Fast FPT Algorithms for Computing Rooted Agreement Forests: Theory and Experiments (Extended Abstract*)

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 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 algorithm is orders of magnitude faster and can handle much larger trees and SPR distances than the best previous methods, **treeSAT** and **sprdist**.

1 Introduction

Phylogenetic trees are used to represent the evolution of a set of species (taxa) [13]. In addition to 'vertical' inheritance from parent to offspring, genetic material can be exchanged between contemporary organisms via lateral gene transfer, recombination and hybridization. These processes enable the rapid spread of antibiotic resistance and other harmful traits in pathogenic bacteria, and more generally allow species to rapidly adapt to new environments. Untangling vertical and lateral evolutionary histories requires the comparison of phylogenetic trees, and metrics that model reticulation events using subtree prune-and-regraft (SPR) [11] or hybridization [1] permutations are of particular interest, since the resulting series of permutations has a direct evolutionary interpretation [14, 15].

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, 16] and fixed-parameter (FPT) algorithms [7, 9], as well as heuristic approaches [3, 12], for computing these distances. The main tool of

* Details can be found in [17].

** Supported by an NSERC PGS-D graduate scholarship.
*** Canada Research Chair; supported in part by NSERC and Genome Atlantic.
† Canada Research Chair; supported in part by NSERC.

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 use a bounded search tree approach and take $O(3^k n)$ and $O(3^k n \log n)$ time, respectively ($O(3^k k + n^3)$ and $O(3^k k \log k + n^3)$ using standard kernelizations [8,9].)

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 SPR distance of [16,18] to prune branches in the search tree that are guaranteed to be unsuccessful. This reduces the size of the search tree substantially and leads to a corresponding decrease in running time. We demonstrate that each of the improved branching rules and the pruning of unsuccessful branches have a marked and distinct effect on the performance of the algorithm. Our experiments confirm that our algorithm is orders of magnitude faster than the currently best exact alternatives [4,20] 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 [20]. 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 and notation. Section 3 presents our 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 [7–9,16, 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. 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 v with only one child and its incident edges. If v was the root of the current tree, its child becomes the new root; otherwise an edge is added between v's parent and child. See Figure 1(c).

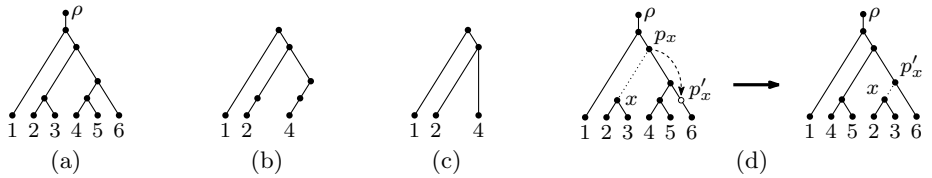

Fig. 1. (a) An X-tree T. (b) The subtree $T(V)$ for $V = \{1, 2, 4\}$. (c) $T|V$. (d) Illustration of an SPR operation.

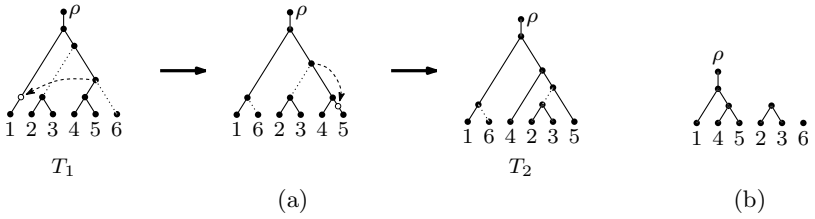

Fig. 2. (a) SPR operations transforming T_1 into T_2. Each operation changes the top endpoint of one of the dotted edges. (b) The corresponding agreement forest, which can be obtained by cutting the dotted edges in both trees.

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 a 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.

For a forest F whose components $T_1, T_2, \ldots, T_k$ have label sets $X_1, X_2, \ldots, X_k$, we say F *yields* the forest 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, $T_i|X_i = \emptyset$. For a subset E of edges of G, we use $F - E$ to denote the forest obtained by deleting the edges in E from F, and $F \div E$ to denote the forest yielded by $F - E$. We say that $F \div E$ is a *forest of* F.

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 $F_1^b = F_2^{b'}$. We refer to a' and b' as a and b for simplicity.

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.

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.

