COMPUTING ROBINSON-FOULDS SUPERTREE FOR TWO TREES

BY

XILIN YU

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2019

Urbana, Illinois

Adviser:

Professor Tandy Warnow

Abstract

Supertree problems are important in phylogeny estimation. Supertree construction takes in a set of input trees on subsets of species and aims to find a supertree containing all species subjective to some combinatorial or statistical criterion. As such, it can be used to combine trees estimated by different research projects, or to construct species trees from gene trees that may not contain all species, or to serve a part in divide-and-conquer pipelines that improve the scalability of large scale phylogeny estimation. Yet the most promising supertree methods, such as the popular Robinson-Foulds Supertree (RFS) methods, not only cannot guarantee an optimal solution but also are computationally intensive by themselves, as they are heuristics for **NP**-hard optimization problems.

We present the first polynomial time algorithm to exactly solve the RFS problem on two binary input trees, and prove that finding the Robinson-Foulds Supertree of three input trees is **NP**-hard. We present GreedyRFS, a greedy heuristic for the Robinson-Foulds Supertree problem that operates by using our exact algorithm for RFS on pairs of trees, until all the trees are merged into a single supertree. Our experiments show that GreedyRFS has better accuracy than FastRFS, the leading heuristic for RFS, when the number of input trees is small, which is the natural case for use within divide-and-conquer pipelines.

To my dearest parents, for your unconditional love and trust.

给我亲爱的老爸老妈，感谢你们这么多年对我无条件的爱和信任。

不管我走到哪里，你们都是我的港湾。

给未来的自己，“去吧，愿你一路都平安，桥都坚固，隧道都光明”。

Acknowledgments

Results in this thesis are adapted from the paper "Advancing Divide-and-Conquer Phylogeny Estimation: Solving the Robinson-Foulds Supertree Problem for Two Trees" currently under submission and co-authored by Thien Le, Erin K. Molloy, Sarah A. Christensen, Professor Tandy Warnow, and me.

First and foremost, I would like to thank my thesis advisor Professor Tandy Warnow, who conceived of and directed this research project. Tandy, thank you for giving me the opportunity to work on such an exciting problem and for always being a source of guidance, inspiration, and encouragement. I want to thank Thien Le for implementing the divide-and-conquer strategy and performing the experimental evaluations for our paper. Thien, you are such a reliable teammate and it was a pleasure to work with you. I also want to thank my group-mates Erin K. Molloy, Sarah A. Christensen, and Mike Nute. Sarah, the time we spent brain-storming ideas was invaluable to me and thank you for your help on checking all the proofs. Erin, thank you for the discussions we have had and the efforts on making the paper more understandable. Mike, although you did not work with us on this project, I truly appreciate your advice and encouragement when I first joined the group, completely new to bioinformatics. This research project is also supported by the Grainger Foundation and the National Science Foundation grants 1458652, 1513629, and 1535977 awarded to Tandy Warnow.

Last but not the least, I want to give a special thank you to my boyfriend, Sahand Mozaffari. Sahand, thanks for always believing in me. Your love and support is what gives me the courage to take on the challenges of life and work.

Table of Contents

List of Notations

Standard Notation

$\mathcal{A}$	a profile, i.e., a set of unrooted source trees $\{T_1, T_2, \ldots, T_N\}$
N	the number of source trees in a profile
$[k]$	the set of integers from 1 to k, i.e., $\{1, 2, \ldots, k\}$
T_i	a source tree in a profile for any $i \in [N]$
$\mathcal{T}_S$	the set of trees with leaf set S
$\mathcal{T}_S^B$	the set of binary trees with leaf set S
$V(T)$	the vertex set of a tree T
$E(T)$	the edge set of a tree T
$L(T)$	the leaf set of a tree T
$N_T(v)$	the set of neighbors of a vertex v in a tree T
π_e	the bipartition of $L(T)$ induced by deleting an edge e from a tree T
$C(T)$	the set of all bipartitions induced by edges in a tree T, i.e., $\{\pi_e \mid e \in E(T)\}$
$[A\|B]$	a bipartition of the set $A \cup B$
$\pi\|_R$	the bipartition $\pi = [A\|B]$ restricted to $R \subseteq A \cup B$, which becomes $[A \cap R \| B \cap R]$
$T\|_R$	the subtree of a tree T induced on leaf set $R \subseteq L(T)$, with degree-two vertices suppressed

Notation for RFS and SFS

$\mathrm{RF}(T, T')$	the Robinson-Foulds (RF) distance between trees T and T' (not necessarily having the same leaf set), calculated by $\lvert C(T\|_X) \backslash C(T'\|_X)\rvert + \lvert C(T'\|_X) \backslash C(T\|_X)\rvert$, where X is the shared leaf set

$\mathrm{SF}(T, T')$ the split support of trees T and T' (not necessarily having the same leaf set), calculated by $|C(T|_X) \cap C(T'|_X)|$, where X is the shared leaf set

$\mathrm{RF}(T, \mathcal{A})$ the RFS score of tree T with respect to a profile $\mathcal{A} = \{T_i \mid i \in [N]\}$, calculated by $\sum_{i \in [N]} \mathrm{RF}(T, T_i)$

$\mathrm{SF}(T, \mathcal{A})$ the split support score of tree T with respect to a profile $\mathcal{A} = \{T_i \mid i \in [N]\}$, calculated by $\sum_{i \in [N]} \mathrm{SF}(T, T_i)$

Notation for Exact-RFS-2

T_1, T_2 two binary source trees

S_1, S_2 the leaf sets of T_1, T_2

X the shared leaf set of T_1 and T_2, i.e., $S_1 \cap S_2$

S the union of leaf sets of T_1 and T_2, i.e., $S_1 \cup S_2$

Π the set of all bipartitions of X such that both sides of the bipartition are non-empty, i.e., $2^X \backslash \{[\emptyset|X]\}$

$C(T_1, T_2, X)$ the set of bipartitions of X in the backbone trees of T_1 and T_2, i.e., $C(T_1|_X) \cup C(T_2|_X)$

$P(e)$ the path in T_i from which the backbone edge $e \in T_i|_X$ is obtained by suppressing degree-two vertices

$w(e)$ the weight of the backbone edge $e \in E(T_i|_X)$ defined as $w(e) = |P(e)|$

$e_i(\pi)$ the edge in T_i that induces the bipartition $\pi \in C(T_1, T_2, X)$

$w^*(\pi)$ the weight of the bipartition $\pi \in \Pi$, calculated by the sum of the weights of the edges in $T_i|_X$ that induce π, i.e., $w^*(\pi) = \sum_{i \in [2]} w(e_i(\pi))$

$T_i - T_i|_X$ the subgraph of T_i obtained by deleting all vertices and edges of the subgraph of T_i induced on X

$\mathrm{Extra}(T_i)$ the set of extra subtrees of T_i defined by $\{t \mid t \text{ is a component in } T_i - T_i|_X\}$

$r(t)$ the root of the extra subtree t, which is the unique vertex in $V(t)$ that is adjacent to a vertex in a backbone tree

$\mathcal{TR}(e)$ the set of extra subtrees attached to $e \in T_i|_X$, i.e., the set of extra subtrees whose roots are adjacent to the internal vertices of $P(e)$

$\mathcal{TR}^*(\pi)$ the set of extra subtrees that are attached to the edges in $T_i|_X$ that induce π, i.e., $\mathcal{TR}^*(\pi) = \bigcup_{i\in[2]} \mathcal{TR}(e_i(\pi))$

$\mathcal{BP}_i(Q)$ the set of bipartitions in $C(T_i|_X)$ with one side being a strict subset of Q, i.e., $\mathcal{BP}_i(Q) = \{[A|B] \in C(T_i|_X) \mid A \subsetneq Q \text{ or } B \subsetneq Q\}$

$\mathcal{BP}(Q)$ the set of bipartitions in $C(T_1, T_2, X)$ with one side being a strict subset of Q, i.e., $\mathcal{BP}(Q) = \mathcal{BP}_1(Q) \cup \mathcal{BP}_2(Q)$

$\mathcal{TRS}_i(Q)$ the set of extra subtrees attached to the edges in $T_i|_X$ that induce bipartitions in $\mathcal{BP}_i(Q)$, i.e., $\mathcal{TRS}_i(Q) = \bigcup_{\pi\in\mathcal{BP}_i(Q)} \mathcal{TR}(e_i(\pi))$

$\mathcal{TRS}(Q)$ the set of extra subtrees attached to the edges in $T_1|_X$ and $T_2|_X$ that induce bipartitions in $\mathcal{BP}(Q)$, i.e., $\bigcup_{\pi\in\mathcal{BP}(Q)} \mathcal{TR}^*(\pi)$, which is equivalent to $\mathcal{TRS}(Q) = \mathcal{TRS}_1(Q) \cup \mathcal{TRS}_2(Q)$

$\mathcal{C}$ the set of bipartitions from input trees, i.e., $C(T_1) \cup C(T_2)$

Π_X the set of bipartitions from input trees that are induced by the edges on the paths connecting vertices of X, i.e., $\Pi_X = \{[A|B] \in \mathcal{C} \mid A \cap X \neq \emptyset \text{ and } B \cap X \neq \emptyset\}$

Π_Y the set of bipartitions from input trees that are induced by the edges in an extra subtree or connecting an extra subtree to a backbone tree, i.e., $\Pi_Y = \{[A|B] \in \mathcal{C} \mid A \cap X = \emptyset \text{ or } B \cap X = \emptyset\}$

List of Figures

Chapter 1: Introduction

Motivation. Phylogenetic trees are graphical models of evolutionary relationships. Besides the immediate value of informing us about how the known species evolved on earth, phylogenies can be used in many ways. For example, phylogenies are helpful in identifying new species, have applications to clinical medicine [1], can be used to identify patterns of evolution of morphological and chemical characteristics [2], and serve to develop ecological and biogeographical conservation strategies [3, 4].

With increasingly fast and cheap sequencing technologies, there has been an explosion of large-scale genomic datasets, with now many hundreds of thousands of sequences available for many species, leading to the construction of very large species trees, and suggesting the possibility of estimating the Tree of Life. However, phylogeny estimation is computationally and statistically challenging on large datasets. Thus, a divide-and-conquer framework has become increasingly important to large-scale phylogeny estimation. The first step of the framework is to decompose the set of all species (and their associated sequence data) into overlapping subsets so that a phylogenetic tree can be estimated on each subset individually and potentially in parallel. Then a supertree method is used to combine the small trees on subsets of species into a supertree containing all species.

However, current leading supertree methods do not scale well to the large datasets they are designed for in the divide-and-conquer pipeline as most supertree methods use heuristics for **NP**-hard optimization problems [5–9]. Divide-and-conquer pipelines can be provably *statistically consistent* under stochastic models of evolution, provided that optimal supertrees are computed. However, current methods use local search heuristics for **NP**-hard optimization problems and so are not guaranteed to find optimal solutions. Thus, the resulting divide-and-conquer pipeline are not statistically consistent. In this context, to design a supertree method that has both good accuracy when used in a divide-and-conquer framework and guaranteed optimality for the statistical consistency of the pipeline can be of both theoretical and empirical interests.

Supertree problems are also interesting outside divide-and-conquer pipelines. For example, supertree methods can be used to compute species trees from a set of gene trees on overlapping subsets of species. Among the established supertree problems, we focus on the Robinson-Foulds Supertree (RFS) problem, which was first proposed in [10, 11]. RFS is a natural supertree problem as it tries to minimize the topological difference between the supertree and the input trees. RFS can also be seen as a heuristic for the Maximum Likelihood Supertree problem [12, 13], and hence has desirable properties. Unfortunately,

RFS is shown to be **NP**-hard when the number of input trees is not bounded [14] and any previous method either provides a heuristic [10, 11, 15, 16] or solves a constrained version of the problem [17]. It was not known before what the complexity of RFS is when the number of input trees is restricted to be a small constant, like two or three. This is the area in which we make progress.

Contributions. In this thesis, we prove that RFS can be solved exactly in polynomial time when given only two binary input trees and that RFS is **NP**-hard for three or more input trees. We show that the Split Fit Supertree problem (SFS), introduced in [18], has the same set of optimal solutions as RFS when the output is required to be binary (by default) but their optimal solutions are different when the output does not need to be binary.

Finally, we present GreedyRFS, a simple greedy heuristic for RFS that operates by applying our exact algorithm for two trees repeatedly until all the trees are merged together. Despite its simplicity, we show (using an experimental performance study on simulated datasets) that GreedyRFS provides improved accuracy over the leading RFS method when the number of input trees is small, which is the natural case for use within divide-and-conquer pipelines.

Structure of thesis. First we give some background information and related work of this thesis in Chapter 2. Chapter 2 also presents essential terminologies and formally define the RFS and SFS problems we study in this thesis. Our main theoretical results, including the polynomial time exact algorithm for RFS with two binary input trees and its proof of correctness, are mostly presented in Chapter 3 with some proofs appear in the appendices for the sake of better exposition. Chapter 3 also contains other theoretical results such as hardness results on RFS and SFS. Chapter 4 gives a brief description of the experiments we performed and their evaluation results. We talk about directions of future work and conclude the thesis in Chapter 5. The appendices contain some additional theorems, lemmas, and proofs needed for the theoretical results and additional details and results on the experiments.

Chapter 2: Background and Related Work

In this chapter, we give some background information for this thesis, including the divide-and-conquer pipeline used in large phylogeny estimation (see Section 2.1) and major supertree problems and their methods (see Section 2.2). We also talk about related work on the Robinson-Foulds Supertree problem and the Split Fit Supertree problem (see Section 2.3). In the end, we formalize the terminologies, the notation, and the problem statements in Section 2.4.

2.1 DIVIDE-AND-CONQUER PIPELINE

With the computational and statistical challenges of phylogeny estimation on large datasets, a divide-and-conquer framework that improves scalability and accuracy of tree estimation methods has become increasingly important. The divide-and-conquer framework takes a set of sequences or gene trees as input and estimates a gene tree or a species tree in three steps. It first decomposes the set of all species into (usually overlapping but sometimes disjoint) subsets of species. This step uses an estimated starting tree to make the decomposition balanced. Then a tree is estimated on each subset of species by a tree estimation method (called the base method) that is either computationally intensive or inaccurate on large datasets. The estimation of trees on the subsets of species can also be recursive, i.e., the divide-and-conquer framework is recursively applied to each subset. In the last step, the small trees on the subsets of species are merged into a large tree containing all species, either using a supertree method for overlapping subsets or using other information for disjoint subsets.

Methods that use this divide-and-conquer framework with overlapping subset decomposition, such that each subset of species has a small enough evolutionary diameter, are called disk-covering-methods (DCMs) [19–22]. Some DCMs have good statistical guarantees such as *statistical consistency* or even *absolute fast convergence* [19, 21]. Most DCMs developed so far are also shown to have good empirical performances. One such example is DACTAL, which improves the accuracy and scalability of tree estimation from unaligned sequences [22] and improves the accuracy of species tree estimation methods when used in the species tree estimation setting [23]. Another example is DCM-NJ [19], which improves the accuracy of Neighbor-Joining [24], a classical distance-based method that has large discrepancy of accuracy between large-diameter and small-diameter datasets. However, the last step of DCMs, which is the supertree construction, can be computationally intensive by itself and most

current supertree methods do not scale well to the large datasets they are designed for in the divide-and-conquer pipeline. (We will talk more about supertree methods in the next section.)

Divide-and-conquer framework can also be used with disjoint leaf set decomposition, thus avoiding the computationally intensive step of supertree construction. Some recent methods that adopted this approach include NJMerge [25], TreeMerge [26], and Constrained-INC [27].

2.2 SUPERTREE PROBLEMS AND METHODS

A supertree construction problem takes in a set of trees on overlapping leaf sets and tries to find a supertree on the union of the leaf sets optimizing some combinatorial or statistical criterion. Supertree methods originated from the need to combine trees already estimated in different research projects into a larger tree. Supertree methods can also be used in the context of species tree estimation if the input trees are genes trees on subsets of species. But most importantly, supertree construction plays a crucial role in divide-and-conquer pipelines that aim at improving the scalability and accuracy of classical phylogeny estimation methods on large datasets.

However, most current supertree methods either have unsatisfying accuracy or are too computationally intensive to scale to the large datasets they are designed for in the divide-and-conquer pipeline as they are heuristics for **NP**-hard optimization problems (e.g., Matrix Representation with Parsimony [5, 6], Matrix Representation with Likelihood [7], Mincut Supertree [8, 9], Quartets MaxCut [28], and Bad Clade Deletion [29]). Since most accurate supertree methods do not solve their corresponding optimization problems optimally, the resulting divide-and-conquer pipelines also do not have good statistical guarantees.

We briefly introduce a few popular supertree problem and their methods. Matrix Representation with Parsimony (MRP) [5, 6] is the current most popular supertree method and it has good accuracy. It creates a matrix representation for each input tree where each column corresponds to an internal edge and each row corresponds to a leaf. It then concatenates the matrices together to form a MRP matrix, in which each row of the matrix is seen as a label for the corresponding leaf. It then finds a supertree such that when the internal vertices of the supertree are labelled optimally, the total Hamming distance on the edges of the supertree is minimized (i.e., the parsimony of the supertree is maximized) among all possible supertrees. The MRP problem is **NP**-hard [30] and heuristics are used to find local optima. However, the search space of the problem grows exponentially with respect to the number of leaves, and thus heuristics can be enormously computationally intensive on large datasets. Another limitation of MRP is that it takes enormous space to explicitly store the

MRP matrix. Nonetheless, MRP is among the most accurate supertree methods and by far the most widely used.

Another method that uses the same matrix representation of the input is Matrix Representation with Likelihood (MRL) [7], which seeks a maximum likelihood supertree with respect to the concatenated input matrix by using RAxML [31], an accurate but computationally intensive maximum likelihood tree estimation method. The MRL method is proved to have an accuracy comparable to or better than that of MRP [7].

Other than using the matrix representation, we can also find a supertree that minimizes its topological difference to the input trees directly. The Robinson-Foulds Supertree problem does exactly that. A closely related and almost complementary problem is the Split Fit Supertree problem which maximizes the topological similarity between the supertree and the input trees. We will talk more about related work on these two problems in the next section.

There are numerous other supertree methods, including quartets-based methods [28, 32–35] and distance-based methods [36–38] . Quartets-based supertree methods usually need to compute all quartet trees of all input trees and thus have a running time of $\Omega(n^4)$ where n is the number of leaves appearing in any input tree. This running time makes quartets-based supertree methods infeasible to use on large datasets despite their good accuracy [35, 39]. Distance-based methods either find an additive matrix, defining the supertree, that has the minimum total distance to the additive matrices defining the input trees [36] or construct a single (not necessarily additive) matrix capturing the distance relationships between all pairs of species and then find an additive matrix as close to it as possible [38]. Since finding the closest additive matrix to a given matrix in terms of most of the natural distance metrics is **NP**-hard (e.g., L_1, L_2-norm [40], and L_∞-norm [41]), distance-based supertree methods use heuristics to find local optima and thus are either not as accurate as MRP or accurate but unscalable.

