdots - E_{i\ell_{g_j}} &\leq 2 \\ E_{i\ell_1} + \cdots + E_{i\ell_{g_j}} - \delta_i^0 - \delta_i^1 - (g_j + 2) \cdot U_{ij} &\leq -2 \end{aligned} \tag{6.10}$$

Note that the i^{th} cluster is always in its own tree. $U_{ip} \equiv 1$ and thus are constants if the i^{th} cluster is originally in T_p.

For reduced loose consensus trees, there are no U variables and related constraints. Instead, there are C variables and constraints defined below.

The clusters of the reduced loose tree are in at least one tree and are compatible with every other cluster. Note that, even if input tree clusters i and j are incompatible, they could be compatible when restricted to the selected taxa. Thus, for reduced loose consensus trees, we define variables C_{ij} where $i = 1, \ldots, g$, $j = g_1 + 1, \ldots, g$, and $i < j$ to represent pairwise compatibility among reduced clusters. The compatibility within the first tree clusters is obvious. When $1 \leq i, j \leq g_1$, $C_{ij} \equiv 1$.

Observe that $\neg C_{ij} \Leftrightarrow B_{ij}^{(00)} \wedge B_{ij}^{(01)} \wedge B_{ij}^{(10)} \wedge B_{ij}^{(11)}$. Since $B_{ij}^{(11)} = 1$ for rooted trees, we have

$$\begin{aligned} B_{ij}^{(00)} + B_{ij}^{(01)} + B_{ij}^{(10)} + 4C_{ij} &\geq 3, \\ B_{ij}^{(00)} + B_{ij}^{(01)} + B_{ij}^{(10)} + C_{ij} &\leq 3. \end{aligned} \tag{6.11}$$

where $1 \leq i \leq g$ and $g_1 + 1 \leq j \leq g$. Note that $C_{ij} = C_{ji}$.

If two original clusters, cluster i and cluster j are compatible, their induced clusters on the same set of taxa are compatible. Hence $C_{ij} \equiv 1$. The variable C_{ij} and the constraints 6.11 are not needed. However, if cluster i and cluster j are incompatible, their induced clusters could be incompatible or compatible, depending on what taxa are selected. Hence C_{ij} and the constraints 6.11 cannot be omitted.

Next, we define W-variables, that indicate which reduced input tree clusters appear in the reduced consensus tree. The number of these variables and the constraints that define them depend on the particular consensus method used.

For reduced strict consensus trees, define variables $W_1, \ldots, W_{g_1}$ where $W_i = 1$ precisely if the i^{th} cluster is in every other tree. That is, $W_i \Leftrightarrow \sum_{j=2}^{m} U_{ij} = m - 1$. This is expressed by

the following two constraints.

$$\begin{aligned}(m-1)\cdot W_i - \sum_{j=2}^{m} U_{ij} \leq 0,\\ \sum_{j=2}^{m} U_{ij} - W_i \leq m-2.\end{aligned} \tag{6.12}$$

For reduced majority-rule consensus trees, define g binary variables $W_1, \ldots, W_g$ where $W_i = 1$ precisely if the i^{th} cluster is in more than half of the trees. That is, $W_i \Leftrightarrow \sum_{j=1}^{m} U_{ij} \geq \lceil \frac{m+1}{2} \rceil$. When m is odd, it becomes $W_i \Leftrightarrow \sum_{j=1}^{m} U_{ij} \geq \frac{m+1}{2}$, or

$$\begin{aligned}\sum_{j=1}^{m} U_{ij} - \frac{m-1}{2}\cdot W_i \geq 1,\\ \sum_{j=1}^{m} U_{ij} - \frac{m+1}{2}\cdot W_i \leq \frac{m-1}{2}.\end{aligned} \tag{6.13}$$

When m is even, it becomes $W_i \Leftrightarrow \sum_{j=1}^{m} U_{ij} \geq \frac{m}{2} + 1$, or

$$\begin{aligned}\sum_{j=1}^{m} U_{ij} - \frac{m}{2}\cdot W_i \geq 1,\\ \sum_{j=1}^{m} U_{ij} - \frac{m}{2}\cdot W_i \leq \frac{m}{2}.\end{aligned} \tag{6.14}$$

For reduced loose consensus trees, define g binary variables $W_1, \ldots, W_g$ where $W_i = 1$ precisely if the i^{th} cluster is compatible with every other cluster. That is, $W_i \Leftrightarrow \sum_{j=1, j\neq i}^{g} C_{ij} = g-1$, or

$$\begin{aligned}(g-1)\cdot W_i - \sum_{j=1, j\neq i}^{g} C_{ij} \leq 0,\\ \sum_{j=1, j\neq i}^{g} C_{ij} - W_i \leq g-2.\end{aligned} \tag{6.15}$$

When $1 \leq i, j \leq g_1$, $C_{ij} \equiv 1$. In other ranges where $j < i$, $C_{ij} \equiv C_{ji}$, which was defined earlier and will be used.

For reduced strict consensus trees, define g_1 V variables to represent unique nontrivial clusters in the reduced consensus tree. For $1 \leq p \leq g_1$, we have $V_p \Leftrightarrow W_p \wedge \neg D_p \wedge \delta_p^0 \wedge \delta_p^1$, or

$$\begin{aligned}4\cdot V_p - W_p + D_p - \delta_p^0 - \delta_p^1 \leq 1\\ W_p - D_p + \delta_p^0 + \delta_p^1 - V_p \leq 2\end{aligned} \tag{6.16}$$

For reduced majority-rule and loose consensus trees, replace g_1 with g.

The objective value is the number of unique reduced strict splits (clusters), or the sum of the V_is. For reduced strict consensus trees, define objective variable obj through the equation

$$obj - \sum_{p=1}^{g_1} V_p = 0 \tag{6.17}$$

For reduced majority-rule and loose consensus functions, replace g_1 with g.

The objective functions for the three reduced consensus functions are the same:

$$\text{maximize} \quad obj \tag{6.18}$$

The order of g_1 is $O(n)$. When the clusters within P are highly heterogeneous, g approaches $m \cdot n$. When the clusters within P are highly homogeneous, g approaches n. We choose the highly heterogeneous cases, it can be shown that the number of variables and constraints is $O(m^2n^2)$ for the reduced majority-rule and loose consensus tree ILPs and $O(mn^2)$ for the reduced strict consensus tree ILP. Thus for the same profile, the reduced strict supertree ILP generally solves faster than the other two. Moreover, the former can also handle larger input profiles. Experiments below confirm it.

