Advancing Divide-and-Conquer Phylogeny Estimation using Robinson-Foulds Supertrees*

Xilin Yu[1] [0000−0002−6323−3755], Thien Le[1] [0000−0001−5476−8451], Sarah A. Christensen[1] [0000−0001−5790−6266], Erin K. Molloy[1] [0000−0001−5553−3312], and Tandy Warnow[1] [0000−0001−7717−3514]

University of Illinois at Urbana-Champaign warnow@illinois.edu

Abstract. One of the Grand Challenges in Science is the construction of the *Tree of Life*, an evolutionary tree containing several million species, spanning all life on earth. However, the construction of the Tree of Life is enormously computationally challenging, as all the current most accurate methods are either heuristics for **NP**-hard optimization problems or Bayesian MCMC methods that sample from tree space. One of the most promising approaches for improving scalability and accuracy for phylogeny estimation uses divide-and-conquer: a set of species is divided into overlapping subsets, trees are constructed on the subsets, and then merged together using a "supertree method". Here, we present Exact-RFS-2, the first polynomial-time algorithm to find an optimal supertree of two trees, using the Robinson-Foulds Supertree (RFS) criterion (a major approach in supertree estimation that is related to maximum likelihood supertrees), and we prove that finding the RFS of three input trees is **NP**-hard. We also present GreedyRFS (a greedy heuristic that operates by repeatedly using Exact-RFS-2 on pairs of trees, until all the trees are merged into a single supertree). We evaluate Exact-RFS-2 and GreedyRFS, and show that they have better accuracy than the current leading heuristic for RFS. Exact-RFS-2 and GreedyRFS are available in open source form on Github at github.com/yuxilin51/GreedyRFS.

Keywords: Phylogenomics · Divide-and-conquer · Robinson-Foulds Supertrees.

* Supported by The University of Illinois at Ubana-Champaign

1 Introduction

Supertree construction (i.e., the combination of a collection of trees, each on a potentially different subset of the species, into a tree on the full set of species) is a natural algorithmic problem that has important applications to computational biology; see [7] for a 2004 book on the subject and [9, 10, 14, 16–18, 27, 40] for some of the recent papers on this subject. Supertree methods are particularly important for large-scale phylogeny estimation, where it can be used as a final step in a divide-and-conquer pipeline [48]: the species set is divided into two or more overlapping subsets, unrooted leaf-labelled trees are constructed (possibly recursively) on each subset, and then these subset trees are combined into a tree on the full dataset, using the selected supertree method. Furthermore, provided that optimal supertrees are computed, divide-and-conquer pipelines can be provably *statistically consistent* under stochastic models of evolution: i.e., as the amount of input data (e.g., sequence lengths when estimating gene trees, or number of gene trees when estimating species trees) increases, the probability that the true tree is returned converges to 1 [26, 47].

Unfortunately, the most accurate supertree methods are typically heuristics for **NP**-hard optimization problems [4,16,27,29,31,35,36,40], and so are computationally intensive. However, divide-and-conquer strategies, especially recursive ones, may only need to apply supertree methods to two trees at a time, and hence the computational complexity of supertree estimation given two trees is of interest. One optimization problem where optimal supertrees can be found on two trees is the **NP**-hard Maximum Agreement Supertree (SMAST) problem (also known as the Agreement Supertree Taxon Removal problem), which removes a minimum number of leaves so that the reduced trees have an agreement supertree [14, 17]. Similarly, the Maximum Compatible Supertree (SMCT) problem, which removes a minimum number of leaves so that the reduced trees have a compatibility supertree [5, 6], can also be solved in polynomial time on two trees (and note that SMAST and SMCT are identical when the input trees are fully resolved). Because SMAST and SMCT remove taxa, methods for these optimization problems are not *true supertree methods*, because they do not return a tree on the entire set of taxa. However, solutions to SMAST and SMCT could potentially be used as *constraints* for other supertree methods, where the deleted leaves are added into the computed SMAST or SMCT trees, so as to optimize the desired criterion.

When restricting to methods that return trees on the full set of taxa, much less seems to be understood about finding supertrees on two trees. However, if the two input trees are compatible (i.e., there is a supertree that equals or refines each tree when restricted to the respective leaf set), then finding that compatibility supertree is solvable in polynomial time, using (for example) the well known BUILD algorithm [1], but more efficient algorithms exist (e.g., [3,6]).

Since compatibility is a strong requirement (rarely seen in biological datasets), optimization problems are more relevant. One optimization problem worth discussing is the Maximum Agreement Supertree Edge Contraction problem (which takes as input a set of rooted trees and seeks a minimum number of edges to collapse so that an agreement supertree exists). This problem is **NP**-hard, but the decision problem can be solved in $O((2k)^p kn^2)$ time when the input has k trees and p is the allowed number of number of edges to be collapsed [14]. Note that this analysis means that their algorithm for AST-EC may be exponential even for two trees, when the number of edges that must be collapsed is $\Omega(n)$ (e.g., imagine two caterpillar trees, where one is obtained from the other by moving the left-most leaf to the rightmost position).

In sum, while supertree methods are important and well studied, when restricted to the major optimization problems that do not remove taxa, polynomial time methods do not seem to be available, even for the special case where the input contains just two trees. This restriction has consequences for large-scale phylogeny estimation, as without good supertree methods, divide-and-conquer pipelines are not guaranteed to be statistically consistent, are not fast, and do not have good scalability [47].

In this paper we examine the well known Robinson-Foulds Supertree (RFS) problem [2], which seeks a supertree that minimizes the total Robinson-Foulds [32] distance to the input trees. Although RFS is **NP**-hard [22], it has several desirable properties: it is closely related to maximum likelihood supertrees [38] and, as shown very recently, has good theoretical performance for species tree estimation in the presence of gene duplication and loss [25]. Because of its importance, there are several methods for RFS supertrees, including PluMiST [18], MulRF [8], and FastRFS [44]. A comparison between FastRFS and other supertree methods (MRL [27], ASTRAL, ASTRID [43], PluMiST, and MulRF) on simulated and biological supertree datasets showed that FastRFS matched or improved on the other methods with respect to topological accuracy and RFS criterion scores [44]. Hence, FastRFS is currently the leading method for the RFS optimization problem.

The main contributions of this paper are:

- We prove that RFS is solvable in $O(n^2|X|)$ time for two trees, where n is the number of leaves and X is the number of shared leaves (Theorem 1) and **NP**-hard for three or more trees (Lemma 6).
- We present Exact-RFS-2, a polynomial time algorithm for the RFS problem when given only two source trees, and explore its performance on simulated data, both within a natural divide-and-conquer pipeline and within a greedy heuristic (Section 3). We show that Exact-RFS-2 outperforms FastRFS [44] on two trees, the current most accurate method for RFS, and that GreedyRFS is better than FastRFS for small to moderate numbers of source trees (Section 4).
- We prove that divide-and-conquer pipelines using Exact-RFS-2 are statistically consistent methods for phylogenetic tree estimation (both gene trees and species trees) under standard sequence evolution models (Theorem 2).
- We establish equivalence between RFS and some other supertree problems (Lemma 1).
- We show critical differences between RFS and SMAST/SMCT problems, that establish that methods for SMAST or SMCT cannot provably be used to constrain the search for RFS supertrees (Lemma 23).

The remainder of the paper is organized as follows. In Section 2, we provide terminology and define the optimization problems we consider. We present the Exact-RFS-2 algorithm and establish theory related to the algorithm in Section 3. Our experimental performance study is presented in Section 4, and we conclude in Section 5 with a discussion of trends and future research directions.

The details of the performance study and commands necessary to reproduce the study are omitted from the main paper but available in the appendices; the proofs are also available in [50]. All datasets used in this study are publicly available from prior studies, and the scripts and codes necessary to reproduce the study are available at `http://github.com/yuxilin51/GreedyRFS`.

2 Terminology and Problem Statements

We let $[N] = \{1, 2, \ldots, N\}$ and $\mathcal{A} = \{T_i \mid i \in [N]\}$ denote the input to a supertree problem, where each T_i is a 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 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".

Robinson-Foulds Supertree Each edge e in a tree T defines a bipartition $\pi_e := [A|B]$ of the leaf set, and each tree is defined by the set $C(T) := \{\pi_e \mid e \in E(T)\}$. The *Robinson-Foulds distance* [32] (also called the bipartition distance) between trees T and T' with the same leaf set is $\mathrm{RF}(T, T') := |C(T) \backslash C(T')| + |C(T') \backslash C(T)|$. We extend the definition of RF distance to allow for T and T' to have different leaf sets as follows: $RF(T, T') := RF(T|_X, T'|_X)$, where X is the shared leaf set and $t|_X$ denotes the homeomorphic subtree of t induced by X. Letting $\mathcal{T}_S$ denote the set of all phylogenetic trees such that $L(T) = S$ and $\mathcal{T}_S^B$ denote the binary trees in $\mathcal{T}_S$, then a Robinson-Foulds supertree [2] of a profile $\mathcal{A}$ is a binary tree

$$T_{\mathrm{RFS}} = \underset{T \in \mathcal{T}_S^B}{\operatorname{argmin}} \sum_{i \in [N]} \mathrm{RF}(T, T_i).$$

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 problem** takes as input the profile $\mathcal{A}$ and seeks a Robinson-Foulds (RF) supertree for $\mathcal{A}$, which we denote by $\mathrm{RFS}(\mathcal{A})$.

Split Fit Supertree The Split Fit (SF) Supertree problem was introduced in [49], and is based on optimizing the number of shared splits (i.e., bipartitions) between the supertree and the source trees. For two trees T, T' with the same leaf set, the *split support* is the number of shared bipartitions, i.e., $\mathrm{SF}(T, T') := |C(T) \cap C(T')|$. For trees with different leaf sets, we restrict them to the shared leaf set before calculating the split support. The Split Fit supertree for a profile

$\mathcal{A}$ of source trees, denoted SFS($\mathcal{A}$), is a tree $T_{\text{SFS}} \in \mathcal{T}_S^B$ such that

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

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

Nomenclature for variants of RFS *and* SFS *problems*

- The relaxed versions of the problems where we do not require the output to be binary (i.e., we allow $T \in \mathcal{T}_S$) 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.