2.3 RELATED WORK

The Robinson-Foulds Supertree (RFS) problem is an extensively studied supertree problem [10, 11, 15–17]. Taking a set of trees on overlapping leaf sets as input, it finds a supertree that minimizes the sum of the Robinson-Foulds (RF) distances between the supertree and each of the input trees (we define the problem formally in the next section). Intuitively, the RF distance of two trees measures the topological differences between them and thus it is natural to minimize the total RF distance between the supertree and the input trees.

Unfortunately, the RFS problem with unbounded number of input trees is **NP**-hard,

as shown in [14]. Therefore, existing RFS methods usually use heuristics (e.g., Robinson-Foulds Supertrees [10] and URF [11], MulRF [16,42], and PluMiST [15]). These heuristics are mostly based on hill-climbing in tree search space defined by rearrangement operations such as nearest-neighbor interchange and subtree pruning and regrafting. On the other hand, a recently developed method, FastRFS [17], solves a constrained version of RFS exactly. When the supertree is only allowed to have bipartitions (which we define in the next section) from the input trees, the constrained version of RFS becomes polynomial-time solvable and FastRFS solves it exactly. FastRFS has been shown to have the best accuracy on large datasets among current RFS methods [17].

The closely related Split Fit Supertree (SFS) problem was introduced in [18], and is based on optimizing the similarity of topologies between the supertree and the input trees. However, no method for SFS is provided in a software form.

2.4 TERMINOLOGY AND PROBLEM STATEMENTS

Throughout this thesis we will make the following assumptions. A phylogenetic tree t is an unrooted tree (but not necessarily binary, so that internal vertices may have degree greater than 3) with leaves that are labelled by distinct elements of a set S of species. For any positive integer N, let $[N]$ denote $\{1, 2, \ldots, N\}$. We will let $\mathcal{A} = \{T_1, T_2, \ldots, T_N\}$ denote the input to a supertree problem, where each T_i for $i \in [N]$ is an unrooted phylogenetic tree on leaf set $L(T_i) = S_i \subseteq S$ (where $L(t)$ denotes the leaf set of t) and the output is a tree T where $L(T)$ is the set of all species that appear as a leaf in at least one tree in $\mathcal{A}$, which by default we will assume is all of S. We use the standard supertree terminology, and refer to the trees in $\mathcal{A}$ as "source trees" and the set $\mathcal{A}$ as a "profile". Let $\mathcal{T}_S$ denote the set of all phylogenetic trees such that $L(T) = S$ and $\mathcal{T}_S^B$ denote the binary trees in $\mathcal{T}_S$.

Robinson-Foulds Supertree. The Robinson-Foulds Supertree (RFS) of a set $\mathcal{A}$ of source trees is a supertree that minimizes the total Robinson-Foulds (RF) distance [43] to the source trees, which we now define. Each edge e in a tree T defines the bipartition $\pi_e := [A|B]$ of the leaf set, where A and B are the sets of leaves produced by deleting e (but not its endpoints) from T. Thus, each tree is defined by the set $C(T) := \{\pi_e \mid e \in E(T)\}$.

The *Robinson-Foulds distance* between two trees T, T' on the same leaf set is the number of bipartitions that are unique to one of the two trees; i.e., $\mathrm{RF}(T, T') := |C(T) \triangle C(T')| = |C(T) \backslash C(T')| + |C(T') \backslash C(T)|$. We extend the definition of RF distance to allow for trees to have different leaf sets as follows: $RF(T, T')$, where T and T' can have different leaf sets, is $RF(T|_X, T'|_X)$, where X is the shared leaf set and $t|_X$ denotes the homeomorphic subtree

of t induced by X (i.e., restricting t to just the vertices and edges on paths between leaves in X and then suppressing vertices of degree two). Note that if t is binary, then so is $t|_X$ for any subset X of leaves.

A Robinson-Foulds Supertree [10] of a profile $\mathcal{A}$, denoted $\mathrm{RFS}(\mathcal{A})$, is a binary tree T^* such that

$$T^* = \operatorname*{argmin}_{T \in \mathcal{T}_S^B} \sum_{i \in [N]} \mathrm{RF}(T, T_i). \tag{2.1}$$

We let $\mathrm{RF}(T, \mathcal{A}) := \sum_{i \in [N]} \mathrm{RF}(T, T_i)$ denote the RFS score of T with respect to profile $\mathcal{A}$. Thus, the **Robinson-Foulds Supertree (RFS) problem** takes as input the set $\mathcal{A}$ and seeks a Robinson-Foulds Supertree.

Split Fit Supertree. For two trees T, T' with the same leaf set, the *split support* is the number of shared bipartitions, i.e., $\mathrm{SF}(T, T') := |C(T) \cap C(T')|$. As with the RF distance, we can extend the definition of split support to trees T and T' that have different leaf sets by replacing the trees T and T' with $T|_X$ and $T'|_X$, respectively, and then computing the split support.

The Split Fit supertree for a profile $\mathcal{A}$ of source trees, denoted $\mathrm{SFS}(\mathcal{A})$, is a binary tree T^* such that

$$T^* = \operatorname*{argmax}_{T \in \mathcal{T}_S^B} \sum_{i \in [N]} \mathrm{SF}(T, T_i). \tag{2.2}$$

We denote the split support score of T with respect to $\mathcal{A}$ as $\mathrm{SF}(T, \mathcal{A}) := \sum_{i \in [N]} \mathrm{SF}(T, T_i)$. Thus, the **Split Fit Supertree (SFS) problem** takes as input the profile $\mathcal{A}$ and seeks a Split Fit supertree.

Variants of RFS and SFS. We consider variants of the two supertree problems in terms of the number of source trees, source tree types and output tree types:

- The relaxed versions of the problems, where we do not require the output to be binary, are named Relax-RFS and Relax-SFS.
- We append "-N" to the name to indicate that we assume there are N source trees. If no number is specified then the number of source trees is unconstrained.
- We append "-B" to the name to indicate that the source trees are required to be binary; hence, we indicate that the source trees are allowed to be non-binary by not appending -B.

For example, the RFS problem with two binary input trees is RFS-2-B and the relaxed SFS problem with three (not necessarily binary) input trees is Relax-SFS-3.

Other notation. For any tree T, we let $V(T)$ and $E(T)$ denote the vertices and edges of T, respectively. For any $v \in V(T)$, we let $N_T(v)$ denote the set of neighbors of v in T. A bipartition $[A|B]$ is non-trivial if $|A|, |B| \geq 2$. A tree T' is a *refinement* of T if T can be obtained from T' by contracting a set of edges. Equivalently, T' is a refinement of T if and only if $C(T) \subseteq C(T')$. A tree is *fully resolved* or *binary* if every non-leaf vertex has degree 3 (we will use fully resolved and binary interchangeably). Equivalently, a tree T with n leaves is fully resolved if and only if $C(T)$ contains $2n-3$ bipartitions, exactly $n-3$ of which are non-trivial. Two bipartitions π_1 and π_2 of the same leaf set are said to be *compatible* if and only if there exists a tree T such that $\pi_1, \pi_2 \in C(T)$. A bipartition $\pi = [A|B]$ restricted to a subset $R \subseteq A \cup B$ is $\pi|_R = [A \cap R | B \cap R]$. For a graph G and a set F of vertices or edges, we use $G + F$ and $G - F$ to represent the graph obtained from adding or deleting the set of vertices or edges to and from G, respectively.

Chapter 3: Theoretical Results

In this chapter, we present our theoretical results. Our main result (Theorem 3.1) establishes that computing the Robinson-Foulds (RF) and Split Fit (SF) supertrees of two binary source trees can be done in polynomial time. We present a polynomial time algorithm to solve RFS on two binary source trees in Section 3.1 and give the proof of correctness in Section 3.2. We also provide additional results on RFS and SFS including hardness results in Section 3.3. The next two results establish that RFS and SFS have the same set of optimal trees and that computing an Split Fit supertree (and thus a Robinson-Foulds supertree) for two binary trees is solvable in polynomial time.

Lemma 3.1. *Given an profile $\mathcal{A}$ of source trees, a tree $T \in \mathcal{T}_S^B$ is an optimal solution for* RFS*$(\mathcal{A})$ if and only if it is an optimal solution for* SFS*$(\mathcal{A})$.*

Theorem 3.1. *Let $\mathcal{A} = \{T_1, T_2\}$ with $L(T_i) = S_i$ $(i = 1, 2)$ and $X := S_1 \cap S_2$.* RFS-2-B*$(\mathcal{A})$ and* SFS-2-B*$(\mathcal{A})$ can be solved in $O(n^2|X|)$ time, where $n := \max\{|S_1|, |S_2|\}$.*

We first provide the proof of Lemma 3.1 before moving on to the algorithm and the proof of correctness for Theorem 3.1. Let $N \geq 2$ be any integer. Let $\mathcal{A} = \{T_1, T_2, \ldots, T_N\}$ and $S_1, S_2, \ldots, S_N$ be defined as from problem statement of RFS. Let $T \in \mathcal{T}_S^B$ be any binary tree of leaf set S. Then $T|_{S_i}$ is also binary and thus $|C(T|_{S_i})| = 2|S_i| - 3$. For any $i \in [N]$, we have

$$\begin{aligned}
&\mathrm{RF}(T, T_i) + 2\mathrm{SF}(T, T_i) && (3.1)\\
=&|C(T|_{S_i}) \backslash C(T_i)| + |C(T_i) \backslash C(T|_{S_i})| + 2|C(T|_{S_i}) \cap C(T_i)| && (3.2)\\
=&|C(T|_{S_i}) \backslash C(T_i) \cup (C(T|_{S_i}) \cap C(T_i))| + |C(T_i) \backslash C(T|_{S_i}) \cup (C(T|_{S_i}) \cap C(T_i))| && (3.3)\\
=&|C(T|_{S_i})| + |C(T_i)| && (3.4)\\
=&2|S_i| - 3 + |C(T_i)|. && (3.5)
\end{aligned}$$

Taking the sum of the equations over all $i \in [N]$, we have

$$\mathrm{RF}(T, \mathcal{A}) + 2\mathrm{SF}(T, \mathcal{A}) = \sum_{i \in [N]} (\mathrm{RF}(T|_{S_i}, T_i) + 2\mathrm{SF}(T|_{S_i}, T_i)) = \sum_{i \in [N]} (2|S_i| - 3 + |C(T_i)|), \tag{3.6}$$

which is a constant for any fixed $\mathcal{A}$. Therefore, for any binary tree T and any profile $\mathcal{A}$ of source trees, the sum of T's RFS score and twice T's split support score is the same,

independent of T. This implies that minimizing the RF score is the same as maximizing the split support score. Although this argument depends on the output tree being binary, it does not depend on the input trees being binary. Hence, we conclude that RFS and SFS have the same set of optimal supertrees.

3.1 EXACT-RFS-2: POLYNOMIAL TIME ALGORITHM FOR RFS-2-B

We present Exact-RFS-2, the algorithm for SFS-2-B, which (by Lemma 3.1) is also an algorithm for RFS-2-B. Exact-RFS-2 is described in detail as Algorithm 3.1 and is illustrated in Figure 3.1 to 3.3. Furthermore, Exact-RFS-2 is the basis for GreedyRFS, a greedy heuristic we develop and evaluate in Chapter 4.

The input to Exact-RFS-2 is a pair of binary trees T_1 and T_2 and we define $X = S_1 \cap S_2$ to be their shared leaf set. At a high level, Exact-RFS-2 constructs a tree T_{init} that has a central vertex that is adjacent to every leaf in X and to the root of every "rooted extra subtree" (which we define below) such that T_{init} contains all taxa in $S = S_1 \cup S_2$. It then modifies T_{init} by repeatedly refining it to add specific desired bipartitions, so as to produce an optimal Split Fit (and also optimal Robinson-Foulds) supertree. The bipartitions that are added are defined by a maximum independent set in a bipartite "weighted incompatibility graph" we compute.

3.1.1 Additional Notation

To explain the algorithmic ideas, we need additional notation. Let $\Pi = 2^X \backslash \{[\emptyset|X]\}$ denote the set of all possible bipartitions of X such that both sides of the bipartition are non-empty. Let $C(T_1, T_2, X)$ denote $C(T_1|_X) \cup C(T_2|_X)$, and let Triv and NonTriv denote the sets of trivial and non-trivial bipartitions in $C(T_1, T_2, X)$, respectively. We will refer to $T_i|_X, i = 1, 2$ as **backbone trees**. In the rest of the paper, we use $i \in [2]$ to index input trees.

Weights of edges of the backbone trees and weights of bipartitions of X. For any backbone edge $e \in E(T_i|_X)$, let the **path of** e, denoted $P(e)$, be the path in the input tree T_i from which e is obtained by suppressing degree-two vertices. We define the **weight of a backbone edge** with the function $w : E(T_1|_X) \cup E(T_2|_X) \to \mathbb{N}_{\geq 0}$ such that $w(e) := |P(e)|$ for any $e \in E(T_i|_X)$. Thus, $w(e)$ is the number of edges on the path in T_i from which e is obtained by suppressing degree two vertices. For $\pi \in C(T_i|_X)$, we define $e_i(\pi)$ to be the edge that induces π in $T_i|_X$. We define the **weight of a bipartition of** X using the function

$w^* : \Pi \to \mathbb{N}_{\geq 0}$ as follows:

$$w^*(\pi) = \begin{cases} 0 & \text{if } \pi \in \Pi \backslash C(T_1, T_2, X), \\ w(e_1(\pi)) & \text{if } \pi \in C(T_1|_X) \setminus C(T_2|_X), \\ w(e_2(\pi)) & \text{if } \pi \in C(T_2|_X) \setminus C(T_1|_X), \\ \sum_{i \in [2]} w(e_i(\pi)) & \text{else.} \end{cases} \tag{3.7}$$

For any set F of bipartitions, we extend the definition so that $w^*(F) = \sum_{\pi \in F} w^*(\pi)$.

Extra subtrees for edges and bipartitions of the backbone trees. The next concept we introduce is the set of **extra subtrees**, which are rooted subtrees of T_1 and T_2, formed by deleting X and all the edges and vertices on the paths between vertices in X (i.e., we delete $T_i|_X$ from T_i). Each component in $T_i - T_i|_X$ is called an **extra subtree** of T_i, and note that the extra subtree t is naturally seen as rooted at the unique vertex $r(t)$ that is adjacent to a vertex in $T_i|_X$. Thus,

$$\text{Extra}(T_i) := \{t \mid t \text{ is a component in } T_i - T_i|_X\}. \tag{3.8}$$

We say that an extra subtree t is **attached to an edge** $e \in E(T_i|_X)$ if the root of t is adjacent to an internal vertex of $P(e)$, and we let $\mathcal{TR}(e)$ denote the set of such extra subtrees attached to edge e. That is,

$$\mathcal{TR}(e) := \{t \mid \exists v \in P(e) \text{ s.t. } r(t) \text{ is adjacent to } v\}. \tag{3.9}$$

Similarly, if $\pi \in C(T_1, T_2, X)$, we let $\mathcal{TR}^*(\pi)$ refer to the set of extra subtrees that attach to backbone edges $e_1(\pi)$ and $e_2(\pi)$ that induce π in $T_1|_X$ and $T_2|_X$, i.e.,

$$\mathcal{TR}^*(\pi) := \bigcup_{i \in [2]} \mathcal{TR}(e_i(\pi)). \tag{3.10}$$

Bipartitions and extra subtrees for subsets of X. For any $Q \subseteq X$, we let $\mathcal{BP}_i(Q)$ denote the set of bipartitions in $C(T_i|_X)$ that have one side being a strict subset of Q, and we let $\mathcal{TRS}_i(Q)$ denote the set of extra subtrees associated with these bipartitions. In other words,

$$\mathcal{BP}_i(Q) := \{[A|B] \in C(T_i|_X) \mid A \subsetneq Q \text{ or } B \subsetneq Q\}, \tag{3.11}$$

$$\mathcal{TRS}_i(Q) := \bigcup_{\pi \in \mathcal{BP}_i(Q)} \mathcal{TR}(e_i(\pi)). \tag{3.12}$$

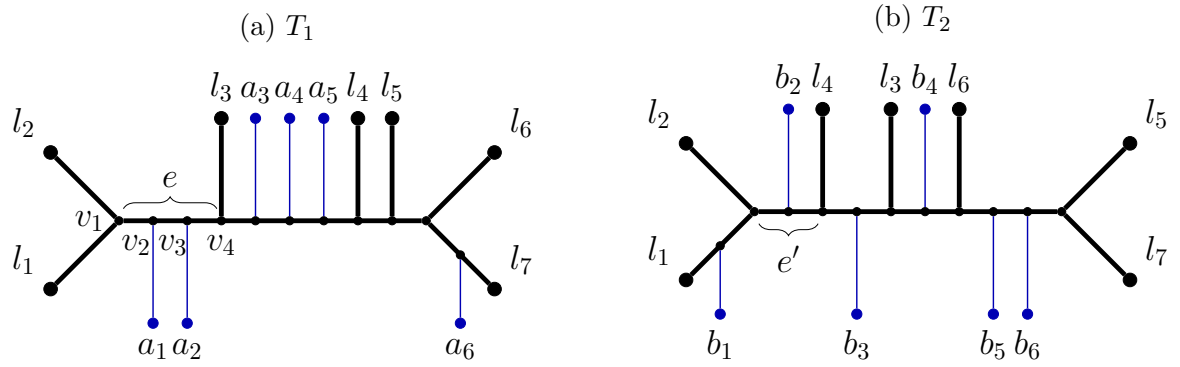


Figure 3.1: T_1 and T_2 depicted in (a) and (b) have an overlapping leaf set $X = \{l_1, l_2, \ldots, l_7\}$. Each of $a_1, \ldots, a_6$ and $b_1, \ldots, b_6$ can represent a multi-leaf extra subtree. Then, $\text{Extra}(T_1) = \{a_1, \ldots, a_6\}$ and $\text{Extra}(T_2) = \{b_1, \ldots, b_6\}$. Let $\pi = [l_1, l_2 \mid l_3, l_4, l_5, l_6, l_7]$. Then $e_1(\pi) = e$ and $e_2(\pi) = e'$. $P(e)$ is the path from v_1 to v_4, so $w(e) = 3$. Similarly, $w(e') = 2$ and thus $w^*(\pi) = 5$. $\mathcal{TR}(e) = \{a_1, a_2\}$ and $\mathcal{TR}(e') = \{b_2\}$, so $\mathcal{TR}^*(\pi) = \{a_1, a_2, b_2\}$. Let $A = \{l_1, l_2, l_3\}$, $B = \{l_4, l_5, l_6, l_7\}$. Ignoring the trivial bipartitions, we have $\mathcal{BP}_1(A) = \mathcal{BP}_2(A) = \mathcal{BP}(A) = \{\pi\}$. Let $\pi_{567} = [l_1, l_2, l_3, l_4 \mid l_5, l_6, l_7]$, $\pi_{67} = [l_1, l_2, l_3, l_4, l_5 \mid l_6, l_7]$, and $\pi_{57} = [l_1, l_2, l_3, l_4, l_6 \mid l_5, l_7]$. Then $\mathcal{BP}_1(B) = \{\pi_{567}, \pi_{67}\}$ and $\mathcal{BP}_2(B) = \{\pi_{567}, \pi_{57}\}$. Thus, $\mathcal{BP}(B) = \{\pi_{567}, \pi_{67}, \pi_{57}\}$. We have $\mathcal{TRS}_1(A) = \{a_1, a_2\}$ and $\mathcal{TRS}_2(A) = \{b_2\}$, so $\mathcal{TRS}(A) = \{a_1, a_2, b_2\}$. We also have $\mathcal{TRS}_1(B) = \emptyset$, so $\mathcal{TRS}_2(B) = \mathcal{TRS}(B) = \{b_4, b_5, b_6\}$.