6.3 Results and discussion

6.3.1 Experiments

Our data sets were derived from those used in a previous study by Pattengale et al. (2011). There are 16 sets of bootstrapped trees constructed from single-gene and multi-gene DNA sequences, with 125-2,554 taxa. Out of these, nine sets were found to be prone to rogue taxon in our earlier analysis (Deepak et al. (2012a)). We refer to these data sets as 1-9, and analyzed them further in our current work. For each of the nine sets of trees, we extracted five sets of trees with 100 trees on 10 taxa, 65 trees on 15 taxa, 50 trees on 20 taxa, 40 trees on 25 taxa and 30 trees on 30 taxa respectively. For this, we randomly selected the required number of trees and restricted them on a random set of taxa of the required size. All trees used in the experiments are available as 'Trees.zip' in the Additional Files section of this manuscript.

We compared our ILP formulation with the rogue taxon solution proposed by Pattengale et al. (2011). The latter tries to uncover new internal edges by merging bipartitions that are sufficiently similar that the merged bipartition becomes frequent enough to be included in the consensus tree on the reduced leafset. Though quite fast, the heuristic cannot guarantee an optimal solution. The default solution in Pattengale et al. (2011) tries to balance the gain in internal edges with the accompanying loss in the number of leaves. However, the authors also give a straightforward modification that attempts to maximize only resolution; i.e., the gain in internal edges. We use this modified approach to compare with our exact solutions. We use the original implementation of the authors — a Python script — for the purpose of comparison.

We ran our ILP formulations for the strict and majority reduced consensus trees on each of the five sets of trees for each each of the 9 data sets. All experiments were run on an Intel Core 2 64 bit quad-core processor (2.83GHz) with 8GB RAM. The ILPs were solved using CPLEX[4]. We used MATLAB to generate the input files for CPLEX and to post-process the output files. Table 6.1, Table 6.2, Table 6.3, Table 6.4 and Table 6.5 show the results for the set of 100 trees on 10 taxa, 50 trees on 20 taxa, 65 trees on 15 taxa, 40 trees on 25 taxa and 30 trees on 30 taxa respectively. A bold faced entry in a table indicates improvement of our ILP approach over the heuristic method. The columns represent the data set number (Dataset), total running time in seconds (Time), original internal edges (OIE), new internal edges after rogue taxon removal (NIE), and total leaf loss (LL) for the exact solution, followed by the heuristic NIE (HNIE) and LL (HLL) results.

6.3.2 Discussion

Based on our experimental results, we can make some preliminary observations. First, the local search heuristics does not seem as effective when the removed rogue taxa account for a high percentage of the total number of taxa. On the other hand, in most cases, the improvement achieved by the exact algorithm over the heuristic method is at most one, showing that the heuristic method is quite accurate.

Second, since our ILP only maximizes NIE, it seems to discard more leaves than the heuristic

[4]CPLEX is a trademark of IBM.

	Strict						Majority-rule					
Data set	Time (sec)	OIE	NIE	HNIE	LL	HLL	Time (sec)	OIE	NIE	HNIE	LL	HLL
1	8	2	2	2	2	0	242	5	**6**	5	2	0
2	7	2	**4**	3	4	1	32	8	8	8	0	0
3	3	3	**4**	3	3	0	11	8	8	8	0	0
4	5	3	3	3	0	0	27	6	**7**	6	1	0
5	3	3	3	3	2	0	14	8	8	8	0	0
6	4	2	4	4	3	2	30	5	**7**	5	1	0
7	2	2	**4**	3	6	2	12	8	8	8	0	0
8	6	1	**2**	1	5	0	26	7	7	7	0	0
9	6	2	**3**	2	5	0	35	8	8	8	0	0

Table 6.1: ILP and Heuristic Results for Strict and Majority-rule Reduced Consensus Trees for the set of 100 trees on 10 taxa. The columns represent data set number (Dataset), total running time in seconds (Time), original internal edges (OIE), new internal edges after rogue taxon removal (NIE), and total leaf loss (LL) for exact solution, followed by the heuristic NIE (HNIE) and LL (HLL) results. A bold faced entry in the table indicates improvement of our ILP approach over the heuristic method.

method, which attempts to minimize the leaf loss as well as maximize NIE. It is straightforward to use an ILP solver to find all optimal solutions and choose the one with minimum leaf loss. Alternatively, it is easy to modify the objective function of our ILPs to make it a weighted sum of the number of leaves and the number of internal edges. No new variables need to be introduced, since the sum of the S_i's gives us the size of the selected set X. We did this and noticed an interesting phenomenon when the number of leaves and the number of internal edges have the same weight, which is effectively what is done in Pattengale et al. (2011). Here, we find that the ILP is reluctant to remove taxa to make gain on internal edges. Indeed, it would appear that the method is too conservative in this case.

Third, we found that reduced majority consensus trees tend to lose many fewer leaves than the reduced strict consensus trees; sometimes no leaves are lost at all. This might be explained as follows. For reduced majority-rule consensus, as long as a taxon is correctly placed in more than half of the input trees it will not be removed. In contrast, for strict reduced consensus, if a taxon is placed incorrectly in just one tree but correctly in others, it often has to be removed from all the input trees to achieve high resolution. Further, there are instances where

	Strict						Majority-rule					
Data set	Time (sec)	OIE	NIE	HNIE	LL	HLL	Time (sec)	OIE	NIE	HNIE	LL	HLL
1	7	5	**6**	5	5	0	36	12	12	12	0	0
2	97	3	**4**	3	4	0	75	13	13	13	0	0
3	24	3	5	5	7	5	1033	9	**10**	9	3	0
4	10	6	**7**	6	4	0	3815	10	10	10	0	0
5	200	3	**4**	3	6	0	1004	10	10	10	0	0
6	103	3	3	3	0	0	36066	9	**10**	9	5	0
7	5	6	6	6	1	0	17	12	12	12	1	0
8	4	8	8	8	0	0	14	11	11	11	0	0
9	19	6	6	6	0	0	983	10	**11**	10	2	0

Table 6.2: ILP and Heuristic Results for Strict and Majority-rule Reduced Consensus Trees for the set of 65 trees on 15 taxa. For column headings' details, see caption of Table 6.1.

two clusters that were non-majority originally, become identical after the removal of rogue taxa and their combined frequency exceeds the threshold value of 50%. However, the combined frequency is still not high enough (i.e., 100%) for the reduced cluster to appear in reduced strict consensus. In passing, we should note that in general the majority-rule reduced consensus ILP takes longer to solve than the strict reduced consensus ILP.

Finally, the ILP is much slower than the heuristic method. The latter typically ran within a fraction of a seconds in most of the cases we studied. In contrast, as the numbers of taxa and trees increase, the exact algorithm quickly becomes impractical. Despite their drawbacks, however, our exact solutions give a good benchmark against which to evaluate and explore the limitations of heuristic methods.

