Fast FPT Algorithms for Computing Rooted Agreement Forests: Theory and Experiments

Chris Whidden*, Robert G. Beiko**, and Norbert Zeh***

Faculty of Computer Science, Dalhousie University, Halifax, Nova Scotia, Canada
{whidden,beiko,nzeh}@cs.dal.ca

Abstract. We improve on earlier FPT algorithms for computing a rooted maximum agreement forest (MAF) or a maximum acyclic agreement forest (MAAF) of a pair of phylogenetic trees. Their sizes give the subtree-prune-and-regraft (SPR) distance and the hybridization number of the trees, respectively. We introduce new branching rules that reduce the running time of the algorithms from $O(3^k n)$ and $O(3^k n \log n)$ to $O(2.42^k n)$ and $O(2.42^k n \log n)$, respectively. In practice, the speed up of the algorithms may be much more than predicted by the worst-case analysis. We confirm this intuition experimentally by computing MAFs for simulated trees and trees inferred from protein sequence data. We show that our MAF algorithm is orders of magnitude faster than the best previous methods and can handle much larger trees and SPR distances.

1 Introduction

Phylogenies (evolutionary trees) are the standard model for representing the evolution of a set of species (taxa) [14]. Unfortunately, it is difficult to determine the correct phylogeny, particularly due to reticulation events—lateral gene transfer, recombination and hybridization—which result in species that are composites of genes with distinct histories. Analyzing different genes from the same set of species can produce different phylogenies. Thus, it has proven essential to compute distances between phylogenies under different metrics to assess the quality of phylogenies proposed by heuristics or to visualize tree space (see, e.g., [13]). Metrics that model reticulation events, such as the *subtree-prune-and-regraft* (SPR) distance [11] and the *hybridization number* [1] are of particular interest, since they often help to discover such events [15, 16].

Although these distance metrics are biologically meaningful, they are NP-hard to compute [8,10, 12]. This has led to significant effort to develop approximation [5, 7, 17] and fixed-parameter (FPT) algorithms [7,9], as well as heuristic approaches [2,12], for computing these distances. The main tool of most such algorithms is the notion of a maximum agreement forest [1, 8, 11]. Recently, Whidden and Zeh [18] presented a unified view of previous methods for computing agreement forests and introduced improvements that led to the theoretically fastest approximation and FPT algorithms for computing such forests so far. The FPT algorithms for SPR distance and hybridization number take a bounded search tree approach and take $O(3^k n)$ and $O(3^k n \log n)$ time, respectively. Using standard kernelizations [8, 9], these times can be reduced to $O(3^k k + n^3)$ and $O(3^k k \log k + n^3)$, respectively.

In this paper, we introduce improved branching rules to be used in the algorithms of [18]. These branching rules reduce the running times of the algorithms for SPR distance and hybridization number to $O(2.42^k n)$ and $O(2.42^k n \log n)$, respectively. Using the same kernelizations as before, the running times can be reduced further to $O(2.42^k k + n^3)$ and $O(2.42^k k \log k + n^3)$, respectively.

While these theoretical improvements are valuable in their own right, our main contribution is to evaluate the practical performance of the algorithm of [18] and the impact of our improved branching rules. An additional optimization we apply is to use the linear-time 3-approximation algorithm for

* Supported by an NSERC PGS-D graduate scholarship.
** Supported in part by NSERC, Genome Atlantic and the Canada Research Chairs programme.
*** Supported in part by NSERC and the Canada Research Chairs programme.

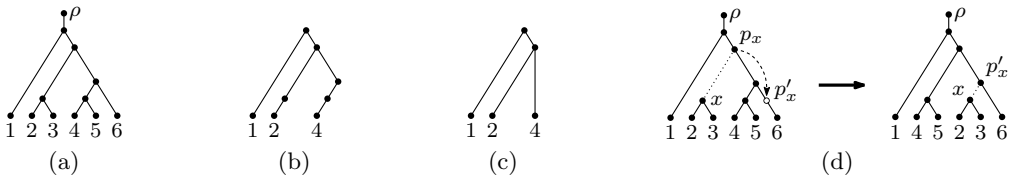

Fig. 1. (a) An X-tree T. (b) The subtree $T(V)$ for $V = \{1, 2, 4\}$. (c) $T|V$. (d) Illustration of an SPR operation.

SPR distance of [17,18] to prune branches in the search tree that are guaranteed to be unsuccessful. This reduces the size of the search tree substantially with a corresponding decrease in running time. We demonstrate that each of the improved branching rules and the pruning of unsuccessful branches have a marked effect on the performance of the algorithm, though the individual contributions differ. Our experiments confirm that our algorithm is orders of magnitude faster than the currently best exact alternatives [4, 19] based on reductions to integer linear programming and satisfiability testing, respectively. The largest distances reported using implementations of previous methods are a hybridization number of 14 on 40 taxa [6] and an SPR distance of 19 on 46 taxa [19]. In contrast, our method took less than 5 hours to compute SPR distances of up to 46 on trees with 144 taxa and 99 on synthetic 1000-leaf trees. This represents a major step forward towards tools that can infer reticulation scenarios for the hundreds of genomes that have been fully sequenced to date.

The rest of this paper is organized as follows. Section 2 introduces the necessary terminology, notation, and some technical lemmas needed in our proofs. Section 3 presents the FPT algorithms using the improved branching rules. Section 4 presents our experimental results.

2 Preliminaries

Throughout this paper, we mostly use the definitions and notation from [7–9, 17, 18]. A *(rooted binary phylogenetic) X-tree* is a rooted tree T whose leaves are the elements of a label set X and whose non-root internal nodes have two children each; see Figure 1(a). The root of T has label ρ and has one child. Throughout this paper, we consider ρ to be a member of X, and we sometimes refer to X as X_T. For a subset V of X, $T(V)$ is the smallest subtree of T that connects all nodes in V; see Figure 1(b). The *V-tree induced by T* is the tree $T|V$ obtained from $T(V)$ using *forced contractions*, each of which removes an unlabelled node with only one child and its incident edges. If the node was the root of the current tree, its child becomes the new root; otherwise it is replaced by an edge between its parent and child. See Figure 1(c).

A *subtree-prune-and-regraft* (SPR) operation on an X-tree T cuts an edge $e_x := xp_x$, where p_x denotes the parent of x. This divides T into subtrees T_x and T_{p_x} containing x and p_x, respectively. Then it introduces a node p'_x into T_{p_x} by subdividing an edge of T_{p_x} and adds an edge xp'_x, making x the child of p'_x. Finally, p_x is removed using a forced contraction. See Figure 1(d).

The distance measure $d_{spr}(T_1, T_2)$ between X-trees is the minimum number of SPR operations required to transform T_1 into T_2. A related distance measure is the *hybridization number*, $hyb(T_1, T_2)$, which is defined in terms of hybrid networks. A *hybrid network* of T_1 and T_2 is a directed acyclic graph H such that both trees, with their edges directed away from the root, can be obtained from H by forced contractions and edge deletions. For a node $x \in H$, let $\deg_{\text{in}}(x)$ be its in-degree and $\deg^-_{\text{in}}(x) = \max(0, \deg_{\text{in}}(x) - 1)$. Then $hyb(T_1, T_2) = \min_H \sum_{x \in H} \deg^-_{\text{in}}(x)$, taking the minimum over all hybrid networks H of T_1 and T_2. These metrics are related to the sizes of appropriately defined agreement forests. To define these, we first introduce some terminology.