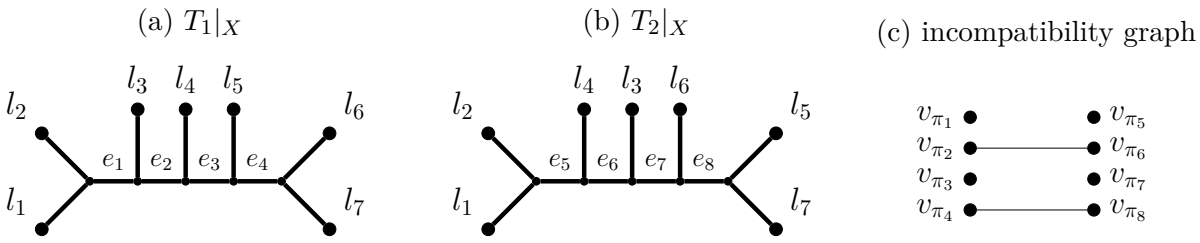


Figure 3.2: We show (a) $T_1|_X$, (b) $T_2|_X$, and (c) their incompatibility graph (without isolated vertices that represent trivial bipartitions), based on the trees T_1 and T_2 in Figure 3.1. Each π_i is the bipartition induced by e_i, and the weights for $v_{\pi_1}, \ldots, v_{\pi_8}$ are $3, 4, 1, 1, 2, 2, 2, 3$, in that order. We note that π_1 and π_5 are the same bipartition, but v_{π_1} and v_{π_5} have different weights as π_1 and π_5 are induced by different edges; similarly for π_3 and π_7. The maximum weight independent set in this graph has all the isolated vertices $(v_{\pi_1}, v_{\pi_3}, v_{\pi_5}, v_{\pi_7})$ and also v_{π_2} and v_{π_8}, achieving a total weight of 15.

Intuitively, $\mathcal{BP}_i(Q)$ denotes the set of bipartitions in $C(T_i|_X)$ that are compatible with the bipartition $[Q|X \backslash Q]$ and are "on the side of Q". By Corollary A.1 (see Appendix A.1), for any $\pi = [A|B] \in C(T_i|_X)$, $\mathcal{BP}_i(A) \cup \mathcal{BP}_i(B)$ is the set of bipartitions in $C(T_i|_X)$ that are compatible with π. Similarly, the intuition behind $\mathcal{TRS}_i(Q)$ is that it represents the set of extra subtrees of T_i that are "on the side of Q", i.e., the set of extra subtrees of T_i attached

to the edges of the subgraph of T_i defined by the bipartitions in $\mathcal{BP}_i(Q)$. Finally, let

$$\mathcal{BP}(Q) = \mathcal{BP}_1(Q) \cup \mathcal{BP}_2(Q), \tag{3.13}$$

$$\mathcal{TRS}(Q) = \mathcal{TRS}_1(Q) \cup \mathcal{TRS}_2(Q). \tag{3.14}$$

We give an example for these terms in Figure 3.1.

Weighted incompatibility graph. The *incompatibility graph* of a set of trees, each on the same set of leaves, has one vertex for each bipartition in any tree (and note that bipartitions can appear more than once) and one edge between each pair of bipartitions if they are incompatible (see [44]). We compute a **weighted incompatibility graph** for the pair of trees $T_1|_X$ and $T_2|_X$, in which the weight of the vertex corresponding to bipartition π appearing in tree $T_i|_X$ is $w(e_i(\pi))$, as defined previously. Formally, the weighted incompatibility graph for $T_1|_X$ and $T_2|_X$ is a graph $G = (V_1 \cup V_2, E)$, such that $V_i = \{v_\pi \mid \pi \in C(T_i|_X)\}$ for $i = 1, 2$, and $E = \{(v_\pi, v_{\pi'}) \mid \pi \text{ is incompatible with } \pi'\}$. For each $v_\pi \in V_i$, $\text{weight}(v_\pi) = w(e_i(\pi))$. Thus, if a bipartition is common to the two trees, it produces two vertices in the weighted incompatibility graph, and each vertex has its own weight (see Figure 3.2). We note that the incompatibility graph of k trees is k-partite, since the set of bipartitions for any given tree is by definition compatible.

Two kinds of bipartitions from input trees. We split the set of bipartitions $\mathcal{C} = C(T_1) \cup C(T_2)$ into two sets:

$$\Pi_X = \{[A|B] \in \mathcal{C} \mid A \cap X \neq \emptyset \text{ and } B \cap X \neq \emptyset\}, \tag{3.15}$$

$$\Pi_Y = \{[A|B] \in \mathcal{C} \mid A \cap X = \emptyset \text{ or } B \cap X = \emptyset\}. \tag{3.16}$$

We notice that Π_X is the set of bipartitions of S_1 or S_2 that are induced by edges on the paths between vertices of X, and Π_Y are all the other input tree bipartitions (i.e., Π_Y is the set of bipartitions of S_1 or S_2 induced by edges inside extra subtrees or connecting extra subtrees to the backbone trees).

Omitting the parameters T_1 and T_2 for brevity, we define $p_X(\cdot)$ and $p_Y(\cdot)$ on any tree $T \in \mathcal{T}_S$ by:

$$p_X(T) = \sum_{i \in [2]} |C(T|_{S_i}) \cap C(T_i) \cap \Pi_X|, \tag{3.17}$$

$$p_Y(T) = \sum_{i \in [2]} |C(T|_{S_i}) \cap C(T_i) \cap \Pi_Y|. \tag{3.18}$$

Observation 3.1. *Note that $p_X(T)$ and $p_Y(T)$ decompose the split support score of T into the score contributed by bipartitions in Π_X and the score contributed by bipartitions in Π_Y; thus, the split support score of T with respect to T_1, T_2 is $p_X(T) + p_Y(T)$.*

As we will show, the two scores can be maximized sequentially without interference and we can use this observation to refine T_{init} so that it achieves the optimal total score.

3.1.2 Overview of Exact-RFS-2.

Exact-RFS-2 (Algorithm 3.1) builds a tree $T \in \mathcal{T}_S^B$ that maximizes its split support score in four phases. In the pre-processing phase (lines 1 – 5), it computes the weight function w and the mappings $\mathcal{TR}, \mathcal{TR}^*, \mathcal{BP}$, and $\mathcal{TRS}$ for use in latter parts of the algorithm. In the initial construction phase (line 6), it constructs a supertree T_{init}, which maximizes the $p_Y(\cdot)$ score. In the refinement phase (lines 7 – 13), it refines T_{init} so that it attains the maximum $p_X(\cdot)$ score. In the last phase (line 14), it arbitrarily refines the supertree to make it binary.

The refinement phase of Algorithm 3.1 has several parts, and begins with the construction

Algorithm 3.1 Exact-RFS-2: Computing a Robinson-Foulds supertree of two trees

Input: two binary trees T_1, T_2 with leaf sets S_1 and S_2 where $S_1 \cap S_2 = X \neq \emptyset$
Output: a binary supertree T on leaf set $S = S_1 \cup S_2$ that maximizes the split support score

1: compute $C(T_1|_X)$ and $C(T_2|_X)$
2: **for** each $\pi = [A|B] \in C(T_1, T_2, X)$ **do**
3: **for** $i \in [2]$ **do**
4: compute $\mathcal{TR}(e_i(\pi))$, $w(e_i(\pi))$, $\mathcal{BP}_i(A)$, $\mathcal{BP}_i(B)$, $\mathcal{TRS}_i(A)$, $\mathcal{TRS}_i(B)$
5: compute $\mathcal{TR}^*(\pi)$, $\mathcal{BP}(A)$, $\mathcal{BP}(B)$, $\mathcal{TRS}(A)$, and $\mathcal{TRS}(B)$
6: construct T as a star tree with leaf set X and center vertex $\hat{v}$ and with the root of each $t \in \text{Extra}(T_1) \cup \text{Extra}(T_2)$ connected to $\hat{v}$ by an edge ▷ let $T_{\text{init}} = T$
7: construct the weighted incompatibility graph G of $T_1|_X$ and $T_2|_X$
8: compute the maximum weight independent set I^* in G and let $I = \{\pi \mid v_\pi \in I^*\}$
9: **for** each $\pi = [\{a\}|B] \in \text{Triv}$ **do**
10: detach all extra subtrees in $\mathcal{TR}^*(\pi)$ from $\hat{v}$ and attach them onto $(\hat{v}, a)$ such that the subtrees from $\mathcal{TR}(e_1(\pi))$ and subtrees from $\mathcal{TR}(e_2(\pi))$ are side by side and the ordering of the attachments of $\mathcal{TR}(e_i(\pi))$ match the ordering of the attachments on $e_i(\pi)$ exactly ▷ let $\tilde{T} = T$ after for loop
11: $H(\hat{v}) = \text{NonTriv}$, set $sv(\pi) = \hat{v}$ for all $\pi \in \text{NonTriv}$
12: **for** each $\pi \in \text{NonTriv} \cap I$ **do**
13: $T \leftarrow \text{Refine}(T, \pi, H, sv)$ ▷ let $T^* = T$ after for loop
14: arbitrarily refine T to make it a binary tree
15: return T

of the weighted incompatibility graph G of $T_1|_X$ and $T_2|_X$ (line 7, see Figure 3.2). It then finds a maximum weight independent set of G that defines a set $I \subseteq C(T_1, T_2, X)$ of compatible bipartitions of X (line 8). Finally, it uses these bipartitions of X in I to refine T_{init} to achieve the optimal $p_X(\cdot)$ score (line 9 – 13). Algorithm 3.1 handles the trivial and non-trivial bipartitions in I differently. For any bipartition $\pi \in \text{Triv} \cap I$ (note that $\text{Triv} \cap I = \text{Triv}$), we know $\pi \in C(T_{\text{init}}|_X)$. Thus, the algorithm attaches all extra subtrees in $\mathcal{TR}^*(\pi)$ onto the edge that induces π in $T_{\text{init}}|_X$ (line 9 – 10), so as to add the desired bipartitions, which become π when restricted to X, to the supertree. For any $\pi \in \text{NonTriv} \cap I$, Algorithm 3.1 invokes Algorithm 3.2 on the supertree and π (line 12 – 13). We note that the order in which the bipartitions in I are handled does not matter. See Figure 3.3 for an example of Exact-RFS-2 given two input source trees.

Algorithm 3.2 Refine

Input: a tree T on leaf set S, a bipartition $\pi = [A|B] \in \text{NonTriv}$, two mappings H and sv
Output: a refinement T' of T such that $C(T'|_{S_i}) = C(T|_{S_i}) \cup \{\pi' \in C(T_i) \mid \pi'|_X = \pi\}$ for both $i = 1, 2$

1: $v \leftarrow sv(\pi)$
2: $T' \leftarrow T + v_a + v_b + (v_a, v_b)$
3: compute $N_A := \{u \in N_T(v) \mid \exists a \in A \text{ s.t. } u \text{ can reach } a \text{ in } T - v\}$ and $N_B := \{u \in N_T(v) \mid \exists b \in B \text{ s.t. } u \text{ can reach } b \text{ in } T - v\}$.
4: **for** each $u \in N_A \cup N_B$ **do**
5: **if** $u \in N_A$ **then** connect u to v_a
6: **else** connect u to v_b
7: detach all extra subtrees in $\mathcal{TR}^*(\pi)$ from v and attach them onto (v_a, v_b) such that the subtrees from $\mathcal{TR}(e_1(\pi))$ and subtrees from $\mathcal{TR}(e_2(\pi))$ are side by side and the ordering of the attachments of $\mathcal{TR}(e_i(\pi))$ match the ordering of their attachments on $e_i(\pi)$ exactly
8: **for** each $t \in \mathcal{TRS}(A)$ **do**
9: if t is attached to v, detach it and attach to v_a
10: **for** each $t \in \mathcal{TRS}(B)$ **do**
11: if t is attached to v, detach it and attach to v_b
12: **for** each remaining extra subtree attached to v **do**
13: detach it from v and attach it to either v_a or v_b
14: $H(v_a) \leftarrow \emptyset, H(v_b) \leftarrow \emptyset$
15: **for** each $\pi' \in H(v)$ **do**
16: **if** $\pi' \in \mathcal{BP}(A)$ **then**
17: $sv(\pi') = v_a$, $H(v_a) \leftarrow H(v_a) \cup \{\pi'\}$
18: **else if** $\pi' \in \mathcal{BP}(B)$ **then**
19: $sv(\pi') = v_b$, $H(v_b) \leftarrow H(v_b) \cup \{\pi'\}$
20: return $T' = T' - v$

Algorithm 3.2 refines the given supertree T on leaf set S with a bipartition of X from $C(T_1, T_2, X)$. Given a bipartition $\pi = [A|B]$ of X, Algorithm 3.2 produces a refinement T' of T such that $T'|S_i$ contains, in addition to all bipartitions in $C(T|_{S_i})$, all bipartitions of T_i that become π when restricted to X. That is, the refinement T' satisfy that $C(T'|_{S_i}) = C(T|_{S_i}) \cup \{\pi' \in C(T_i) \mid \pi'|_X = \pi\}$ for both $i = 1, 2$.

To do this, we first find the unique vertex v such that no component of $T - v$ has leaves from both A and B. We create two new vertices v_a and v_b with an edge between them. We divide the neighbor set of v into three sets: N_A is the set of neighbors that split v from leaves in A, N_B is the set of neighbors that split v from leaves in B, and N_{other} contains the remaining neighbors. Then, we make all the vertices in N_A adjacent to v_a and all the vertices in N_B adjacent to v_b.

We note that $N_{\text{other}} = \emptyset$ if $X = S$ and thus there are no extra subtrees. In the case where $X \neq S$, N_{other} contains the roots of the extra subtrees adjacent to v and we handle them in the following four different cases in order to make T' include the desired bipartitions:

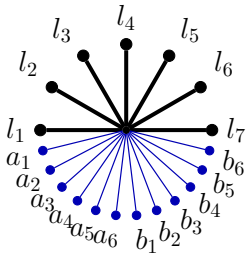
- those vertices that root extra subtrees in $\mathcal{TR}^*(\pi)$ are moved onto the edge (v_a, v_b) (by subdividing the edge to create new vertices, and then making these vertices adjacent to the new vertices) in an order that induces all the desired bipartitions
- those vertices that root extra subtrees in $\mathcal{TRS}(A)$ are made adjacent to v_a
- those that root extra subtrees in $\mathcal{TRS}(B)$ are made adjacent to v_b
- the remaining vertices can be made adjacent to either v_a or v_b, and the choice does not impact the split support score

Note that to perform these modifications, we assume that the weight function w and the mappings $\mathcal{TR}, \mathcal{TR}^*, \mathcal{BP}$, and $\mathcal{TRS}$ computed in the pre-processing stage of Algorithm 3.1 are accessible to Algorithm 3.2. Algorithms 3.1 and 3.2 also use two data structures H and sv, which can also be thought of as functions or mappings. For a given vertex $v \in V(T)$, $H(v) \subseteq C(T_1, T_2, X)$ is the set of bipartitions of X that can be added to $T|_X$ by a refinement at v. Given $\pi \in C(T_1, T_2, X)$, $sv(\pi) = v$ means that there exists a refinement T' of T at v, so that $C(T'|_X) = C(T|_X) \cup \{\pi\}$.

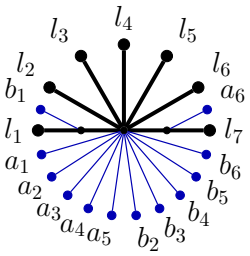
3.2 PROOF OF CORRECTNESS FOR EXACT-RFS-2

In this section, we prove Theorem 3.1, i.e., Exact-RFS-2 solves RFS and SFS correctly. We first present a lemma that upperbounds the $p_Y(\cdot)$ score for any $T \in \mathcal{T}_S$ and show that the tree T_{init} we build in line 6 of Exact-RFS-2 (Algorithm 3.1) maximizes the $p_Y(\cdot)$ score.

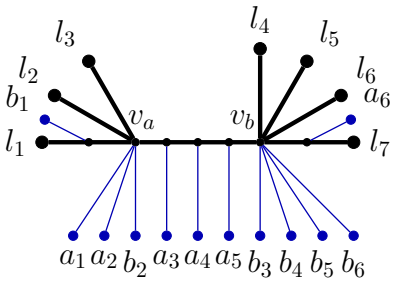
(a) T_{init}: star with leaf set X and all extra subtrees attached to center



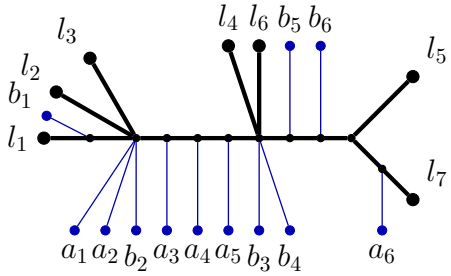
(b) $\tilde{T}$: after adding all Triv to $T|_X$



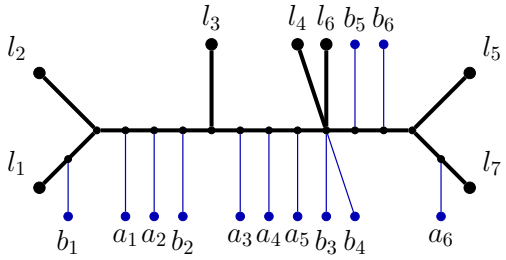
(c) After adding $\pi_2 = [123|4567]$



(d) After adding $\pi_8 = [12346|57]$



(e) After adding $\pi_1 = \pi_5 = [12|34567]$



(f) After adding $\pi_3 = \pi_7 = [1234|567]$

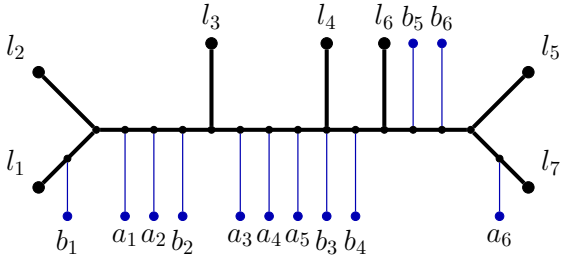


Figure 3.3: Algorithm 3.1 working on T_1 and T_2 from Figure 3.1 as source trees. $X = \{l_1, l_2, \ldots, l_7\}$ are the shared leaves and the notation of $\pi_1, \ldots, \pi_8$ is from Figure 3.2. In (a) to (f), the $p_X(\cdot)$ score of the trees are $14, 16, 20, 23, 27, 29$, in that order. We explain how the algorithm obtain the tree in (g) from $\tilde{T}$ by adding $\pi_2 = [123|4567]$ to the backbone of $\tilde{T}$. Let $A = \{l_1, l_2, l_3\}$ and $B = \{l_4, l_5, l_6, l_7\}$. The center vertex c of $\tilde{T}$ is split into two vertices v_a, v_b with an edge between them. Then all neighbors of c between c and A are made adjacent to v_a while the neighbors between c and B are made adjacent to v_b. All neighbors of c which are roots of extra subtrees are moved around such that all extra subtrees in $\mathcal{TR}^*(\pi_2)$ are attached onto (v_a, v_b); all extra subtrees in $\mathcal{TRS}(A) = \{a_1, a_2, b_2\}$ are attached to v_a and all extra subtrees in $\mathcal{TRS}(B) = \{b_4, b_5, b_6\}$ are attached to v_b. We note that in this step, b_3 can attach to either v_a or v_b because it is not in $\mathcal{TRS}(A)$ or $\mathcal{TRS}(B)$. However, when obtaining the tree in (d) from the tree in (g), b_3 can only attach to the left side because for $A' = \{l_1, l_2, l_3, l_4, l_6\}$, $[l_1, l_2, l_4|l_3, l_5, l_6, l_7] \in \mathcal{BP}(A')$ and thus $b_3 \in \mathcal{TRS}(A')$.