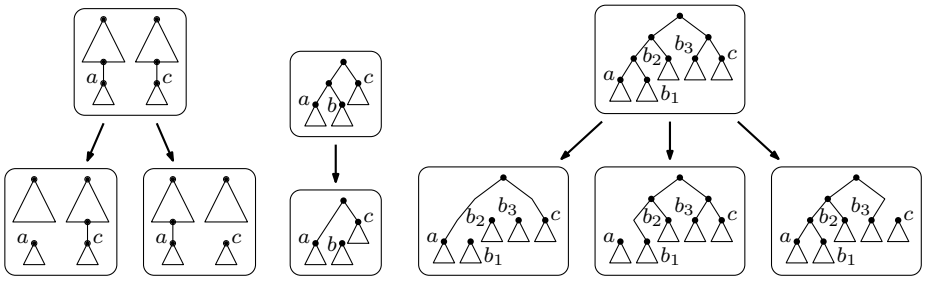

Fig. 3. The cases in Step 6 of the rooted MAF algorithm. Only $\dot{F}_2$ is shown. Each box represents a recursive call.

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, thus, 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 (see Figure 3).
   - 6.1. If $a \not\sim_{F_2} c$, 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 calls $\textsc{Maf}(F_1, F_2 \div \{e_{b_1}, e_{b_2}, \ldots, e_{b_q}\}, k-q)$, $\textsc{Maf}(F_1, F_2 \div \{e_a\}, k-1)$, and $\textsc{Maf}(F_1, F_2 \div \{e_c\}, k-1)$.

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

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 $\dot{F}_2$ 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 by distinguishing between Cases 6.1–6.3 we achieve an improved running time.

Theorem 1. *For two rooted X-trees T_1 and T_2 and a parameter k, it takes $O((1+\sqrt{2})^k n) = O(2.42^k n)$ 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 $O(n)$ time. Thus, the running time claimed in Theorem 1 follows if we can bound the number of recursive calls by $O((1+\sqrt{2})^k)$. The number of recursive calls spawned by an invocation depends only on k with the following recurrence

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

because Case 6.3 dominates the other two cases and $q \geq 2$ in this case. Simple substitution shows that this recurrence solves to $I(k) = O((1+\sqrt{2})^k)$. 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 $F_2 \div E$ is an AF of T_1 and T_2. $F_2 \div E$ 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. Due to the lack of space, proofs are omitted but can be found in [17].

Lemma 1 (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 $E \cap \{e_a, e_c\} \neq \emptyset$.*

Lemma 2 (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 $e_b \in E$.*

Lemma 3 (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 $\{e_{b_1}, e_{b_2}, \ldots, e_{b_q}\} \subseteq E$.*

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 1 and 3 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 replace it with the following case when computing an MAAF:

6.2′. If the path from a to c in F_2 has one pendant node 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 4 below instead of Lemma 2 to show that we cut the right edges in Case 6.2′. To bound the running time of the algorithm, we observe that the recurrence for the number of recursive calls in Case 6.2′ is $I(k) = 1 + 2I(k-1)$, which is still dominated by the recurrence for Case 6.3. Using that the running time of the MAAF procedure is $T(n, k) = O(2^k n \log n)$, substitution yields the claimed bound. Again, a proof of this lemma can be found in [17].

Lemma 4 (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 $e_c \in E$.*

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** [20] 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** [3] because **sprdist** outperformed it and other heuristics at finding the exact SPR distance between binary rooted phylogenies [20].

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 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]. Source code for our algorithm is available at [19].

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 $e(T_1, T_2, T_2) \geq \lceil k'/3 \rceil$; 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.

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] contains 5689 protein trees with 10 to 144 leaves (each corresponding to a different microbial genome); each of these was compared in turn to a rooted reference tree covering all 144 genomes. 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 created by first generating a random tree T_1 and then transforming it into a second tree T_2 using a known number of random SPR operations. Note that the SPR distance may be lower because the sequence of SPR operations we generated 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. Random SPR operations were 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 with $25, 30, \ldots, 50$ SPR operations. We also constructed pairs of 1000-leaf trees with 1–20 SPR operations and with $25, 30, \ldots, 100$ SPR operations. For each tree size and number of SPR operations we generated ten pairs of trees.

4.2 Results

Our experiments were performed on a 3.16Ghz Xeon system with 4GB of RAM and 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 instances solved. Figure 4 shows the number of solved protein tree instances for the given ranges of tree sizes. 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 instances with 20 or fewer leaves and only **treeSAT** did not solve all instances with 40 or fewer leaves. **sprdist** solved most of the instances with 41-50 leaves, and half of the instances with 51-100 leaves, but very few of the larger instances. **fpt** performed similarly to **sprdist** but solved all of the instances with 41-50 leaves and more of the larger instances than **sprdist**. **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 this data set.