Given a forest F and a subset E of its edges, $F - E$ denotes the forest obtained by deleting the edges in E from F. If F has components $T_1, T_2, \ldots, T_k$ with label sets $X_1, X_2, \ldots, X_k$, F *yields*

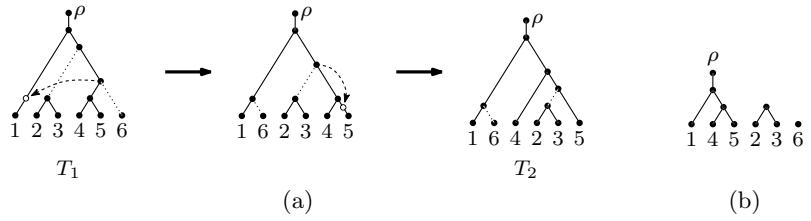

Fig. 2. (a) SPR operations transforming T_1 into T_2. Each operation changes the top endpoint of one of the dotted edges. (b) The corresponding agreement forest, which can be obtained by cutting the dotted edges in both trees.

the forest F' with components $T_1|X_1, T_2|X_2, \ldots, T_k|X_k$; if $X_i = \emptyset$, then $T_i(X_i) = \emptyset$ and, hence, $T_i|X_i = \emptyset$. We use $F \div E$ to denote the forest yielded by $F - E$, and say that it is a *forest of* F.

A *triple* $ab|c$ of a forest F is defined by a set $\{a, b, c\}$ of three leaves in the same component of F and such that the path from a to b in F is disjoint from the path from c to the root of the component. A triple of a forest F_1 is *compatible* with a forest F_2 if it is also a triple of F_2; otherwise it is *incompatible* with F_2. For two forests F_1 and F_2 with the same label set, two components T_1 and T_2 of F_1 are said to *overlap* in F_2 if there exist leaves $a, b \in T_1$ and $c, d \in T_2$ such that the paths from a to b and from c to d in F_2 exist and are non-disjoint. The following lemma is an easy extension of a lemma of [7], which states the same result for a tree T_2 instead of a forest F_2.

Lemma 1. *Let F_1 and F_2 be forests of two X-trees T_1 and T_2, and denote the label sets of the components of F_1 by $X_1, X_2, \ldots, X_k$ and the label sets of the components of F_2 by $Y_1, Y_2, \ldots, Y_l$. F_1 is a forest of F_2 if and only if (1) for every X_i, there exists a Y_j such that $X_i \subseteq Y_j$, (2) no two components of F_1 overlap in F_2, and (3) no triple of F_1 is incompatible with F_2.*

Given X-trees T_1 and T_2 and forests F_1 of T_1 and F_2 of T_2, a forest F is an *agreement forest* (AF) of F_1 and F_2 if it is a forest of both F_1 and F_2; see Figure 2. F is a *maximum agreement forest* (MAF) of F_1 and F_2 if there is no AF of F_1 and F_2 with fewer components. We denote the number of components in an MAF of F_1 and F_2 by $m(F_1, F_2)$, and the size of the smallest edge set E such that $F \div E$ is an AF of F_1 and F_2 by $e(F_1, F_2, F)$, where F is a forest of F_2. Bordewich and Semple [8] showed that $d_{spr}(T_1, T_2) = e(T_1, T_2, T_2) = m(T_1, T_2) - 1$.

Hybridization numbers correspond to MAFs with an additional constraint. For two forests F_1 and F_2 of T_1 and T_2 and an AF F of F_1 and F_2, each node x in F can be mapped to nodes $\phi_1(x)$ in T_1 and $\phi_2(y)$ in T_2 by defining X^x to be the set of labelled descendants of x in F and $\phi_i(x)$ to be the lowest common ancestor in T_i of all nodes in X^x. We say that F contains a cycle if there exist nodes x and y (a *cycle pair* (x, y)) that are roots of trees in F and such that $\phi_1(x)$ is an ancestor of $\phi_1(y)$ and $\phi_2(y)$ is an ancestor of $\phi_2(x)$. Otherwise, F is an *acyclic agreement forest* (AAF). A *maximum acyclic agreement forest* (MAAF) of F_1 and F_2 is an AAF with the minimum number of components among all AAFs of F_1 and F_2. We denote its size by $\tilde{m}(F_1, F_2)$ and the number of edges in a forest F of F_2 that must be cut to obtain an AAF of F_1 and F_2 by $\tilde{e}(F_1, F_2, F)$. Baroni et al. [1] showed that $hyb(T_1, T_2) = \tilde{e}(T_1, T_2, T_2) = \tilde{m}(T_1, T_2) - 1$.

We write $a \sim_F b$ when there exists a path between two nodes a and b of a forest F. For a node $x \in F$, F^x denotes the subtree of F induced by all descendants of x, inclusive. For forests F_1 and F_2 and nodes $a, b \in F_1$ with a common parent, we say (a, b) is a *sibling pair* of F_1 if there exist nodes $a', b' \in F_2$ such that $F_1^a = F_2^{a'}$ and $F_1^b = F_2^{b'}$. For simplicity, we refer to a' and b' as a and b.

The correctness proofs of our algorithms in the next section make use of the following two lemmas. Lemma 2 was shown by Bordewich et al. [7]. A proof of Lemma 3 can be found in Appendix A.

Lemma 2 (Shifting Lemma). *Let F be a forest of an X-tree, e and f edges of F, and E a subset of edges of F such that $f \in E$ and $e \notin E$. Let v_f be the end vertex of f closest to e, and v_e an end vertex of e. If $v_f \sim_{F-E} v_e$ and $x \nsim_{F-(E\cup\{e\})} v_f$, for all $x \in X$, then $F \div E = F \div (E \setminus \{f\} \cup \{e\})$.*

Lemma 3. *Let F_1 and F_2 be forests of X-trees T_1 and T_2, respectively. Let F_1 be the union of trees $\dot{T}_1, \dot{T}_2, \ldots, \dot{T}_k$ and F_2 be the union of forests $\dot{F}_1, \dot{F}_2, \ldots, \dot{F}_k$ such that $X_{\dot{T}_i} = X_{\dot{F}_i}$, for all $1 \leq i \leq k$. $F_2 \div E$ is an AF of T_1 and T_2 if and only if it is an AF of F_1 and F_2.*

3 The Algorithms

In this section, we present our improved FPT algorithms for computing an MAF or MAAF of two phylogenies. As is customary for FPT algorithms, we focus on the decision version of the problem: "Given two X-trees T_1 and T_2, a distance measure $d(\cdot,\cdot)$, and a parameter k, is $d(T_1, T_2) \leq k$?" To compute the distance between two trees, we start with $k = 0$ and increase it until we receive an affirmative answer. This does not asymptotically increase the running time of the algorithm, as the dependence on k is exponential. Our implementations employ this strategy.

We begin with the MAF algorithm. The algorithm is recursive. Each invocation takes as input two forests F_1 and F_2 of T_1 and T_2 and a parameter k, and decides whether $e(T_1, T_2, F_2) \leq k$. We denote such an invocation by $\textsc{Maf}(F_1, F_2, k)$. The forest F_1 is the union of a tree $\dot{T}_1$ and a forest F, while F_2 is the union of the same forest F and another forest $\dot{F}_2$ with the same label set as $\dot{T}_1$. We maintain two sets of labelled nodes: R_d contains the roots of F, and R_t contains roots of subtrees that agree between $\dot{T}_1$ and $\dot{F}_2$. We refer to the nodes in these sets by their labels. For the top-level invocation, $F_1 = \dot{T}_1 = T_1$, $F_2 = \dot{F}_2 = T_2$, and $F = \emptyset$; R_d is empty, and R_t contains all leaves of F_2.