Lemma 3.2. *For any tree $T \in \mathcal{T}_S$, $p_Y(T) \le |\Pi_Y|$. In particular, let T_{init} be the tree defined in line 6 of Algorithm 3.1. Then, $p_Y(T_{\text{init}}) = |\Pi_Y|$.*

Since T_1 and T_2 have different leaf sets, $C(T_1)$ and $C(T_2)$ are disjoint. Since $\Pi_Y \subseteq C(T_1) \cup C(T_2)$, $C(T_1) \cap \Pi_Y$ and $C(T_2) \cap \Pi_Y$ form a disjoint decomposition of Π_Y. By definition of $p_Y(\cdot)$, for any tree T of leaf set S,

$$p_Y(T) = \sum_{i\in[2]} |C(T|_{S_i}) \cap C(T_i) \cap \Pi_Y| \le \sum_{i\in[2]} |C(T_i) \cap \Pi_Y| = |\Pi_Y|. \tag{3.19}$$

Fix any $\pi = [A|B] \in \Pi_Y$. Suppose $\pi \in C(T_i)$ and is induced by $e \in E(T_i)$ for some $i \in [2]$. By definition of Π_Y, either $A \cap X = \emptyset$ or $B \cap X = \emptyset$. By Lemma A.1, $e \notin P(e')$ for any backbone edge $e' \in E(T_i|_X)$. Therefore, either e is an internal edge in an extra subtree in $\text{Extra}(T_i)$, or e connects one extra subtree in $\text{Extra}(T_i)$ to the backbone tree. In either case, the construction of T_{init} ensures that e is also present in $T_{\text{init}}|_{S_i}$ and thus $\pi \in C(T_{\text{init}}|_{S_i})$. Therefore, each bipartition $\pi \in \Pi_Y$ contributes 1 to $|C(T_{\text{init}}|_{S_i}) \cap C(T_i) \cap \Pi_Y|$ for exactly one index $i \in [2]$ and thus it contributes 1 to $p_Y(T_{\text{init}})$. Hence, $p_Y(T_{\text{init}}) = |\Pi_Y|$.

The next lemma shows that for any $\pi \in \Pi$ (not necessarily in $C(T_1, T_2, X)$), $w^*(\pi)$ represents the maximum potential increase in $p_X(\cdot)$ as a result of adding the bipartition π to $T|_X$. The proof of Lemma 3.3 follows the idea that for any bipartition π of X, there are at most $w^*(\pi)$ edges in either T_1 or T_2 whose induced bipartitions become π when restricted to X. Therefore, by only adding π to $T|_X$, at most $w^*(\pi)$ bipartitions in Π_X get included in $C(T'|_{S_1})$ or $C(T'|_{S_2})$ so that the $p_X(\cdot)$ score is increased by at most $w^*(\pi)$.

Lemma 3.3. *Let $\pi = [A|B] \in \Pi$. Let $T \in \mathcal{T}_S$ such that $\pi \notin C(T|_X)$ but π is compatible with $C(T|_X)$. Let T' be a refinement of T such that for all $\pi' \in C(T'|_{S_i}) \backslash C(T|_{S_i})$ for some $i \in [2]$, $\pi'|_X = \pi$. Then, $p_X(T') - p_X(T) \le w^*(\pi)$.*

By definition of $p_X(\cdot)$,

$$p_X(T') - p_X(T) = \sum_{i\in[2]} |C(T'|_{S_i}) \cap C(T_i) \cap \Pi_X| - \sum_{i\in[2]} |C(T|_{S_i}) \cap C(T_i) \cap \Pi_X| \tag{3.20}$$

$$= \sum_{i\in[2]} |(C(T'|_{S_i}) \backslash C(T|_{S_i})) \cap C(T_i) \cap \Pi_X|. \tag{3.21}$$

Therefore, we only need to prove that

$$\sum_{i\in[2]} |(C(T'|_{S_i}) \backslash C(T|_{S_i})) \cap C(T_i) \cap \Pi_X| \le w^*(\pi). \tag{3.22}$$

We perform a case analysis, as follows: Case (1): $\pi \notin C(T_1, T_2, X)$, Case (2): $\pi \in C(T_1|_X)\Delta C(T_2|_X)$, and Case (3): $\pi \in C(T_1|_X) \cap C(T_2|_X)$.

Case 1): Let $\pi \notin C(T_1, T_2, X)$. We recall that $w^*(\pi) = 0$. Assume for contradiction that there exists a bipartition $\pi' \in (C(T'|_{S_i})\backslash C(T|_{S_i})) \cap C(T_i) \cap \Pi_X$ for some $i \in [2]$. Since $\pi \notin C(T_1, T_2, X)$ and $\pi'|_X = \pi$, by Corollary A.2, $\pi' \notin C(T_i)$ for any $i \in [2]$. This contradicts the fact that $\pi' \in C(T_i)$ for some $i \in [2]$. Therefore, the assumption that there exists such a bipartition π' is wrong and $\sum_{i\in[2]} |(C(T'|_{S_i})\backslash C(T|_{S_i})) \cap C(T_i) \cap \Pi_X| = 0 \leq w^*(\pi)$.

Case 2): Let $\pi \in C(T_1|_X)\Delta C(T_2|_X)$. We can assume without loss of generality that $\pi \in C(T_1|_X)\backslash\ C(T_2|_X)$ since the other possibility is symmetrical. Then, we have $w^*(\pi) = w(e_1(\pi))$. Let $\pi' \in (C(T'|_{S_i})\backslash C(T|_{S_i})) \cap C(T_i) \cap \Pi_X$ for some $i \in [2]$. Then we have $\pi'|_X = \pi$ by assumption of the lemma. Since $\pi \notin C(T_2|_X)$, by Corollary A.2, we have $\pi' \notin C(T_2)$ and thus $\pi' \in C(T_1)$. By Lemma A.1, the edge which induces π' in T_1 is an edge on $P(e_1(\pi))$. Since there are $w(e_1(\pi))$ edges on $P(e_1(\pi))$, there are at most $w(e_1(\pi))$ distinct bipartitions π', proving the claim.

Case 3): Let $\pi \in C(T_1|_X) \cap C(T_2|_X)$. Then we have $w^*(\pi) = w(e_1(\pi)) + w(e_2(\pi))$. Fix any $\pi' \in (C(T'|_{S_i})\backslash C(T|_{S_i})) \cap C(T_i) \cap \Pi_X$ for any $i \in [2]$. Since $\pi' \in C(T_i)$ and $\pi'|_X = \pi \in C(T_i|_X)$, by Lemma A.1, the edge e' that induces π' is an edge on $P(e_i(\pi))$. Since there are $w(e_i(\pi))$ edges on $P(e_i(\pi))$, there are at most $w(e_i(\pi))$ distinct bipartitions π' in $(C(T'|_{S_i})\backslash C(T|_{S_i})) \cap C(T_i) \cap \Pi_X$. Therefore, for any $i \in [2]$,

$$|(C(T'|_{S_i})\backslash C(T|_{S_i})) \cap C(T_i) \cap \Pi_X| \leq w(e_i(\pi)). \tag{3.23}$$

Taking sum of the inequalities over $i \in [2]$, we have

$$\sum_{i\in[2]} |(C(T'|_{S_i})\backslash C(T|_{S_i})) \cap C(T_i) \cap \Pi_X| \leq w(e_1(\pi)) + w(e_2(\pi)) = w^*(\pi). \tag{3.24}$$

Lemma 3.4 extends the results of Lemma 3.3 to a set of compatible bipartitions. Naturally, the proof of Lemma 3.4 uses Lemma 3.3 repeatedly by adding the compatible bipartitions to the tree in an arbitrary order.

Lemma 3.4. *For any compatible set $F \subseteq \Pi$, let $T \in \mathcal{T}_S$ such that $C(T|_X) = F$. Then $p_X(T) \leq w^*(F) = \sum_{\pi\in F} w^*(\pi)$.*

Fix an arbitrary ordering of bipartitions in F and let them be $\pi_1, \pi_2, \ldots, \pi_k$, where $k = |F|$. Let $F_j = \{\pi_1, \ldots, \pi_j\}$ for any $j \in \{0, 1, \ldots, k\}$. In particular, $F_0 = \emptyset$ and $F_k = F$. Let T^j be obtained by contracting all edges in $P(e)$ for any $e \in E(T|_X)$ such that $\pi_e \notin F_j$. Then, $C(T^j|_X) = F_j$. For each $j \in [k]$, $C(T^j|_X)\backslash C(T^{j-1}|_X) = \{\pi_j\}$. Fix $j \in [k]$ and fix any

$\pi' \in C(T^j|_{S_i}) \backslash C(T^{j-1}|_{S_i})$ for some $i \in [2]$. By Lemma A.1, we have $\pi'|_X \in C(T^j|_X)$. We also know $\pi'|_X \notin C(T^{j-1}|_X)$ as otherwise $\pi' \in C(T^j|_{S_i})$ by construction, which is a contradiction. Then $\pi'|_X \in C(T^j|_X) \backslash C(T^{j-1}|_X) = \{\pi_j\}$. Therefore, for any $j \in [k]$, T^j is a refinement of T^{j-1} such that for any $\pi' \in C(T^j|_{S_i}) \backslash C(T^{j-1}|_{S_i})$ for some $i \in [2]$, $\pi'|_X = \pi_j$. Hence we can apply Lemma 3.3 and we have $p_X(T^j) - p_X(T^{j-1}) \leq w^*(\pi_j)$. Therefore, by telescoping sum,

$$p_X(T) - p_X(T^0) = \sum_{j=1}^{k} p_X(T^j) - p_X(T^{j-1}) \leq \sum_{j=1}^{k} w^*(\pi_j). \tag{3.25}$$

Since $C(T^0|_X) = \emptyset$, by Corollary A.2, $C(T^0|_{S_i}) \cap \Pi_X = \emptyset$ for both $i \in [2]$. Then, $C(T^0|_{S_i}) \cap C(T_i) \cap \Pi_X = \emptyset$ for both $i \in [2]$, which implies $p_X(T^0) = 0$. Thus, $p_X(T) \leq \sum_{\pi \in F} w^*(\pi)$, as desired.

Claim 3.1. *Let G be the weighted incompatibility graph on $T_1|_X$ and $T_2|_X$ and let I^* be a maximum weight independent set of G. Let I be the set of bipartitions associated with vertices in I^*, i.e., $I = \{\pi \mid v_\pi \in I^*\}$. Let F be any compatible subset of $C(T_1, T_2, X)$. Then $w^*(I) \geq w^*(F)$.*

We extend the weight function of vertices in G and let weight(U) denote the total weight of any set U of vertices of G. For any compatible subset of bipartitions $F \subseteq C(T_1, T_2, X)$, let $V(F)$ be the set of vertices of G associated with the bipartitions in F. We first claim that $w^*(F) = \text{weight}(V(F))$. For each $\pi \in C(T_1|_X) \cap C(T_2|_X)$, there are two vertices associated with it in G with a total weight of $w(e_1(\pi)) + w(e_2(\pi))$, which is exactly $w^*(\pi)$. For each $\pi \in C(T_i|_X) \backslash C(T_j|_X)$ for $i, j \in [2]$ and $i \neq j$, $\text{weight}(v_\pi) = w(e_i(\pi)) = w^*(\pi)$. Since $C(T_1, T_2, X) = (C(T_1|_X) \cap C(T_2|_X) \cup (C(T_1|_X) \Delta C(T_2|_X))$, we have shown that the $w^*(\pi) = \text{weight}(V(\{\pi\}))$ for any $\pi \in C(T_1, T_2, X)$. Then taking the sum over all $\pi \in F$, we have $w^*(F) = \text{weight}(V(F))$.

Since bipartitions in $C(T_1|_X) \cap C(T_2|_X)$ are compatible with bipartitions in $C(T_1, T_2, X)$, each bipartition in $C(T_1|_X) \cap C(T_2|_X)$ becomes two isolated vertices in the weighted incompatibility graph, all of which must be included in the maximum weight independent set I^*. Therefore, $V(I) = I^*$ and thus $\text{weight}(I^*) = w^*(I)$.

Fix any compatible subset F of $C(T_1, T_2, X)$. Let $F' = F \backslash (C(T_1|_X) \cap C(T_2|_X))$ and let $F'' = V(F' \cup (C(T_1|_X) \cap C(T_2|_X)))$. Then we have

$$\begin{aligned} w^*(F) =& w^*(F') + w^*(F \cap C(T_1|_X) \cap C(T_2|_X)) && (3.26)\\ \leq& w^*(F') + w^*(C(T_1|_X) \cap C(T_2|_X)) && (3.27)\\ =& w^*(F' \cup (C(T_1|_X) \cap C(T_2|_X))) && (3.28)\\ =& \text{weight}(F''). && (3.29) \end{aligned}$$

Since F is compatible, $F' \cup (C(T_1|_X) \cap C(T_2|_X))$ is also compatible, and thus F'' is an independent set in G. Therefore, $\text{weight}(F'') \leq \text{weight}(I^*)$, since I^* is a maximum weight independent set in G. We conclude that $w^*(F) \leq \text{weight}(F'') \leq \text{weight}(I^*) = w^*(I)$.

We have the following claim about $p_X(T_{\text{init}})$, which is needed for the proof of Claim 3.3.

Claim 3.2. *Let T_{init} be the tree defined in Algorithm 3.1. Then, $p_X(T_{\text{init}}) = 2|X|$.*

For each $v \in X$, consider the bipartition $\pi_v = [\{v\} \mid S \backslash \{v\}]$ of T_{init} induced by the edge that connects the leaf v to the center $\hat{v}$. It is easy to see that $\pi_v|_{S_i} = [\{v\} \mid S_i \backslash \{v\}] \in C(T_i)$ for any $i \in [2]$ as $\pi_v|_{S_i}$ is a trivial bipartition of S_i. By construction, we also have $\pi_v|_{S_i} \in C(T_{\text{init}}|_{S_i})$. We also know $\pi_v|_{S_i} \in \Pi_X$ as both sides of π_v have non-empty intersections with X. Thus, $\pi_v|_{S_i} \in C(T_{\text{init}}|_{S_i}) \cap C(T_i) \cap \Pi_X$ for any $i \in [2]$. So for each $v \in X$, $\pi_v|_{S_1}$ and $\pi_v|_{S_2}$ each contributes 1 to $p_X(T_{\text{init}})$. Therefore, $p_X(T_{\text{init}}) \geq 2|X|$.

Fix any bipartition $\pi = [A|B]$ induced by any other edge $e \in E(T_{\text{init}}|_{S_i})$ for any $i \in [2]$. By construction of T_{init}, e must be an edge in an extra subtree or connecting an extra subtree to the center $\hat{v}$, i.e., one component in $T - e$ does contain any leaf of X. Therefore, either $A \subseteq S \backslash X$ or $B \subseteq S \backslash X$, which implies $\pi|_{S_i} \notin \Pi_X$ for any $i \in [2]$. Hence, there is no other bipartition of T_{init} such that when restrict to S_i contributes to $p_X(T_{\text{init}})$. Therefore, $p_X(T_{\text{init}}) = 2|X|$.

The following claim states that the algorithm adds the trivial bipartitions of X (which are all in I) to $T|_X$ in such a way that $p_X(T)$ reaches the full potential of adding those trivial bipartitions. The proof naturally follows from the way we attach extra subtrees to the edges inducing trivial bipartitions (Line 8 of Algorithm 3.1).

Claim 3.3. *Let $\tilde{T}$ be the tree constructed after line 11 of Algorithm 3.1, then $p_X(\tilde{T}) = \sum_{\pi \in \text{Triv}} w^*(\pi)$.*

Let $\pi = [\{a\}|B]$ be a trivial bipartition of X. We know both $e_1(\pi)$ and $e_2(\pi)$ exist, and we abbreviate them with e_1 and e_2. We number the extra subtrees in $\mathcal{TR}(e_1)$ as $t_1, t_2, \ldots, t_p$, where $p = w(e_1) - 1$, such that t_1 is the closest to a in T_1. Similarly, we number extra subtrees in $\mathcal{TR}(e_2)$ as $t'_1, t'_2, \ldots, t'_q$, where $q = w(e_2) - 1$, such that t'_1 is the closest to a in T_2. For each $k \in [w(e_1)]$, we define

$$A_k := \bigcup_{i=1}^{k-1} L(t_i) \cup \{a\}, \ \pi_k := [A_k | S_1 \backslash A_k], \tag{3.30}$$

and for each $k \in [w(e_2)]$, we define

$$A'_k := \bigcup_{i=1}^{k-1} L(t'_i) \cup \{a\},\ \pi'_k := [A'_k | S_2 \backslash A'_k]. \tag{3.31}$$

It follows by definition that π_k for any $k \in [w(e_1)]$ is the bipartition induced by the kth edge on $P(e_1)$ in T_1, where the edges are numbered starting from the side of a. This implies $\pi_k \in C(T_1)$ for any $k \in [w(e_1)]$. Similarly, $\pi'_k \in C(T_2)$ for any $k \in [w(e_2)]$. In particular, we notice that $\pi_1 = [\{a\}|S_1\backslash\{a\}]$ and $\pi'_1 = [\{a\}|S_2\backslash\{a\}]$. Clearly, all these bipartitions (π_k and π'_k for any k) are in Π_X because both sides have none empty intersection with X.

