A Unifying View on Approximation and FPT of Agreement Forests

Chris Whidden
Norbert Zeh

Technical Report CS-2009-02

June 15, 2009

Faculty of Computer Science
6050 University Ave., Halifax, Nova Scotia, B3H 1W5, Canada

A Unifying View on Approximation and FPT of Agreement Forests

Chris Whidden[⋆] and Norbert Zeh[⋆⋆]

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

Abstract. We provide a unifying view on the structure of different types of agreement forests using a "shifting lemma" proved by Bordewich et al. We consider rooted and unrooted maximum agreement forests (MAF's) and rooted maximum acyclic agreement forests (MAAF's). Their sizes are known to respectively correspond to the subtree prune and regraft distance between two rooted phylogenies, the tree bisection and reconnection distance between two unrooted phylogenies, and the hybridization number of two rooted phylogenies. With the exception of approximation of rooted MAF, we obtain improved approximation and fixed-parameter algorithms for the sizes of all of the above types of agreement forests, most of which are substantial improvements over the best previous results. To the best of our knowledge, our 3-approximation algorithm for MAAF size is the first approximation result for MAAF. For rooted MAF size, we obtain an alternative and, in our opinion, simpler correctness proof for the 3-approximation algorithm by Rodrigues et al.

[⋆] Supported by an NSERC CGS-M graduate scholarship.

[⋆⋆] Supported in part by NSERC, the Canadian Foundation for Innovation, and the Canada Research Chairs program.

1 Introduction

Phylogenies, or evolutionary trees, are a standard model to represent the evolutionary history of a set of species and are an indispensable tool in evolutionary biology [20]. Unfortunately, most useful optimization criteria for creating phylogenies, such as maximum likelihood [1] and Steiner tree [14], are NP-hard. Assessing the quality of proposed phylogenies found using approximate or heuristic methods often involves measuring the similarity of many computed phylogenies using an appropriate distance metric. This is feasible only if distances under the used metric can be computed efficiently. Fast distance computations are also required for the analysis and visualization of tree space [3, 19].

A number of metrics are commonly used to define the distance between phylogenies. The Robinson-Foulds distance [24] is popular, as it can be calculated in linear time [13]. Other metrics, such as the *tree bisection and reconnection* (TBR) and *subtree prune and regraft* (SPR) distances [20], are often more meaningful. Not all groups of species follow a simple tree-like evolutionary pattern. Collectively known as reticulation events, non-tree-like evolutionary processes such as hybridization, horizontal gene transfer, and recombination result in species being composites of genes derived from different ancestors. Given two trees on the same set of species but derived by analyzing different genes, the rooted SPR (rSPR) distance and the hybridization number of the two trees are important tools that often help to discover such events. In particular, the rSPR distance provides a lower bound on the number of reticulation events [4, 5], and this metric has been regularly used to model reticulate evolution [21, 22]. The close relationship between rSPR operations and reticulation events has also led to advances in network models of evolution [4, 11, 22].

While TBR distance, SPR distance, and hybridization number capture biologically meaningful notions of similarity between phylogenies, their practical use has been limited by the fact that they are NP-hard to compute [2, 9, 11, 18]. This has led to an interest in approximation and fixed-parameter algorithms for computing these distances (see [23, 27] for an introduction to approximation algorithms and fixed-parameter tractability), as well as heuristic approaches [5, 18]. Hein et al. [16] claimed a 3-approximation algorithm for computing SPR distances and proposed the notion of a maximum agreement forest (MAF) as the main tool underlying both the approximation algorithm and a proposed NP-hardness proof for computing SPR distances. The central claim was that the number of components in an MAF of two phylogenies is one more than the number of SPR operations needed to transform one into the other. Unfortunately, there were subtle mistakes in the proofs. Allen and Steel [2] proved that the number of components in an MAF is in fact one more than the *TBR* distance between the two trees. Rodrigues et al. [25] provided instances where the algorithm of [16] provides an approximation guarantee no better than 4 for the size of an MAF, thereby disproving the 3-approximation claim of [16]. They also proposed a modification to the algorithm, which they claimed to produce a 3-approximation for TBR. A counterexample to this claim was provided by Bonet et al. [7], who showed, however, that both the algorithm of [16] and of [25] compute 5-approximations of the rSPR distance between two rooted phylogenies, and that the algorithms can be implemented in linear time. The approximation ratio was improved to three by Bordewich et al. [8], but at the expense of an increased running time of $\mathrm{O}(n^5)$.[1] A second 3-approximation algorithm presented in [25] achieves a running time of $\mathrm{O}(n^2)$. Using entirely different ideas, Chataigner [12] obtained an 8-approximation algorithm for TBR distances of two or more trees.