$\textsc{Maf}(F_1, F_2, k)$ identifies a small collection $\{E_1, E_2, \ldots, E_q\}$ of subsets of edges of $\dot{F}_2$ such that $e(T_1, T_2, F_2) \leq k$ if and only if $e(T_1, T_2, F_2 - E_i) \leq k - |E_i|$, for at least one $1 \leq i \leq q$. It makes a recursive call $\textsc{Maf}(F_1, F_2 - E_i, k - |E_i|)$, for each subset E_i, and returns "yes" if and only if one of these calls does. The steps of this procedure are as follows. Figure 3 illustrates the important steps.

1. (Failure) If $k < 0$, there is no subset E of at most k edges of F_2 such that $F_2 - E$ yields an AF of T_1 and T_2. Return "no" in this case.
2. (Success) If $|R_t| \leq 2$, then $\dot{F}_2 \subseteq \dot{T}_1$. Hence, $\dot{F}_2 \cup F$ is an AF of F_1 and F_2 and, by Lemma 3, also of T_1 and T_2. Return "yes" in this case.
3. (Prune maximal agreeing subtrees) If there is no node $r \in R_t$ that is a root in $\dot{F}_2$, go to Step 4. Otherwise remove r from R_t and add it to R_d, thereby moving the corresponding subtree of $\dot{F}_2$ to F. Cut the edge e_r in $\dot{T}_1$ and apply a forced contraction to remove r's parent from $\dot{T}_1$. This does not alter F_2 and, thus, neither $e(T_1, T_2, F_2)$. Return to Step 2.
4. Choose a sibling pair (a, c) in $\dot{T}_1$ such that $a, c \in R_t$.
5. (Grow agreeing subtrees) If (a, c) is not a sibling pair of $\dot{F}_2$, go to Step 6. Otherwise remove a and c from R_t, label their parent in both trees with (a, c), and add it to R_t. Return to Step 2.
6. (Cut edges) Distinguish three cases.
   - 6.1. If $a \nsim_{F_2} c$, make two recursive calls $\textsc{Maf}(F_1, F_2 \div \{e_a\}, k-1)$ and $\textsc{Maf}(F_1, F_2 \div \{e_c\}, k-1)$.
   - 6.2. If $a \sim_{F_2} c$ and the path from a to c in $\dot{F}_2$ has one pendant node b, make one recursive call $\textsc{Maf}(F_1, F_2 \div \{e_b\}, k-1)$.
   - 6.3. If $a \sim_{F_2} c$ and the path from a to c in $\dot{F}_2$ has $q \geq 2$ pendant nodes $b_1, b_2, \ldots, b_q$, make three recursive calls $\textsc{Maf}(F_1, F_2 \div \{e_a\}, k-1)$, $\textsc{Maf}(F_1, F_2 \div \{e_c\}, k-1)$, and $\textsc{Maf}(F_1, F_2 \div \{e_{b_1}, e_{b_2}, \ldots, e_{b_q}\}, k-q)$.

   Return "yes" if one of the recursive calls does; otherwise return "no".

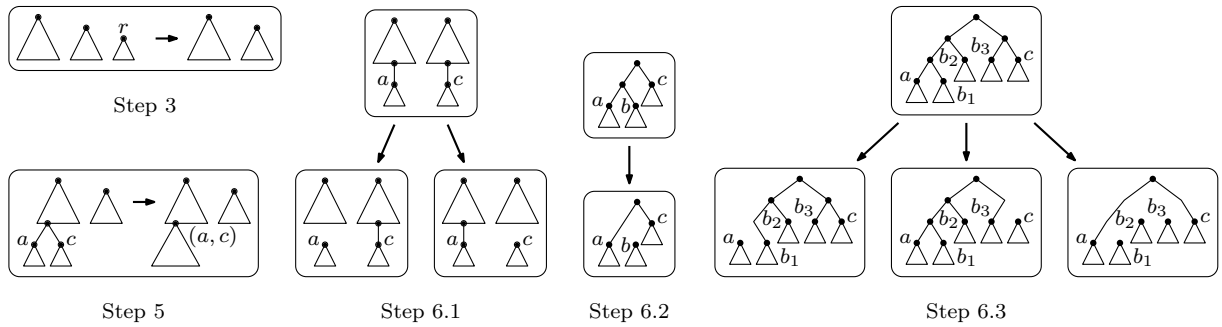

Fig. 3. The cases of the rooted MAF algorithm. Only $\dot{F}_2$ is shown. Each box represents a recursive call.

The above algorithm is identical to the one presented in [18], with the exception of Step 6. In this step, the algorithm of [18] chooses a and c so that the distance of a from the root of T_2 is no less than that of c, identifies the sibling b of a in $\dot{F}_2$ (which exists because otherwise Step 3 would have removed a from R_t) and then makes three recursive calls $\textsc{Maf}(F_1, F_2 \div \{e_a\}, k-1)$, $\textsc{Maf}(F_1, F_2 \div \{e_b\}, k-1)$, and $\textsc{Maf}(F_1, F_2 \div \{e_c\}, k-1)$. Next we show that distinguishing between Cases 6.1–6.3 as we do here leads to an improved running time.

Theorem 1. *For two rooted X-trees T_1 and T_2 and a parameter k, it takes $\mathrm{O}((1+\sqrt{2})^k n) = \mathrm{O}(2.42^k n)$ time to decide whether $e(T_1, T_2, T_2) \leq k$.*

Using the same data structures as in the algorithm of [18], each recursive call takes $\mathrm{O}(n)$ time. Thus, the running time claimed in Theorem 1 follows if we can bound the number of recursive calls by $\mathrm{O}((1+\sqrt{2})^k)$. The number of recursive calls spawned by an invocation depends only on k and is given by the following recurrence

$$I(k) = \begin{cases} 1 + 2I(k-1) & \text{Case 6.1} \\ 1 + I(k-1) & \text{Case 6.2} \\ 1 + 2I(k-1) + I(k-q) & \text{Case 6.3} \end{cases}$$
$$\leq 1 + 2I(k-1) + I(k-2)$$

because Case 6.3 dominates the other two cases and $q \geq 2$ in this case. Simple substitution shows that this recurrence solves to $I(k) = \mathrm{O}((1+\sqrt{2})^k)$. It remains to prove the correctness of the algorithm. Our strategy is as follows. Consider an edge set E of size $e(T_1, T_2, F_2)$ and such that $F \div E$ is an AF of T_1 and T_2. By Lemma 3, $F_2 \div E$ is also an AF of F_1 and F_2. Now we consider each of the three cases of Step 6. We show that, in each case, there exists a set E as above and a recursive call $\textsc{Maf}(F_1, F_2 \div E_i, k - |E_i|)$ made in this case such that $E_i \subseteq E$. This implies by induction that (1) none of the other recursive calls we make returns "yes" unless $e(T_1, T_2, F_2) \leq k$ (because each recursive call $\textsc{Maf}(F_1, F_2 \div E_j, k')$ satisfies $k' = k - |E_j|$) and (2) the recursive call $\textsc{Maf}(F_1, F_2 \div E_i, k - |E_i|)$ returns "yes" if and only if $e(T_1, T_2, F_2) = k$. Thus, the current invocation gives the correct answer. Each of the following three lemmas considers one case.