Recall that Algorithm 3.1 moves all extra subtrees in $\mathcal{TR}^*(\pi)$ onto the edge $(\hat{v}, a)$ and orders them in a way to add our desired bipartitions. In particular, the extra subtrees are ordered such that subtrees from $\mathcal{TR}(e_1)$ and subtrees from $\mathcal{TR}(e_2)$ are side by side and the attachments of $\mathcal{TR}(e_i)$ match their attachment on e_i exactly (i.e., t_1 or t'_1, respectively, is closest to a). It is easy to see that as a result of such ordering of the extra subtrees, we have $\pi_k \in C(T|_{S_1})$ for any $k \in [w(e_1)]$ and $\pi'_k \in C(T|_{S_2})$ for any $k \in [w(e_2)]$, where T is the tree obtained after adding π to the backbone through line 8 of Algorithm 3.1. Therefore, the algorithm increases $|C(T|_{S_1}) \cap C(T_1|_X) \cap \Pi_X|$ by $w(e_1) - 1$, because $\pi_k \notin C(T|_{S_1})$ before the step for all $k \in [w(e_1)]$ except $k = 1$ (since $\pi_1 = [\{a\}|S_1\backslash\{a\}] \in C(T|_{S_1})$). Similarly, the algorithm increases $|C(T|_{S_2}) \cap C(T_2|_X) \cap \Pi_X|$ by $w(e_2) - 1$. Overall $p_X(T)$ is increased by $w(e_1) + w(e_2) - 2 = w^*(\pi) - 2$ by running one execution of line 8 in Algorithm 3.1 on T and π.

It is easy to see that line 8 of Algorithm 3.1 never destroys bipartitions of S_1 or S_2 already in T, so we have

$$\begin{aligned}
p_X(\tilde{T}) &= p_X(T_{\text{init}}) + \sum_{\pi \in \text{Triv}} (w^*(\pi) - 2) && (3.32)\\
&= 2|X| + \sum_{\pi \in \text{Triv}} (w^*(\pi) - 2) && \text{(by Claim 3.2)} \quad (3.33)\\
&= \sum_{\pi \in \text{Triv}} w^*(\pi). && \text{(since } |\text{Triv}| = |X|) \quad (3.34)
\end{aligned}$$

The following lemma (Lemma 3.5) shows that Algorithm 3.2 adds any non-trivial bipartition $\pi \in I$ of X to $T|_X$ in a way that realizes the maximum potential increase of $p_X(T)$ of adding π. The proof mainly relies on three invariants of Algorithm 3.1 and 3.2, which we describe intuitively here and prove in Lemma A.3 in Appendix A.2. One invariant is that the auxiliary data structures H and sv ensure that we can correctly find the vertex to split

to add any bipartition to $T|_X$. Second is that the extra subtrees of any bipartition π to be added (i.e., $\mathcal{TR}^*(\pi)$) are attached to the splitting vertex $v = sv(\pi)$ so that we can move them from v onto the new edge (v_a, v_b), which will induce π in $T'|_X$. Third is that we always make sure that for any bipartition $\pi = [A|B] \in C(T|_X)$, the extra subtrees in $\mathcal{TRS}(A)$ or $\mathcal{TRS}(B)$ are attached to the right side of the tree (see Figure 3.3). With these invariants, for any bipartition π to be added, Algorithm 3.2 is able to split the vertex correctly and move extra subtrees around in a way such that each bipartition in T_1 or T_2 that is induced by an edge in $P(e_1(\pi))$ or $P(e_2(\pi))$ is present in $T|_{S_1}$ or $T|_{S_2}$ after the refinement. Since there are exactly $w^*(\pi)$ such bipartitions, they increase $p_X(\cdot)$ by $w^*(\pi)$.

Lemma 3.5. *Let T be a supertree computed within Algorithm 3.1 at line 14 immediately before a refinement step. Let $\pi = [A|B] \in \text{NonTriv} \cap I$. Let T' be a refinement of T obtained from running Algorithm 3.2 with supertree T, bipartition π, and the auxiliary data structures H and sv. Then, $p_X(T') - p_X(T) = w^*(\pi)$.*

Since I corresponds to an independent set in the incompatibility graph G, all bipartitions in I are compatible. Since $C(T|_X) \subseteq \text{Triv} \cup (\text{NonTriv} \cap I) = I$, $\pi \in \text{NonTriv} \cap I$ must be compatible with $C(T|_X)$, then there is a vertex to split to add π to $C(T|_X)$. By invariant 1 of Lemma A.3, $v = sv(\pi)$ is the vertex to split to add π to $T|_X$ and thus Algorithm 3.2 correctly splits v into v_a and v_b and connects them to appropriate neighbors such that in $T'|_X$, (v_a, v_b) induces π.

We abbreviate $e_1(\pi)$ and $e_2(\pi)$ by e_1 and e_2. We number the extra subtrees attached to e_1 as $t_1, t_2, \ldots, t_p$, where $p = w(e_1) - 1$ and t_1 is the closest to A in T_1. Similarly, we number the extra subtrees attached to e_2 as $t'_1, t'_2, \ldots, t'_q$, where $q = w(e_2) - 1$ and t'_1 is the closest to A in T_2.

For any set $\mathcal{T}$ of trees, let $L(\mathcal{T})$ denote the union of the leaf set of trees in $\mathcal{T}$. We note that if e_i exists, $\text{Extra}(T_i) = \mathcal{TRS}_i(A) \cup \mathcal{TRS}_i(B) \cup \mathcal{TR}(e_i)$. Thus, $A \cup L(\mathcal{TRS}_i(A)) \cup L(\mathcal{TR}(e_i)) \cup L(\mathcal{TRS}_i(B)) \cup B = S_i$ for $i \in [2]$.

For each $k \in [w(e_1)]$, we define

$$A_k := \bigcup_{i=1}^{k-1} L(t_i) \cup L(\mathcal{TRS}_1(A)) \cup A, \ \pi_k := [A_k|S_1 \backslash A_k], \tag{3.35}$$

and for each $k \in [w(e_2)]$, we define

$$A'_k := \bigcup_{i=1}^{k-1} L(t'_i) \cup L(\mathcal{TRS}_2(A)) \cup A, \ \pi'_k := [A'_k|S_2 \backslash A'_k]. \tag{3.36}$$

We know that for each $k \in [w(e_1)]$,

$$S_1 \backslash A_k = \bigcup_{i=k}^{p} L(t_i) \cup L(\mathcal{TRS}_1(B)) \cup B. \tag{3.37}$$

Thus, for any $k \in [w(e_1)]$, π_k is the bipartition induced by the kth edge on $P(e_1)$ in T_1, where the edges are numbered from the side of A. Therefore, $\pi_k \in C(T_1)$ for any $k \in [w(e_1)]$. Similarly, $\pi'_k \in C(T_2)$ for any $k \in [w(e_2)]$.

Since for any $k \in [w(e_1)]$, $A_k \cap X = A \neq \emptyset$ and $(S_1 \backslash A_k) \cap X = B \neq \emptyset$, we have $\pi_k|_X = \pi$ and $\pi_k \in \Pi_X$. Similarly, for each $k \in [w(e_2)]$, $\pi'_k \in \Pi_X$ and $\pi'_k|_X = \pi$. We also know that since $\pi \notin C(T|_X)$, by Corollary A.2, $\pi_k \notin C(T|_{S_1})$ for any $k \in [w(e_1)]$ and $\pi'_k \notin C(T|_{S_2})$ for any $k \in [w(e_2)]$. We claim that $\pi_k \in C(T'|_{S_1})$ for all $k \in [w(e_1)]$ and $\pi'_k \in C(T'|_{S_2})$ for all $k \in [w(e_2)]$. Then assuming the claim is true, we have $|C(T'|_{S_1}) \cap C(T_1) \cap \Pi_X| - |C(T|_{S_1}) \cap C(T_1) \cap \Pi_X| = w(e_1)$ and $|C(T'|_{S_2}) \cap C(T_2) \cap \Pi_X| - |C(T|_{S_2}) \cap C(T_2) \cap \Pi_X| = w(e_2)$, and thus $p_X(T') - p_X(T) = w(e_1) + w(e_2) = w^*(\pi)$.

Now we only need to prove the claim. Fix $k \in [w(e_1)]$, we will show that $\pi_k \in C(T'|_{S_1})$. The claim of $\pi'_k \in C(T'|_{S_2})$ for any $k \in [w(e_2)]$ follows by symmetry. By invariant 2 of Lemma A.3, we know that all extra subtrees of $\mathcal{TR}(e_1)$ were attached to v at the beginning of Algorithm 3.2 and thus the algorithm attaches them all onto (v_a, v_b) in the order of $t_1, t_2, \ldots, t_p$, such that t_1 is closest to A. Let the attaching vertex of t_i onto (v_a, v_b) be u_i for any $i \in [w(e_1)]$. Then we note $P((v_a, v_b))$ is the path from v_a to $u_1, u_2, \ldots, u_p$ and then to v_b. For any $t \in \mathcal{TRS}_1(A)$, by invariant 3 of Lemma A.3, t attached to $C(A)$, the component containing A in $T'|_X - (v_a, v_b)$. Therefore, if we delete any edge of $P((v_a, v_b))$ from T', t is in the same component as A. Similarly, for any $t \in \mathcal{TRS}_1(B)$, t is in the same component as B if we delete any edge of $P((v_a, v_b))$ from T. In particular, consider $T'|_{S_1} - (u_{k-1}, u_k)$. The component containing u_{k-1} and A contains all of $\mathcal{TRS}_1(A)$ and $\{t_i \mid i \in [k-1]\}$, thus the leaves of that component is

$$A \cup L(\mathcal{TRS}_1(A)) \cup \bigcup_{i=1}^{k-1} L(t_i) = A_k. \tag{3.38}$$

Therefore, the edge (u_{k-1}, u_k) induces the bipartition $[A_k | S_1 \backslash A_k]$ in $T'|_{S_1}$. Hence, $\pi_k \in C(T'|_{S_1})$ as desired.

Next, we restate our main theorem and present the proof using the lemmas and claims we have shown.

Theorem 3.1. *Let $\mathcal{A} = \{T_1, T_2\}$ with $L(T_i) = S_i$ $(i = 1, 2)$ and $X := S_1 \cap S_2$.* RFS-2-B*$(\mathcal{A})$ and* SFS-2-B*$(\mathcal{A})$ can be solved in $O(n^2|X|)$ time, where $n := \max\{|S_1|, |S_2|\}$.*

First we claim that $p_X(T^*) \geq p_X(T)$ for any tree $T \in \mathcal{T}_S$, where T^* is defined as from line 14 of Algorithm 3.1. Fix arbitrary $T \in \mathcal{T}_S$ and let $F = C(T|_X)$. Then by Lemma 3.4,

$$p_X(T) \leq \sum_{\pi \in F} w^*(\pi) = \sum_{F \cap C(T_1,T_2,X)} w^*(\pi). \tag{3.39}$$

The equality follows from that $w^*(\pi) = 0$ for any $\pi \notin C(T_1, T_2, X)$. Since $F \cap C(T_1, T_2, X)$ is a compatible subset of $C(T_1, T_2, X)$, we have $w^*(F \cap C(T_1, T_2, X)) \leq w^*(I)$ by Claim 3.1. Since $\mathrm{Triv} \subseteq C(T_1|_X) \cap C(T_2|_X) \subseteq I$, we have

$$I = (\mathrm{NonTriv} \cap I) \cup (\mathrm{Triv} \cap I) = (\mathrm{NonTriv} \cap I) \cup \mathrm{Triv}. \tag{3.40}$$

Therefore, by Claim 3.3 and Lemma 3.5, we have

$$p_X(T^*) = p_X(\tilde{T}) + \sum_{\pi \in \mathrm{NonTriv} \cap I} w^*(\pi) = \sum_{\pi \in \mathrm{Triv}} w^*(\pi) + \sum_{\pi \in \mathrm{NonTriv} \cap I} w^*(\pi) = \sum_{\pi \in I} w^*(\pi) = w^*(I). \tag{3.41}$$

Therefore, $p_X(T^*) = w^*(I) \geq p_X(T)$.

From Lemma 3.2 and the fact that a refinement of a tree never decreases $p_X(\cdot)$ and $p_Y(\cdot)$, we also know that $p_Y(T^*) \geq p_Y(T_{\mathrm{init}}) \geq p_Y(T)$ for any tree $T \in \mathcal{T}_S$. By Observation 3.1, for any $T \in \mathcal{T}_S$, we have $\mathrm{SF}(T, \mathcal{A}) = p_X(T) + p_Y(T)$. Therefore, T^* achieves the maximum split support score with respect to $\mathcal{A}$ among all trees in $\mathcal{T}_S$. Thus, T^* is a solution to Relax–SFS-2- (Corollary 3.1). If T^* is not binary, Algorithm 3.1 arbitrarily resolves any polytomy in T^* until it is a binary tree. Because a refinement of a tree does not decreases its split support score, Algorithm 3.1 returns a tree that achieves the maximum split support score among all binary trees of leaf set S. See Appendix A.2 for the running time analysis.

Corollary 3.1. *Let $\mathcal{A} = \{T_1, T_2\}$ with S_i the leaf set of T_i $(i = 1, 2)$ and $X := S_1 \cap S_2$.* Relax–SFS-2-*can be solved in $O(n^2|X|)$ time, where $n := \max\{|S_1|, |S_2|\}$.*

3.3 OTHER RESULTS

We present additional results on the relationship between Relax–RFS and Relax–SFS and the hardness of RFS, SFS, and Relax–RFS.

Lemma 3.6. *There exist instances of* Relax–RFS *and* Relax–SFS *in which an optimal solution to* Relax–RFS *is not an optimal solution to* Relax–SFS*, and vice-versa.*

Let $N \geq 5$ be any integer. Let $\pi_i = [1, 2, \ldots, i+1 \mid i+2, \ldots, N]$ for any $i \in [N-3]$. Let $\mathcal{A} = \{T_1, T_2, \ldots, T_{n-3}\}$ be a profile, where for all $i \in [N-3]$, the leaf set of T_i is $[N]$ and T_i contains a single internal edge defining π_i. Let $\Pi_{[N]}$ denote the set of trivial bipartitions of $[N]$. Let T be the star tree with leaf set $[N]$ (i.e., T has no internal edges). We note that $C(T) = \Pi_{[N]})$. Let $\Pi' = \{\pi_i \mid i \in [N-3]\}$ (i.e., Π' contains all the nontrivial bipartitions from the trees in $\mathcal{A}$). Let T' be the caterpillar tree on leaf set $[N]$ (i.e., T' is formed by taking a path of length $N-2$ with vertices $v_2, v_3, \ldots, v_{N-1}$ in that order, and making leaf 1 adjacent to v_2, leaf i adjacent to v_i, and leaf N adjacent to v_{N-1}). We note that T' the unique tree such that $C(T') = \Pi_{[N]} \cup \Pi'$ and thus a compatibility supertree for $\mathcal{A}$.

We will show that (1) T is an optimal solution for Relax–RFS($\mathcal{A}$), but not an optimal solution for Relax–SFS($\mathcal{A}$), and (2) that T' is an optimal solution for Relax–SFS($\mathcal{A}$), but not an optimal solution for Relax–RFS($\mathcal{A}$).

(1) We first show that T is not an optimal solution for Relax–SFS($\mathcal{A}$). Since T‘ is a compatibility supertree of trees in $\mathcal{A}$, it achieves the maximum split support score possible. In particular, $C(T') \cap C(T_i) = \Pi_{[N]} \cup \{\pi_i\}$ and thus $\mathrm{SF}(T', T_i) = N+1$ for all $i \in [N-3]$. Overall, the split support score of T' is

$$\mathrm{SF}(T', \mathcal{A}) = \sum_{i \in [N-3]} \mathrm{SF}(T', T_i) = (N-3)(N+1). \tag{3.42}$$

Since $C(T) \cap C(T_i) = \Pi_{[N]}$, we have

$$\mathrm{SF}(T, \mathcal{A}) = \sum_{i \in [N-3]} \mathrm{SF}(T, T_i) = (N-3)N < (N-3)(N+1) \tag{3.43}$$

for any $N \geq 5$. Therefore, T is not an optimal solution for Relax–SFS($\mathcal{A}$).

Since $|C(T) \backslash C(T_i)| + |C(T_i) \backslash C(T)| = 1$ for all $i \in [N-3]$, the RFS score of T is

$$\mathrm{RF}(T, \mathcal{A}) = \sum_{i \in [N-3]} \mathrm{RF}(T, T_i) = N-3. \tag{3.44}$$

Now consider any tree $t \neq T$ with leaf set $[N]$, and suppose t contains p bipartitions in Π' and q bipartitions in $2^{[N]} \backslash (\Pi' \cup \Pi_{[N]})$ where $p, q \in \mathbb{N}$. Since $t \neq T$, at least one of p and q is nonzero. Therefore,

$$\mathrm{RF}(t, \mathcal{A}) = \sum_{i \in [N-3]} \mathrm{RF}(t, T_i) \tag{3.45}$$

$$= \sum_{i \in [N-3]} |C(t) \backslash C(T_i)| + |C(T_i) \backslash C(t)| \tag{3.46}$$

$$=q(N-3)+(p-1)p+p(N-3-p)+(N-3-p) \tag{3.47}$$
$$=(N-3)+q(N-3)+p(N-5). \tag{3.48}$$

Since $N \geq 5$ and both p and q are non-negative with at least one of them nonzero, we know the RFS score of t is strictly greater than that of T. Therefore, T is an optimal solution to Relax–RFS($\mathcal{A}$).

For (2), the analysis above already shows that T' has the largest possible split support score. Hence, T' is an optimal solution to the relaxed Split Fit Supertree problem. However, the RFS score for the star tree T is $N-3$ and the RFS score for T' is $(N-4)(N-3)$, which is strictly larger than $N-3$ for $N > 5$; hence, T' is not an optimal solution for the relaxed RF supertree problem.

We show that the Split Fit Supertree problem and the Asymmetric Median Supertree (AMS) problem, which was introduced in [45] and which we will present below, have the same set of optimal solutions and thus the hardness of one implies hardness of another.

The Asymmetric Median Supertree problem takes a profile $\mathcal{A} = \{T_1, T_2, \ldots, T_N\}$ with leaf sets S_i for T_i and finds a binary tree T^* on leaf set $S := \bigcup_{i\in[N]} S_i$ such that

$$T^* = \operatorname*{argmin}_{T\in\mathcal{T}_S} \sum_{i\in[N]} |C(T_i) \setminus C(T|_{S_i})|. \tag{3.49}$$

In other words, the asymmetric median supertree T^* minimizes the total number of bipartitions that are in the source trees and not in the supertree (equivalently, it minimizes the total number of false negatives).

Lemma 3.7. *Given a profile $\mathcal{A} = \{T_1, T_2, \ldots, T_N\}$ of source trees with leaf sets S_i for T_i and $S := \bigcup_{i\in[N]} S_i$, a tree $T \in \mathcal{T}_S$ is a Split Fit Supertree for $\mathcal{A}$ if and only if it is an Asymmetric Median Supertree for $\mathcal{A}$.*

Let $\mathrm{FN}(T, \mathcal{A}) = \sum_{i\in[N]} |C(T_i) \setminus C(T|_{S_i})|$ be the total number of false negatives of T with respect to $\mathcal{A}$, and we refer to this as the false negative score of T. Then,

$$\mathrm{SF}(T, \mathcal{A}) + \mathrm{FN}(T, \mathcal{A}) = \sum_{i\in[N]} |C(T_i) \cap C(T|_{S_i})| + |C(T_i) \setminus C(T|_{S_i})| \tag{3.50}$$
$$= \sum_{i\in[N]} |C(T_i)|. \tag{3.51}$$

Since the sum of the split support score of T and the false negative score of T is the same, regardless of T, minimizing the false negative score is the same as maximizing the split

support score. Hence any tree T is an Asymmetric Median supertree if and only if it is a Split Fit supertree, for all profiles $\mathcal{A}$.