Author's contributions

AD and DFB conceived the reduced consensus problem and the NP-complete reduction. AD worked out the reduction and contributed fixed parameter tractable solution. JD and DFB conceived application of ILP for exact solutions. JD contributed ILP formulations. AD, JD and DFB designed the experiments. AD arranged for the datasets. JD implemented the experiments. AD and JD carried out the experiments. DFB, JD and AD drafted the

	Strict						Majority-rule					
Data set	Time (sec)	OIE	NIE	HNIE	LL	HLL	Time (sec)	OIE	NIE	HNIE	LL	HLL
1	334	4	**7**	4	10	0	1366	15	15	15	0	0
2	78	5	**8**	7	6	3	1762	15	**16**	15	1	0
3	243	5	**8**	6	8	3	62	18	18	18	0	0
4	79	7	**9**	8	7	1	6929	14	16	16	1	1
5	1897	6	**7**	6	3	0	7659	15	**16**	15	1	0
6	20	7	**10**	9	5	2	1417	15	**16**	15	2	0
7	22	8	**9**	8	2	0	28	18	18	18	0	0
8	9	9	9	9	0	0	46	15	**16**	15	1	0
9	163	8	8	8	1	0	30550	14	**15**	14	1	0

Table 6.3: ILP and Heuristic Results for Strict and Majority-rule Reduced Consensus Trees for the set of 50 trees on 20 taxa. For column heading's details, see caption of Table 6.1.

manuscript. DFB coordinated the project. All authors read and approved the final manuscript.

Acknowledgments

This research was supported in part by National Science Foundation grants DEB-0830012 and CCF-106029.

	Strict					
Data set	Time (sec)	OIE	NIE	HNIE	LL	HLL
1	2566	5	**8**	7	9	2
2	743	7	**9**	7	8	0
3	356	9	**11**	9	6	0
4	13320	6	**9**	8	7	2
5	1149	6	**8**	6	7	0
6	10394	4	**6**	5	3	1
7	44	12	**14**	12	7	0
8	424	10	**12**	11	3	1
9	4058	10	11	11	5	1

Table 6.4: ILP and Heuristic Results for Strict Reduced Consensus Trees for the set of 40 trees on 25 taxa. For column heading's details, see caption of Table 6.1. Results for Majority-rule Reduced Consensus Trees are not reported because more than often the computation went beyond the 24 hours time limit that we had set for practical purposes.

	Strict					
Data set	Time (sec)	OIE	NIE	HNIE	LL	HLL
1	11557	6	**8**	6	19	0
2	2284	7	**11**	9	14	4
3	4714	8	**10**	9	13	2
4	4039	12	**13**	12	6	0
5	120799	6	**11**	8	12	7
6	20861	7	**9**	8	7	1
7	366	14	**16**	15	2	1
8	53239	6	11	11	11	4
9	80304	8	**11**	10	7	2

Table 6.5: ILP and Heuristic Results for Strict Reduced Consensus Trees for the set of 30 trees on 30 taxa. For column heading's details, see caption of Table 6.1. Results for Majority-rule Reduced Consensus Trees are not reported because more than often the computation went beyond the 24 hours time limit that we had set for practical purposes.

CHAPTER 7. CONCLUSIONS

This thesis addresses a range of issues related to large phylogenetic databases: from their deployment (storing and querying), such as in STBase, to analyzing collections of phylogenetic trees for common and conflict-free information, and identification of rogue taxa.

Through STBase, we introduced a new infrastructure and methodologies to manage and analyze large phylogenetic databases. Through a combination of techniques from information retrieval, notably inverted indexing, and from computational phylogenetics, especially for constructing consensus trees, and use of efficient hash functions, STBase search engine achieves fast response times — responding to user queries within few seconds. Further, such a database consisting of a billion trees can be efficiently hosted on just a modern desktop computer. Apart from STBase itself, we are optimistic that our results would be very useful for other projects in Tree of Life initiatives such as: http://www.phylo.org/atol/ and http://opentreeoflife.org/.

Identifying common information in collections of phylogenetic trees is a very frequently encountered problem in phylogenetic analyses. We improved upon an existing approach for mining frequent agreement trees both in terms of the time taken (more than 100 times improvement!) and the size of the leafset of the input collection. We further extended our work to propose two new approaches for the same task: one based on agreement subtrees, called maximal agreement subtrees, the other on frequent subtrees, called maximal frequent subtrees. These approaches can return subtrees on a larger set of taxa than maximum agreement subtrees, and are capable of revealing new common phylogenetic relationships not present in either maximum agreement subtrees or the majority rule tree.

We introduced a novel approach to reduce MUL-trees to singly-labeled trees based on preserving conflict-free quartets from the original MUL-tree. The proposed approach is conservative with respect to species relationships, i.e., does not introduce quartets not already present

in the original tree, and is purely topology-based — unaffected by the underlying cause of label multiplicity. In the experiments, the reduced singly-labeled tree is often much smaller than the original tree, yet retains most of the taxa. Further, the reduction algorithm is quadratic in the number of leaves and its complexity is unaffected by the multiplicity of leaf labels or the degree of the nodes.

Rogue taxon problem is often encountered at various stages of phylogenetic reconstruction. Our results are the first one to prove that identification of rogue taxon through reduced consensus approach is an NP-hard problem. Our exact solutions — both constrained polynomial-time algorithm and integer linear programming formulations — are also the first of their kind. Results show that they can be a useful benchmark for evaluating heuristic approaches and, identify interesting characteristics and limitations of the various consensus methods.

LIST OF PUBLICATIONS

Published:

1. **Akshay Deepak**, Jianrong Dong, David Fernndez-Baca. Identifying rogue taxa through reduced consensus: Np-hardness and exact algorithms. In Proceedings of the 8th *International Conference on Bioinformatics Research and Applications*, Lecture Notes in Computer Science, pages 87-98, Berlin, Heidelberg, 2012. Springer-Verlag.

2. **Akshay Deepak**, Michelle M. McMahon, and David Fernndez-Baca. Extracting conflict-free information from multi-labeled trees. In Proceedings of the 12th *Workshop on Algorithms in Bioinformatics*, Lecture Notes in Computer Science, pages 81-92, Berlin, Heidelberg, 2012. Springer-Verlag.

Under Review/In Preparation:

1. **Akshay Deepak**, David Fernndez-Baca, Srikanta Tirthapura, Michael J. Sanderson and Michelle M. McMahon. EvoMiner: Frequent Subtree Mining in Phylogenetic Databases. *Knowledge and Information Systems*, under review, 2013.

2. Michelle M. McMahon, **Akshay Deepak**, David Fernndez-Baca, Darren Boss and Michael J. Sanderson. STBase: One billion trees for comparative biology. *Systematic Biology*, under review, 2013.

3. **Akshay Deepak**, Michelle M. McMahon, and David Fernndez-Baca. Extracting conflict-free information from multi-labeled trees. *Algorithms for Molecular Biology* (invited paper), under review, 2013.

4. **Akshay Deepak** and David Fernndez-Baca. Enumerating All Maximal Frequent Subtrees. In preparation.

5. **Akshay Deepak**, Jianrong Dong, David Fernndez-Baca. Identifying rogue taxa through reduced consensus: Np-hardness and exact algorithms. In preparation, 2013.