Thus, 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 $v \in V(T)$, we let $N_T(v)$ denote the set of neighbors of v in T. A tree T' is a *refinement* of T iff T can be obtained from T' by contracting a set of edges. 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 $\pi_i \in C(T), i = 1, 2$. A bipartition $\pi = [A|B]$ restricted to a subset R is $\pi|_R = [A \cap R | B \cap R]$. For a graph G and a set F of vertices or edges, we use $G + F$ to represent the graph obtained from adding the set F of vertices or edges to G, and $G - F$ is defined for deletions, similarly.

3 Theoretical Results

In this section we establish the main theoretical results, with detailed proofs provided in [50] or in the appendix of the full version on bioRxiv.

3.1 Solving RFS and SFS on two trees

Lemma 1. *Given an input set $\mathcal{A}$ 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*($\mathcal{A}$).*

The main result of this paper is Theorem 1 (correctness is proved later within the main body of the paper, and the running time is established in the Appendix):

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

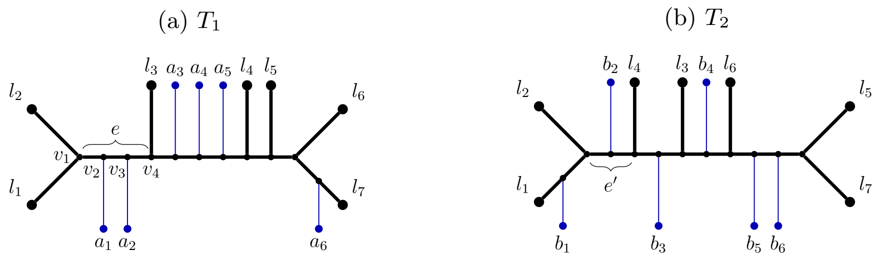


Fig. 1: T_1 and T_2 depicted in (a) and (b) have an overlapping leaf set $X = \{l_1, l_2, \ldots, l_7\}$. Each of $a_1, \ldots, a_6$ and $b_1, \ldots, b_6$ can represent a multi-leaf extra subtree. Using indices to represent the shared leaves, let $\pi = [12|34567]$; then $e_1(\pi) = e$ and $e_2(\pi) = e'$. $P(e)$ is the path from v_1 to v_4, so $w(e) = 3$. $\mathcal{TR}(e) = \{a_1, a_2\}$, $\mathcal{TR}(e') = \{b_2\}$, so $\mathcal{TR}^*(\pi) = \{a_1, a_2, b_2\}$. Let $A = \{1, 2, 3\}$, $B = \{4, 5, 6, 7\}$. Ignoring the trivial bipartitions, we have $\mathcal{BP}(A) = \{[12|34567]\}$ and $\mathcal{BP}(B) = \{[1234|567], [12345|67], [12346|57]\}$. $\mathcal{TRS}(A) = \{a_1, a_2, b_2\}$ and $\mathcal{TRS}(B) = \{b_4, b_5, b_6\}$.

Exact-RFS-2: Polynomial time algorithm for RFS-2-B and SFS-2-B The input to Exact-RFS-2 is a pair of binary trees T_1 and T_2. Let X denote the set of shared leaves. At a high level, Exact-RFS-2 constructs a tree T_{init} that has a central node that is adjacent to every leaf in X and to the root of every "rooted extra subtree" (a term we define below) so that T_{init} contains all the leaves in S. It then modifies T_{init} by repeatedly refining it to add specific desired bipartitions, to produce an optimal Split Fit (and optimal Robinson-Foulds) supertree (Figure 3). The bipartitions that are added are defined by a maximum independent set in a bipartite "weighted incompatibility graph" we compute.

Additional notation Let $\Pi = 2^X$ denote the set of all bipartitions of X; any bipartition that splits a single leaf from the remaining $|X|-1$ leaves will be called "trivial" and the others will be called "non-trivial". 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 refer to $T_i|_X, i = 1, 2$ as **backbone trees** (Figure 2). Recall that we suppress degree-two vertices when restricting a tree T_i to a subset X of the leaves; hence, every edge e in $T_i|_X$ will correspond to an edge or a path in T (see Fig. 1 for an example). We will let $P(e)$ denote the path associated to edge e, and let $w(e) := |P(e)|$ (the number of edges in $P(e)$). Finally, for $\pi \in C(T_i|_X)$, we define $e_i(\pi)$ to be the edge that induces π in $T_i|_X$ (Fig. 1).

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

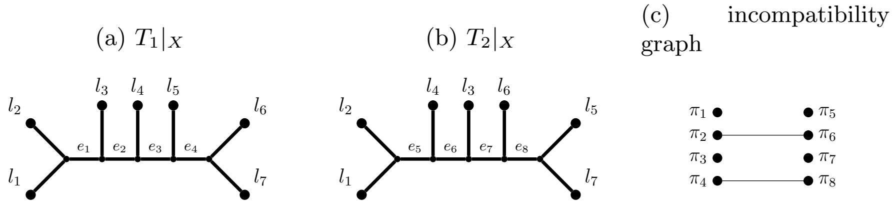


Fig. 2: We show (a) $T_1|_X$, (b) $T_2|_X$, and (c) their incompatibility graph, based on the trees T_1 and T_2 in Figure 1 (without the trivial bipartitions). Each π_i is the bipartition induced by e_i, and the weights for $\pi_1, \ldots, \pi_8$ are $3, 4, 1, 1, 2, 2, 2, 3$, in that order. We note that π_1 and π_5 are the same bipartition, but they have different weights as they 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 $(\pi_1, \pi_3, \pi_5, \pi_7)$ and also π_2, π_8, and so has total weight 15.

We can now define the initial tree T_{init} computed by Exact-RFS-2: T_{init} has a center node that is adjacent to every $x \in X$ and also to the root $r(t)$ for every extra subtree $t \in \text{Extra}(T_1) \cup \text{Extra}(T_2)$. Note that T_{init} has a leaf for every element in S, and that $T_{\text{init}}|_{S_i}$ is a contraction of T_i, formed by collapsing all the edges in the backbone tree $T_i|_X$.

We say that an extra subtree t is **attached to edge** $e \in E(T_i|_X)$ if the root of t is adjacent to an internal node of $P(e)$, and we let $\mathcal{TR}(e)$ denote the set of such extra subtrees attached to edge e. 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 edges in a backbone tree that induce π in either $T_1|_X$ or $T_2|_X$. For example, if both trees T_1 and T_2 contribute extra subtrees to π, then $\mathcal{TR}^*(\pi) := \bigcup_{i \in [2]} \mathcal{TR}(e_i(\pi))$.

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, $\mathcal{BP}_i(Q) := \{[A|B] \in C(T_i|_X) \mid A \subsetneq Q \text{ or } B \subsetneq Q\}$, and $\mathcal{TRS}_i(Q) := \bigcup_{\pi \in \mathcal{BP}_i(Q)} \mathcal{TR}(e_i(\pi))$. Intuitively, $\mathcal{TRS}_i(Q)$ denotes the set of extra subtrees in T_i that are "on the side of Q". By Corollary 2, which appears in the Appendix, 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. Finally, let $\mathcal{BP}(Q) = \mathcal{BP}_1(Q) \cup \mathcal{BP}_2(Q)$, and $\mathcal{TRS}(Q) = \mathcal{TRS}_1(Q) \cup \mathcal{TRS}_2(Q)$. We give an example for these terms in Figure 1.

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 edges between bipartitions if they are incompatible (see [30]). 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. 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 (Figure 2).

We divide $\mathcal{C} = C(T_1) \cup C(T_2)$ into two sets: $\Pi_X = \{[A|B] \in \mathcal{C} \mid A \cap X \neq \emptyset \text{ and } B \cap X \neq \emptyset\}$, and $\Pi_Y = \{[A|B] \in \mathcal{C} \mid A \cap X = \emptyset \text{ or } B \cap X = \emptyset\}$. Intuitively,

Algorithm 1 Exact-RFS-2: Computing a Robinson-Foulds supertree of two trees (see Figure 3)