Figure 5 shows the number of protein trees found with a given SPR distance from the reference tree. The "number solved" axis is a $\log_3$-scale to allow easy comparison of the trees with small and large SPR distances, as the majority had small SPR distances. **treeSAT** was unable to solve any instances with 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, including instances with an SPR distance of 46, we were able to verify that **sprdist** and **fpt** solved all instances with SPR distance up to 15 and 18, respectively.

Running time. Figure 6 shows the mean running time of the algorithms on *solved* protein tree instances with the given SPR distance. The time axis here and in the following figures 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 some of the algorithms 'dip" for higher distance values, which is a result of taking the average running time only over solved 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 number of instances as **fpt**, as shown in Figure 5, it took much longer to solve them on average. The two instances that **sprdist** solved with an SPR distance of 19 and 20 are an exception to this, but that is likely an artifact

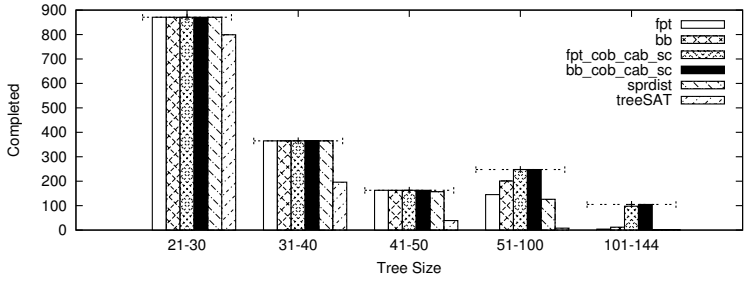

Fig. 4. Completion on the protein tree runs grouped by tree size. Abbreviations are defined in the text.

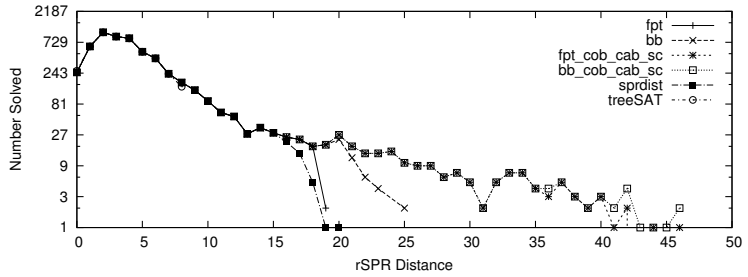

Fig. 5. Number of protein trees solved by rSPR distance.

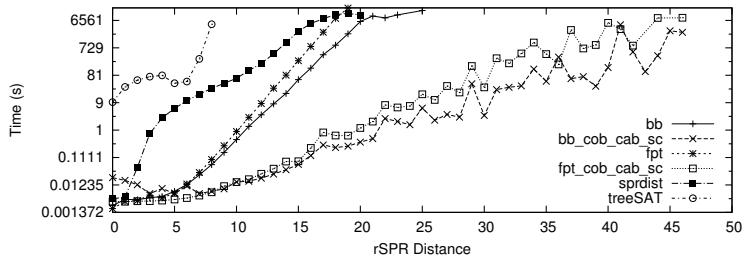

Fig. 6. Mean running time of the FPT, sprdist, and treeSAT protein tree runs.

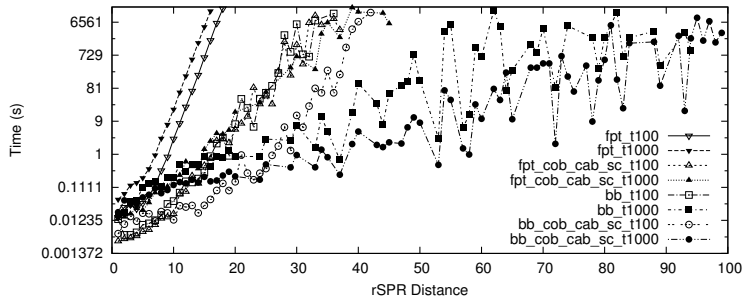

Fig. 7. Mean running time of the FPT algorithm on the data set of randomly permuted trees. The trees with 100 and 1000 leaves are shown separately.