Fixed-parameter algorithms provide exact answers efficiently, as long as the distance between the two trees is not too big. The currently best such algorithm for rSPR distance is due to Bordewich et al. [8] and runs in $\mathrm{O}(4^k \cdot k^4 + n^3)$ time, where k is the distance between the two trees. For TBR distance, the currently best result is due to Hallett and McCartin [15], who provide an algorithm with running time $\mathrm{O}(4^k \cdot k^5 + p(n))$, where $p(\cdot)$ is a polynomial function. An earlier algorithm for this problem by Allen and Steel [2] had running time $\mathrm{O}(k^{3k} + p(n))$. For unrooted SPR, Hickey et al. [17] first claimed a fixed-parameter algorithm,

[1] Using non-trivial but standard data structures, the running time can be reduced to $\mathrm{O}(n^4)$.

	Approximation	FPT
TBR	Previous: 8-approximation in polynomial time [12] New: 3-approximation in linear time	Previous: $O(4^k k^5 + p(n))$ time [15] New: $O(4^k k + n^3)$ or $O(4^k n)$ time
rSPR	Previous: 3-approximation in $O(n^2)$ time [25] New: 3-approximation in linear time	Previous: $O(4^k k^4 + n^3)$ time [8] New: $O(3^k k + n^3)$ or $O(3^k n)$ time
Hybridization	Previous: — New: 3-approximation in $O(n \log n)$ time	Previous: $O((28k)^k + n^3)$ time [10] New: $O(3^k k \log k + n^3)$ or $O(3^k n \log n)$ time

Table 1. Previous and new results on rSPR distance, TBR distance, and hybridization number.

but the correctness proof was flawed. Recently, St. John [26] proposed a correction of the central technical lemma in Hickey et al.'s result. In [10], Bordewich and Semple provided a fixed-parameter algorithm for the hybridization number of two rooted phylogenies.

The contribution of this paper is to provide a unifying view on the results discussed in the previous paragraph and improve on most of them along the way, substantially in most cases. In particular, we show that the framework of the algorithms of [7, 16, 25] can be used not only to approximate the rSPR distance between two rooted phylogenies but also to obtain approximation and FPT algorithms for rSPR distance, TBR distance, and hybridization number. To the best of our knowledge, an approximation algorithm for hybridization number had not been obtained before. The key to showing these results is a "shifting lemma" proved by Bordewich et al. [8]. Table 1 shows our new results in comparison to the best previous results. The 3-approximation algorithm for rSPR is the algorithm of Rodrigues et al. [25], with a minor modification to reduce its running time to linear. We believe that the correctness proof obtained using our approach is simpler than the one presented in [25].

The rest of the paper is organized as follows. We present the necessary terminology and notation in Section 2. Section 3 presents the main structural theorems that are at the heart of both the approximation and fixed-parameter algorithms. Section 4 presents the approximation algorithms. Section 5 discusses briefly how to turn the approximation algorithms into fixed-parameter algorithms based on bounded search trees.

2 Preliminaries

This section introduces the terminology and notation used throughout the paper. We mostly follow the definitions from [2, 7–9, 25]. An *unrooted binary phylogenetic X-tree* is a tree T whose leaves are the elements of X and each of whose internal nodes has degree three. We refer to these trees simply as X-trees. The set X is called the *label set* of T and contains the species whose evolutionary relationship the tree represents. For a subset V of X, we use $T(V)$ to denote 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 $T(V)$ by splicing out all nodes of degree two (i.e., replacing the vertex and its two incident edges with a single edge); see Figure 1(c). Such a splice operation is called a *forced contraction*.