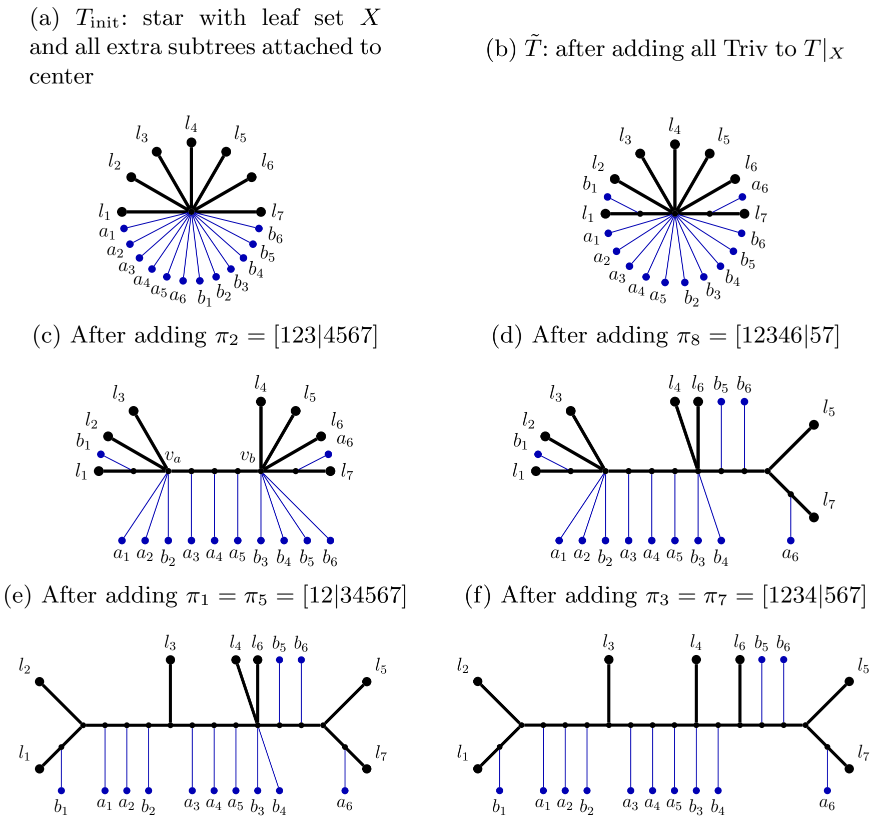
```
    Input: two binary trees $T_1$, $T_2$ with leaf sets $S_1$ and $S_2$ where $S_1 \cap S_2 = X \neq \emptyset$
    Output: a binary supertree $T$ on leaf set $S = S_1 \cup S_2$ that maximizes the split support score
 1: compute $C(T_1|_X)$ and $C(T_2|_X)$
 2: for each $\pi = [A|B] \in C(T_1, T_2, X)$ do
 3:     for $i \in [2]$ do
 4:         compute $\mathcal{TR}(e_i(\pi))$, $w(e_i(\pi))$
 5:     compute $\mathcal{BP}(A)$, $\mathcal{BP}(B)$, $\mathcal{TRS}(A)$, $\mathcal{TRS}(B)$, and $\mathcal{TR}^*(\pi)$,
 6: construct $T$ as a star tree with leaf set $X$ and center vertex $\hat{v}$ and with the root of each $t \in$
    Extra$(T_1) \cup$ 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 $G$
 9: let $I$ be the set of bipartitions associated with vertices in $I^*$
10: for each $\pi = [\{a\}|B] \in$ Triv do
11:     detach all extra subtrees in $\mathcal{TR}^*(\pi)$ from $\hat{v}$ and attach them onto $(\hat{v}, a)$ such that $\mathcal{TR}(e_1(\pi))$
    are attached first with their ordering matching their attachments on $e_1(\pi)$ and $\mathcal{TR}(e_2(\pi))$ are
    attached to the right of all subtrees in $\mathcal{TR}(e_1(\pi))$ with the ordering of them also matching their
    attachments on $e_2(\pi)$
                                                                               ▷ let $\tilde{T} = T$ after for loop
12: $H(\hat{v}) =$ NonTriv, set $sv(\pi) = \hat{v}$ for all $\pi \in$ NonTriv
13: for each $\pi \in$ NonTriv $\cap I$ (in any order) do
14:     $T \leftarrow$ Refine$(T, \pi, H, sv)$                                    ▷ let $T^* = T$ after for loop
15: arbitrarily refine $T$ to make it a binary tree
16: return $T$
```

$\mathit{\Pi}_X$ is the set of bipartitions from the input trees that are induced by edges in the minimal subtree of T_1 or T_2 spanning X, and $\mathit{\Pi}_Y$ are all the other input tree bipartitions. We define $p_X(\cdot)$ and $p_Y(\cdot)$ on trees $T \in \mathcal{T}_S$ by:

$$p_X(T) = \sum_{i\in[2]} |C(T|_{S_i}) \cap C(T_i) \cap \mathit{\Pi}_X|, \quad p_Y(T) = \sum_{i\in[2]} |C(T|_{S_i}) \cap C(T_i) \cap \mathit{\Pi}_Y|.$$

Note that $p_X(T)$ and $p_Y(T)$ decompose the split support score of T into the score contributed by bipartitions in $\mathit{\Pi}_X$ and the score contributed by bipartitions in $\mathit{\Pi}_Y$; 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 independently and we can use this observation to refine T_{init} so that it achieves the optimal total score.

Overview of Exact-RFS-2 Exact-RFS-2 (Algorithm 1) has 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 Algorithm 1 and Algorithm 2. In the initial construction phase (line 6), it constructs a tree T_{init} (as described earlier), and we note that T_{init} maximizes $p_Y(\cdot)$ score (Lemma 2). In the refinement phase (lines 7–14), it refines T_{init} so that it attains the maximum $p_X(\cdot)$ score. In the last phase (line 15), it arbitrarily refines T to make it binary. The refinement phase begins with the construction of a weighted incompatibility graph G of $T_1|_X$ and $T_2|_X$ (see Figure 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. Finally, it uses these bipartitions of X in I to refine T_{init} to achieve the optimal $p_X(\cdot)$ score, by repeatedly applying Algorithm 2 for each $\pi \in I$ (and we note that the order does not matter). See Figure 3 for an example of Exact-RFS-2 given two input source trees.


l_1 l_2 l_3 l_4 l_5 l_6 l_7
a_1 a_2 a_3 a_4 a_5 a_6
b_1 b_2 b_3 b_4 b_5 b_6
v_a v_b

(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 $\pi_3 = \pi_7 = [1234|567]$

Fig. 3: Algorithm 1 working on T_1 and T_2 from Figure 1 as source trees; the indices of leaves in $X = \{l_1, l_2, \ldots, l_7\}$ represent the leaves and the notation of $\pi_1, \ldots, \pi_8$ is from Figure 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 (c) 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 (c), b_3 can only attach to the left side because for $A' = \{l_1, l_2, l_3, l_4, l_6\}$, $[124|3567] \in \mathcal{BP}(A')$ and thus $b_3 \in \mathcal{TRS}(A')$.

Algorithm 2 refines the given tree T on leaf set S with bipartitions on X from $C(T_1, T_2, X) \setminus C(T|_X)$. Given bipartition $\pi = [A|B]$ on X, Algorithm 2 produces 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

Algorithm 2 Refine

```
Input: a tree $T$ on leaf set $S$, a nontrivial bipartition $\pi = [A|B]$ of $X$, two data structures $H$ and $sv$
Output: a tree $T'$ which is a refinement of $T$ such that for both $i = 1, 2$, $C(T'|_{S_i}) = C(T|_{S_i}) \cup \{\pi' \in C(T_i) \mid \pi'|_X = \pi\}$
$v \leftarrow sv(\pi)$
$T' \leftarrow T + v_a + v_b + (v_a, v_b)$
compute $N_A := \{u \in N_T(v) \mid \exists a \in A$ s.t. $u$ can reach $a$ in $T - v\}$ and $N_B := \{u \in N_T(v) \mid \exists b \in B$ s.t. $u$ can reach $b$ in $T - v\}$.
for each $u \in N_A \cup N_B$ do
    if $u \in N_A$ then connect $u$ to $v_a$
    else connect $u$ to $v_b$
detach all extra subtrees in $\mathcal{TR}^*(\pi)$ from $v$ and attach them onto $(v_a, v_b)$ such that $\mathcal{TR}(e_1(\pi))$ are attached first with their ordering matching their attachments on $e_1(\pi)$ and $\mathcal{TR}(e_2(\pi))$ are attached to the right of all subtrees in $\mathcal{TR}(e_1(\pi))$ with the ordering of them also matching their attachments on $e_2(\pi)$
for each $t \in \mathcal{TRS}(A)$ do
    if $t$ is attached to $v$, detach it and attach to $v_a$
for each $t \in \mathcal{TRS}(B)$ do
    if $t$ is attached to $v$, detach it and attach to $v_b$
for each remaining extra subtree attached to $v$ do
    detach it from $v$ and attach it to either $v_a$ or $v_b$
$H(v_a) \leftarrow \emptyset, H(v_b) \leftarrow \emptyset$
for each $\pi' \in H(v)$ do
    if $\pi' \in \mathcal{BP}(A)$ then
        $sv(\pi') = v_a$, $H(v_a) \leftarrow H(v_a) \cup \{\pi'\}$
    else if $\pi' \in \mathcal{BP}(B)$ then
        $sv(\pi') = v_b$, $H(v_b) \leftarrow H(v_b) \cup \{\pi'\}$
    else
        discard $\pi'$
return $T' = T' - v$
```

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 vertices in N_A adjacent to v_a and 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 four different cases 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)
- 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

Algorithms 1 and 2 also use two data structures (functions) H and sv: (1) For a given node $v \in V(T)$, $H(v) \subseteq C(T_1, T_2, X)$ is the set of bipartitions of X that can be added to $T|_X$ by refining $T|_X$ at v, and (2) Given $\pi \in C(T_1, T_2, X)$, $sv(\pi) = v$ means $\exists T'$, a refinement of T at v, so that $C(T'|_X) = C(T|_X) \cup \{\pi\}$.

Lemma 2. *For any tree $T \in \mathcal{T}_S$, $p_Y(T) \leq |\Pi_Y|$. In particular, let T_{init} be the tree defined in line 6 of Algorithm 1. Then, $p_Y(T_{\text{init}}) = |\Pi_Y|$.*

Lemma 2 formally states that the tree T_{init} we build in line 6 of Exact-RFS-2 (Algorithm 1) maximizes the $p_Y(\cdot)$ score. This lemma is true because there are only $|\Pi_Y|$ bipartitions that can contribute to $p_Y(\cdot)$ and T_{init} contains all of them by construction. We define the function $w^* : \Pi \to \mathbb{N}_{\geq 0}$ as follows:

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

For any set F of bipartitions, we let $w^*(F) = \sum_{\pi \in F} w^*(\pi)$.

Lemma 3. *Let $\pi = [A|B] \in \Pi$. Let $T \in \mathcal{T}_S$ be any tree with leaf set 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, $p_X(T') - p_X(T) \leq w^*(\pi)$.*

Lemma 4. *For any compatible set $F \subseteq \Pi$, let $T \in \mathcal{T}_S$ be any tree with leaf set S such that $C(T|_X) = F$. Then $p_X(T) \leq w^*(F)$.*

Lemma 3 shows that $w^*(\pi)$ represents the maximum potential increase in $p_X(\cdot)$ as a result of adding bipartition π to $T|_X$. The proof of Lemma 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 $w^*(\pi)$ more bipartitions get included in $C(T|_{S_1})$ or $C(T|_{S_2})$ so that they contribute to the increase of $p_X(T)$. The proof of Lemma 4 uses Lemma 3 repeatedly by adding the compatible bipartitions to the tree in an arbitrary order.