Corollary 3.2. RFS-3, SFS-3, *and* Relax–SFS-3 *are all* ***NP****-hard.*

By Lemma 3.7 and Lemma 3.1, we know that for any profile $\mathcal{A}$, the Robinson-Foulds, Split Fit, and Asymmetric Median supertrees all have the same set of optimal solutions. We also note that the Asymmetric Median Tree problem was shown to be **NP**-hard for three trees [44], which is the same as the Asymmetric Median Supertree problem when all three trees have the same set of species. Therefore, SFS-3 and RFS-3 are both **NP**-hard. Since refining a tree never decreases its split support score, SFS-3 trivially reduces to Relax–SFS-3, and thus Relax–SFS-3 is also **NP**-hard.

Chapter 4: Experiments and Results

We propose GreedyRFS, a simple heuristic that takes as input a profile $\mathcal{A}$ of source trees and then merges pairs of them (using Exact-2-RFS) until the trees are merged into a single supertree. The choice of which pair to merge follows the technique used in SuperFine [46] for computing the Strict Consensus Merger, which selects the pair that maximizes the number of shared taxa between the two trees (other techniques could be used, potentially with better accuracy [47]). Note that GreedyRFS is identical to Exact-2-RFS when the profile has only two trees. We compare GreedyRFS to FastRFS [17], the leading RFS heuristic, with respect to RFS criterion scores.

We used two different types of simulated datasets: the first based on a traditional supertree paradigm called "SMIDgen", using online datasets from prior published studies [46], and the second based on a divide-and-conquer approach applied to multi-locus species tree estimation with newly generated datasets.

4.1 EXPERIMENT 1: EVALUATION ON SUPERTREE DATASETS

We use a subset of the datasets that were used originally in [46] and then in later studies [7, 48], including the study for FastRFS [17]. See [49] for the full description of the simulation protocol.

These are multi-locus simulated supertree datasets with a total of 500 species and varying numbers of source trees. Each source tree is computed using maximum likelihood heuristics, with several clade-based source trees and a single scaffold source tree (i.e., species sampled randomly from across the tree). We selected the hardest of these 500-leaf conditions, where the scaffold tree has only 20% of the leaves. Because all the source trees miss some leaves, the number of leaves per supertree dataset varied. The source trees were then given to FastRFS and GreedyRFS to combine into a supertree.

We use the first 10 replicates out of a total of 30 replicates. Note that since replicate number 8 requires combining two trees with less than 2 shared taxa, supertree construction does not make sense on this replicate. After eliminating this replicate, we end up with 9 replicates in total. To make inputs with k source trees, for $k \in \{2, 4, 6, 8, 10, 12, 14\}$, we take the *first* k source trees in each replicate. Since the first tree is always the scaffold tree, all of our replicates contain the scaffold tree.

We explored the impact of changing the number of source trees. The result for two source trees is predicted by theory (i.e., GreedyRFS is optimal for this case), but even when the number of source trees was greater than two, GreedyRFS dominated FastRFS in terms of

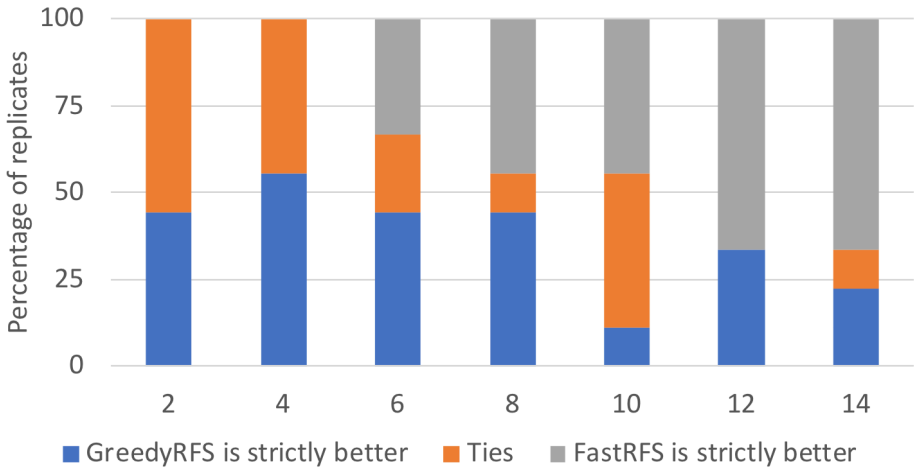


Figure 4.1: **Experiment 1**. The percentage of datasets (y-axis) that each method (FastRFS and GreedRFS) ties with or is strictly better than the other in terms of RFS criterion score is shown for varying numbers of source trees (x-axis), based on nine replicate supertree 500-leaf datasets (from [46]) with 20% scaffold trees.

criterion score provided that the number of source trees was not too large (Figure 4.1).

4.2 EXPERIMENT 2: EVALUATION ON MULTI-LOCUS DATASETS WITH ILS

The second collection of simulated datasets were generated for this study to evaluate GreedyRFS and FastRFS within a divide-and-conquer pipeline for multi-locus species tree estimation where gene trees can differ from each other due to incomplete lineage sorting (ILS), as described in [50].

We used SimPhy [51] to generate species trees and gene trees with 501 species under the multi-species coalescent model, producing a set of true gene trees that differ from the true species tree by on average 68% of their branches due to ILS [50]. The number of genes varied from 25 to 1000 with ten replicate datasets per number of genes. (We vary the number of true gene trees by selecting the first 100 and 25 true gene trees from the datasets with 1000 true gene trees.)

For each replicate dataset, we used the model species tree and a technique similar to DACTAL [22] to divide the species set into two overlapping subsets, each containing slightly more than half the species. ASTRAL [52–54] is a leading method for species tree estimation in the presence of ILS for large numbers of species, and so we used ASTRAL v5.6.3 (i.e., ASTRAL-III) to construct subset trees on the model gene trees, restricted to the relevant subset of species. Finally, the two ASTRAL subset trees were merged together using Exact-

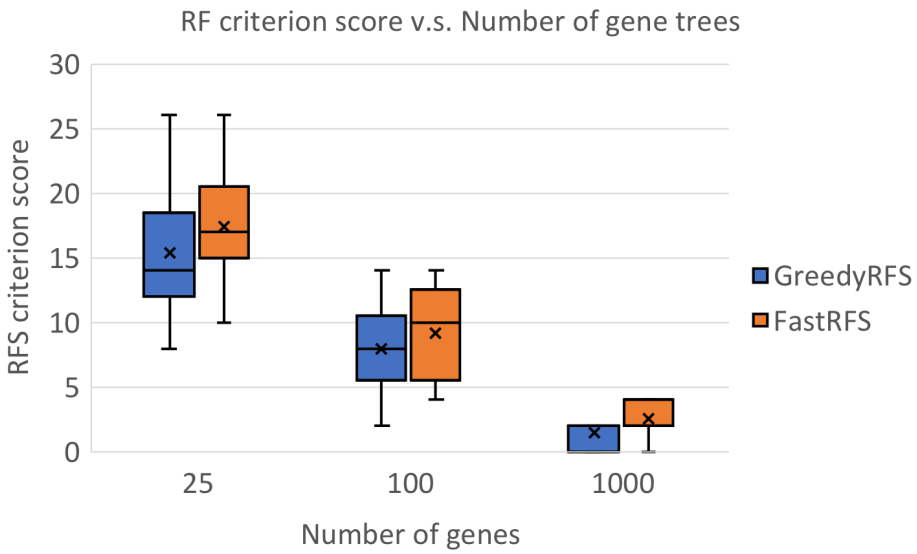


Figure 4.2: **Experiment 2**. The RFS criterion scores (y-axis) are shown for FastRFS and GreedyRFS within divide-and-conquer strategies for multi-locus species tree estimation, where gene trees can differ from the species tree due to ILS. Each dataset had 501 species and varying numbers of gene trees (x-axis). Species trees were estimated on two overlapping subsets of species using ASTRAL and then combined using the specified supertree method. Box plots show data from ten replicates, and no outliers are excluded from the box plot.

RFS-2 and FastRFS.

In general, the average criterion scores for GreedyRFS were better than FastRFS, and GreedyRFS was clearly more accurate than FastRFS for 1000 genes.

This improvement for larger numbers of genes is relevant to practice, as phylogenomic datasets typically contain hundreds to tens of thousands of genes (e.g., the Avian phylogenomic dataset had fewer than 50 species but more than 14,000 loci [55]).

Chapter 5: Future Work and Conclusion

Future work. Due to the limited time for this project, we restricted our attention to the RFS and SFS problem with binary source trees. However, we strongly believe that the polynomial time results also apply to the two problems with general source trees that are not necessarily binary. Thus we give the following conjecture.

Conjecture 5.1. RFS-*2 and* SFS-*2 can be solved in polynomial time.*

We also do not know what is the complexity of Relax–RFS-2 due to the fact that Relax–RFS and Relax–SFS have different optimal solutions and we leave this as an open problem.

Conclusion. Supertree construction is immensely valuable in the context of phylogeny estimation as it not only serves to combine phylogenetic trees estimated by different research groups but also plays a crucial role in divide-and-conquer pipelines, which have become more and more important in large-scale phylogeny estimation. In particular, the Robinson-Foulds Supertree problem is well-established and has desirable proprieties.

This thesis mainly presents Exact-RFS-2, a polynomial time algorithm to solve the Robinson-Foulds Supertree problem (and also the Split Fit Supertree problem) with two binary source trees. We also proved that the Robinson-Foulds and Split Fit Supertree problems are **NP**-hard when there are at least three source trees. We offer a greedy heuristic, GreedyRFS, that takes unlimited number of sources trees as input and applies Exact-RFS-2 repeatedly until all trees are merged into one supertree. Our experimental study showed that GreedyRFS dominates the leading RFS heuristic, FastRFS, when the number of source trees are small.

Thus, our study advances the theoretical understanding of several important supertree problems and also provides a new method that should improve scalability of phylogeny estimation methods.

Appendix A: Appendix for Theoretical Results

A.1 GENERAL THEOREMS AND LEMMAS ON TREES AND BIPARTITIONS

The following theorem and corollary gives alternative characterizations of compatibility between two bipartitions.

Theorem A.1. *[56] A pair of bipartitions $[A|B]$ and $[A'|B']$ of the same set is compatible if and only if at least one of the four pairwise intersections $A \cap A'$, $A \cap B'$, $B \cap A'$, $B \cap B'$ is empty.*

Corollary A.1. *A pair of bipartitions $[A|B]$ and $[A'|B']$ on the same leaf set is compatible if and only if one side of $[A|B]$ is a subset of one side of $[A'|B']$.*

We now provide a lemma and corollary that formalize the relationship between two distinct, yet closely related entities: bipartitions from a tree on leaf set $R \subseteq S$ and bipartitions restricted to R from a tree on leaf set S.

Lemma A.1. *Let $T \in \mathcal{T}_S$ and let $\pi = [A|B] \in C(T)$ be a bipartition induced by $e \in E(T)$. Let $R \subseteq S$.*

1. *If $R \cap A = \emptyset$ or $R \cap B = \emptyset$, then $e \notin P(e')$ for any $e' \in E(T|_R)$.*
2. *If $R \cap A \neq \emptyset$ and $R \cap B \neq \emptyset$, then for any $\pi' \in C(T|_R)$ induced by $e' \in E(T|_R)$, $\pi|_R = \pi'$ if and only if $e \in P(e')$.*

Let T_R be the minimal subtree of T that spans R. It follows that the leaf set of T_R is R and $T|_R$ is obtained from T_R by suppressing all degree-two nodes.

(Proof of 1) We first claim that if $R \cap A = \emptyset$ or $R \cap B = \emptyset$, then $e \notin E(T_R)$. Assume by way of contradiction that $e \in E(T_R)$. There are then two non-empty components in $T_R - e$. Since e induces $[A|B]$ in T, the two components in $T_R - e$ have leaf set $R \cap A$ and $R \cap B$, which contradicts the fact that one intersection is empty. Therefore, $e \notin E(T_R)$. Furthermore, every edge $e' \in E(T|_R)$ comes from a path in T_R. Since $e \notin E(T_R)$, then $e \notin P(e')$ for any $e' \in E(T|_R)$.

(Proof of 2) If $R \cap A \neq \emptyset$ and $R \cap B \neq \emptyset$, then e is required to connect $R \cap A$ with $R \cap B$ in T (since e connects A with B). Thus, e is in any subtree of T spanning R; in particular, $e \in E(T_R)$. Fix any $\pi' \in C(T|_R)$ induced by $e' \in E(T|_R)$. Note that the bipartition induced by $P(e')$ in T_R equals the bipartition induced by e' in $T|_R$, i.e., π'. For one direction of the proof, suppose $e \in P(e')$. Because internal nodes of $P(e')$ in T_R do not connect to any leaves,

(a) T: before adding $[abedfhl|cgijk]$ (b) T': after adding $[abedfhl|cgijk]$

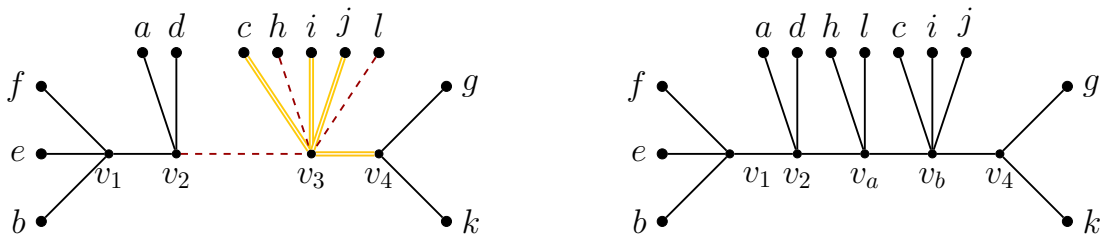


Figure A.1: Splitting a vertex in a tree T to add a compatible bipartition $[A|B] = [abedfhl|cgijk]$. The vertex v_3 satisfies the requirement that no component in $T - v_3$ has leaves from both A and B. Let N_A (N_B) denote the neighbors of v_3 that are in a component containing a leaf in A (B) in $T - v_3$. Then $N_A = \{v_2, h, l\}$ and $N_B = \{c, i, j, v_4\}$. We split v_3 into v_a and v_b. We then make N_A the neighbors of v_a, and N_B the neighbors of v_b. Then (v_a, v_v) induces $[abedfhl|cgijk]$ in T'.

the bipartition induced by the path $P(e')$ in T_R equals the bipartition induced by any of its edges (in particular, e). Since e induces $[A|B]$ in T, it induces $[R \cap A|R \cap B]$ in T_R. Then $\pi' = [R \cap A|R \cap B] = \pi|_R$. On the other hand, if $\pi|_R = \pi'$, then π' induces $[R \cap A|R \cap B]$ in $T|_R$. It follows that $P(e')$ also induces $[R \cap A|R \cap B]$ in T_R. Suppose $e \in P(e^*)$ for some edge $e^* \in E(T|_R)$ such that $e^* \neq e'$. Then, by the previous argument, $\pi_{e^*} = [R \cap A|R \cap B]$, which contradicts the fact that e^* and e' are different edges. Therefore, $e \in P(e')$.

The next corollary follows easily from Lemma A.1.

Corollary A.2. *Let T be a tree with leaf set S and let $\pi = [A|B] \in C(T)$ be a bipartition induced by $e \in E(T)$. Let $R \subseteq S$ such that $R \cap A \neq \emptyset$ and $R \cap B \neq \emptyset$. Then $\pi|_R \in C(T|_R)$.*

In the following lemma, we characterize the vertex that we can split to add a compatible bipartition into a tree. An example can be seen in Figure A.1.

Lemma A.2. *Let T be a tree with leaf set S. Let $\pi = [A|B]$ be a bipartition of S such that $\pi \notin C(T)$, but π is compatible with $C(T)$. Then there exists a unique vertex $v \in V(T)$ such that no component of $T - v$ has leaves from both A and B. Furthermore, we can split the neighbors of v, $N_T(v)$, into two sets N_A and N_B, where N_A contains neighbors whose corresponding components contain a leaf in A and N_B contains neighbors whose corresponding components contain a leaf from B. By replacing v with two vertices v_a and v_b, making v_a adjacent to all the vertices in N_A and v_b adjacent to all the vertices in N_B, and then adding an edge between v_a and v_b, we create a tree T' such that $C(T') = C(T) \cup \{\pi\}$.*

By definition of compatibility, there exists a tree T' such that $C(T') = C(T) \cup \{\pi\}$. Let $e = (v_a, v_b)$ be the edge that induces π in T' such that the component containing v_a

in $T' - (v_a, v_b)$ has leaf set A and the component containing v_b in $T' - (v_a, v_b)$ has leaf set B. Since $\pi \notin C(T)$, when we contract (v_a, v_b), then T' becomes T. Let v be the vertex of T corresponding to the vertex of T' created from contracting (v_a, v_b). Let N_a, N_b be the neighbors of v_a and v_b in $T' - (v_a, v_b)$, respectively. Let N_A, N_B be vertices in T corresponding to N_a and N_b. We note that $N_A \cup N_B = N_T(v)$. Since in $T' - (v_a, v_b)$, no vertex in N_a can reach any vertex of B, the same is true in $T' - v_a - v_b$. Since v_a is in the component of A in $T' - (v_a, v_b)$, so are all vertices of N_a. Then each vertex in N_a must be able to reach some vertex of A in $T' - v_a - v_b$ by either being a leaf in A or in the same component of some leaf in A. Similarly, in $T' - v_a - v_b$, no vertex of N_b can reach any vertex of A, but every vertex of N_b can reach some vertex of B. By construction, $T' - v_a - v_b$ is identical to $T - v$, and thus N_A (and N_B respectively) is a set of neighbors of v that can reach some vertex of A (B) but no vertex of B (A). Therefore, v is the vertex desired.

To obtain T' from T, we can replace v by two new vertices v_a, v_b with an edge between them. We also connect all vertices in N_A to v_a and all vertices in N_B to v_b. Then it is easy to see that (v_a, v_b) induces π in T'.

A.2 LEMMAS AND PROOFS FOR CHAPTER 3

Lemma A.3 proves that the auxiliary data structures of Algorithm 3.1 and 3.2 are maintaining the desired information and that the algorithm can split the vertex and perform the detaching and reattaching of the extra subtrees correctly. These invariants are important to the proof of Lemma 3.5.