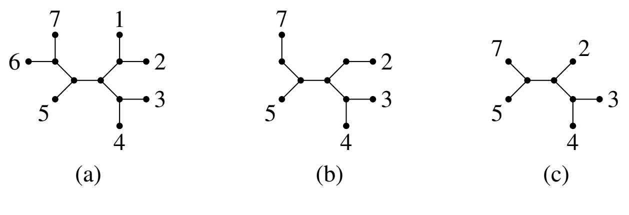


Fig. 1. (a) An X-tree T. (b) The subtree $T(V)$ for $V = \{2,3,4,5,7\}$. (c) The tree $T|V$ obtained by forced contractions on $T(V)$.

As discussed in the introduction, several distance measures between X-trees have been defined in the literature, reflecting different biological concepts, such as horizontal gene transfers. Of interest here are distance measures based on SPR and TBR operations. Given an X-tree T, an SPR operation cuts an edge xy in T, thereby dividing T into two subtrees T_x and T_y containing x and y, respectively. Then it subdivides an edge of T_y using a new vertex y' and adds an edge between x and y' to reconnect T_x and T_y. Finally, vertex y is removed using a forced contraction. A TBR operation also starts by cutting an edge xy, but it subdivides an edge in *each* of the two trees, T_x and T_y, and then adds an edge $x'y'$ to reconnect T_x and T_y, where x' and y' are the two vertices created in T_x and T_y by these edge subdivisions. Vertices x and y are then removed using forced contractions. Figure 2 illustrates both operations.

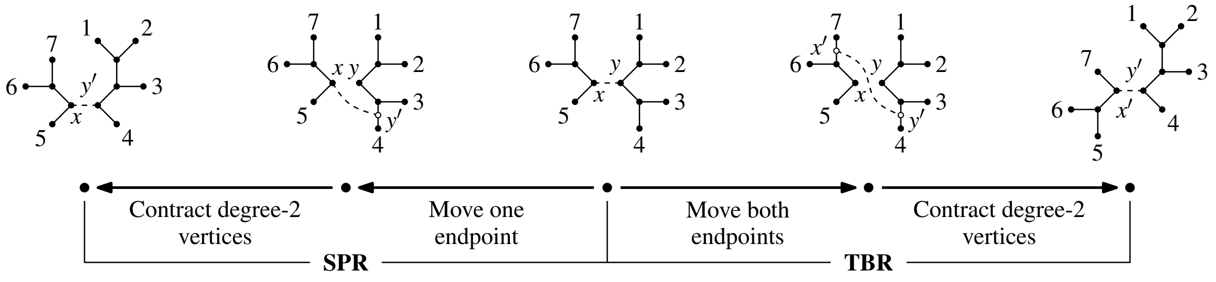


Fig. 2. Illustration of TBR and SPR operations.

A *rooted* X-tree is obtained from an unrooted one, T, by subdividing one of T's edges, declaring the node this introduces to be the root, and defining parent-child and ancestor-descendant relations in the standard way with respect to the chosen root. While TBR operations seem inherently unrooted, as applying such an operation to a rooted tree would require rerooting at least part of the tree, SPR operations translate naturally to the rooted case: a rooted SPR (rSPR) operation performs the same transformation as an SPR operation for unrooted trees; however, in the description of the operation above, vertex y is chosen to be the parent of vertex x. Thus, rSPR operations leave ancestor-descendant relations in the two subtrees intact. Moreover, at the end of the operation, forced contractions are applied only to non-root nodes of degree two, and the root is removed if edge xy was one of its incident edges, thereby making its child the new root of the tree.

TBR, SPR, and rSPR operations define distance measures $d_{\mathrm{TBR}}(\cdot,\cdot)$, $d_{\mathrm{SPR}}(\cdot,\cdot)$, and $d_{\mathrm{rSPR}}(\cdot,\cdot)$ between (unrooted or rooted) X-trees, where the distance between two trees is the number of such operations required to transform one into the other. A related distance measure for rooted X-trees is their *hybridization number*, $\mathrm{hyb}(T_1, T_2)$. This distance is defined in terms of hybrid networks of the two trees, where a hybrid network of T_1 and T_2 is a directed acyclic graph H such that both T_1 and T_2 can be obtained from H by deleting edges and performing forced contractions. For a vertex $x \in H$, let $\deg_{\mathrm{in}}(x)$ be its in-degree and $\deg^-_{\mathrm{in}}(x) = \max(0, \deg_{\mathrm{in}}(x) - 1)$. Then the hybridization number of T_1 and T_2 is $\min_H \sum_{x \in H} \deg^-_{\mathrm{in}}(x)$, where the minimum is taken over all hybrid networks of T_1 and T_2.