Proposition 1. *Let $\tilde{T}$ be the tree constructed after line 11 of Algorithm 1, then $p_X(\tilde{T}) = w^*(\text{Triv})$.*

The proof naturally follows by construction (Line 8 of Algorithm 1), and implies that the algorithm adds the trivial bipartitions of X (which are all in I) to $T|_X$ so that $p_X(T)$ reaches the full potential of adding those trivial bipartitions.

Lemma 5. *Let T be a supertree computed within Algorithm 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 2 with supertree T, bipartition π, and the auxiliary data structures H and sv. Then, $p_X(T') - p_X(T) = w^*(\pi)$.*

The idea for the proof of Lemma 5 is that for any non-trivial bipartition $\pi \in I$ of X to be added to $T|_X$, Algorithm 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 $P(e_2(\pi))$, which is not in $T|_{S_1}$ or $T|_{S_2}$ before the refinement, becomes 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 $w^*(\pi)$.

Proposition 2. *Let G be the weighted incompatibility graph on $T_1|_X$ and $T_2|_X$, and let I be the set of bipartitions associated with vertices in I^*, which is a maximum weight independent set of G. Let F be any compatible subset of $C(T_1, T_2, X)$. Then $w^*(I) \geq w^*(F)$.*

We now restate and prove Theorem 1:

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

Proof. 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 1. Fix arbitrary $T \in \mathcal{T}_S$ and let $F = C(T|_X)$. Then by Lemma 4, $p_X(T) \leq w^*(F)$. We know that $w^*(\pi) = 0$ for any $\pi \notin C(T_1, T_2, X)$, so $w^*(F) = w^*(F \cap C(T_1, T_2, X))$ and thus $p_X(T) \leq w^*(F \cap C(T_1, T_2, X))$. 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 Proposition 2. Then $p_X(T) \leq w^*(I)$. Since $\text{Triv} \subseteq C(T_1|_X) \cap C(T_2|_X) \subseteq I$, we have $I = (\text{NonTriv} \cap I) \cup (\text{Triv} \cap I) = (\text{NonTriv} \cap I) \cup \text{Triv}$. Therefore, by Proposition 1 and Lemma 5, we have

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

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

From Lemma 2 and the fact that a refinement of a tree never decreases $p_X(\cdot)$ and $p_Y(\cdot)$, we also know that $p_Y(T^*) \geq p_Y(T_{\text{init}}) \geq p_Y(T)$ for any tree $T \in \mathcal{T}_S$. Since for any $T \in \mathcal{T}_S$, $\text{SF}(T, \mathcal{A}) = p_X(T) + p_Y(T)$, 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-B (Corollary 1). If T^* is not binary, Algorithm 1 arbitrarily resolves every high degree node in T^* until it is a binary tree and then returns a tree that achieves the maximum split support score among all binary trees of leaf set S. See the Appendix for the running time analysis. □

Corollary 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-B *can be solved in $O(n^2|X|)$ time, where $n := \max\{|S_1|, |S_2|\}$.*

Lemma 6. RFS-3*,* SFS-3*, and* Relax–SFS-3 *are all* **NP***-hard.*

3.2 DACTAL-Exact-RFS-2

Let Φ be a model of evolution (e.g., GTR) for which statistically consistent methods exist, and we have some data (e.g., sequences) generated by the model and wish to construct a tree. We construct an initial estimate of the tree, divide the dataset into two overlapping subsets (by removing an edge e, letting X be the set of the p nearest leaves to e, and letting the subsets be $A \cup X$ and $B \cup X$, where $\pi_e = [A|B]$), re-estimate trees on the subsets (perhaps using a recursive approach that is statistically consistent), and then combine the trees together using Exact-RFS-2. We call this the DACTAL-Exact-RFS-2 pipeline, due to its similarity to the DACTAL pipeline [26].

Before we prove that DACTAL-Exact-RFS-2 can enable statistically consistent pipelines, we begin with some definitions. Given a tree T and an internal edge e in T, the deletion of the edge e and its endpoints defines four subtrees. A **short quartet around** e is a set of four leaves, one from each subtree, selected to be closest to the edge. Note that due to ties, there can be multiple short quartets around some edges. The set of short quartets for a tree T is the set of all short quartets around the edges of T. The **short quartet trees of** T is the set of quartet trees on short quartets induced by T. It is well known that the short quartet trees of a tree T define T, and furthermore T can be computed from this set in polynomial time [11–13].

Lemma 7. *Let T be a binary tree on leaf set S and let $A, B \subseteq S$. Let $T_A = T|_A$ and $T_B = T|_B$ (i.e., T_A and T_B are correct induced subtrees). If every short quartet tree is induced in T_A or in T_B, then T is the unique compatibility supertree for T_A and T_B and Exact-2-RFS(T_A, T_B) $= T$.*

Proof. Because T_A and T_B are true trees, it follows that T is a compatibility supertree for T_A and T_B. Furthermore, because every short quartet tree appears in at least one of these trees, T is the unique compatibility supertree for T_A and T_B (by results from [12, 13], mentioned above). Finally, because T is a compatibility supertree, the RFS score of T with respect to T_A, T_B is 0, which is the best possible. Since Exact-2-RFS solves the RFS problem on two binary trees, Exact-2-RFS returns T given input T_A and T_B. □

Thus, Exact-2-RFS is guaranteed to return the true tree when given two correct trees that have sufficient overlap (in that all short quartets are included). We continue with proving that these pipelines are statistically consistent.

Theorem 2. *The DACTAL-Exact-RFS-2 pipeline is statistically consistent under any model Φ for which statistically consistent tree estimation methods exist.*

Proof. Let Φ be the sequence evolution model. To establish statistical consistency of the pipeline, we need to prove that as the amount of data increases the tree that is returned by the pipeline converge to the true tree. Hence, let F be the method used to compute the starting tree and let G be the method used to compute the subset trees. Because F is statistically consistent under Φ, it follows that as the amount of data increases, the starting tree computed by F will converge to the true tree T. Now consider the decomposition into two sets produced by the algorithm, when applied to the true tree. Let e be the edge that is deleted and let the four subtrees around e have leaf sets A_1, A_2, B_1, and B_2. Note in particular that all the leaves appearing in any short quartet around e are placed in the set X. Now, subset trees are computed using G on $A_1 \cup A_2 \cup X$ and $B_1 \cup B_2 \cup X$, which we will refer to as T_A and T_B, respectively. Since G is statistically consistent, as the amount of data increases, T_A converges to the true tree on its leaf set ($T|_{L(T_A)}$) and T_B converges to the true tree on its leaf set

($T|_{L(T_B)}$). When T_A and T_B are equal to the true trees on their leaf sets, then every short quartet tree of T is in T_A or T_B, and by Lemma 7, T is the only compatibility supertree for T_A and T_B and Exact-2-RFS(T_A, T_B) returns T. □

Hence, DACTAL+Exact-2-RFS is statistically consistent under all standard molecular sequence evolution models and also under the MSC+GTR model [33, 45] where gene trees evolve within species trees under the multi-species coalescent model (which addresses gene tree discordance due to incomplete lineage sorting [19]) and then sequences evolve down each gene tree under the GTR model.

Note that all that is needed for F and G to guarantee that the pipeline is statistically consistent is that they should be statistically consistent under Φ. However, for the sake of improving empirical performance, F should be fast so that it can run on the full dataset but G can be more freely chosen, since it will only be run on smaller datasets. Indeed, the user can specify the size of the subsets that are analyzed, with smaller datasets enabling the use of more computationally intensive methods.

For example, when estimating trees under the GTR [42] model, F could be a fast but statistically consistent distance-based method such as neighbor joining [34] and G could be RAxML [37], a leading maximum likelihood method. For the MSC+GTR model, F and G could be polynomial time summary methods (i.e., methods that estimate the species tree by combining gene trees), with F being ASTRID [43] (a very fast summary method) and G being ASTRAL [23, 24, 51], which is slower than ASTRID but often more accurate. However, if the subsets are chosen to be very small, then other choices for G include StarBeast2 [28], a Bayesian method for co-estimating gene trees and species trees under the MSC+GTR model.

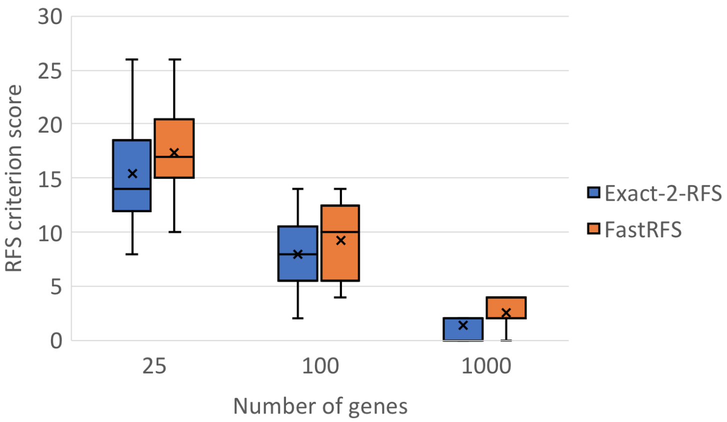


Fig. 4: Results for Experiment 1: Exact-2-RFS has better RFS criterion scores than FastRFS (lower is better) in ILS-based species tree estimation (using ASTRAL-III [51], for 501 species with varying numbers of genes.

4 Experiments and Results

We performed two experiments: Experiment 1, where we used Exact-2-RFS within a divide-and-conquer strategy for large scale phylogenomic species tree estimation where gene trees differ from the species tree due to Incomplete Lineage Sorting (ILS), and Experiment 2, where we used Exact-2-RFS as part of a greedy heuristic, GreedyRFS, for large-scale supertree estimation.