BIBLIOGRAPHY

Aggarwal, C. C. and Wang, H. (2010). *Managing and Mining Graph Data*, volume 40 of *Advances in Database Systems*. Springer. 20

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., and Verkamo, A. (1996). Fast Discovery of Association Rules. *Advances in Knowledge Discovery and Data Mining*, 12:307–328. 20, 25, 30, 32, 36

Altschul, S., Gish, W., Miller, W., Myers, E., Lipman, D., et al. (1990). Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410. 7

Amenta, N., Clarke, F., and John, K. S. (2003). A linear-time majority tree algorithm. In *Proceedings of the 3rd Workshop on Algorithms in Bioinformatics (WABI'03)*, pages 216–227. 10, 21, 23, 50, 110, 116

Amir, A. and Keselman, D. (1994). Maximum agreement subtree in a set of evolutionary trees. *SIAM Journal on Computing*, 26:758–769. 21, 23, 58, 60, 108

Asai, T., Abe, K., Kawasoe, S., Arimura, H., Sakamoto, H., and Arikawa, S. (2002). Efficient substructure discovery from large semi-structured data. In *Proceedings of the SIAM International Conference on Data Mining*, pages 158–174. 18, 19, 20

Asai, T., Arimura, H., Uno, T., and Nakano, S.-i. (2003). Discovering frequent substructures in large unordered trees. In *Proceedings of the 6th International Conference on Discovery Science*, pages 47–61. 18, 19, 20, 33

Avis, D. and Fukuda, K. (1996). Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1):21–46. 63

Ayres, J., Flannick, J., Gehrke, J., and Yiu, T. (2002). Sequential pattern mining using a bitmap representation. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 429–435. 48, 78

Bader, D., Roshan, U., and Stamatakis, A. (2006). Computational grand challenges in assembling the tree of life: Problems and solutions. *Advances in computers*, 68:127–176. 6

Bansal, M., Burleigh, J. G., Eulenstein, O., and Fernández-Baca, D. (2010). Robinson-Foulds supertrees. *Algorithms for Molecular Biology*, 5(1):18. 88

Barns, S., Delwiche, C., Palmer, J., and Pace, N. (1996). Perspectives on archaeal diversity, thermophily and monophyly from environmental rRNA sequences. *Proceedings of the National Academy of Sciences*, 93:9188–9193. 17

Baum, B. R. (1992). Combining trees as a way of combining data sets for phylogenetic inference, and the desirability of combining gene trees. *Taxon*, 41(1):pp. 3–10. 88

Baum, D. (2008). Reading a phylogenetic tree: The meaning of monophyletic groups. *Nature Education*, 1(1). 17, 26

Baum, D. and Smith, S. (2012). *Tree Thinking: An introduction to phylogenetic biology*. Roberts and Company Publishers, 1st edition. 6

Bei, Y., Chen, G., Shou, L., Li, X., and Dong, J. (2009). Bottom-up discovery of frequent rooted unordered subtrees. *Information Sciences*, 179:70–88. 18

Bender, M. and Farach-Colton, M. (2000). The LCA problem revisited. In *Proceedings of the 4th Latin American Symposium on Theoretical Informatics*, pages 88–94. 31

Benson, D., Karsch-Mizrachi, I., Lipman, D., Ostell, J., and Wheeler, D. (2008). Genbank. *Nucleic acids research*, 36(suppl 1):D25. 1

Bhaskar, R., Laxman, S., Smith, A., and Thakurta, A. (2010). Discovering frequent patterns in sensitive data. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 503–512. 20

Bininda-Emonds, O., Cardillo, M., Jones, K., MacPhee, R., Beck, R., Grenyer, R., Price, S., Vos, R., Gittleman, J., and Purvis, A. (2007). The delayed rise of present-day mammals. *Nature*, 446(7135):507–512. 6

Bryant, D. (1997). *Building trees, hunting for trees and comparing trees*. PhD thesis, University of Canterbury, New Zealand. 23, 60

Bryant, D. (2003a). A classification of consensus methods for phylogenetics. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 61:163–184. 21, 47, 52, 58

Bryant, D. (2003b). A classification of consensus methods for phylogenetics. In Janowitz, M., Lapointe, F.-J., McMorris, F., B. Mirkin, B., and Roberts, F., editors, *Bioconsensus*, volume 61 of *Discrete Mathematics and Theoretical Computer Science*, pages 163–185. American Mathematical Society, Providence, RI. 111

Chi, Y., Muntz, R., Nijssen, S., and Kok, J. (2004a). Frequent Subtree Mining — An Overview. *Fundamenta Informaticae*, 66:161–198. 18, 20, 44, 55, 108

Chi, Y., Xia, Y., Yang, Y., and Muntz, R. (2005). Mining closed and maximal frequent subtrees from databases of labeled rooted trees. *IEEE Transactions on Knowledge and Data Engineering*, 17:190–202. 18, 19, 47, 55, 60

Chi, Y., Yang, Y., and Muntz, R. (2004b). Hybridtreeminer: An efficient algorithm for mining frequent rooted trees and free trees using canonical forms. In *Proceedings of the 16th International Conference on Scientific and Statistical Database Management*, pages 11–20. 19, 20

Chi, Y., Yang, Y., and Muntz, R. R. (2003). Indexing and mining free trees. In *Proceedings of the IEEE International Conference on Data Mining*, pages 509–512. 19, 20, 32

Cole, R., Farach-Colton, M., Hariharan, R., Przytycka, T., and Thorup, M. (2000). An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees. *SIAM Journal on Computing*, 30:1385–1404. 23

Cormen, T. (2001). *Introduction to algorithms*. The MIT press. 11

Cranston, K. and Rannala, B. (2007). Summarizing a posterior distribution of trees using agreement subtrees. *Systematic biology*, 56(4):578. 108, 114

Currie, T. E., Greenhill, S. J., Gray, R. D., Hasegawa, T., and Mace, R. (2010). Rise and fall of political complexity in island South-East Asia and the Pacific. *Nature*, 467:801–804. 17

Daubin, V., Gouy, M., and Perrière, G. (2002). A phylogenomic approach to bacterial phylogeny: evidence of a core of genes sharing a common history. *Genome Research*, 12:1080–1090. 52, 58, 82

de Queiroz, A. and Gatesy, J. (2007). The supermatrix approach to systematics. *Trends in Ecology & Evolution*, 22(1):34–41. 88

De Vienne, D., Giraud, T., and Martin, O. (2007). A congruence index for testing topological similarity between trees. *Bioinformatics*, 23:3119–3124. 52, 58, 82

Deepak, A. (2010). Searchtree: Mining robust phylogenetic trees. Master's thesis, Iowa State University. 3, 11