TBR distance, rSPR distance, and hybridization number are known to be one less than the number of connected components in appropriately defined maximum agreement forests (MAF's) [2,4,9]. To define these MAF's, we first introduce some terminology.

Given a forest F and a subset E of its edges, we write $F - E$ to denote the forest obtained by deleting the edges in E from F. If forest F has components $T_1, T_2, \ldots, T_k$ with label sets $X_1, X_2, \ldots, X_k$, we say that forest F *yields* forest F' if F' has components $T'_1, T'_2, \ldots, T'_k$ (some of them possibly empty) and, for all $1 \le i \le k$, $T'_i = T_i|X_i$; if all nodes of a component T_i are unlabelled (that is, $X_i = \emptyset$), we define $T_i|X_i = \emptyset$. For an X-tree T, we say that F is a *forest of* T if there exists a subset E of T's edges such that $T - E$ yields F.

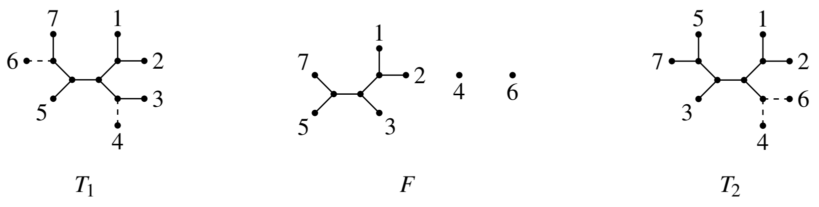


Fig. 3. Two X-trees T_1 and T_2 and an agreement forest F of T_1 and T_2. F is obtained from each tree by cutting the dashed edges.

Given two X-trees T_1 and T_2 and two forests F_1 of T_1 and F_2 of T_2, a forest F is an *agreement forest* of F_1 and F_2 if there exist edge sets E_1 and E_2 such that $F_1 - E_1$ and $F_2 - E_2$ yield F; see Figure 3. Forest F is a *maximum agreement forest* (MAF) if there is no agreement forest of F_1 and F_2 with fewer connected components than F. We use $m(F_1, F_2)$ to denote the number of connected components in an MAF of F_1 and F_2 and $e(F_1, F_2)$ to denote the size of the smallest edge set E_2 such that $F_2 - E_2$ yields F. The following theorem by Allen and Steel [2] establishes the relationship between TBR distances and unrooted MAF sizes.

Theorem 1. *For two unrooted X-trees T_1 and T_2, $d_{TBR}(T_1, T_2) = e(T_1, T_2) = m(T_1, T_2) - 1$.*

In the rooted case, MAF's are similarly related to rSPR distances. This, however, is true only if the MAF is defined with respect to augmented versions of the two trees, obtained by adding a new root node with label ρ to both trees and making the original root of each tree the child of ρ. An agreement forest of two forests F_1 and F_2 of T_1 and T_2 is then defined as a collection $\{T'_\rho, T'_1, T'_2, \ldots, T'_k\}$ of rooted trees with label sets $X_\rho, X_1, X_2, \ldots, X_k$ that satisfy the following conditions [9]:

1. The label sets $X_\rho, X_1, X_2, \ldots, X_k$ partition $X \cup \{\rho\}$, and $\rho \in X_\rho$.
2. For all $i \in \{\rho, 1, 2, \ldots, k\}$, $T'_i = F_1|X_i = F_2|X_i$ in the rooted sense.
3. The graphs in each of the sets $\{F_1(X_i) \mid i \in \{\rho, 1, 2, \ldots, k\}\}$ and $\{F_2(X_i) \mid i \in \{\rho, 1, 2, \ldots, k\}\}$ are vertex-disjoint trees.

An MAF is again one with the minimum number of connected components. Bordewich and Semple [9] proved the following theorem, where $m(F_1, F_2)$ and $e(F_1, F_2)$ are defined as in the unrooted case.