4.1 Experiment 1: Phylogenomic species tree estimation

In this experiment, the input is a set of gene trees that can differ from the species tree due to Incomplete Lineage Sorting [19], ASTRAL [23, 24, 51] is used to construct species trees on the two overlapping subsets in the DACTAL pipeline described above, and the two overlapping estimated species trees are then merged together using either Exact-2-RFS or FastRFS. Because the divide-and-conquer strategy produces two source trees, the RFS criterion score for Exact-2-RFS cannot be worse than the score obtained by FastRFS; here we examine the degree of improvement. The simulation protocol produced datasets with high variability (especially for small numbers of genes), so that there was substantial range in the optimal criterion scores for 25 and 100 genes (Figure 4). On average, Exact-2-RFS produces better RFS scores than FastRFS for all numbers of genes, and strictly dominates FastRFS for 1000 genes (Fig. 4), showing that divide-and-conquer pipelines are improved using Exact-2-RFS compared to FastRFS.

4.2 Experiment 2: Exploring GreedyRFS for supertree estimation

We developed GreedyRFS, a greedy heuristic that takes a profile $\mathcal{A}$ as input, and then merges pairs of trees until all the trees are merged into a single tree. The choice of which pair to merge follows the technique used in SuperFine [41] 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 [15]). Thus, GreedyRFS is identical to Exact-2-RFS when the profile has only two trees.

We use a subset of the SMIDgen [39] datasets with 500 species and varying numbers of source trees (each estimated using maximum likelihood heuristics) that have been used to evaluate supertree methods in several studies [27, 39–41, 44]. See Appendix (in the full version of the paper on arXiv) for full details of this study.

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 the same as Exact-2-RFS for two source trees, and so is guaranteed optimal for this case), but even when the number of source trees was greater than two, GreedyRFS dominated FastRFS in terms of criterion score, provided that the number of source trees was not too large (Fig. 5).

This establishes that the advantage in criterion score is not limited to the case of two source trees, suggesting that using Exact-2-RFS within GreedyRFS (or some other heuristics) may be useful for supertree estimation more generally.

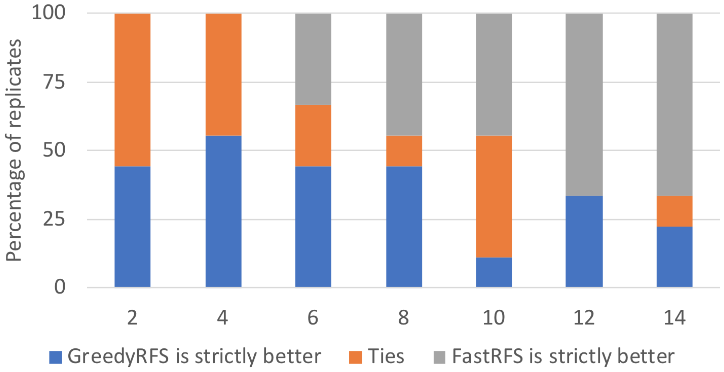


Fig. 5: Results for Experiment 2: The percentage of datasets (y-axis) that each method (FastRFS and GreedRFS) ties with or is strictly better than the other in terms of RFS criterion score is shown for varying numbers of source trees (x-axis), based on nine replicate supertree 500-leaf 20% scaffold datasets (from [41]) .

5 Conclusions

The main contribution of this paper is Exact-2-RFS, a polynomial time algorithm for the Robinson-Foulds Supertree (RFS) of two trees that enables divide-and-conquer pipelines to be provably statistically consistent under sequence evolution models (e.g., GTR [42] and MSC+GTR [33]). Our experimental study showed that Exact-2-RFS dominates the leading RFS heuristic, FastRFS, when used within divide-and-conquer species tree estimation using genome-scale datasets, a problem of increasing importance in biology. We also showed that a greedy heuristic using Exact-2-RFS produced better criterion scores than FastRFS when the number of source trees was small to moderate, showing the potential for Exact-2-RFS to be useful in other settings. Overall, 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.

This study suggests several directions for future work. For example, although we showed that Exact-2-RFS produced better RFS criterion scores than FastRFS when used in divide-and-conquer species tree estimation (and similarly GreedyRFS was better than FastRFS for small numbers of source trees in supertree estimation), additional studies are needed to explore its performance, including additional datasets (both simulated and biological datasets) and other leading supertree methods. Similarly, other heuristics using Exact-2-RFS besides GreedyRFS should be developed and studied.

Acknowledgments

This research was supported by NSF grants 1458652, 1513629, and 1535977 to TW, by the NSF Graduate Research Fellowship DGE-1144245 to EKM, and by the Ira and Debra Cohen Fellowship in Computer Science at the University of Illinois to SAC and EKM. This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois.

References

1. Aho, A.V., Sagiv, Y., Szymanski, T.G., Ullman, J.D.: Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. SIAM Journal on Computing **10**(3), 405–421 (1981)
2. Bansal, M.S., Burleigh, J.G., Eulenstein, O., Fernández-Baca, D.: Robinson-foulds supertrees. Algorithms for Molecular Biology **5**(1), 18 (2010)
3. Baste, J., Paul, C., Sau, I., Scornavacca, C.: Efficient FPT algorithms for (strict) compatibility of unrooted phylogenetic trees. Bulletin of Mathematical biology **79**(4), 920–938 (2017)
4. Baum, B.R.: Combining trees as a way of combining data sets for phylogenetic inference, and the desirability of combining gene trees. Taxon pp. 3–10 (1992)
5. Berry, V., Nicolas, F.: Maximum agreement and compatible supertrees. In: Annual Symposium on Combinatorial Pattern Matching. pp. 205–219. Springer (2004)
6. Berry, V., Nicolas, F.: Improved parameterized complexity of the maximum agreement subtree and maximum compatible tree problems. IEEE/ACM Transactions on Computational Biology and Bioinformatics **3**(3), 289–302 (2006)
7. Bininda-Emonds, O.R.: Phylogenetic supertrees: combining information to reveal the tree of life, vol. 4. Springer Science & Business Media (2004)
8. Chaudhary, R., Fernández-Baca, D., Burleigh, J.G.: MulRF: a software package for phylogenetic analysis using multi-copy gene trees. Bioinformatics **31**(3), 432–433 (2014)
9. Cotton, J.A., Wilkinson, M.: Majority-rule supertrees. Systematic biology **56**(3), 445–452 (2007)
10. De Oliveira Martins, L., Mallo, D., Posada, D.: A Bayesian supertree model for genome-wide species tree reconstruction. Systematic biology **65**(3), 397–416 (2016)
11. Erdős, P., Steel, M.A., Szekely, L.A., Warnow, T.J.: Local quartet splits of a binary tree infer all quartet splits via one dyadic inference rule. Computers and Artifical Intelligence **16**(2), 217–227 (1997)
12. Erdős, P.L., Steel, M.A., Székely, L.A., Warnow, T.J.: A few logs suffice to build (almost) all trees (I). Random Structures & Algorithms **14**(2), 153–184 (1999)
13. Erdös, P.L., Steel, M.A., Székely, L.A., Warnow, T.J.: A few logs suffice to build (almost) all trees (II). Theoretical Computer Science **221**(1-2), 77–118 (1999)
14. Fernández-Baca, D., Guillemot, S., Shutters, B., Vakati, S.: Fixed-parameter algorithms for finding agreement supertrees. SIAM Journal on Computing **44**(2), 384–410 (2015)
15. Fleischauer, M., Böcker, S.: Collecting reliable clades using the greedy strict consensus merger. PeerJ **4**, e2172 (2016)
16. Fleischauer, M., Böcker, S.: Bad clade deletion supertrees: a fast and accurate supertree algorithm. Molecular biology and evolution **34**(9), 2408–2421 (2017)
17. Guillemot, S., Berry, V.: Fixed-parameter tractability of the maximum agreement supertree problem. IEEE/ACM Transactions on Computational Biology and Bioinformatics **7**(2), 342–353 (2010)
18. Kupczok, A.: Split-based computation of majority-rule supertrees. BMC evolutionary biology **11**(1), 205 (2011)
19. Maddison, W.P.: Gene trees in species trees. Systematic Biology **46**(3), 523–536 (1997)
20. Mallo, D., de Oliveira Martins, L., Posada, D.: SimPhy: phylogenomic simulation of gene, locus, and species trees. Systematic biology **65**(2), 334–344 (2015)

21. McMorris, F.: On the compatibility of binary qualitative taxonomic characters. Bulletin of Mathematical Biology **39**(2), 133–138 (1977)
22. McMorris, F., Steel, M.A.: The complexity of the median procedure for binary trees. In: New Approaches in Classification and Data Analysis, pp. 136–140. Springer (1994)
23. Mirarab, S., Reaz, R., Bayzid, M.S., Zimmermann, T., Swenson, M.S., Warnow, T.: ASTRAL: genome-scale coalescent-based species tree estimation. Bioinformatics **30**(17), i541–i548 (2014), special issue for ECCB (European Conference on Computational Biology), 2014
24. Mirarab, S., Warnow, T.: ASTRAL-II: coalescent-based species tree estimation with many hundreds of taxa and thousands of genes. Bioinformatics **31**(12), i44–i52 (2015), special issue for ISMB 2015
25. Molloy, E.K., Warnow, T.: FastMulRFS: fast and accurate species tree estimation under generic gene duplication and loss models. Bioinformatics (2020), to appear, special issue for ISMB 2020
26. Nelesen, S., Liu, K., Wang, L.S., Linder, C.R., Warnow, T.: DACTAL: divide-and-conquer trees (almost) without alignments. Bioinformatics **28**(12), i274–i282 (2012), special issue for ISMB 2012
27. Nguyen, N., Mirarab, S., Warnow, T.: MRL and SuperFine+MRL: new supertree methods. Algorithms for Molecular Biology **7**(1), 3 (2012)
28. Ogilvie, H.A., Heled, J., Xie, D., Drummond, A.J.: Computational performance and statistical accuracy of *BEAST and comparisons with other methods. Systematic Biology **65**(3), 381–396 (2016)
29. Page, R.D.: Modified mincut supertrees. In: Proceedings WABI (International Workshop on Algorithms in Bioinformatics). pp. 537–551. Springer-Verlag (2002)
30. Phillips, C., Warnow, T.J.: The asymmetric median tree—a new model for building consensus trees. Discrete Applied Mathematics **71**(1-3), 311–335 (1996)
31. Ragan, M.A.: Phylogenetic inference based on matrix representation of trees. Molecular Phylogenetics and Evolution **1**(1), 53–58 (1992)
32. Robinson, D.F., Foulds, L.R.: Comparison of phylogenetic trees. Mathematical biosciences **53**(1-2), 131–147 (1981)
33. Roch, S., Nute, M., Warnow, T.: Long-branch attraction in species tree estimation: Inconsistency of partitioned likelihood and topology-based summary methods. Systematic Biology **68**(2), 281–297 (09 2018). https://doi.org/10.1093/sysbio/syy061, `https://doi.org/10.1093/sysbio/syy061`
34. Saitou, N., Nei, M.: The neighbor-joining method: a new method for reconstructing phylogenetic trees. Molecular biology and evolution **4**(4), 406–425 (1987)
35. Semple, C., Steel, M.: A supertree method for rooted trees. Discrete Applied Mathematics **105**(1-3), 147–158 (2000)
36. Snir, S., Rao, S.: Quartets MaxCut: a divide and conquer quartets algorithm. IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB) **7**(4), 704–718 (2010)
37. Stamatakis, A.: RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. Bioinformatics **30**(9), 1312–1313 (2014)
38. Steel, M., Rodrigo, A.: Maximum likelihood supertrees. Systematic biology **57**(2), 243–250 (2008)
39. Swenson, M.S., Barbançon, F., Warnow, T., Linder, C.R.: A simulation study comparing supertree and combined analysis methods using SMIDGen. Algorithms for Molecular Biology **5**(1), 8 (2010)