Lemma 4 (Case 6.1). *If $a \nsim_{F_2} c$, there exists an edge set E of size $e(T_1, T_2, F_2)$ (resp. $\tilde{e}(T_1, T_2, F_2)$) such that $F_2 \div E$ is an AF (resp. AAF) of T_1 and T_2 and $E \cap \{e_a, e_c\} \neq \emptyset$.*

Proof. Consider an edge set E' of size $e(T_1, T_2, F_2)$ and such that $F \div E'$ is an AF of F_1 and F_2, and assume that $E' \cap \{e_a, e_c\} = \emptyset$. Since (a, c) is a sibling pair of F_1 but $a \nsim_{F_2} c$, we must have $a \nsim_{F_2 - E'} x$, for all $x \notin F_2^a$, or $c \nsim_{F_2 - E'} x$, for all $x \notin F_2^c$. W.l.o.g. assume the former. Since a is

not a root in F_2, there exists a leaf $l \notin F_2^a$ such that $a \sim_{F_2} l$. For each such leaf l, the path from a to l in F_2 contains an edge in E' because $a \nsim_{F_2 - E'} l$. Pick an arbitrary such leaf l, and let f be the edge in E' on the path from a to l closest to a. The edges e_a and f satisfy the conditions of Lemma 2, and $F_2 \div E' = F_2 \div (E' \setminus \{f\} \cup \{e_a\})$, that is, the set $E := E' \setminus \{f\} \cup \{e_a\}$ satisfies the lemma. The second claim follows using the same arguments after choosing E' of size $\tilde{e}(T_1, T_2, F_2)$ and such that $F \div E'$ is an AAF of T_1 and T_2. □

Lemma 5 (Case 6.2). *If $a \sim_{F_2} c$ and the path from a to c in F_2 has only one pendant node b, there exists an edge set E of size $e(T_1, T_2, F_2)$ such that $F_2 \div E$ is an AF of T_1 and T_2 and $e_b \in E$.*

Proof. Again, consider an edge set E' of size $e(T_1, T_2, F_2)$ and such that $F \div E'$ is an AF of F_1 and F_2. Assume that b is a's sibling and that E' contains the maximum number of edges from $\{e_a, e_b, e_c\}$ among all sets E'' such that $|E''| = e(T_1, T_2, F_2)$ and $F \div E''$ is an AF of F_1 and F_2. As shown in [18, Theorem 1], $E' \cap \{e_a, e_b, e_c\} \neq \emptyset$. If $e_b \in E'$, there is nothing to prove. So assume that $e_b \notin E'$. Let $v = p_a = p_b$, and $u = p_v = p_c$. We distinguish two cases.

If $|E' \cap \{e_a, e_c, e_v\}| \geq 2$, we define $E := E' \setminus \{e_a, e_c, e_v\} \cup \{e_b, e_u\}$. Clearly, $|E| \leq |E'|$. By Lemma 1, it suffices to prove that no triple of $F_2 \div E$ is incompatible with F_1 and that no two components of $F_2 \div E$ overlap in F_1, that is, there are no two paths in F_1 between leaves in different components of $F_2 \div E$ that share an edge. Since all triples of $F_2 \div E'$ are compatible with F_1, it suffices to consider only triples of $F_2 \div E$ that do not exist in $F_2 \div E'$. All such triples, however, involve only leaves in $F_2^a \cup F_2^c$ and, thus, are also triples of F_1 because (a, c) is a sibling pair of F_1. Every path in $F_2 \div E$ that does not exist in $F_2 \div E'$ connects a leaf $a' \in F_2^a$ with a leaf $c' \in F_2^c$. The corresponding path in F_1 stays completely inside $F_1^{p_a}$ and can overlap only a path P between two leaves in $F_1^{p_a}$. This, however, is impossible because P would also exist in $F_2^{p_a}$ and would overlap the path from a' to c' in F_2, contradicting that the two paths belong to different components of $F_2 - E$. Thus, no two components of $F_2 \div E$ overlap in F_1.

If $|E' \cap \{e_a, e_v, e_c\}| = 1$, then either $e_a \in E'$ or $e_c \in E'$. W.l.o.g. assume the latter. In this case, we define $E := E' \setminus \{e_c\} \cup \{e_b\}$. Again, $|E| = |E'|$. Next we prove that there are no triples in $F_2 \div E$ incompatible with F_1 nor paths in F_1 between leaves in different components of $F_2 \div E$ that share an edge. Again, it suffices to consider only triples of $F_2 \div E$ that are not triples of $F_2 \div E'$. Each such triple $xy|z$ involves at least one leaf $c' \in F_2^c$. If all leaves of the triple belong to $F_2^a \cup F_2^c$, the argument from the previous case shows that the triple is compatible with F_1. Otherwise, the triple involves at least one node outside $F_2^a \cup F_2^c$. We choose two nodes $a' \in F_2^a$ and $b' \in F_2^b$ such that $a' \sim_{F_2 - E'} u \sim_{F_2 - E'} b'$. By Lemma 2 and because $e_a, e_b, e_v \notin E'$, such nodes exist.

If the triple $xy|z$ includes two nodes $c_1, c_2 \in F_2^c$, it is of the form $c_1c_2|z$. By the choice of a' and b', the triple $a'b'|z$ is a triple of $F_2 \div E'$ and, hence, of F_1. In F_1, the lowest common ancestor of a' and b' is an ancestor of a and c. Thus, $c_1c_2|z$ is also a triple of F_1.

If the triple $xy|z$ includes exactly one node $c' \in F_2^c$, it is of the form $a''c'|z$, $c'y|z$, or $xy|c'$, where $a'' \in F_2^a$ and $x, y, z \notin F_2^a \cup F_2^c$. In the first case, we observe that $a''b'|z$ is a triple of $F_2 \div E'$ and, hence, of F_1, and the lowest common ancestor of a'' and b' in F_1 is an ancestor of a and c. Thus, $a''c'|z$ is also a triple of F_1. In the last two cases, $a'y|z$ or $xy|a'$, respectively, is a triple of F_1. Since $x, y, z \notin F_2^a \cup F_2^c$, this implies that $c'y|z$ or $xy|c'$, respectively, is also a triple of F_1.

Finally, assume there are two components of $F_2 \div E$ that overlap in F_1. One of these components must include leaves $a' \in F_2^a$ and $c' \in F_2^c$, and there must exist a path in F_1 between c' and a leaf z such that $c' \sim_{F_2 - E} z$ that shares an edge f with another path between nodes x and y such that $x \sim_{F_2 - E} y$ but $x \nsim_{F_2 - E} c'$. If f belongs to $F_1^{p_a}$, the same argument as in the case $|E' \cap \{e_a, e_c, e_v\}| \geq 2$ leads to a contradiction. If f does not belong to $F_1^{p_a}$, then $z \notin F_1^{p_a}$. This implies that $a' \sim_{F_2 - E'} z$ and the edge f belongs to the path from a' to z. Thus, the component of $F_2 - E'$ containing a' and z overlaps the component containing x and y, a contradiction. □

Lemma 6 (Case 6.3). *If $a \sim_{F_2} c$ and the path from a to c in F_2 has $q \geq 2$ pendant nodes $b_1, b_2, \ldots, b_q$, there exists an edge set E of size $e(T_1, T_2, F_2)$ (resp. $\tilde{e}(T_1, T_2, F_2)$) such that $F_2 \div E$ is an AF (resp. AAF) and either $E \cap \{e_a, e_c\} \neq \emptyset$ or $\{e_{b_1}, e_{b_2}, \ldots, e_{b_q}\} \subseteq E$.*

Proof. By induction on q. See Appendix B. □

As shown in [18], an algorithm for deciding whether T_1 and T_2 have a maximum *acyclic* agreement forest of size at most $k+1$ can be obtained using a two-phased approach. In the first phase, we employ the MAF algorithm. Whenever the MAF algorithm would return "yes" in Step 2, however, we invoke a second algorithm that tests whether all cycles in the obtained agreement forest can be eliminated by cutting at most k edges:

2′. If $|R_t| \leq 2$, then $F_2 = \dot{F}_2 \cup F$ is an AF of T_1 and T_2. Now invoke an algorithm $\textsc{Maaf}(F_2, k)$ that decides whether all cycles in F_2 can be eliminated by cutting at most k edges.