Lemma A.3. *At any stage of the Algorithm 3.1 after line* 12*, we have the following invariants of T and the auxiliary data structures H and sv:*

1. *For any bipartition $\pi \in$ NonTriv, $sv(\pi)$ is the vertex to split to add π to $C(T|_X)$. For any internal vertex v, the set of bipartitions $H(v) \subseteq$ NonTriv is the set of bipartitions which can be added to $C(T|_X)$ by splitting v.*

2. *For any $\pi = [A|B] \in H(v)$, for all $t \in \mathcal{TR}^*(\pi)$, the root of t is a neighbor of v.*

3. *For any $\pi = [A|B] \in C(T|_X)$ induced by edge e, let $C(A), C(B)$ be the components containing the leaves of A and B in $T|_X - e$. Then,*

   (a) *all $t \in \mathcal{TRS}(A)$ are attached to an edge or a vertex in $C(A)$*

   (b) *all $t \in \mathcal{TRS}(B)$ are attached to an edge or a vertex in $C(B)$.*

We prove the invariants by induction on the number of refinement steps k performed on T. When $k = 0$, we have $T = \tilde{T}$ and $T|_X$ is a star with leaf set X and center vertex $\hat{v}$. Thus all bipartitions in NonTriv are compatible with $C(T|_X)$. For any $\pi \in$ NonTriv, $\hat{v}$ is the vertex to refine in $T|_X$ to add π to $C(T|_X)$. Therefore, it is correct that $sv(\pi) = \hat{v}$ for every $\pi \in$ NonTriv and $H(\hat{v}) =$ NonTriv. The roots of all extra subtrees in $\mathcal{TR}^*(\pi)$ for any $\pi \in$ NonTriv are all neighbors of $\hat{v}$, so invariant 2 also holds. For any $\pi \in C(T|_X) =$ Triv, let $\pi = [\{a\}|B]$. It is easy to see that since a is a leaf, $\mathcal{TRS}_i(\{a\}) = \emptyset$ and $\mathcal{TRS}_i(B) =$ Extra$(T_i)\backslash\mathcal{TR}(e_i(\pi))$ for both $i \in [2]$. Then $\mathcal{TRS}(\{a\}) = \emptyset$ and $\mathcal{TRS}(B) =$ (Extra$(T_1) \cup$ Extra$(T_2))\backslash\mathcal{TR}^*(\pi)$. Therefore, invariant 3(a) trivially holds as $\mathcal{TRS}(\{a\}) = \emptyset$. Since $C(\{a\})$ is the vertex a and $C(B)$ is the rest of the star of $T|_X$, all $t \in \mathcal{TRS}(B)$ are attached to an edge or a vertex in $C(B)$, then invariant 3(b) holds. This proves invariant 3 and thus concludes our proof for the base case.

Assume that all invariants hold after any $k' < k$ steps of refinement. Let $\pi = [A|B]$ be the bipartition to add in the kth refinement step. We will show that after the kth refinement step, i.e., one execution of Algorithm 3.2, the invariants still hold for the resulting tree T'. Since $v = sv(\pi)$ at the beginning of Algorithm 3.2, π can be added to $C(T|_X)$ by splitting v. By Lemma A.2, there exists a division of neighbors of v in $T|_X$ into $N_A \cup N_B$ such that N_A (or N_B respectively) consists of neighbors of v which can reach vertices of A (or B) but not B (or A) in $T|_X - v$. Then, the algorithm correctly finds N_A and N_B and connects N_A to v_a and N_B to v_b so the new edge (v_a, v_b) induces the bipartition $\pi = [A|B]$ in $T|_X$. For any vertex u other than v and any bipartition $\pi' \in H(u)$, the invariants 1 and 2 still hold after Algorithm 3.2 as we do not change $H(u)$, $sv(\pi')$, or the extra subtrees attached to u. For any bipartition $\pi' =\in H(v)$ such that $\pi' \neq \pi$, if π' is not compatible with π, then it cannot be added to $C(T'|_X)$ since π is added, so the algorithm correctly discards π' and does not add it to $H(v_a)$ or $H(v_b)$. If π' is compatible with π, we will show that the invariants 1 and 2 still hold for π'.

Fix any $\pi' = [A'|B'] \in H(v)$ s.t. $\pi' \neq \pi$. By Corollary A.1, one of A' and B' is a subset of one side of $[A|B]$. Assume without loss of generality that $A' \subseteq A$ (other cases are symmetric). Then we have $B \subseteq B'$. In this case, Algorithm 3.2 adds π' to $H(v_a)$ and set $sv(\pi) = v_a$. We will show that this step preserves the invariants. Since $\pi' \in H(v)$, before adding π we can split v to add π' to $C(T|_X)$. Then there exists a division of neighbors of v in $T|_X$ into $N_{A'}$ and $N_{B'}$ such that $N_{A'}$ (or $N_{B'}$, respectively) consists of neighbors of v which can reach vertices of A' (or B') in $T|_X - v$. It is easy to see that $N_{A'} \subseteq N_A$ and $N_B \subseteq N_{B'}$. Since $N_A \cup N_B = N_{A'} \cup N_{B'} = N_{T|_X}(v)$, we have $N_A\backslash N_{A'} = N_{B'}\backslash N_B$. Since all vertices in N_B are connected to v_b in T' while vertices in $N_{B'}\backslash N_B$ are connected to v_a, $N_{B'}\backslash N_B \cup \{v_b\}$ is the set of all neighbors of v_a which can reach leaves of B' in $T'|_X - v_a$. Then

$N_{T'|_X}(v_a) = N_A \cup \{v_b\} = N_{A'} \cup (N_A \backslash N_{A'} \cup \{v_b\}) = N_{A'} \cup (N_{B'} \backslash N_B \cup v_b)$ implies that $N_{A'}$ and $N_{B'} \backslash N_B \cup \{v_b\}$ gives an division of neighbors of v_a such that $N_{A'}$ are the neighbors that can reach leaves of A' in $T'|_X - v_a$ and $N_{B'} \backslash N_B \cup \{v_b\}$ are the neighbors that can reach leaves of B' in $T'|_X - v_a$. Such a division proves that v_a is the correct vertex to refine in $T'|_X$ to add π' to $C(T'|_X)$ after the kth refinement. Therefore, invariant 1 holds with respect to π'. Since $\pi' \in H(v)$ before adding π, we also have for all $t \in \mathcal{TR}^*(\pi')$, the root of t is a neighbor of v before adding π. Since $A' \subseteq A$, $\pi' \in BP(A)$ and thus $\mathcal{TR}^*(\pi) \subseteq \mathcal{TRS}(A)$. Then, Algorithm 3.2 correctly attaches roots of all trees in $\mathcal{TR}^*(\pi')$ to v_a. Therefore invariant 2 holds for π'.

We have shown that invariants 1 and 2 hold for the tree T' with the auxiliary data structures H and sv. Next, we show that invariant 3 holds. Since π is the only bipartition in $C(T'|_X)$ that is not in $C(T|_X)$, we only need to show two things: i) for any $\pi' \in C(T|_X)$, the invariant 3 still holds, ii) invariant 3 holds for π. We first show i). Fix $\pi' = [A'|B'] \in C(T|_X)$. Since π is compatible with π', by Corollary A.1, one of A' and B' is a subset of one of A and B. We assume without loss of generality that $A' \subseteq A$. Therefore, $B \subseteq B'$. Let $C(A'), C(B')$ be the components containing the leaves of A' and B' in $T|_X - e'$, where e' induces π'. Since $C(A')$ is unchanged after the refinement, invariant 3(a) is trivially true. Since $B \subseteq B'$, $C(B)$ is a subgraph of $C(B')$ and $v \in C(B')$. During the refinement, v is split into v_a and v_b, both of which are still part of $C(B')$. Since all $t \in \mathcal{TRS}(B)$ are attached to an edge or a vertex in $C(B')$ before refinement and any extra subtree attached to v before is now on either v_a, or v_b, or (v_a, v_b), they are all still attached to an edge or a vertex in $C(B')$. Thus, the invariant 3 holds with respect to π'.

For ii), we show invariant 3(a) holds for π and 3(b) follows the same argument. For any extra subtree in $t \in \mathcal{TRS}(A)$, if it was attached to v before refinement, then it is now attached to v_a, which is in $C(A)$. If it was not attached to v before refinement, then let N_B be as defined from Algorithm 3.2. For any bipartition $\pi' = [A'|B']$ induced by (v, u) where $u \in N_B$. We know that $(v, u) \in C(B)$ and thus either $A' \subseteq B$ or $B' \subseteq B$. Assume without loss of generality that $B' \subseteq B$. Then we have $\mathcal{BP}(B') \cup \{\pi'\} \subseteq \mathcal{BP}(B)$ and thus $\mathcal{TRS}(B') \cup \mathcal{TR}^*(\pi') \subseteq \mathcal{TRS}(B)$. We note that $\mathcal{TRS}(A)$ and $\mathcal{TRS}(B)$ are disjoint. Since $t \in \mathcal{TRS}(A)$, we know $t \notin \mathcal{TRS}(B)$, then $t \notin \mathcal{TRS}(B') \cup \mathcal{TRS}^*(\pi')$. Let $C(A'), C(B')$ be the components containing the leaves of A' and B' in $T|_X - (v, u)$. Then $C(A')$ contains v and $C(B')$ contains u. Since $t \notin \mathcal{TR}^*(\pi')$, it cannot be attached to (v, u). Also by the invariant 3 with respect to π', t is not attached any vertex or edge in $C(B')$. Since this is true for every neighbor of v in N_B, $t \notin C(B)$ as $C(B)$ consists of only edges connecting v to a neighbor $u \in N_B$ and the component containing u. Since t was not attached to v before the refinement, t is not attached to (v_a, v_b) or $C(B)$ after the refinement, then t must

be attached to some edge or vertex in $C(A)$. This proves invariant 3(a) for π and thus the inductive proof.

We give the running time analysis for Algorithm 3.1 to complete the proof of Theorem 3.1. First we analyze the running time of Algorithm 3.2, i.e., one refinement step. Dividing the neighbors of v and connecting them to v_a and v_b appropriately in line 3 – 6 take $O(|X|^2)$ time. We can do a depth-first-search in $T|_X - v$ from every neighbor u of v and check in $O(|X|)$ time if any newly discovered vertex is in A or B and connect u to v_a or v_b accordingly. Moving extra subtrees in $\mathcal{TR}^*(\pi)$ in line 7 takes $O(n)$ time as T_i has at most n leafs and thus there are $O(n)$ extra subtrees in total, so $|\mathcal{TR}^*(\pi)|$ is $O(n)$. Line 8 – 13 take $O(n)$ time as the mappings are pre-calculated and there are again $O(n)$ extra subtrees to be moved. Updating the data structures in line 15 – 21 takes $O(|X|^2)$ time as there are at most $O(|X|)$ bipartitions in $H(v)$ and each of the containment conditions is checkable in $O(|X|)$ time by checking whether one side of π' is a subset of one side of π (assuming that labels of leaves in both sides of the bipartitions are stored as pre-processed sorted lists instead of sets). The rest of the algorithm takes constant time. Overall, Algorithm 3.2 runs in $O(n + |X|^2)$ time.

Next we analyze the running time for Algorithm 3.1, i.e., Exact-RFS-2. Computing $C(T_1|_X)$ and $C(T_2|_X)$ in line 1 takes $O(n^2 + n|X|^2)$ time as we need to compute $\pi_e|_X$ for all $e \in E(T_1) \cup E(T_2)$ and then take the union. There are $O(n)$ edges in $E(T_1) \cup E(T_2)$. Computing $\pi_e|_X$ for each edge takes $O(n)$ time by running DFS on $T_i - e$ to obtain π_e and then taking intersection of both sides of π_e with X, separately. Together it takes $O(n^2)$ time. Taking union of the bipartitions takes $O(n|X|^2)$ time as there are $O(n)$ bipartitions to add and whenever we add a new bipartition, it needs to be compared to the $O(|X|)$ distinct existing ones in the set. Since all bipartitions have size $O(|X|)$, the comparison can be done in $O(|X|)$ time (if each of them is represented by two sorted lists instead of two sets). In this step, we can always maintain a set of edges in T_i for each bipartition $\pi \in C(T_1, T_2, X)$ such that $\pi_e|_X = \pi$.

Line 2 – 5 compute the mappings and values we need in latter part of the algorithm. We analyze the running time for each $\pi = [A|B] \in C(T_1, T_2, X)$ first. We can compute the path $P(e_i(\pi))$ by assembling the set of edges associated with π in T_i from the last step into a path. This takes $O(n^2)$ time by counting the times any vertex appear as an end vertex in the set of edges. The two vertices appearing once are the end vertices of the path while those appearing twice are internal vertices of the path. Then $w(e_i(\pi)) = |P(e_i(\pi))|$ can be found in constant time. Then we can find $\mathcal{TR}(e_i(\pi))$ by DFS in $T_i - v$ for every internal node v of $P(e_i(\pi))$, starting the search from the unique neighbor u of v such that u does not appear in the path. This takes $O(n)$ time. We compute $\mathcal{BP}_i(A)$ and $\mathcal{BP}_i(B)$ by iterating

over $O(|X|)$ bipartitions in $C(T_i|_X)$ and check if one side of any bipartition is a subset of A or B in $O(|X|)$ time, this takes $O(|X|^2)$ time together. Next, we compute $\mathcal{TRS}_i(A)$ (or $\mathcal{TRS}_i(B)$) by taking unions of extra subtrees in $\mathcal{TR}(e_i(\pi))$ for any $\pi \in \mathcal{BP}_i(A)$ (or $\mathcal{BP}_i(B)$) in $O(n)$ time. Extra subtrees are unique identified by their roots and $\mathcal{TR}(e_i(\pi))$ is disjoint from the set of extra subtrees associated with other edges, so taking union of at most $O(n)$ extra subtrees takes $O(n)$ time. Therefore, all the mappings and values can be computed in $O(n^2)$ time for each bipartition and thus it takes $O(n^2|X|)$ time overall. With all the extra subtrees calculated for each partition, we can compute Extra(T_i) in $O(n^2)$ time.

Constructing T_{init} in line 6 takes $O(n)$ time. Line 7 constructs an incompatibility graph with $O(|X|)$ vertices and $O(|X|^2)$ edges in $O(|X|^3)$ time as compatibility of any pair of bipartitions of size $O(|X|)$ can be checked in $O(|X|)$ time. For line 8, we can reduce Maximum Weight Independent Set to Minimum Cut problem in a directed graph with a dummy source and sink. Then the Minimum Cut problem can be solved by a standard Maximum Flow Algorithm. Since the best Maximum Flow algorithm runs in $O(|V||E|)$ time and the graph has $O(|X|)$ vertices and $O(|X|^2)$ edges, this line runs in $O(|X|^3)$ time. Line 9 – 10 essentially runs line 7 of Algorithm 3.2 $O(|X|)$ times using a total of $O(n|X|)$ time. Line 11 initiates the data structure H and sv in $O(|X|)$ time. Line 12 – 13 runs Algorithm 3.2 $O(|X|)$ times with a total of $O(n|X| + |X|^3)$ time. To perform line 14, we first find the set of high degree vertices (with degree greater than 3) in $O(n)$ time. Then we repeatedly refine at any high degree vertex v by splitting it into two new vertices and distributing the neighbors of v and adding the two new vertices back to the set if needed, until there is no high degree vertex left. Since the number of internal edges of the tree increase by 1 for every refinement and there are at most $n-3$ internal edges, we know this step takes $O(n^2)$ time as each arbitrary refinement (along with updating the set of high degree vertices) can be done in $O(n)$ time.

Since $|X| \leq n$, $|X|^3 \leq n|X|^2 \leq n^2|X|$, and thus, the overall running time of Algorithm 3.1 is dominated by the running time of line 2 – 5, which is $O(n^2|X|)$.

A.3 MAXIMUM WEIGHT INDEPENDENT SET IN BIPARTITE GRAPHS

Given an undirected bipartite graph G, with vertices $V = A \cup B$, edges E, and vertex weights $w : V \to \mathbb{N}$, the Maximum Weighted Independent Set problem tries to find a independent set $I \subseteq V$ that maximizes $w(I)$, where $w(S) = \sum_{v \in S} w(v)$ for any $S \subseteq V$. It is well known (in folklore) that maximum weight independent set can be solved in polynomial time through reduction to the maximum flow problem. We reproduce a proof for completeness.

We first turn the graph into a directed flow network $G' = (V \cup \{s, t\}, E')$ where s, t are the newly added source and sink, respectively. To obtain E', we direct all edges in E from

A to B, add an edge from s to each vertex $u \in A$ and add an edge from each vertex $v \in B$ to t. We set the capacities $c : E' \to \mathbb{N}$ such that $c(e) = \infty$ if $e \in E$, $c(e) = w(u)$ if $e = (s, u)$ and $c(e) = w(v)$ if $e = (v, t)$. We claim that any s, t-cut (S, T) in G' has a finite capacity k if and only if $(S \cap A) \cup (T \cap B)$ is an independent set of weight $w(V) - k$ in G.

We first observe that $((S \cap A) \cup (T \cap B)) \cup ((S \cap B) \cup (T \cap A)) = (S \cup T) \cap (A \cup B) = A \cup B = V$.

Suppose $(S \cap A) \cup (T \cap B)$ is an independent set of weight $w(V) - k$ in G. Since $((S \cap A) \cup (T \cap B)) \cup ((S \cap B) \cup (T \cap A)) = V$, the weight of $(S \cap B) \cup (T \cap A)$ is $w(V) - (w(V) - k) = k$. Since $(S \cap A) \cup (T \cap B)$ is an independent set, there is no edge from $S \cap A$ to $T \cap B$. There is also no edge from $S \cap B$ to $T \cap A$ since edges in E are directed from A to B. Thus, the cut (S, T) consists of only edges from s to $T \cap A$ and from $S \cap B$ to t. Together the capacities of those edges equal the weight of the set $(S \cap B) \cup (T \cap A)$, which is k.

For the other direction of the proof, suppose (S, T) is an s, t-cut of finite capacity k. Since the cut has finite capacity, it does not contain any edge derived from E. In particular, there is no edge from $S \cap A$ to $T \cap B$ in G', which implies there is no edge between $S \cap A$ and $T \cap B$ in G. Since there is also no edge among $S \cap A$ and $T \cap B$ in G, $(S \cap A) \cup (T \cap B)$ is an independent set. Since the edges in (S, T) solely consist of edges from s to $T \cap A$ and from $S \cap B$ to t, the sum of their capacities is k. Therefore, the weight of the set $(S \cap B) \cup (T \cap A)$ is k and the weight of $(S \cap A) \cup (T \cap B)$ is $w(V) - k$.

Since $w(V)$ is a fixed constant, we conclude that any s, t-cut (S, T) is a minimum cut in G' if and only if $(S \cap A) \cup (T \cap B)$ is an maximum weight independent set in G. By the standard Max-flow Min-cut theorem, a minimum s, t-cut in a directed graph is equivalent to the maximum s, t-flow. Thus, we can solve the Maximum Weighted Independent Set problem on bipartite graphs using a maximum flow algorithm in polynomial time.