Theorem 2. *For two rooted X-trees T_1 and T_2, $d_{rSPR}(T_1, T_2) = e(T_1, T_2) = m(T_1, T_2) - 1$.*

The hybridization number of two rooted X-trees T_1 and T_2 corresponds to an MAF of the two trees with an additional constraint. For two forests F_1 and F_2 of T_1 and T_2, an agreement forest F of F_1 and F_2 is said to contain a cycle if there exist two nodes x and y that are roots of trees in F and such that x is an ancestor of y in T_1, while y is an ancestor of x in T_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 set of labelled descendants of x in F and defining $\phi_i(x)$ to be the lowest common ancestor in T_i of all nodes in X_x.) A *maximum acyclic agreement forest* (MAAF) of F_1 and F_2 is an agreement forest with the minimum number of connected components among all acyclic agreement forests of F_1 and F_2. We denote the size of such a forest by $\bar{m}(F_1, F_2)$ and the number of edges that need to be cut in a forest F of F_2 to obtain such a forest by $\bar{e}(F_1, F_2, F)$. The following result by Baroni et al. [4] relates $\bar{e}(T_1, T_2, T_2)$ to $\mathrm{hyb}(T_1, T_2)$.

Theorem 3. *For two rooted X-trees T_1 and T_2, $hyb(T_1, T_2) = \bar{e}(T_1, T_2, T_2) = \bar{m}(T_1, T_2) - 1$.*

By Theorems 1–3, it suffices to compute or approximate the size of the right kind of MAF in order to compute or approximate the distance of two trees under one of the three metrics, $d_{\mathrm{TBR}}(\cdot,\cdot)$, $d_{\mathrm{rSPR}}(\cdot,\cdot)$ or $\mathrm{hyb}(\cdot,\cdot)$. Thus, we focus on MAF's in the remainder of this paper.

For a forest F and two nodes a and b of F, we write $a \sim_F b$ to indicate that a and b belong to the same connected component of F, that is, there exists a path from a to b in F. We say that (a,c) is a *sibling pair* of a forest F if a and c are labelled leaves of T with the same neighbour, which we denote by r_{ac}. As previous algorithms for rooted MAF's, our algorithms rely on the following two lemmas, which were proved by Bordewich and Semple [9] and Bonet et al. [7] for the rooted case, but are easily seen to apply also to the unrooted case and to acyclic agreement forests.

Lemma 1. *Let (F_1,F_2) be a pair of forests, let (a,c) be a sibling pair that exists in both forests, and let (F_1',F_2') be the forests obtained by removing a and c from F_1 and F_2 and assigning label (a,c) to r_{ac}. We refer to this as* contracting *the sibling pair (a,c). Then $e(F_1,F_2) = e(F_1',F_2')$ and $\bar{e}(F_1,F_2,F_2) = \bar{e}(F_1',F_2',F_2')$.*

Lemma 2. *Let (F_1,F_2) be a pair of forests, let c be a singleton in F_2, and let (F_1',F_2') be the forests obtained by removing c from both forests. Then $e(F_1,F_2) = e(F_1',F_2')$ and $\bar{e}(F_1,F_2,F_2) = \bar{e}(F_1',F_2',F_2')$.*

3 The Structure of Agreement Forests

This section presents the structural results that provide the intuition and correctness proofs for the algorithms presented in Sections 4 and 5. All these algorithms start with a pair of trees (T_1,T_2), cut edges, remove singletons, and contract sibling pairs in both trees until they have one or two nodes left and are identical. The intermediate state is that T_1 has been reduced to a tree T, and T_2 to a forest F. Each iteration has to decide which edges in F to cut next. The results in this section identify small edge sets in F so that at least one edge in each of these sets has the property that cutting it reduces $e(T,F)$ by one. Thus, the approximation algorithm cuts all edges in the identified set, and the size of the edge set cut in each step gives the approximation ratio of the algorithm. The FPT algorithm tries each edge in the set in turn, so that the size of the set gives the branching factor for a bounded search tree algorithm. The following lemma by Bordewich et al. [8] is the central tool used in all our proofs. They proved this result for rooted trees. It is trivial, however, to verify that it holds also for unrooted trees.