Such an algorithm $\textsc{Maaf}(F, k)$ with running time $O(2^k n \log n)$ time is presented in [18]. The correctness of this two-phased MAAF procedure follows if we can show that in each of the three cases in Step 6, there exists a recursive call $\textsc{Maf}(F_1, F_2 \div E_i, k - |E_i|)$ such that E_i is a subset of a set E of size $\tilde{e}(T_1, T_2, F_2)$ and such that $F_2 \div E$ is an AAF of T_1 and T_2. For Cases 6.1 and 6.3, Lemmas 4 and 6 state that this is the case. In Case 6.2, however, edge e_b may not belong to such a set. To fix this, we use the following alternative to Step 6.2 when computing an MAAF.

6.2′. If the path from a to c in F_2 has one pendant child b, make two recursive calls $\textsc{Maf}(F_1, F_2 \div \{e_b\}, k-1)$ and $\textsc{Maf}(F_1, F_2 \div \{e_c\}, k-1)$.

Theorem 2. *For two rooted X-trees T_1 and T_2 and a parameter k, it takes $O(((1+\sqrt{2})^k n \log n) = O(2.42^k n \log n)$ time to decide whether $\tilde{e}(T_1, T_2, T_2) \leq k$.*

The correctness proof of the MAAF algorithm obtained from our MAF algorithm using the above modifications is identical to the correctness proof of the MAF algorithm; however, we use Lemma 7 below instead of Lemma 5 to show that we cut the right edges in Step 6.2′. To bound the running time of the algorithm, we observe that the recurrence for the number of recursive calls in Step 6.2′ is $I(k) = 2I(k-1)$, which is still dominated by the recurrence for Step 6.3. Using that the running time of the MAAF procedure is $T(n, k) = O(2^k n \log n)$, simple substitution yields the bound claimed in the theorem. It remains to prove the correctness of Step 6.2′.

Lemma 7 (Case 6.2′). *If $a \sim_{F_2} c$ and the path from a to c in F_2 has only one pendant node b, there exists an edge set E of size $\tilde{e}(T_1, T_2, F_2)$ and such that $F_2 \div E$ is an AAF of T_1 and T_2 and either $e_b \in E$ or $e_c \in E$.*

Proof. Let E' be an edge set of size $\tilde{e}(T_1, T_2, F_2)$ and such that $F_2 \div E'$ is an AAF of T_1 and T_2. Assume further that there is no such set containing more edges from $\{e_a, e_b, e_c\}$ than E' and that b is a's sibling. As shown in [18, Theorem 1], $E' \cap \{e_a, e_b, e_c\} \neq \emptyset$. If $e_b \in E'$ or $e_c \in E'$, we are done. So assume that $E' \cap \{e_b, e_c\} = \emptyset$ and $e_a \in E'$. Let $v = p_a = p_b$ and $u = p_c = p_v$. If $\{e_a, e_v\} \subseteq E'$, Lemma 2 implies that we could replace e_v with e_b in E' without changing $F \div E'$, contradicting the choice of E'. Thus, $e_v \notin E'$. Now, let $E := E' \setminus \{e_a\} \cup \{e_b\}$. We have $|E| \leq |E'|$, $e_b \in E$ and, as shown in the proof of Lemma 5, $F \div E$ is an AF of T_1 and T_2. Next we show that $F \div E$ is acyclic.

It suffices to consider roots of $F_2 \div E$ that do not exist in $F_2 \div E'$ or whose mapping to nodes in T_1 changes because their sets of descendant leaves in $F_2 \div E$ differ from the ones in $F_2 \div E'$. The only root of $F_2 \div E$ that does not exist in $F \div E'$ is a result of cutting edge e_b and is a descendant

z of b in F_2. The only root of $F_2 \div E$ that can map to a different node in T_1 because its sets of descendants in $F_2 \div E$ and $F_2 \div E'$ differ is u, if it is a root.

First assume that u is a root of $F_2 \div E$ and consider a cycle pair (u, x) of $F_2 \div E$ with $x \neq z$. Let u' be the node in T_1 that u maps to based on its descendants in $F_2 \div E'$, and let u'' be the node in T_1 that u maps to based on its descendants in $F_2 \div E$. Then $u'' = p_a = p_c$, and u' is an ancestor of u''. Since (u, x) is a cycle pair of $F_2 \div E$ and not a cycle pair of $F_2 \div E'$, x belongs to the path from u'' to u' in T_1 and, thus, to the path from u' to a leaf $c' \in F_1^c$ such that $c' \sim_{F_2 - E'} u$. This implies that the components of $F_2 \div E'$ with roots u and x overlap in T_1, a contradiction.

It remains to prove that there are no cycle pairs of $F_2 \div E$ involving z. Assume that there exists such a pair (x, z). If x is a descendant of z in T_2, then $x \neq u$ and x is an ancestor of z in T_1. Since z becomes a root as a result of cutting edge e_b, it belongs to a component of $F_2 \div E'$ with root $r \neq z$. Since x is a descendant of z in T_2, it is also a descendant of r in T_2 and, thus, no ancestor of r in T_1 because $F_2 \div E'$ is acyclic. Since x is an ancestor of z in T_1, this implies that the components of $F_2 \div E'$ with roots x and r overlap in T_1, contradicting that $F_2 \div E'$ is a forest of T_1.

If x is an ancestor of z in T_2, it is a descendant of z in T_1. Observe that z cannot be an ancestor of c in T_1 because otherwise z is the lowest common ancestor of two leaves $b_1, b_2 \in F_2^b$ such that $b_1 \sim_{F_2 - E'} c \sim_{F_2 - E'} b_2$ and, hence, $b_1 \sim_{F_2 - E'} c' \sim_{F_2 - E'} b_2$, for some leaf $c' \in F_2^c$. These leaves form the triples $b_1 b_2 | c'$ in T_2 and $c' b_1 | b_2$ in T_1, a contradiction because $F_2 \div E'$ is an AF of T_1 and T_2. Since z is not an ancestor of c in T_1, we have $x \neq u$. This implies that either $x \in F_2^b$ or x is an ancestor of u in F_2. In the former case, the components of $F_2 \div E'$ containing x and z overlap in F_2, a contradiction. In the latter case, x is an ancestor of the root r of the component of $F_2 \div E'$ containing z. In T_1, on the other hand, x is a descendant of r because it is a descendant of z, a contradiction because $F_2 \div E'$ is acyclic. □

As in [18], Theorems 1 and 2 along with known kernelizations [8,9] imply the following corollary.

Corollary 1. *For two rooted X-trees T_1 and T_2 and a parameter k, it takes $O(2.42^k k + n^3)$ time to decide whether $e(T_1, T_2, T_2) \leq k$ and $O(2.42^k k \log k + n^3)$ time to decide whether $\tilde{e}(T_1, T_2, T_2) \leq k$.*

4 Evaluation of SPR Distance Algorithms

In this section, we present an experimental evaluation of our MAF (SPR distance) algorithm that compares the algorithm's performance to that of two competitors using the protein tree data set examined in [2,3] and using synthetic trees. We also investigate the impact of the improved branching rules in Step 6 on the performance of the algorithm. Our competitors were **sprdist** [19] and **treeSAT** [4], which reduce the problem of computing SPR distances to integer linear programming and satisfiability testing, respectively. We do not provide a comparison with **EEEP** [2] because **sprdist** outperformed **EEEP** at finding the exact SPR distance between binary rooted phylogenies [19]. However, **EEEP** is designed for more general scenarios, such as multifurcating trees.