40. Swenson, M.S., Suri, R., Linder, C.R., Warnow, T.: An experimental study of Quartets MaxCut and other supertree methods. Algorithms for Molecular Biology **6**(1), 7 (2011)
41. Swenson, M.S., Suri, R., Linder, C.R., Warnow, T.: SuperFine: fast and accurate supertree estimation. Systematic Biology **61**(2), 214 (2011)
42. Tavaré, S.: Some probabilistic and statistical problems in the analysis of DNA sequences. In: Miura, R. (ed.) Lectures on mathematics in the life sciences–DNA sequences, vol. 17, pp. 57–86. American Mathematical Society, Providence, RI (1986)
43. Vachaspati, P., Warnow, T.: ASTRID: accurate species trees from internode distances. BMC genomics **16**(10), S3 (2015)
44. Vachaspati, P., Warnow, T.: FastRFS: fast and accurate robinson-foulds supertrees using constrained exact optimization. Bioinformatics **33**(5), 631–639 (2016)
45. Warnow, T.: Concatenation analyses in the presence of incomplete lineage sorting. PLOS Currents Tree of Life (2015). https://doi.org/10.1371/currents.tol.8d41ac0f13d1abedf4c4a59f5d17b1f7
46. Warnow, T.: Computational Phylogenetics: an introduction to designing methods for phylogeny estimation. Cambridge University Press (2017)
47. Warnow, T.: Divide-and-conquer tree estimation: Opportunities and challenges. In: Bioinformatics and Phylogenetics: Seminal contributions of Bernard Moret, pp. 121–150. Springer (2019)
48. Wilkinson, M., Cotton, J.A.: Supertree methods for building the tree of life: divide-and-conquer approaches to large phylogenetic problems. SYSTEMATICS ASSOCIATION SPECIAL VOLUME **72**, 61 (2007)
49. Wilkinson, M., Cotton, J.A., Creevey, C., Eulenstein, O., Harris, S.R., Lapointe, F.J., Levasseur, C., Mcinerney, J.O., Pisani, D., Thorley, J.L.: The Shape of Supertrees to Come: Tree Shape Related Properties of Fourteen Supertree Methods. Systematic Biology **54**(3), 419–431 (2005)
50. Yu, X.: Computing Robinson-Foulds supertree for two trees. Master's thesis, University of Illinois at Urbana-Champaign, Urbana, IL (2019), available online at `http://hdl.handle.net/2142/105698`
51. Zhang, C., Rabiee, M., Sayyari, E., Mirarab, S.: ASTRAL-III: polynomial time species tree reconstruction from partially resolved gene trees. BMC Bioinformatics **19**(6), 153 (2018), special issue for RECOMB-CG 2017

6 Notation

6.1 Standard Notation

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

6.2 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 $|C(T|_X) \setminus C(T'|_X)| + |C(T'|_X) \setminus C(T|_X)|$, 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 Robinson-Foulds (RF) score of tree T with respect to a profile $\mathcal{A}$, 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 $\mathcal{A}$, calculated by $\sum_{i \in [N]} \mathrm{SF}(T, T_i)$.

6.3 Notation for Exact-RFS-2

We assume T_1 and T_2 are two binary trees with leaf set S_1 and S_2.

- $X = S_1 \cap S_2$: the shared leaf set
- $S = S_1 \cup S_2$: the union of leaf set
- $\mathit{\Pi} = 2^X$: the set of all bipartitions of X
- $C(T_1, T_2, X) := C(T_1|_X) \cup C(T_2|_X)$: the set of bipartitions of X in the backbone trees of T_1 and T_2
- $P(e)$: the path in T_i from which a backbone edge $e \in T_i|_X$ is obtained by suppressing degree-two vertices
- $w(e) = |P(e)|$: the weight of edge $e \in E(T_i|_X)$
- $e_i(\pi)$: the edge in T_i that induces the bipartition $\pi \in C(T_1, T_2, X)$
- $w^*(\pi) := \sum_{i \in [2]} w(e_i(\pi))$: the weight of bipartition $\pi \in \mathit{\Pi}$, which is the sum of weights of edges in $T_i|_X$ that induces π

- $T_i - T_i|_X$: the subgraph obtained by deleting all vertices and edges of the subgraph of T_i induced on X
- $\text{Extra}(T_i) := \{t \mid t \text{ is a component in } T_i - T_i|_X\}$: the set of extra subtrees of T_i
- $r(t)$: the root of extra subtree t, which is the unique vertex in $V(t)$ that is adjacent to a vertex in the 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 internal vertices of $P(e)$
- $\mathcal{TR}^*(\pi) := \bigcup_{i \in [2]} \mathcal{TR}(e_i(\pi))$: the set of extra subtrees that are attached to edges in $T_i|_X$ that induce π
- $\mathcal{BP}_i(Q) := \{[A|B] \in C(T_i|_X) \mid A \subsetneq Q \text{ or } B \subsetneq Q\}$: the set of bipartitions in $C(T_i|_X)$ with one side being a strict subset of Q
- $\mathcal{BP}(Q) := \mathcal{BP}_1(Q) \cup \mathcal{BP}_2(Q)$: the set of bipartitions in $C(T_1, T_2, X)$ with one side being a strict subset of Q
- $\mathcal{TRS}_i(Q) := \bigcup_{\pi \in \mathcal{BP}_i(Q)} \mathcal{TR}(e_i(\pi))$: the set of extra subtrees attached to the edges in $T_i|_X$ inducing bipartitions in $\mathcal{BP}_i(Q)$
- $\mathcal{TRS}(Q) := \mathcal{TRS}_1(Q) \cup \mathcal{TRS}_2(Q)$: the set of extra subtrees attached to edges in $T_1|_X$ and $T_2|_X$ inducing bipartitions in $\mathcal{BP}(Q)$, equivalent to $\bigcup_{\pi \in \mathcal{BP}(Q)} \mathcal{TR}^*(\pi)$
- $\mathcal{C} = C(T_1) \cup C(T_2)$: the set of bipartitions from input trees T_1 and
- $\Pi_X := \{[A|B] \in \mathcal{C} \mid A \cap X \neq \emptyset \text{ and } B \cap X \neq \emptyset\}$: the set of bipartitions from input trees that are induced by edges on paths connecting vertices of X
- $\Pi_Y := \{[A|B] \in \mathcal{C} \mid A \cap X = \emptyset \text{ or } B \cap X = \emptyset\}$: the set of bipartitions from input trees that are induced by edges not on paths connecting vertices of X, i.e., edges in an extra subtree or connecting an extra subtree to the backbone trees
- $p_X(T) := \sum_{i \in [2]} |C(T|_{S_i}) \cap C(T_i) \cap \Pi_X|$: the split support score of T contributed by bipartitions in Π_X
- $p_Y(T) := \sum_{i \in [2]} |C(T|_{S_i}) \cap C(T_i) \cap \Pi_Y|$: the split support score of T contributed by bipartitions in Π_Y

7 General Theorems and Lemmas on Trees and Bipartitions

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

Theorem 3. *[21] 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 2. *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 8. *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 $e \in P(e')$.*

Proof. 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 vertices.

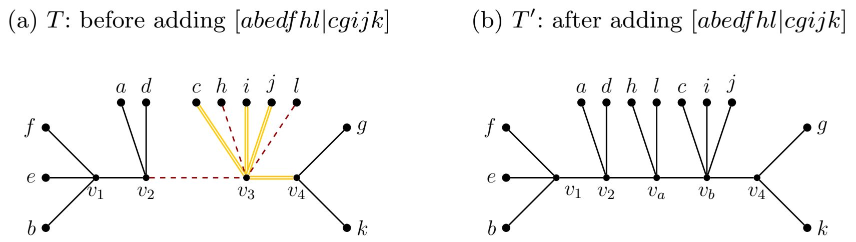
(Proof of 1) We first claim that if $R \cap A = \emptyset$ or $R \cap B = \emptyset$, then $e \notin E(T_R)$. Assume by way of contradiction that $e \in E(T_R)$. There are then two non-empty components in $T_R - e$. Since e induces $[A|B]$ in T, the two components in $T_R - e$ have leaf set $R \cap A$ 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 vertices of $P(e')$ in T_R do not connect to any leaves, 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 8.

Corollary 3. *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 6.


(a) T: before adding $[abedfhl|cgijk]$

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

Fig. 6: 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'.