Deepak, A., Dong, J., and Fernández-Baca, D. (2012a). Identifying rogue taxa through reduced consensus: Np-hardness and exact algorithms. In *Bioinformatics Research and Applications*, volume 7292 of *Lecture Notes in Computer Science*, pages 87–98. Springer Berlin / Heidelberg. 124

Deepak, A., Fernández-Baca, D., and McMahon, M. (2012b). Extracting conflict-free information from multi-labeled trees. In *Proceedings of the 12th Workshop on Algorithms in Bioinformatics (WABI'12)*, pages 81–92. Springer. 12

Do, T., Laurent, A., and Termier, A. (2010). Pglcm: Efficient parallel mining of closed frequent gradual itemsets. In *Proceedings of the 10th IEEE International Conference on Data Mining*, pages 138–147. 55

Dong, J. and Fernández-Baca, D. (2011). Constructing large conservative supertrees. In *WABI*, pages 61–72. 109, 118

Dong, J., Fernández-Baca, D., and McMorris, F. R. (2010). Constructing majority-rule supertrees. *Algorithms in Molecular Biology*, 5(2). 109, 118

Dong, S. and Kraemer, E. (2004). Calculation, visualization, and manipulation of masts (maximum agreement subtrees). In *Proceedings of the IEEE Computational Systems Bioinformatics Conference CSB*, pages 405–414. 52, 58

Driskell, A., Ané, C., Burleigh, J., McMahon, M., O'Meara, B., and Sanderson, M. (2004). Prospects for building the tree of life from large sequence databases. *Science*, 306(5699):1172–1174. 14

Edgar, R. (2004). Muscle: a multiple sequence alignment method with reduced time and space complexity. *BMC bioinformatics*, 5(1):113. 12

Edwards, E. (2011). New grass phylogeny resolves deep evolutionary relationships and discovers c4 origins. *New Phytologist*, 193:304–312. 6

Farach, M., Przytycka, T., and Thorup, M. (1995a). On the agreement of many trees. *Information Processing Letters*, 55:297–301. 21, 23

Farach, M., Przytycka, T., and Thorup, M. (1995b). On the agreement of many trees. *Information Processing Letters*, 55(6):297–301. 60

Farach, M., Przytycka, T. M., and Thorup, M. (1995c). On the agreement of many trees. *Inf. Process. Lett.*, 55(6):297–301. 108

Farach, M. and Thorup, M. (1994). Fast comparison of evolutionary trees. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 481–488. 52, 58

Fellows, M., Hallett, M., and Stege, U. (2003). Analogs & duals of the mast problem for sequences & trees. *Journal of Algorithms*, 49(1):192 – 216. 1998 European Symposium on Algorithms. 12, 85

Felsenstein, J. (1985). Confidence limits on phylogenies: An approach using the bootstrap. *Evolution*, 39:783–791. 17, 52

Felsenstein, J. (2003). *Inferring Phylogenies*. Sinauer Assoc., Sunderland, Mass. 6

Felsenstein, J. (2004). *Phylogenetics*. Sunderland, Massachusetts: Sinauer Associates. 61

Feng, B., Xu, Y., Zhao, N., and Xu, H. (2010). A new method of mining frequent closed trees in data streams. In *Proceedings of the Seventh International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, pages 2245–2249. 18, 55

Finden, C. and Gordon, A. (1985). Obtaining common pruned trees. *Journal of Classification*, 2:255–276. 21, 58, 60, 108

Finn, R., Mistry, J., Tate, J., Coggill, P., Heger, A., Pollington, J., Gavin, O., Gunasekaran, P., Ceric, G., Forslund, K., et al. (2010). The pfam protein families database. *Nucleic acids research*, 38(suppl 1):D211–D222. 1, 7

Flint-Garcia, S., Thuillet, A., Yu, J., Pressoir, G., Romero, S., Mitchell, S., Doebley, J., Kresovich, S., Goodman, M., and Buckler, E. (2005). Maize association population: a high-resolution platform for quantitative trait locus dissection. *The Plant Journal*, 44:1054–1064. 17

Flynn, S., Turner, R., and Stuppy, W. (2006). Seed information database (release 7.0, oct. 2006). 6

Ganapathy, G., Goodson, B., Jansen, R., Le, H., Ramachandran, V., and Warnow, T. (2006). Pattern identification in biogeography. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3:334–346. 85, 88

Ganapathysaravanabavan, G. and Warnow, T. (2001). Finding a maximum compatible tree for a bounded number of trees with bounded degree is solvable in polynomial time. In *Proceedings of the International Workshop on Algorithms in Bioinformatics*, pages 156–163. 22, 23, 24

Geerts, F., Goethals, B., and Bussche, J. (2005). Tight upper bounds on the number of candidate patterns. *ACM Transactions on Database Systems (TODS)*, 30:333–363. 45

Goddard, W., Kubicka, E., Kubicki, G., and McMorris, F. (1994). The agreement metric for labeled binary trees. *Mathematical Biosciences*, 123:215–226. 52, 58, 82

Goldman, N. and Yang, Z. (2008). Introduction. statistical and computational challenges in molecular phylogenetics and evolution. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 363(1512):3889–3892. 6

Gray, R., Drummond, A., and Greenhill, S. (2009). Language phylogenies reveal expansion pulses and pauses in Pacific settlement. *Science*, 323:479–483. 17

Grundt, H., Popp, M., Brochmann, C., and Oxelman, B. (2004). Polyploid origins in a circumpolar complex in draba (brassicaceae) inferred from cloned nuclear dna sequences and fingerprints. *Molecular Phylogenetics and Evolution*, 32(3):695 – 710. 12, 85

Guillemot, S. and Berry, V. (2010). Fixed-parameter tractability of the maximum agreement supertree problem. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7:342–353. 24

Gusfield, D., Frid, Y., and Brown, D. (2007). Integer programming formulations and computations solving phylogenetic and population genetic problems with missing or genotypic data. In Lin, G., editor, *Computing and Combinatorics, 13th Annual International Conference, COCOON 2007, Banff, Canada, July 16-19, 2007, Proceedings*, volume 4598, pages 51–64. Springer. 109

Hadzic, F., Tan, H., Dillon, T., Hadzic, F., Tan, H., and Dillon, T. (2010). Mining maximal and closed frequent subtrees. In *Mining of Data with Complex Structures*, volume 333 of *Studies in Computational Intelligence*, pages 191–199. Springer Berlin / Heidelberg. 18, 55

Han, J. and Pei, J. (2000). Mining frequent patterns by pattern-growth: methodology and implications. *ACM SIGKDD Explorations Newsletter*, 2:14–20. 45

Han, J., Pei, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., and Hsu, M. (2001). Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proceedings of the 17th International Conference on Data Engineering*, pages 215–224. 46

Han, J., Pei, J., and Yin, Y. (2000). Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 1–12. 18, 44, 46

Han, J., Pei, J., Yin, Y., and Mao, R. (2004). Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery*, 8:53–87. 18, 44

Harel, D. and Tarjan, R. (1984). Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13:338–355. 31