For **sprdist** and **treeSAT**, we used publicly available implementations of these algorithms. For our own algorithm, we developed an implementation in C++ that allowed us to individually turn the optimized branching rules in Step 6 of the algorithm on and off. When the optimized branching rule in one of the cases is turned off, the algorithm uses the 3-way branching of [18] described on page 5 in this case. In particular, with all optimizations off, the algorithm is the one of [18]. Our algorithm explores the search tree in depth-first order. Thus, it needs to remember the state of the invocations along only one search path in the tree at any point in time, which requires $O(kn)$ space.

We also implemented the linear-time 3-approximation algorithm for MAF of [18] and used it to implement two additional optimizations of our FPT algorithm. The FPT algorithm searches for the correct value of $e(T_1, T_2, T_2)$ by starting with a lower bound k of $e(T_1, T_2, T_2)$ and incrementing

k until it determines that $k = e(T_1, T_2, T_2)$. If the 3-approximation algorithm returns a value of k', then $e(T_1, T_2, F_2) \geq \lceil k'/3 \rceil$; by using this as the starting value of our search, we can skip early iterations of the algorithm and thereby obtain a small improvement in the running time. The same approach can be used in a branch-and-bound strategy that prunes unsuccessful branches from the search tree. In particular, we extended Step 1 of the FPT algorithm as follows:

1′. (Failure) If $k < 0$, return "no". Otherwise compute a 3-approximation k' of $e(T_1, T_2, F_2)$. If $k' > 3k$, then $e(T_1, T_2, F_2) > k$; return "no" in this case.

Again, we allowed this optimization of Step 1 to be turned on or off in our algorithm to investigate its effect on the running time, but our implementation always uses the 3-approximation algorithm to provide a starting guess of $e(T_1, T_2, T_2)$.

4.1 Data Sets

The protein tree data set of [2, 3] consists of 22,437 protein trees, each constructed from a set of proteins covering from 4 to 144 microbial genomes. We compared the 5689 protein trees with 10 or more leaves to the corresponding subtree of a rooted reference tree covering all 144 microbial genomes considered in [2,3]. The protein trees were unrooted, so we selected a rooting for each tree that gave the minimum SPR distance according to the 3-approximation algorithm of [18].

The synthetic tree pairs were generated by first generating a random tree T_1 and then transforming it into a second tree T_2 using a known number of SPR operations. Note that the SPR distance may be lower because the sequence of SPR operations we use to obtain T_2 from T_1 may not be the shortest such sequence. For n taxa, the label set of T_1 was represented using integers 1 through n, and T_1 was generated by splitting the interval $[1, n]$ into two sub-intervals uniformly at random, recursively generating two trees with these two intervals as label sets and then adding a root to merge these trees. Every SPR operation in the construction of T_2 from T_1 was generated by choosing an edge xp_x to cut uniformly at random and then choosing the new parent p'_x of x uniformly at random from among all valid locations of p'_x. We constructed pairs of 100-leaf trees with 1–20 SPR operations and $25, 30, \ldots, 50$ SPR operations. We also constructed pairs of 1000-leaf trees with 1–20 SPR operations and $25, 30, \ldots, 100$ SPR operations. Ten pairs of trees were generated for each tree size and number of SPR operations.

4.2 Results

Our experiments were performed on a 3.16Ghz Xeon system with 4GB of RAM running CentOS 2.6 Linux in a Rocks 5.1 cluster. Our code was compiled using gcc 4.4.3 and optimization -O2. Each run of an algorithm was limited to 5 hours of running time. If it did not produce an answer in this time limit, we say the algorithm did not solve the given input instance in the following discussion. We refer to the FPT algorithm with all optimizations off as **fpt**, and with only the branch-and-bound optimization turned on as **bb**. The activation of the improved branching rules in Step 6 is indicated using suffixes **sc** (Case 6.1: separate components), **cob** (Case 6.2: cut only b), and **cab** (Case 6.3: cut a or b). Thus, the algorithm with all optimizations on is labelled **bb_cob_cab_sc**.

Number of cases solved. Figure 8 in Appendix C shows the number of completed protein tree runs for the given ranges of tree sizes. For this data set, our experiments showed that the average SPR distance for trees of the same size ranged between one sixth and one third of the number of leaves. All of the algorithms solved all cases with fewer than 20 leaves. **treeSAT** solved most cases with 21-30 leaves and half of the cases with 31-40 leaves but few of the larger instances. **sprdist** solved all of the cases with less than 40 leaves, most of the cases with 41-50 leaves, and half of the

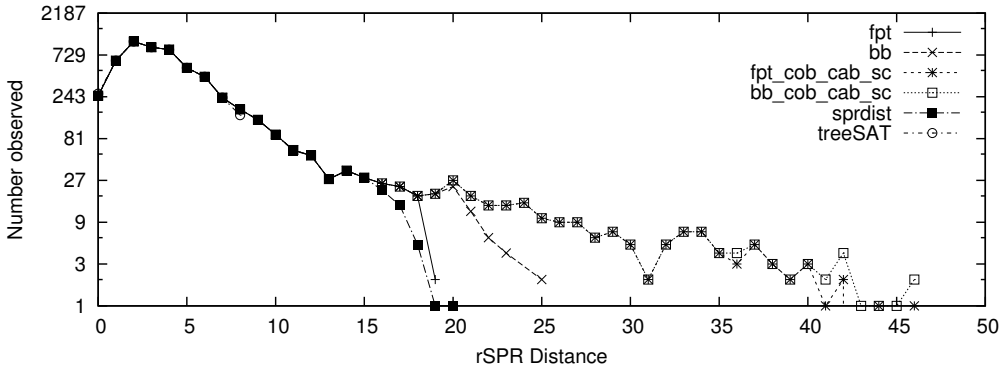

Fig. 4. Number solved by k.

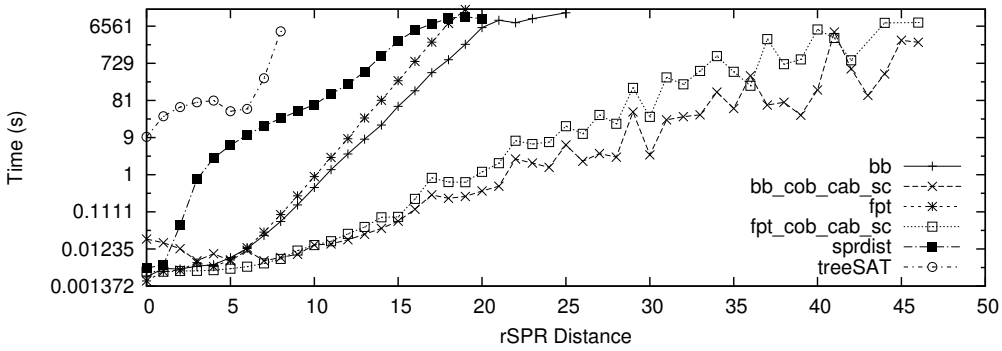

Fig. 5. Mean running time of the FPT, sprdist, and treeSAT protein tree runs.

cases with 51-100 leaves, but very few of the larger cases. **fpt** performed similarly to **sprdist** but solved all of the cases with 41-50 leaves and more of the larger instances. **bb** improved upon this somewhat. However, adding our new branching rules improved the results greatly. In particular, **bb_cob_cab_sc** solved all of the instances in the protein tree data set.