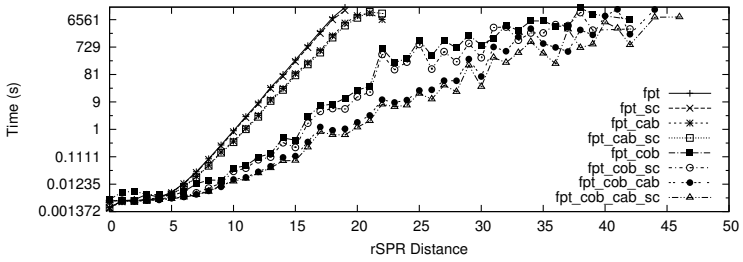

Fig. 8. Mean running time for combinations of the new cases on the protein tree runs.

of considering only solved instances. **bb_cob_cab_sc** solved all input instances with SPR distance of 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 7 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 its 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 8 shows the mean running time of the fixed-parameter algorithms without branch-and-bound on the protein tree data set and using only some of the improved branching rules. Case 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 algorithm efficiently solves 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 approach showed a marked improvement on larger trees, allowing us to compute distance values up to 42 on 100-leaf synthetic trees and 99 on 1000-leaf synthetic trees.

We expect experimental results using an implementation of the hybridization algorithm would 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.

References

1. Baroni, M., Grünewald, S., Moulton, V., Semple, C.: Bounding the number of hybridisation events for a consistent evolutionary history. J. Math. Biol. 51(2), 171–182 (2005)
2. Beiko, R.G., Harlow, T.J., Ragan, M.A.: Highways of gene sharing in prokaryotes. P. Natl. Acad. Sci. USA 102(40), 14332–14337 (2005)
3. Beiko, R.G., Hamilton, N.: Phylogenetic identification of lateral genetic transfer events. BMC Evol. Biol. 6(1), 15 (2006)
4. Bonet, M.L., St. John, K.: Efficiently Calculating Evolutionary Tree Measures Using SAT. In: SAT 2009. LNCS, vol. 5584, pp. 4–17. Springer-Verlag (2009)
5. Bonet, M.L., St. John, K., Mahindru, R., Amenta, N.: Approximating subtree distances between phylogenies. J. Comp. Biol. 13(8), 1419–1434 (2006)
6. Bordewich, M., Linz, S., St. John, K., Semple, C.: A reduction algorithm for computing the hybridization number of two trees. Evol. Bioinform. 3, 86–98 (2007)
7. Bordewich, M., McCartin, C., Semple, C.: A 3-approximation algorithm for the subtree distance between phylogenies. J. Disc. Alg. 6(3), 458–471 (2008)
8. Bordewich, M., Semple, C.: On the computational complexity of the rooted subtree prune and regraft distance. Annals of Comb. 8(4), 409–423 (2005)
9. Bordewich, M., Semple, C.: Computing the hybridization number of two phylogenetic trees is fixed-parameter tractable. IEEE/ACM T. Comp. Biol. 4(3), 458–466 (2007)
10. Bordewich, M., Semple, C.: Computing the minimum number of hybridization events for a consistent evolutionary history. Disc. Appl. Math. 155(8), 914–928 (2007)
11. Hein, J., Jiang, T., Wang, L., Zhang, K.: On the complexity of comparing evolutionary trees. Disc. Appl. Math. 71(1-3), 153–169 (1996)
12. Hickey, G., Dehne, F., Rau-Chaplin, A., Blouin, C.: SPR distance computation for unrooted trees. Evol. Bioinf. 4, 17–27 (2008)
13. Hillis, D.M., Moritz, C., Mable, B.K. (eds.): Molecular Systematics. Sinauer Associates (1996)
14. Maddison, W.P.: Gene trees in species trees. Syst. Biol. 46(3), 523–536 (1997)
15. Nakhleh, L., Warnow, T., Lindner, C.R., St. John, K.: Reconstructing reticulate evolution in species—theory and practice. J. Comp. Biol. 12(6), 796–811 (2005)
16. Rodrigues, E.M., Sagot, M.F., Wakabayashi, Y.: The maximum agreement forest problem: Approximation algorithms and computational experiments. Theor. Comp. Sci. 374(1-3), 91–110 (2007)
17. Whidden, C., Beiko R.G., Zeh, N.: Fast FPT algorithms for computing rooted agreement forests: Theory and experiments. Tech. Rep. CS-2010-03, Faculty of Computer Science, Dalhousie University (2010)
18. Whidden, C., Zeh, N.: A unifying view on approximation and FPT of agreement forests. In: WABI 2009. LNCS, vol. 5724, pp. 390–401. Springer-Verlag (2009)
19. Whidden, C., http://kiwi.cs.dal.ca/Software/RSPR : rSPR FPT Software
20. Wu, Y.: A practical method for exact computation of subtree prune and regraft distance. Bioinformatics 25(2), 190–196 (2009)