Hromkovič, J. (2005). Abundance of witnesses. In *Design and Analysis of Randomized Algorithms*, Texts in Theoretical Computer Science. An EATCS Series, pages 183–207. Springer Berlin Heidelberg. 39

Huan, J., Wang, W., Prins, J., and Yang, J. (2004). Spin: mining maximal frequent subgraphs from graph databases. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 581–586. ACM. 60

Huber, K., Lott, M., Moulton, V., and Spillner, A. (2008). The complexity of deriving multi-labeled trees from bipartitions. *Journal of Computational Biology*, 15(6):639–651. 88

Huber, K. and Moulton, V. (2006). Phylogenetic networks from multi-labelled trees. *Journal of Mathematical Biology*, 52:613–632. 12, 85

Huber, K., Spillner, A., Suchecki, R., and Moulton, V. (2011). Metrics on multilabeled trees: Interrelationships and diameter bounds. *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, 8(4):1029 –1040. 88

Huelsenbeck, J., Ronquist, F., et al. (2001). Mrbayes: Bayesian inference of phylogenetic trees. *Bioinformatics*, 17(8):754–755. 8

Huelsenbeck, J. P. and Ronquist, F. (2001). MRBAYES: Bayesian inference of phylogenetic trees. *Bioinformatics*, 17:754–755. 17, 52

Izquierdo-Carrasco, F., Smith, S., and Stamatakis, A. (2011). Algorithms, data structures, and numerics for likelihood-based phylogenetic inference of huge trees. *BMC bioinformatics*, 12(1):470. 6

Jenkins, B. (1997). Algorithm alley: Hash functions. *Dr. Dobb's Journal of Software Tools*, 22(9). 11

Jia, Y., Zhang, J., and Huan, J. (2011). An efficient graph-mining method for complicated and noisy data with real-world applications. *Knowledge and Information Systems*, 28:423–447. 20

Jimenez, A., Berzal, F., and Cubero, J. (2010a). Frequent tree pattern mining: A survey. *Intelligent Data Analysis*, 14:603–622. 18, 21

Jimenez, A., Berzal, F., and Cubero, J. (2010b). Potminer: Mining ordered, unordered, and partially-ordered treesining ordered, unordered, and partially-ordered trees. *Knowledge and Information Systems*, 23:199–224. 19

Johnson, K., Adams, R., Page, R., and Clayton, D. (2003). When do parasites fail to speciate in response to host speciation? *Syst. Biol.*, 52:37–47. 85

Kao, M., Lam, T., Sung, W., and Ting, H. (2001). An even faster and more unifying algorithm for comparing trees via unbalanced bipartite matchings. *Journal of Algorithms*, 40:212–233. 21, 60

Karp, R. (1972). Reducibility among combinatorial problems. In Miller, R. E. and Thatcher, J. W., editors, *Complexity of Computer Computations*. Plenum, New York. 111

Karp, R. and Rabin, M. (1987). Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31:249–260. 36

Ke, Y., Cheng, J., and Yu, J. (2009). Efficient discovery of frequent correlated subgraph pairs. In *Proceedings of the Ninth IEEE International Conference on Data Mining*, pages 239–248. 55

Kubicka, E., Kubicki, G., and McMorris, F. (1992). On agreement subtrees of two binary trees. *Congressus Numerantium*, 88:217–217. 55

Lanfear, R., Bromham, L., et al. (2011). Estimating phylogenies for species assemblages: A complete phylogeny for the past and present native birds of new zealand. *Molecular phylogenetics and evolution*, 61(3):958–963. 6

Lapointe, F. and Rissler, L. (2005). Congruence, consensus, and the comparative phylogeography of codistributed species in California. *The American Naturalist*, 166:290–299. 52, 58

Lee, C.-M., Hung, L.-J., Chang, M.-S., Shen, C.-B., and Tang, C.-Y. (2005). An improved algorithm for the maximum agreement subtree problem. *Information Processing Letters*, 94(5):211–216. 108

Lewis, L. and Lewis, P. (2005). Unearthing the molecular phylodiversity of desert soil green algae (Chlorophyta). *Systematic Biology*, 54:936–947. 47

Liu, H., Lin, Y., and Han, J. (2011). Methods for mining frequent items in data streams: an overview. *Knowledge and Information Systems*, 26:1–30. 20

Liu, L. and Liu, J. (2011). Mining frequent embedded subtree from tree-like databases. In *Proceedings of the International Conference on Internet Computing & Information Services (ICICIS)*, pages 3–7. 18

Liu, L., Yu, L., Kubatko, L., Pearl, D., and Edwards, S. (2009). Coalescent methods for estimating phylogenetic trees. *Molecular Phylogenetics and Evolution*, 53(1):320. 6

Lott, M., Spillner, A., Huber, K., Petri, A., Oxelman, B., and Moulton, V. (2009). Inferring polyploid phylogenies from multiply-labeled gene trees. *BMC evolutionary biology*, 9(1):216. 85

Manning, C., Raghavan, P., and Schütze, H. (2008). *Introduction to information retrieval*. Cambridge University Press. 10

Marcet-Houben, M. and Gabaldón, T. (2011). Treeko: a duplication-aware algorithm for the comparison of phylogenetic trees. *Nucleic Acids Research*, 39:e66. 88, 89

Margush, T. and McMorris, F. (1981a). Consensus n-trees. *Bulletin of Mathematical Biology*, 43:239–244. 21

Margush, T. and McMorris, F. (1981b). Consensus n-trees. *Bulletin of Mathematical Biology*, 43(2):239–244. 107

Mau, B., Newton, M., and Larget, B. (1999). Bayesian phylogenetic inference via Markov chain Monte Carlo methods. *Biometrics*, 55:1–12. 48

McMahon, M., Deepak, A., Fernandez-Baca, D., Boss, D., and Sanderson, M. J. (2013). Stbase: One billion species trees for comparative biology. *Systematic biology*. Submitted manuscript. 1

Michael, S., Michelle, M., and Mike, S. (2010). Phylogenomics with incomplete taxon coverage: the limits to inference. *BMC Evolutionary Biology*, 10:155. 5

Moles, A., Ackerly, D., Webb, C., Tweddle, J., Dickie, J., and Westoby, M. (2005). A brief history of seed size. *Science*, 307(5709):576–580. 6

Motwani, R. and Raghavan, P. (1995). *Randomized algorithms*, chapter 7. Cambridge: Cambridge Univ. 11, 37

Nadler, S., Carreno, R., Mejía-Madrid, H., Ullberg, J., Pagan, C., Houston, R., and Hugot, J. (2007). Molecular phylogeny of clade III nematodes reveals multiple origins of tissue parasitism. *Parasitology*, 134(10):1421–1442. 107