Figure 4 shows the number of protein trees found with a given SPR distance from the reference tree. The "number observed" axis is a log scale to allow easy comparison of the trees with small and large SPR distances, as the majority of the trees had small SPR distances. **treeSAT** was unable to solve any instances with an SPR distance greater than 8. **sprdist** and **fpt** solved instances with a distance as large as 20. Since **bb_cob_cab_sc** solved all the instances in this data set, we were able to verify that **sprdist** solved all instances with SPR distance up to 15, and **fpt** solved all instances with SPR distance up to 18. In contrast, **bb_cob_cab_sc** solved all instances in the data set, including instances with SPR distance 46.

Running time. Figure 5 shows the mean running time of the algorithms on *solved* protein tree instances with the given SPR distance. The time axis is a $\log_3$-scale to highlight the exponential running time of the algorithms and to allow easy comparison of the runs. The curves for the algorithms that were not able to solve all input instances "dip" for higher distance values. This is a result of taking the average running time only over solved instances, and these are likely to be the "easiest" of the input instances. The slope of the curve for **fpt** is close to 1, indicating that the algorithm is close to its worst-case running time of $O(3^k n)$. **bb** shows a marked improvement over **fpt**; however, the improvement achieved using the new branching rules is much more dramatic. **treeSAT** was much slower than all the other algorithms and although **sprdist** solved a similar

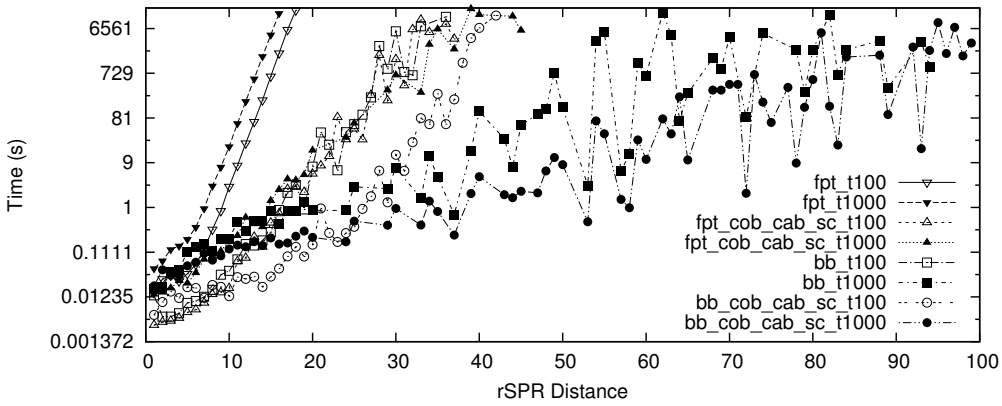

Fig. 6. Mean running time of the FPT algorithm on the random data set.

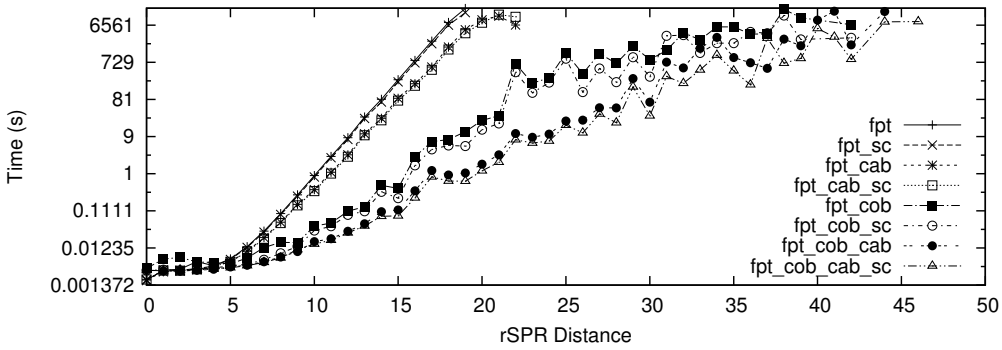

Fig. 7. Mean running time for combinations of the new cases on the protein tree runs.

number of cases as **fpt**, as shown in Figure 4, it took much longer to solve them on average. **sprdist** appears slightly faster for the cases with an SPR distance of 19 or 20, but this is likely to be an artifact of considering only solved cases, as just discussed. **bb_cob_cab_sc** solved all input instances with SPR distance up to 20 in 5.5 seconds or less, and solved instances with SPR distance up to 46 in well under 2 hours, while none of the previous methods was able to solve instances with SPR distance greater than 20 in under 5 hours.

Figure 6 shows the mean running time of the fixed-parameter algorithms on the random data set. As expected, **fpt** took 10 times longer on average for the 1000-leaf trees as for the 100-leaf trees, given the same SPR distance. **fpt_cob_cab_sc** did not show this difference, which suggests that the improved branching rules have a more pronounced impact on larger trees. **bb_cob_cab_sc** was able to solve instances with SPR distances up to 99 on the 1000-leaf trees, while a distance of 42 was the limit on 100-leaf trees. We believe that, since the proportion of SPR operations to the number of leaves is smaller for the bigger trees, the randomly generated SPR operations are more likely to operate on independent subtrees, which brings the approximation ratio of the approximation algorithm closer to the worst-case bound of 3 on these inputs. In our case, this provides better lower bounds on the true SPR distance and, thus, allows us to prune more branches in the search tree than is the case for the smaller trees.

Figure 7 shows the mean running time of the fixed-parameter algorithms without branch-and-bound on the protein tree data set when only some of the improved branching rules are used. Cases 6.1, Case 6.3, and Case 6.2 provide small, moderate and large improvements, respectively. Using all of the cases gives another large improvement, since each occurs under different conditions.

5 Conclusions

Our theoretical results improve on previous work, and our experiments confirm that these improvements have a tremendous impact in practice. Our experiments demonstrate that our algorithm can efficiently handle moderate SPR distances, solving problems with up to 144 leaves and an SPR distance of 20 in less than a second on average; for distance values up to 46, the running time was less than two hours. Our branch-and-bound improvement showed a marked improvement on larger trees, allowing us to compute distance values up to 99 on 1000-leaf synthetic trees, while we were able to compute distances up to 42 on 100-leaf synthetic trees.

While our experiments evaluated only the SPR algorithm, we expect experimental results using an implementation of the hybridization algorithm to be similar, as only Case 6.2 is more costly than in the SPR algorithm. Thus, our hybridization algorithm should also be able to solve instances beyond the reach of current hybridization approaches. Producing an implementation of the hybridization algorithm is future work. Other open problems include extending our results to multifurcating trees or the related problem of finding maximum agreement forests of multiple trees. Additionally, implementing the known kernelizations could improve the running time of the fixed-parameter algorithms for larger values of k.