Lemma 9. *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 $C(T') = C(T) \cup \{\pi\}$.*

Proof. 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'. □

8 Proofs for Section 3

Lemma 1. *Given an input set $\mathcal{A}$ 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*($\mathcal{A}$).*

Proof. Let $N \geq 2$ be any integer. Let $T_1, T_2, \ldots, T_N$ and $S_1, S_2, \ldots, S_N$ be defined as from problem statement of RFS. Let T 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

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

Taking the sum of the equations, we have

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

which is a constant. 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 RFS 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. □

Lemma 2. *For any tree $T \in \mathcal{T}_S$, $p_Y(T) \leq |\Pi_Y|$. In particular, let T_{init} be the tree defined in line 6 of Algorithm 1. Then, $p_Y(T_{\mathrm{init}}) = |\Pi_Y|$.*

Proof. 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,

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

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 8, $e \notin P(e')$ for any backbone edge $e' \in E(T_i|_X)$. Therefore, either e is an internal edge in an extra subtree in $\mathrm{Extra}(T_i)$, or e connects one extra subtree in $\mathrm{Extra}(T_i)$ to the backbone tree. In either case, the construction of T_{init} ensures that e is also present in $T_{\mathrm{init}}|_{S_i}$ and thus $\pi \in C(T_{\mathrm{init}}|_{S_i})$. Therefore, each bipartition $\pi \in \Pi_Y$ contributes 1 to $|C(T_{\mathrm{init}}|_{S_i}) \cap C(T_i) \cap \Pi_Y|$ for exactly one index $i \in [2]$ and thus it contributes 1 to $p_Y(T_{\mathrm{init}})$. Hence, $p_Y(T_{\mathrm{init}}) = |\Pi_Y|$. □

Lemma 3. *Let $\pi = [A|B] \in \mathit{\Pi}$. Let $T \in \mathcal{T}_S$ be any tree with leaf set 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, $p_X(T') - p_X(T) \leq w^*(\pi)$.*

Proof. By definition of $p_X(\cdot)$,

$$\begin{aligned} p_X(T') - p_X(T) &= \sum_{i\in[2]} |C(T'|_{S_i}) \cap C(T_i) \cap \mathit{\Pi}_X| - \sum_{i\in[2]} |C(T|_{S_i}) \cap C(T_i) \cap \mathit{\Pi}_X| \\ &= \sum_{i\in[2]} |(C(T'|_{S_i}) \backslash C(T|_{S_i})) \cap C(T_i) \cap \mathit{\Pi}_X|. \end{aligned}$$

Therefore, we only need to prove that

$$\sum_{i\in[2]} |(C(T'|_{S_i}) \backslash C(T|_{S_i})) \cap C(T_i) \cap \mathit{\Pi}_X| \leq w^*(\pi).$$

We perform a case analysis, as follows: Case (1): $\pi \notin C(T_1, T_2, X)$, Case (2): $\pi \in C(T_1|_X) \mathit{\Delta} C(T_2|_X) = (C(T_1|_X) \setminus C(T_2|_X)) \cup (C(T_2|_X) \setminus C(T_1|_X))$, 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 by way of contradiction that there exists a bipartition $\pi' \in (C(T'|_{S_i}) \backslash C(T|_{S_i})) \cap C(T_i) \cap \mathit{\Pi}_X$ for some $i \in [2]$. Since $\pi \notin C(T_1, T_2, X)$ and $\pi'|_X = \pi$, by Corollary 3, $\pi' \notin C(T_i)$ 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 $\sum_{i\in[2]} |(C(T'|_{S_i}) \backslash C(T|_{S_i})) \cap C(T_i) \cap \mathit{\Pi}_X| = 0 \leq w^*(\pi)$.

Case (2): Let $\pi \in C(T_1|_X) \mathit{\Delta} C(T_2|_X)$. Assume without loss of generality that $\pi \in C(T_1|_X) \backslash C(T_2|_X)$. Then, we have $w^*(\pi) = w(e_1(\pi))$. Let $\pi' \in \mathit{\Pi}_X \cap C(T_i) \cap (C(T'|_{S_i}) \backslash C(T|_{S_i}))$ 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 3, we have $\pi' \notin C(T_2)$ and thus $\pi' \in C(T_1)$. By Lemma 8, 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 $\pi' \in C(T_i) \cap \mathit{\Pi}_X \cap (C(T'|_{S_i}) \backslash C(T|_{S_i}))$ for any $i \in [2]$. Since $\pi' \in C(T_i)$ and $\pi'|_X = \pi \in C(T_i|_X)$, by Lemma 8, 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 $C(T_i) \cap \mathit{\Pi}_X \cap (C(T'|_{S_i}) \backslash C(T|_{S_i}))$. Therefore,

$$|(C(T'|_{S_i}) \backslash C(T|_{S_i})) \cap C(T_i) \cap \mathit{\Pi}_X| \leq w(e_i(\pi)).$$

Hence,

$$\sum_{i\in[2]} |(C(T'|_{S_i}) \backslash C(T|_{S_i})) \cap C(T_i) \cap \mathit{\Pi}_X| \leq w^*(\pi).$$

□

Lemma 4. *For any compatible set $F \subseteq \mathit{\Pi}$, let $T \in \mathcal{T}_S$ be any tree with leaf set S such that $C(T|_X) = F$. Then $p_X(T) \le w^*(F)$.*

Proof. 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 8, 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 and we have $p_X(T^j) - p_X(T^{j-1}) \le w^*(\pi_j)$. Therefore, by telescoping sum,

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

Since $C(T^0|_X) = \emptyset$, by Corollary 3, $C(T^0|_{S_i}) \cap \mathit{\Pi}_X = \emptyset$ for both $i \in [2]$. Then, $C(T^0|_{S_i}) \cap C(T_i) \cap \mathit{\Pi}_X = \emptyset$ for both $i \in [2]$, which implies $p_X(T^0) = 0$. Thus, $p_X(T) \le \sum_{\pi \in F} w^*(\pi)$, as desired. □

Let T_{init} be the tree defined in Algorithm 1. We have the following proposition about $p_X(T_{\text{init}})$, which is needed for the proof of Proposition 1.

Proposition 3. $p_X(T_{\text{init}}) = 2|X|$.

Proof. 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 $\pi_v|_{S_i} \in \mathit{\Pi}_X$ as both sides of π_v have non-empty intersections with X. Thus, $\pi_v|_{S_i} \in C(T_{\text{init}}|_{S_i}) \cap C(T_i) \cap \mathit{\Pi}_X$ 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, $p_X(T_{\text{init}}) \ge 2|X|$.

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 $\pi|_{S_i} \notin \mathit{\Pi}_X$ 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, $p_X(T_{\text{init}}) = 2|X|$. □

Proposition 1. *Let $\tilde{T}$ be the tree constructed after line 11 of Algorithm 1, then $p_X(\tilde{T}) = w^*(\text{Triv})$.*

Proof. 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

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

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

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

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 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 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 1 on T and π.

It is easy to see that line 8 of Algorithm 1 never destroys bipartitions of S_1 or S_2 already in T, so we have

$$\begin{aligned} p_X(\tilde{T}) &= p_X(T_{\text{init}}) + \sum_{\pi \in \text{Triv}} (w^*(\pi) - 2) \\ &= 2|X| + \sum_{\pi \in \text{Triv}} (w^*(\pi) - 2) && \text{(by Proposition 3)} \\ &= \sum_{\pi \in \text{Triv}} w^*(\pi). && \text{(since } |\text{Triv}| = |X|) \end{aligned}$$

□

Lemma 10 proves that the auxiliary data structures of Algorithm 1 and 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 5.

Lemma 10. *At any stage of the Algorithm 1 after line 12, we have the following invariants of T and the auxiliary data structures H and sv:*

1. *For any bipartition $\pi \in \mathrm{NonTriv}$, $sv(\pi)$ is the vertex to split to add π to $C(T|_X)$. For any internal vertex v, the set of bipartitions $H(v) \subseteq \mathrm{NonTriv}$ 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 $C(B)$.

Proof. 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 $\pi \in \mathrm{NonTriv}$, $\hat{v}$ 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 $\pi \in \mathrm{NonTriv}$ and $H(\hat{v}) = \mathrm{NonTriv}$. The roots of all extra subtrees in $\mathcal{TR}^*(\pi)$ for any $\pi \in \mathrm{NonTriv}$ are all neighbors of $\hat{v}$, so invariant 2 also holds. For any $\pi \in C(T|_X) = \mathrm{Triv}$, let $\pi = [\{a\}|B]$. It is easy to see that since a is a leaf, $\mathcal{TRS}_i(\{a\}) = \emptyset$ and $\mathcal{TRS}_i(B) = \mathrm{Extra}(T_i) \backslash \mathcal{TR}(e_i(\pi))$ for both $i \in [2]$. Then $\mathcal{TRS}(\{a\}) = \emptyset$ and $\mathcal{TRS}(B) = (\mathrm{Extra}(T_1) \cup \mathrm{Extra}(T_2)) \backslash \mathcal{TR}^*(\pi)$. 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 2, the invariants still hold for the resulting tree T'. Since $v = sv(\pi)$ at the beginning of Algorithm 2, π can be added to $C(T|_X)$ by splitting v. By Lemma 9, 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 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. $\pi' \neq \pi$ and π' is compatible with π. By Corollary 2, 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 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 $A' \subseteq A$, $\pi' \in \mathcal{BP}(A)$ and thus $\mathcal{TR}^*(\pi) \subseteq \mathcal{TRS}(A)$. Then, Algorithm 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 2, 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 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. □

Lemma 5. *Let T be a supertree computed within Algorithm 1 at line* 14 *immediately before a refinement step. Let $\pi = [A|B] \in \mathrm{NonTriv} \cap I$. Let T' be a refinement of T obtained from running Algorithm 2 with supertree T, bipartition π, and the auxiliary data structures H and sv. Then, $p_X(T') - p_X(T) = w^*(\pi)$.*

Proof. Since I corresponds to an independent set in the incompatibility graph G, all bipartitions in I are compatible. Since $C(T|_X) \subseteq \mathrm{Triv} \cup (\mathrm{NonTriv} \cap I) = I$, $\pi \in \mathrm{NonTriv} \cap I$ 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 10, $v = sv(\pi)$ is the vertex to split to add π to $T|_X$ and thus Algorithm 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, $\mathrm{Extra}(T_i) = \mathcal{TRS}_i(A) \cup \mathcal{TRS}_i(B) \cup \mathcal{TR}(e_i)$. 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