Appendix B: Appendix for Experimental Study

Datasets. We provide additional detailed procedure of how the datasets for Experiment 2 are generated:

1. Identify the centroid edge (a, b) of the true species tree (i.e., the edge that, upon deletion, creates 2 subtrees T_a and T_b with leaf sets A and B of roughly equal size)
2. Let X be the set of 25 closest (in term of path distance on the weighted tree) leaves in T_a to a and in T_b to b
3. Let $A' = A \cup X$ and $B' = B \cup X$
4. Restrict all 1000 true gene trees to leaf set A' and use ASTRAL-III [54,57] to compute a tree $A1$ on the restricted true gene trees
5. Restrict all 1000 true gene trees to leaf set B' and use ASTRAL-III to compute a tree $B1$ on the restricted true gene trees
6. Apply supertree methods FastRFS and GreedyRFS to input pair $A1$ and $B1$, and compare to the true species tree

Additional results. In Figure B.1, we present an additional perspective on the comparison of FastRFS and GreedyRFS for Experiment 1, showing the difference in RFS criterion score for the two methods, so that negative scores means that GreedyRFS produces a tree with a better (lower) RFS criterion score, positive scores means that FastRFS has a better score, and a score of zero means the two methods produce trees with the same score. Note that, as predicted by theory, GreedyRFS is never worse than FastRFS when there are only two source trees. The ability to match or improve on FastRFS with respect to criterion score holds also for four source trees, and then is increasingly lost as the number of source trees increases. For the largest number of source trees, FastRFS regularly dominates GreedyRFS, but there are still some inputs where GreedyRFS produces better RFS criterion scores.

Our scripts and commands. Our scripts and other utilities (developed by authors of this paper) are available at http://github.com/yuxilin51/GreedyRFS.

- *GreedyRFS on a set of source trees*

```
GreedyRFS.py -t <source_trees> -o <output_tree>
```

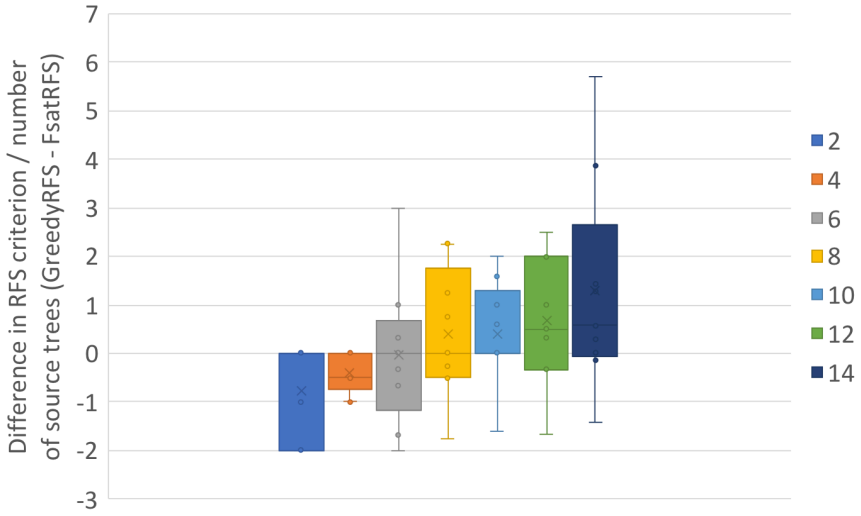


Figure B.1: **Experiment 1**. The difference in the RFS criterion score (y-axis) between FastRFS and GreedyRFS for the 500-taxon SMIDgen datasets, normalized by the number of source trees; a positive score indicates that FastRFS is better than GreedyRFS, a negative score indicates that GreedyRFS is better than FastRFS, and a score of zero indicates they are tied for criterion score. On average, GreedyRFS produces better RFS criterion scores than FastRFS given at most 4 source trees, they are tied for 6 source trees, and then FastRFS produces better RFS criterion scores.

Note that when the input has two source trees, then GreedyRFS is identical to Exact-2-RFS.

- *RFS criterion score*
  To compute the RFS criterion score for a supertree T with respect to a profile $\mathcal{A}$, we add the RF distances between T and every tree $t \in \mathcal{A}$, as follows:

  ```
  compare_trees.py <tree1> <tree2>
  ```

- *Centroid decomposition (1 round)*

  ```
  split_tree.py -t <input_tree> -o <output_directory>
  ```

- *Find overlapping leaf set X*

  ```
  find_x.py -t <input_tree> -o <output_directory>
  ```

- *Restricting tree to a leaf set (Newick Utilities v1.6.0)*

  ```
  nw_prune -v  <input_tree> $(cat <label_of_leaves>) /
      > <output_tree>
  ```

External software commands. We used the following commands in the specified versions of external softwares.

- *FastRFS v1.0*

```
FastRFS -i <source_trees> -o <output_prefix>
```

- *SimPhy v1.0.2*

```
simphy -rs 10 -rl F:1000 -rg 1 -st F:500000 /
   -si F:1 -sl F:500 -sb F:0.0000001 /
   -sp F:200000 -hs LN:1.5,1 -hl LN:1.2,1 /
    -hg LN:1.4,1 -su E:10000000 -so F:1/
    -od 1 -v 3 -cs 293745 /
    -o <output_directory>
```

- *ASTRAL v5.6.3*

```
java -jar <path_to_astral_jar> -i <input_gene_trees> /
    -o <output_est_species_tree>
```

References

[1] M. Abu-Asab, M. Chaouchi, and H. Amri, "Evolutionary medicine: a meaningful connection between omics, disease, and treatment," *PROTEOMICS–Clinical Applications*, vol. 2, no. 2, pp. 122–134, 2008.

[2] D. E. Soltis and P. S. Soltis, "The role of phylogenetics in comparative genetics," *Plant Physiology*, vol. 132, no. 4, pp. 1790–1800, 2003.

[3] A. Purvis, J. L. Gittleman, and T. Brooks, *Phylogeny and Conservation*. Cambridge University Press, 2005.

[4] J. P. Torres-Florez, W. E. Johnson, M. F. Nery, E. Eizirik, M. A. Oliveira-Miranda, and P. M. Galetti, "The coming of age of conservation genetics in latin america: what has been achieved and what needs to be done," *Conservation Genetics*, vol. 19, no. 1, pp. 1–15, Feb 2018.

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

[6] M. A. Ragan, "Phylogenetic inference based on matrix representation of trees," *Molecular phylogenetics and evolution*, vol. 1, no. 1, pp. 53–58, 1992.

[7] N. Nguyen, S. Mirarab, and T. Warnow, "MRL and SuperFine+ MRL: new supertree methods," *Algorithms for Molecular Biology*, vol. 7, no. 1, p. 3, 2012.

[8] C. Semple and M. Steel, "A supertree method for rooted trees," *Discrete Applied Mathematics*, vol. 105, no. 1-3, pp. 147–158, 2000.

[9] R. D. Page, "Modified mincut supertrees," in *International workshop on algorithms in bioinformatics*. Springer, 2002, pp. 537–551.

[10] M. S. Bansal, J. G. Burleigh, O. Eulenstein, and D. Fernández-Baca, "Robinson-Foulds supertrees," *Algorithms for molecular biology*, vol. 5, no. 1, p. 18, 2010.

[11] R. Chaudhary, J. G. Burleigh, and D. Fernandez-Baca, "Fast local search for unrooted Robinson-Foulds supertrees," *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, vol. 9, no. 4, pp. 1004–1013, 2012.

[12] M. Steel and A. Rodrigo, "Maximum likelihood supertrees," *Systematic biology*, vol. 57, no. 2, pp. 243–250, 2008.

[13] D. Bryant and M. Steel, "Computing the distribution of a tree metric," *IEEE/ACM transactions on computational biology and bioinformatics*, vol. 6, no. 3, pp. 420–426, 2009.

[14] F. McMorris and M. A. Steel, "The complexity of the median procedure for binary trees," in *New Approaches in Classification and Data Analysis*. Springer, 1994, pp. 136–140.

[15] A. Kupczok, "Split-based computation of majority-rule supertrees," *BMC evolutionary biology*, vol. 11, no. 1, p. 205, 2011.

[16] R. Chaudhary, J. G. Burleigh, and D. Fernández-Baca, "Inferring species trees from incongruent multi-copy gene trees using the robinson-foulds distance," *Algorithms for Molecular Biology*, vol. 8, no. 1, p. 28, 2013.

[17] P. Vachaspati and T. Warnow, "FastRFS: fast and accurate Robinson-Foulds Supertrees using constrained exact optimization," *Bioinformatics*, vol. 33, no. 5, pp. 631–639, 2016.

[18] M. Wilkinson, J. A. Cotton, C. Creevey, O. Eulenstein, S. R. Harris, F.-J. Lapointe, C. Levasseur, J. O. McInerney, D. Pisani, and J. L. Thorley, "The Shape of Supertrees to Come: Tree Shape Related Properties of Fourteen Supertree Methods," *Systematic Biology*, vol. 54, no. 3, pp. 419–431, 06 2005. [Online]. Available: https://doi.org/10.1080/10635150590949832

[19] D. H. Huson, S. M. Nettles, and T. J. Warnow, "Disk-covering, a fast-converging method for phylogenetic tree reconstruction," *Journal of Computational Biology*, vol. 6, no. 3-4, pp. 369–386, 1999.

[20] D. H. Huson, L. Vawter, and T. Warnow, "Solving large scale phylogenetic problems using DCM2," in *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*. AAAI Press, 1999. [Online]. Available: http://dl.acm.org/citation.cfm?id=645634.660809 pp. 118–129.

[21] T. Warnow, B. M. Moret, and K. St John, "Absolute convergence: true trees from short sequences," in *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2001, pp. 186–195.

[22] S. Nelesen, K. Liu, L.-S. Wang, C. R. Linder, and T. Warnow, "DACTAL: divide-and-conquer trees (almost) without alignments," *Bioinformatics*, vol. 28, no. 12, pp. i274–i282, 2012.

[23] M. S. Bayzid, T. Hunt, and T. Warnow, "Disk covering methods improve phylogenomic analyses," *BMC genomics*, vol. 15, no. 6, p. S7, 2014.

[24] N. Saitou and M. Nei, "The neighbor-joining method: a new method for reconstructing phylogenetic trees." *Molecular biology and evolution*, vol. 4, no. 4, pp. 406–425, 1987.

[25] E. K. Molloy and T. Warnow, "NJMerge: a generic technique for scaling phylogeny estimation methods and its application to species trees," in *RECOMB International conference on Comparative Genomics*. Springer, 2018, pp. 260–276.

[26] E. K. Molloy and T. Warnow, "TreeMerge: a new method for improving the scalability of species tree estimation methods," *Bioinformatics*, vol. 35, no. 14, pp. i417–i426, 07 2019. [Online]. Available: https://doi.org/10.1093/bioinformatics/btz344

[27] T. Le, A. Sy, E. K. Molloy, Q. R. Zhang, S. Rao, and T. Warnow, "Using INC within divide-and-conquer phylogeny estimation," in *International Conference on Algorithms for Computational Biology*. Springer, 2019, pp. 167–178.

[28] S. Snir and S. Rao, "Quartets MaxCut: a divide and conquer quartets algorithm," *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, vol. 7, no. 4, pp. 704–718, 2010.

[29] M. Fleischauer and S. Böcker, "Bad Clade Deletion supertrees: a fast and accurate supertree algorithm," *Molecular biology and evolution*, vol. 34, no. 9, pp. 2408–2421, 2017.

[30] L. R. Foulds and R. L. Graham, "The Steiner problem in phylogeny is NP-complete," *Advances in Applied mathematics*, vol. 3, no. 1, pp. 43–49, 1982.

[31] A. Stamatakis, "RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies," *Bioinformatics*, vol. 30, no. 9, pp. 1312–1313, 2014.

[32] K. Strimmer and A. Von Haeseler, "Quartet puzzling: a quartet maximum-likelihood method for reconstructing tree topologies," *Molecular biology and evolution*, vol. 13, no. 7, pp. 964–969, 1996.

[33] A. Ben-Dor, B. Chor, D. Graur, R. Ophir, and D. Pelleg, "Constructing phylogenies from quartets: elucidation of eutherian superordinal relationships," *Journal of computational Biology*, vol. 5, no. 3, pp. 377–390, 1998.

[34] E. Avni, R. Cohen, and S. Snir, "Weighted quartets phylogenetics," *Systematic biology*, vol. 64, no. 2, pp. 233–242, 2014.

[35] R. Piaggio-Talice, J. G. Burleigh, and O. Eulenstein, "Quartet supertrees," in *Phylogenetic Supertrees*. Springer, 2004, pp. 173–191.

[36] A. Criscuolo, V. Berry, E. J. Douzery, and O. Gascuel, "SDM: a fast distance-based approach for (super) tree building in phylogenomics," *Systematic biology*, vol. 55, no. 5, pp. 740–755, 2006.

[37] S. J. Willson, "Constructing rooted supertrees using distances," *Bulletin of mathematical biology*, vol. 66, no. 6, pp. 1755–1783, 2004.

[38] F.-J. Lapointe and G. Cucumel, "The average consensus procedure: combination of weighted trees containing identical or overlapping sets of taxa," *Systematic Biology*, vol. 46, no. 2, pp. 306–312, 1997.

[39] M. S. Swenson, R. Suri, C. R. Linder, and T. Warnow, "An experimental study of Quartets MaxCut and other supertree methods," *Algorithms for Molecular Biology*, vol. 6, no. 1, p. 7, 2011.

[40] W. H. Day, "Computational complexity of inferring phylogenies from dissimilarity matrices," *Bulletin of Mathematical Biology*, vol. 49, no. 4, pp. 461–467, 1987.

[41] R. Agarwala, V. Bafna, M. Farach, M. Paterson, and M. Thorup, "On the approximability of numerical taxonomy (fitting distances by tree metrics)," *SIAM Journal on Computing*, vol. 28, no. 3, pp. 1073–1085, 1998.

[42] R. Chaudhary, D. Fernández-Baca, and J. G. Burleigh, "MulRF: a software package for phylogenetic analysis using multi-copy gene trees," *Bioinformatics*, vol. 31, no. 3, pp. 432–433, 2014.

[43] D. F. Robinson and L. R. Foulds, "Comparison of phylogenetic trees," *Mathematical biosciences*, vol. 53, no. 1-2, pp. 131–147, 1981.

[44] C. Phillips and T. J. Warnow, "The asymmetric median tree—a new model for building consensus trees," *Discrete Applied Mathematics*, vol. 71, no. 1-3, pp. 311–335, 1996.

[45] T. Warnow, *Computational phylogenetics: an introduction to designing methods for phylogeny estimation*. Cambridge University Press, 2017.

[46] M. S. Swenson, R. Suri, C. R. Linder, and T. Warnow, "SuperFine: fast and accurate supertree estimation," *Systematic biology*, vol. 61, no. 2, p. 214, 2011.

[47] M. Fleischauer and S. Böcker, "Collecting reliable clades using the greedy strict consensus merger," *PeerJ*, vol. 4, p. e2172, 2016.

[48] M. S. Swenson, R. Suri, C. R. Linder, and T. Warnow, "An experimental study of Quartets MaxCut and other supertree methods," *Algorithms for Molecular Biology*, vol. 6, no. 1, p. 7, 2011.

[49] M. S. Swenson, F. Barbançon, T. Warnow, and C. R. Linder, "A simulation study comparing supertree and combined analysis methods using SMIDGen," *Algorithms for Molecular Biology*, vol. 5, no. 1, p. 8, 2010.

[50] W. P. Maddison, "Gene trees in species trees," *Systematic Biology*, vol. 46, no. 3, pp. 523–536, 1997.

[51] D. Mallo, L. de Oliveira Martins, and D. Posada, "SimPhy: phylogenomic simulation of gene, locus, and species trees," *Systematic biology*, vol. 65, no. 2, pp. 334–344, 2015.

[52] S. Mirarab, R. Reaz, M. S. Bayzid, T. Zimmermann, M. S. Swenson, and T. Warnow, "ASTRAL: genome-scale coalescent-based species tree estimation," *Bioinformatics*, vol. 30, no. 17, pp. i541–i548, 2014, special issue for ECCB (European Conference on Computational Biology), 2014.

[53] S. Mirarab and T. Warnow, "Astral-ii: coalescent-based species tree estimation with many hundreds of taxa and thousands of genes," *Bioinformatics*, vol. 31, no. 12, pp. i44–i52, 2015.

[54] C. Zhang, M. Rabiee, E. Sayyari, and S. Mirarab, "ASTRAL-III: polynomial time species tree reconstruction from partially resolved gene trees," *BMC Bioinformatics*, vol. 19, no. 6, p. 153, 2018, special issue for RECOMB-CG 2017.

[55] E. D. Jarvis, S. Mirarab, A. J. Aberer, B. Li, P. Houde, C. Li, S. Y. W. Ho, B. C. Faircloth, B. Nabholz, J. T. Howard, A. Suh, C. C. Weber, R. R. da Fonseca, J. Li, F. Zhang, H. Li, L. Zhou, N. Narula, L. Liu, G. Ganapathy, B. Boussau, M. S. Bayzid, V. Zavidovych, S. Subramanian, T. Gabaldón, S. Capella-Gutiérrez, J. Huerta-Cepas, B. Rekepalli, K. Munch, M. Schierup, B. Lindow, W. C. Warren, D. Ray, R. E. Green, M. W. Bruford, X. Zhan, A. Dixon, S. Li, N. Li, Y. Huang, E. P. Derryberry, M. F. Bertelsen, F. H. Sheldon, R. T. Brumfield, C. V. Mello, P. V. Lovell, M. Wirthlin, M. P. C. Schneider, F. Prosdocimi, J. A. Samaniego, A. M. V. Velazquez, A. Alfaro-Núñez, P. F. Campos, B. Petersen, T. Sicheritz-Ponten, A. Pas, T. Bailey, P. Scofield, M. Bunce, D. M. Lambert, Q. Zhou, P. Perelman, A. C. Driskell, B. Shapiro, Z. Xiong, Y. Zeng, S. Liu, Z. Li, B. Liu, K. Wu, J. Xiao, X. Yinqi, Q. Zheng, Y. Zhang, H. Yang, J. Wang, L. Smeds, F. E. Rheindt, M. Braun, J. Fjeldsa, L. Orlando, F. K. Barker, K. A. Jønsson, W. Johnson, K.-P. Koepfli, S. O'Brien, D. Haussler, O. A. Ryder, C. Rahbek, E. Willerslev, G. R. Graves, T. C. Glenn, J. McCormack, D. Burt, H. Ellegren, P. Alström, S. V. Edwards, A. Stamatakis, D. P. Mindell, J. Cracraft, E. L. Braun, T. Warnow, W. Jun, M. T. P. Gilbert, and G. Zhang, "Whole-genome analyses resolve early branches in the tree of life of modern birds," *Science*, vol. 346, no. 6215, pp. 1320–1331, 2014. [Online]. Available: https://science.sciencemag.org/content/346/6215/1320

[56] F. McMorris, "On the compatibility of binary qualitative taxonomic characters," *Bulletin of Mathematical Biology*, vol. 39, no. 2, pp. 133–138, 1977.

[57] C. Zhang, E. Sayyari, and S. Mirarab, "ASTRAL-III: increased scalability and impacts of contracting low support branches," in *RECOMB International Workshop on Comparative Genomics*. Springer, 2017, pp. 53–75.