References

1. M. Baroni, S. Grünewald, V. Moulton, and C. Semple. Bounding the number of hybridisation events for a consistent evolutionary history. *Journal of Mathematical Biology*, 51(2):171–182, 2005.
2. R. G. Beiko and N. Hamilton. Phylogenetic identification of lateral genetic transfer events. *BMC Evolutionary Biology*, 6(1):15, 2006.
3. R. G. Beiko, T. J. Harlow, and M. A. Ragan. Highways of gene sharing in prokaryotes. *Proceedings of the National Academy of Sciences*, 102(40):14332–14337, 2005.
4. M. Bonet and K. John. Efficiently Calculating Evolutionary Tree Measures Using SAT. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*, page 17. Springer, 2009.
5. M. L. Bonet, K. St. John, R. Mahindru, and N. Amenta. Approximating subtree distances between phylogenies. *Journal of Computational Biology*, 13(8):1419–1434, 2006.
6. M. Bordewich, S. Linz, K. John, and C. Semple. A reduction algorithm for computing the hybridization number of two trees. *Evolutionary Bioinformatics Online*, 3:86, 2007.
7. M. Bordewich, C. McCartin, and C. Semple. A 3-approximation algorithm for the subtree distance between phylogenies. *Journal of Discrete Algorithms*, 6(3):458–471, 2008.
8. M. Bordewich and C. Semple. On the computational complexity of the rooted subtree prune and regraft distance. *Annals of Combinatorics*, 8(4):409–423, 2005.
9. M. Bordewich and C. Semple. Computing the hybridization number of two phylogenetic trees is fixed-parameter tractable. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(3):458–466, 2007.
10. M. Bordewich and C. Semple. Computing the minimum number of hybridization events for a consistent evolutionary history. *Discrete Applied Mathematics*, 155(8):914–928, 2007.
11. J. Hein, T. Jiang, L. Wang, and K. Zhang. On the complexity of comparing evolutionary trees. *Discrete Applied Mathematics*, 71(1-3):153–169, 1996.
12. G. Hickey, F. Dehne, A. Rau-Chaplin, and C. Blouin. SPR distance computation for unrooted trees. *Evolutionary Bioinformatics*, 4:17–27, 2008.
13. D. M. Hillis, T. A. Heath, and K. St. John. Analysis and visualization of tree space. *Systematic Biology*, 54(3):471–482, 2005.
14. D. M. Hillis, C. Moritz, and B. K. Mable, editors. *Molecular Systematics*. Sinauer Associates, 1996.
15. W. P. Maddison. Gene trees in species trees. *Systematic Biology*, 46(3):523–536, 1997.
16. L. Nakhleh, T. Warnow, C. R. Lindner, and K. St. John. Reconstructing reticulate evolution in species—theory and practice. *Journal of Computational Biology*, 12(6):796–811, 2005.
17. E. M. Rodrigues, M.-F. Sagot, and Y. Wakabayashi. The maximum agreement forest problem: Approximation algorithms and computational experiments. *Theoretical Computer Science*, 374(1-3):91–110, 2007.
18. C. Whidden and N. Zeh. A unifying view on approximation and FPT of agreement forests. In *Proceedings of the 9th International Workshop, WABI 2009*, volume 5724 of *Lecture Notes in Bioinformatics*, pages 390–401. Springer-Verlag, 2009.
19. Y. Wu. A practical method for exact computation of subtree prune and regraft distance. *Bioinformatics*, 25(2):190, 2009.

A Proof of Lemma 3

If $F_2 \div E$ is an agreement forest of F_1 and F_2, it is an agreement forest of T_1 and T_2 because F_1 is a forest of T_1 and F_2 is a forest of T_2. So assume that $F_2 \div E$ is an agreement forest of T_1 and T_2. Since $F_2 \div E$ is a forest of F_2, it suffices to prove that there exists an edge set E_1 such that $F_1 \div E_1 = F_2 \div E$. Since $F_2 \div E$ is a forest of T_1, there exists a set E_1' of edges such that $T_1 \div E_1' = F_2 \div E$. We choose E_1 to be the subset of edges in E_1' that are present in F_1. Now it suffices to prove that $x \sim_{F_1 - E_1} y$ if and only if $x \sim_{T_1 - E_1'} y$, for all $x, y \in X$.

If $x \sim_{F_1 - E_1} y$, then none of the edges on the path from x to y in F_1 belongs to E_1'. This path also exists in T_1 because F_1 is a forest of T_1. Hence, $x \sim_{T_1 - E_1'} y$.

If $x \nsim_{F_1 - E_1} y$, we distinguish two cases. If $x, y \in \dot{T}_i$, for some $1 \leq i \leq k$, then there exists an edge $e \in E_1$ on the path from x to y in $\dot{T}_i$. Since $E_1 \subseteq E_1'$, we have $e \in E_1'$, and $x \nsim_{T_1 - E_1'} y$. If $x \in \dot{T}_i$ and $y \in \dot{T}_j$, for $i \neq j$, then $x \in \dot{F}_i$ and $y \in \dot{F}_j$. Hence, $x \nsim_{F_2 \div E} y$. This implies that $x \nsim_{T_1 - E_1'} y$ because $T_1 - E_1'$ yields $F_2 \div E$.

B Proof of Lemma 6

We prove the lemma by induction on q. For $q = 1$, the claim holds by Lemma 5. So assume that $q > 1$ and that the claim holds for $q - 1$. Assume further that b_1 is a sibling of a. As shown in [18, Theorem 1], there exists a set E' of size $e(T_1, T_2, F_2)$ such that $F \div E'$ is an AF of F_1 and F_2 and $E' \cap \{e_a, e_{b_1}, e_c\} \neq \emptyset$. If $E' \cap \{e_a, e_c\} \neq \emptyset$, we are done; otherwise $e_{b_1} \in E'$. Let $F_2' := F_2 \div \{e_{b_1}\}$. In F_2', the path from a to c has $q - 1$ pendant nodes, namely $b_2, b_3, \ldots, b_q$. Thus, by the induction hypothesis, there exists a set E'' of size $e(T_1, T_2, F_2') = e(T_1, T_2, F_2) - 1$ such that $F_2' \div E''$ is an AF of F_1 and F_2' and such that $E'' \cap \{e_a, e_c\} \neq \emptyset$ or $\{e_{b_2}, e_{b_3}, \ldots, e_{b_q}\} \subseteq E''$. The set $E := E'' \cup \{e_{b_1}\}$ has size $|E''| + 1 = e(T_1, T_2, F_2)$, and $F_2 \div E = F_2' \div E''$ is an agreement forest of F_1 and F_2 (because it is an agreement forest of F_1 and F_2'). Since $E'' \cap \{e_a, e_c\} \neq \emptyset$ or $\{e_{b_2}, e_{b_3}, \ldots, e_{b_q}\} \subseteq E''$, we have $E \cap \{e_a, e_c\} \neq \emptyset$ or $\{e_{b_1}, e_{b_2}, \ldots, e_{b_q}\} \subseteq E$; that is, E satisfies the lemma.

The proof for AAF is identical to the one for AF, as Theorem 1 of [18] holds for both AF and AAF. However, we have to use Lemma 7 instead of Lemma 5 for the base case.

C Supplementary Figures

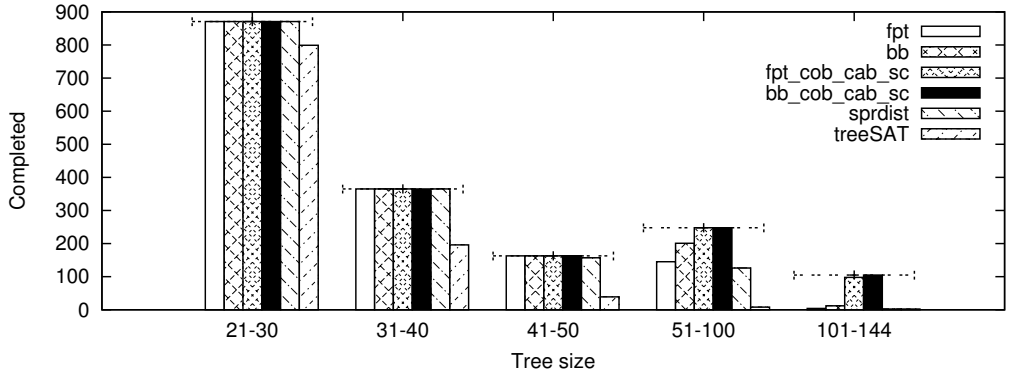

Fig. 8. Completion on the protein tree runs grouped by tree size. Abbreviations are defined in the text.