NCBI (2002). Tree facts: Rooted versus unrooted trees. Online. 26

Nguyen, V. and Yamamoto, A. (2010). Incremental mining of closed frequent subtrees. In Pfahringer, B., Holmes, G., and Hoffmann, A., editors, *Discovery Science*, volume 6332 of *Lecture Notes in Computer Science*, pages 356–370. Springer, Berlin / Heidelberg. 18

Nijssen, S. and Kok, J. (2003). Efficient discovery of frequent unordered trees. In *Proceedings of the International Workshop on Mining Graphs, Trees and Sequences*, pages 55–64. 18, 19, 20

Nyakatura, K. and Bininda-Emonds, O. (2012). Updating the evolutionary history of carnivora (mammalia): a new species-level supertree complete with divergence time estimates. *BMC biology*, 10(1):12. 6

Pattengale, N., Aberer, A., Swenson, K., Stamatakis, A., and Moret, B. (2011). Uncovering hidden phylogenetic consensus in large datasets. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 8:902–911. 22, 47, 53, 59, 79, 108, 109, 114, 124, 125, 126

Pei, J. and Han, J. (2002). Constrained frequent pattern mining: A pattern-growth view. *ACM SIGKDD Explorations Newsletter*, 4:31–39. 18

Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., and Hsu, M. (2004). Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Transactions on Knowledge and Data Engineering*, 16:1424–1440. 46

Pei, J., Han, J., and Wang, W. (2007). Constraint-based sequential pattern mining: the pattern-growth methods. *Journal of Intelligent Information Systems*, 28:133–160. 46

Peters, R., Meyer, B., Krogmann, L., Borner, J., Meusemann, K., Schütte, K., Niehuis, O., and Misof, B. (2011). The taming of an impossible child: a standardized all-in approach to the phylogeny of hymenoptera using public database sequences. *BMC biology*, 9(1):55. 6

Piel, W., Donoghue, M., and Sanderson, M. (2002). Treebase: a database of phylogenetic knowledge. In J. Shimura, K. L. Wilson and D. Gordon, eds. *To the Interoperable "Catalog of Life" with Partners, Species 2000 Asia Oceania*, number 171, pages 41–47. Research Report from the National Institute for Environmental Studies , Tsukuba, Japan. 1, 7, 17, 24

Popp, M. and Oxelman, B. (2001). Inferring the history of the polyploid silene aegaea (caryophyllaceae) using plastid and homoeologous nuclear dna sequences. *Molecular Phylogenetics and Evolution*, 20(3):474 – 481. 12, 85

Pringle, E., Alvarez-Loayza, P., and Terborgh, J. (2007). Seed characteristics and susceptibility to pathogen attack in tree seeds of the peruvian amazon. *Plant Ecology*, 193(2):211–222. 6

Puigbò, P., Garcia-Vallvé, S., and McInerney, J. (2007). Topd/fmts: a new software to compare phylogenetic trees. *Bioinformatics*, 23(12):1556. 88, 89

Ragan, M. (1992). Phylogenetic inference based on matrix representation of trees. *Molecular phylogenetics and evolution*, 1(1):53–58. 88

Raissi, C. and Pei, J. (2011). Towards bounding sequential patterns. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1379–1387. 55

Rannala, B. and Yang, Z. (2008). Phylogenetic inference using whole genomes. *Annual Review of Genomics and Human Genetics*, 9:217–231. 17

Rasmussen, M. and Kellis, M. (2012). Unified modeling of gene duplication, loss, and coalescence using a locus tree. *Genome Research*, 22:755–765. 85

Redelings, B. (2009). Bayesian phylogenies unplugged: Majority consensus trees with wandering taxa. 107, 109

Roure, B., Baurain, D., and Philippe, H. (2012). Impact of missing data on phylogenies inferred from empirical phylogenomic datasets. *Molecular Biology and Evolution*, 30:197—214. 6

Sanderson, M. (2008). Phylogenetic signal in the eukaryotic tree of life. *Science*, 321(5885):121–123. 6

Sanderson, M., Boss, D., Chen, D., Cranston, K., and Wehe, A. (2008). The PhyLoTA browser: processing GenBank for molecular phylogenetics research. *Systematic Biology*, 57:335–346. 1, 7, 12, 17, 24, 85, 102

Sanderson, M., Driskell, A., Ree, R., Eulenstein, O., and Langley, S. (2003). Obtaining maximal concatenated phylogenetic data sets from large sequence databases. *Molecular biology and evolution*, 20(7):1036–1042. 14

Sanderson, M., McMahon, M., and Steel, M. (2010). Phylogenomics with incomplete taxon coverage: the limits to inference. *BMC Evolutionary Biology*, 10(1):155. 13

Sanderson, M., McMahon, M., and Steel, M. (2011). Terraces in phylogenetic tree space. *Science*, 333:448–450. 6, 13, 52, 58

Saslis-Lagoudakis, C., Savolainen, V., Williamson, E., Forest, F., Wagstaff, S., Baral, S., Watson, M., Pendry, C., and Hawkins, J. (2012). Phylogenies reveal predictive power of traditional medicine in bioprospecting. *Proceedings of the National Academy of Sciences*, 109(39):15835–15840. 6

Schieber, B. and Vishkin, U. (1988). On finding lowest common ancestors: simplification and parallelization. *SIAM Journal on Computing*, 17:1253–1262. 31

Scornavacca, C. (2009). *Supertree methods for phylogenomics*. PhD thesis, University of Montpellier II, Montpellier, France. 21

Scornavacca, C., Berry, V., and Ranwez, V. (2011). Building species trees from larger parts of phylogenomic databases. *Information and Computation*, 209(3):590 – 605. Special Issue: 3rd International Conference on Language and Automata Theory and Applications (LATA 2009). 12, 85, 88, 89, 104

Semple, C. and Steel, M. (2003a). *Phylogenetics*. Oxford Lecture Series in Mathematics. Oxford University Press, Oxford. 27, 110

Semple, C. and Steel, M. (2003b). *Phylogenetics*. Oxford University Press, Oxford. 89

Slowinski, J. and Keogh, J. (2000). Phylogenetic relationships of elapid snakes based on cytochrome b mtDNA sequences. *Molecular Phylogenetics and Evolution*, 15:157–164. 17

Smith, M. and Patton, J. (1999). Phylogenetic relationships and the radiation of sigmodontine rodents in South America: Evidence from cytochrome b. *Journal of Mammalian Evolution*, 6:89–128. 17

Smith, S., Beaulieu, J., and Donoghue, M. (2009). Mega-phylogeny approach for comparative biology: an alternative to supertree and supermatrix approaches. *BMC evolutionary biology*, 9(1):37. 6

Smith, S., Beaulieu, J., Stamatakis, A., and Donoghue, M. (2011). Understanding angiosperm diversification using small and large phylogenetic trees. *American Journal of Botany*, 98(3):404–414. 6

Sridhar, S., Lam, F., Blelloch, G. E., Ravi, R., and Schwartz, R. (2008). Mixed integer linear programming for maximum-parsimony phylogeny inference. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 5(3):323–331. 109