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

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

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

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

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

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 $\pi_k \in \mathit{\Pi}_X$. Similarly, for each $k \in [w(e_2)]$, $\pi'_k \in \mathit{\Pi}_X$ and $\pi'_k|_X = \pi$. We also know that since $\pi \notin C(T|_X)$, by Corollary 3, $\pi_k \notin C(T|_{S_1})$ 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 10, we know that all extra subtrees of $\mathcal{TR}(e_1)$ were attached to v at the beginning of Algorithm 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 10, 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

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

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

Proposition 2. *Let G be the weighted incompatibility graph on $T_1|_X$ and $T_2|_X$, and let I be the set of bipartitions associated with vertices in I^*, which is a maximum weight independent set of G. Let F be any compatible subset of $C(T_1, T_2, X)$. Then $w^*(I) \geq w^*(F)$.*

Proof. Let $weight(U)$ denote the total weight of any set U of vertices of G. We first claim that $w^*(I) = weight(I^*)$. Since all bipartitions in $C(T_1|_X) \cap C(T_2|_X)$ are compatible with all bipartitions in $C(T_1, T_2, X)$, each of them become two isolated vertices in the weighted incompatibility graph, all of which are included in the maximum weight independent set I^*. For each $\pi \in C(T_1|_X) \cap C(T_2|_X)$, the two vertices associated with it in G has total weight $w(e_1(\pi)) + w(e_2(\pi))$, which is exactly $w^*(\pi)$. For each $\pi \in C(T_1|_X) \Delta C(T_2|_X)$, the vertex associated with it also has weight exactly $w^*(\pi)$. Therefore, the $weight(I^*) = w^*(I)$.

Fix any compatible subset F of $C(T_1, T_2, X)$. Let $F' = F \setminus (C(T_1|_X) \cap C(T_2|_X))$, and let F'' be a set of vertices constructed by combining the vertices associated with F' and the two vertices associated with each of $\pi \in C(T_1|_X) \cap C(T_2|_X)$. Then $weight(F'') = w^*(F') + w^*(C(T_1|_X) \cap C(T_2|_X)) \geq w^*(F') + w^*(F \cap C(T_1|_X) \cap C(T_2|_X)) = w^*(F)$. Since F is compatible, F' is also compatible, and thus F'' is an independent set in G. Therefore, $weight(F'') \leq weight(I^*)$, since I^* is a maximum weight independent set in G. We conclude that $w^*(F) \leq weight(F'') \leq weight(I^*) = w^*(I)$. □

We now present a lemma with the running time analysis for Algorithm 1, which complete the proof of Theorem 1.

Lemma 11. *Algorithm 1 runs in $O(n^2|X|)$ time.*

Proof. First we analyze the running time of Algorithm 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 2 runs in $O(n+|X|^2)$ time.

Next we analyze the running time for Algorithm 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 $\text{Extra}(T_i)$ 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 10-11 essentially runs line 7 of Algorithm 2 $O(|X|)$ times using a total of $O(n|X|)$ time. Line 12 initiates the data structure H and sv in $O(|X|)$ time. Line $13-14$ runs Algorithm 2 $O(|X|)$ times with a total of $O(n|X|+|X|^3)$ time. Since $|X| \leq n$, $|X|^3 \leq n|X|^2 \leq n^2|X|$, and thus, the overall running time of the algorithm is dominated by the running time of line $2-5$, which is $O(n^2|X|)$. □

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

Lemma 12. *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.*

Proof. Let $n \geq 5$ be any integer. Let $S_i = [n]$ be the leaf set of T_i for all $i \in [n-3]$. Let $\pi_i = [1, 2, \ldots, i+1 \mid i+2, \ldots, n]$ for any $i \in [n-3]$. We let T_i denote the tree with leaf set $[n]$ that contains a single internal edge defining π_i, and let $\mathcal{A} = \{T_1, T_2, \ldots, T_{n-3}\}$. Let T be the star tree with leaf set $[n]$ (i.e., T has no internal edges) and let T' be the unique tree defined by $C(T_i) \subseteq C(T')$ for all $i \in [n]$ (i.e., T' is a compatibility supertree for $\mathcal{A}$). Note that T' is the caterpillar tree on $1, 2, \ldots, n$ (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 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($\mathcal{A}$). Let $\Pi_{[n]}$ denote the set of trivial bipartitions of $[n]$. Then $C(T) = \Pi_{[n]}$. Let $\Pi' = \{\pi_i \mid i \in [n-3]\}$ (i.e., Π' contains the nontrivial bipartitions from the trees in profile $\mathcal{A}$). Note that the set $\Pi' \cup \Pi_{[n]}$ is compatible and that the caterpillar tree T' (defined above) satisfies $C(T') = \Pi' \cup \Pi_{[n]}$. Then $C(T') \cap C(T_i) = \Pi_{[n]} \cup \{\pi_i\}$ and thus $\text{SF}(T', T_i) = n+1$ for all $i \in [n-3]$. Overall, the split support score of T' is

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

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

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

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

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

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,

$$\begin{aligned}
\mathrm{RF}(t, \mathcal{A}) = &\sum_{i \in [n-3]} \mathrm{RF}(t, T_i) \\
= &\sum_{i \in [n-3]} |C(t)\backslash C(T_i)| + |C(T_i)\backslash C(t)| \\
= &q(n-3) + (p-1)p + p(n-3-p) + (n-3-p) \\
= &(n-3) + q(n-3) + p(n-5).
\end{aligned}$$

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 shows that T' (since it is a compatibility supertree for $\mathcal{A}$) 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 T' is $(n-4)(n-3)$, which is strictly larger than $n-3$ for $n > 5$, and the RFS score for the star tree T is $n-3$; 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 [46] and which we will present below, have the same set of optimal solutions and thus the hardness of one implies hardness of another. The input to the AMS problem is a profile $\mathcal{A} = \{T_i \mid i \in [N]\}$ and the output is a binary tree

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

Thus, T_{AMS} minimizes the total number of bipartitions that are in the source trees and not in the supertree (i.e., T_{AMS} minimizes the total number of false negatives).

Lemma 13. *Given profile $\mathcal{A} = \{T_1, T_2, \ldots, T_N\}$ and $S := \bigcup_{i \in [N]} L(T_i)$, tree $T \in \mathcal{T}_S$ is a Split Fit Supertree for $\mathcal{A}$ iff T is an Asymmetric Median Supertree for $\mathcal{A}$.*

Proof. Let $\mathrm{FN}(T, \mathcal{A}) = \sum_{i \in [N]} |C(T_i) \setminus C(T|_{S_i})|$. Then,

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

Hence, T is an Asymmetric Median supertree for $\mathcal{A}$ iff T is a Split Fit supertree for $\mathcal{A}$. □

Lemma 6. RFS-3*,* SFS-3*, and* Relax–SFS-3 *are all* **NP***-hard.*

Proof. By Lemmas 1 and 13, for any profile $\mathcal{A}$, the Robinson-Foulds, Split Fit, and Asymmetric Median supertree problems all have the same set of optimal solutions. Also, the Asymmetric Median Supertree problem is **NP**-hard for three trees when they have the same leaf set [30]; 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. □

9 Relationship between SMAST, SMCT, and RFS supertree problems

The SMAST and SMCT problems seek trees that are obtained after deleting minimal numbers of leaves from the input trees so that an agreement supertree or compatible supertree can be constructed from the reduced input trees. Here, we examine the possibility of using these output trees as constraint trees on the search for RFS supertrees, so that the removed taxa could be introduced into the constraint trees. We show that exact solutions to the SMAST and SMCT (Maximum Agreement Supertree and Maximum Compatible Supertree) problems are not directly relevent to solving the Robinson-Foulds supertree problem.

Lemma 14. *There exists a pair of binary trees T_1 and T_2 for which some optimal SMAST or SMCT supertree cannot be extended to any optimal RFS supertree through the insertion of missing taxa.*

Proof. Consider the following two trees (both unrooted binary trees): Let T_1 be given by the Newick string $(A, ((B, x), ((C, y), (D, E))))$ and let T_2 be given by the Newick string $(A, (C, (z, (B, (D, E)))))$.

An RFS supertree for this pair T_1, T_2 is given by $(A, ((C, y), (z, ((B, x), (D, E)))))$, and has total RF distance to T_1 and T_2 equal to 2.

Note that at least one of A, B, C must be deleted to form an agreement supertree. Suppose C is deleted. Then $((A, z), ((B, x), (y, (D, E))))$ is an optimal SMAST.

Observe that any way of adding C into this tree produces a supertree that has total RFS score greater than 2. Hence, for this pair T_1 and T_2 of input trees, for at least one optimal SMAST supertree, there is no way to extend that optimal supertree into an optimal RFS supertree. □

10 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 and 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.

11 Experimental study

We present additional details about the experimental performance study.

11.1 Datasets

Experiment 1 To produce the datasets for the experiment on multi-locus datasets, we used SimPhy [20] 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 68% of their branches on average due to ILS [19]. The number of genes varied from 25 to 1000 with ten replicate datasets per number of genes.

For each replicate dataset, we used the model species tree and a technique similar to DACTAL [26] (described below) to divide the species set into two overlapping subsets, each containing slightly more than half the species. ASTRAL [23, 24, 51] 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-2-RFS and FastRFS. The following steps describe the procedure in details.

1. Identify a centroid edge (a, b) in the true species tree (i.e., an edge that, upon deletion, creates two 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 [51] 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 Exact-2-RFS to input pair $A1$ and $B1$, and compare to the true species tree

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.

Experiment 2 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. The average number of leaves per each source tree per replicate for these datasets is (rounded to the nearest integer) $\{87, 79, 78, 73, 74, 73, 73\}$, for each corresponding k.

11.2 Scripts and Commands

Our scripts and other utilities (developed by the 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>
  ```

  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>
  ```

11.3 External software

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>
```