Stamatakis, A. (2006). Raxml-vi-hpc: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics*, 22(21):2688–2690. 12

Steel, M. (1992). The complexity of reconstructing trees from qualitative characters and subtrees. *Journal of Classification*, 9(1):91–116. 86

Steel, M. and Sanderson, M. (2010). Characterizing phylogenetically decisive taxon coverage. *Applied Mathematics Letters*, 23(1):82–86. 13

Steel, M. and Warnow, T. (1993). Kaikoura tree theorems: computing the maximum agreement subtree. *Information Processing Letters*, 48:77–82. 23, 60

Stolzer, M., Lai, H., Xu, M., Sathaye, D., Vernot, B., and Durand, D. (2012). Inferring duplications, losses, transfers and incomplete lineage sorting with nonbinary species trees. *Bioinformatics*, 28(18):i409–i415. 86

Sul, S. and Williams, T. (2009). An experimental analysis of consensus tree algorithms for large-scale tree collections. In *Proceedings of the International Symposium on Bioinformatics Research and Applications*, pages 100–111. 47, 54

Sullivan, J. and Swofford, D. (1997). Are guinea pigs rodents? The importance of adequate models in molecular phylogenetics. *Journal of Mammalian Evolution*, 4(2):77–86. 107

Swenson, K., Chen, E., Pattengale, N., and Sankoff, D. (2011). The Kernel of Maximum Agreement Subtrees. In *Proceedings of the International Symposium on Bioinformatics Research and Applications*, pages 123–135. 23, 24, 48, 59, 108

Swenson, M., Suri, R., Linder, C., and Warnow, T. (2012). Superfine: fast and accurate supertree estimation. *Systematic biology*, 61(2):214–227. 88

Swofford, D. (2003). {PAUP*. Phylogenetic Analysis Using Parsimony (* and Other Methods). Version 4.}. 12

Termier, A., Rousset, M., and Sebag, M. (2004). Dryade: a new approach for discovering closed frequent trees in heterogeneous tree databases. In *Proceedings of the IEEE International Conference on Data Mining*, pages 543–546. 18, 19, 55

Thomas, L., Valluri, S., and Karlapalem, K. (2006). Margin: Maximal frequent subgraph mining. In *Proc. IEEE International Conference on Data Mining*, pages 1097–1101. IEEE. 60

Thomson, R. and Shaffer, H. (2010). Sparse supermatrices for phylogenetic inference: taxonomy, alignment, rogue taxa, and the phylogeny of living turtles. *Systematic Biology*, 59(1):42. 107, 108

Wang, C., Hong, M., Pei, J., Zhou, H., Wang, W., and Shi, B. (2004). Efficient pattern-growth methods for frequent tree pattern mining. In Dai, H., Srikant, R., and Zhang, C., editors, *Advances in Knowledge Discovery and Data Mining*, volume 3056 of *Lecture Notes in Computer Science*, pages 441–451. Springer, Berlin / Heidelberg. 18, 19, 44, 46, 47

Wang, J., Shan, H., Shasha, D., and Piel, W. (2005). Fast structural search in phylogenetic databases. *Evolutionary Bioinformatics Online*, 1:37–46. 40

Wang, K. and Liu, H. (1998). Discovering typical structures of documents: a road map approach. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 146–154. ACM. 60

Wang, S., Hong, Y., and Yang, J. (2012). XML document classification using closed frequent subtree. In Bao, Z., Gao, Y., Gu, Y., Guo, L., Li, Y., Lu, J., Ren, Z., Wang, C., and Zhang, X., editors, *Web-Age Information Management*, volume 7419 of *Lecture Notes in Computer Science*, pages 350–359. Springer Berlin, Heidelberg. 18, 55

Webb, C. and Donoghue, M. (2004). Phylomatic: tree assembly for applied phylogenetics. *Molecular Ecology Notes*, 5(1):181–183. 6

Wiens, J. J. and Reeder, T. W. (1995). Combining data sets with different numbers of taxa for phylogenetic analysis. *Systematic Biology*, 44(4):pp. 548–558. 88

Wilkinson, M. (1994). Common cladistic information and its consensus representation: reduced adams and reduced cladistic consensus trees and profiles. *Systematic Biology*, 43(3):343. 107, 108

Wilkinson, M. (1995). More on reduced consensus methods. *Systematic Biology*, 44(3):435. 108, 109

Wilkinson, M. (1996). Majority-rule reduced consensus trees and their use in bootstrapping. *Molecular Biology and Evolution*, 13(3):437. 108

Wu, X., Kumar, V., Ross Quinlan, J., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G., Ng, A., Liu, B., Yu, P., et al. (2008). Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14:1–37. 20

Xiao, Y. and Yao, J. (2003). Efficient data mining for maximal frequent subtrees. In *Proceedings of the IEEE International Conference on Data Mining*, pages 379–386. 18, 19, 20, 55, 60, 108

Yang, L. H., Lee, M. L., Hsu, W., and Acharya, S. (2003). Mining frequent query patterns from XML queries. In *Proceedings of the Eighth International Conference on Database Systems for Advanced Applications*, pages 355–362. 18

Yule, G. (1925). A mathematical theory of evolution, based on the conclusions of Dr. JC Willis, F.R.S. *Philosophical Transactions of the Royal Society of London. Series B, Containing Papers of a Biological Character*, 213:21–87. 47

Zaki, M. (2004). Efficiently mining frequent embedded unordered trees. *Fundamenta Informaticae*, 66:33–52. 19, 20

Zaki, M. (2005). Efficiently mining frequent trees in a forest: Algorithms and applications. *IEEE Transactions on Knowledge and Data Engineering*, 17:1021–1035. 18, 19, 20, 25, 27, 31, 44, 108

Zhang, S. and Wang, J. (2008). Discovering frequent agreement subtrees from phylogenetic data. *IEEE Transactions on Knowledge and Data Engineering*, 20:68–82. 19, 20, 23, 25, 29, 36, 47, 48, 59, 60, 62, 63, 65, 79, 80, 81, 108

Zhang, S., Yang, J., and Li, S. (2009). Ring: An integrated method for frequent representative subgraph mining. In *Proceedings of the Ninth IEEE International Conference on Data Mining*, pages 1082–1087. 55

Zobel, J. and Moffat, A. (2006). Inverted files for text search engines. *ACM Computing Surveys (CSUR)*, 38(2):6–es. 9

Zou, X., Zhang, F., Zhang, J., Zang, L., Tang, L., Wang, J., Sang, T., and Ge, S. (2008). Analysis of 142 genes resolves the rapid diversification of the rice genus. *Genome Biology*, 9:R49. 17

Zou, Z., Gao, H., and Li, J. (2010). Discovering frequent subgraphs over uncertain graph databases under probabilistic semantics. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 633–642. 20