Lemma 3 (Shifting Lemma). *Let T be an X-tree, F a forest of T, e and f edges in the same component 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 (1) $v_f \sim_{F-E} v_e$ and (2) $x \nsim_{F-(E\cup\{e\})} v_f$ for all $x \in X$, then $F-E$ and $F-(E\setminus\{f\}\cup\{e\})$ yield the same forest.*[2]

The other tool we need is an observation that relates incompatible triples and quartets to agreement forests. A *triple* $ab|c$ in a rooted tree T is defined by three leaves a, b, c such that the path from a to b is vertex-disjoint from the path from c to the root. A *quartet* $ab|cd$ in an unrooted tree T is defined by four leaves a, b, c, d such that the two paths from a to b and from c to d are vertex-disjoint. Given a tree T and a forest F, we say a triple $ab|c$ or quartet $ab|cd$ of T is *incompatible* with F if its leaves either do not all belong to the same component of F or define a different triple or quartet; for example, $ac|b$ or $ac|bd$.

Observation 1. *(i) Let T_1 and T_2 be two rooted X-trees, T and F defined as above, and F' an agreement forest of T and F. If $ab|c$ is a triple of T incompatible with F, then $a \nsim_{F'} b$ or $a \nsim_{F'} c$.*

(ii) Let T_1 and T_2 be two unrooted X-trees, T and F defined as above, and F' an agreement forest of T and F. If $ab|cd$ is a quartet of T incompatible with F, then $a \nsim_{F'} b$, $a \nsim_{F'} c$ or $c \nsim_{F'} d$.

Now consider a tree T and a forest F as above, and let (a,c) be a sibling pair of T that does not exist in F and such that neither a nor c is a singleton in F. If a and c belong to the same tree of F, the *sibling* b of a in F is the node adjacent to a's neighbour that does not belong to the path from a to c in F. Otherwise, b is

[2] In the rooted case, it is assumed that $\rho \in X$.

any node at distance two from a in F. Note that b is not necessarily a leaf. We use e_a and e_b to denote the edges connecting a and b to their common neighbour r_{ab}, and B to denote the subtree of F induced by all nodes b' such that edge e_b belongs to the path from b' to a. The sibling d of c and its attached subtree D are defined analogously. In the rooted case, if a and c belong to the same component of F, we assume that the distance from the root to a is no less than the distance from the root to c. This implies that r_{ab} is the parent of a and b in F.

With these tools in hand, we are now ready to prove three results characterizing edges that need to be cut in order to make progress towards an M(A)AF. The first result considers rooted MAF's.

Theorem 4. *Let T_1 and T_2 be two rooted X-trees, and let F be a forest of T_2. Let (a,c) be a sibling pair of T_1 that is not a sibling pair of F, and assume that neither a nor c is a singleton in F. Then $e(T_1, F - \{e_x\}) = e(T_1, F) - 1$ for some $x \in \{a,b,c\}$. In particular, $e(T_1, F - \{e_a, e_b, e_c\}) \leq e(T_1, F) - 1$.*

Proof. It suffices to prove that there exists an edge set E such that $F - E$ yields an MAF of T_1 and F and $E \cap \{e_a, e_b, e_c\} \neq \emptyset$. So let E be chosen such that $F - E$ yields an MAF F' and assume that $E \cap \{e_a, e_b, e_c\} = \emptyset$. We prove that there exists an edge $f \in E$ such that $F - E$ and $F - (E \setminus \{f\} \cup \{e_x\})$ yield the same forest, for some $x \in \{a,b,c\}$. Thus, since $F - E$ yields an MAF, so does $F - E'$, where $E' = E \setminus \{f\} \cup \{e_x\}$, and $E' \cap \{e_a, e_b, e_c\} \neq \emptyset$.

By Lemma 3, we can assume that there exists a leaf $b' \in B$ such that $b' \sim_{F-E} r_{ab}$ because otherwise we can choose an arbitrary leaf $b' \in B$ and replace the first edge in E on the path from r_{ab} to b' with e_b without altering the forest yielded by $F - E$.

Since $e_a \notin E$, we have $a \sim_{F-E} r_{ab}$ and, hence, $a \sim_{F-E} b'$. Since (a,c) is a sibling pair of T_1, $ac|b'$ is a triple of T_1, while $c \notin B$ implies that either $ab'|c$ is a triple of F or $a \nsim_F c$. In either case, the triple $ac|b'$ is incompatible with F. Since $F - E$ yields an agreement forest F' of T_1 and F, Observation 1(i) now implies that $a \nsim_{F-E} c$. Therefore, if $a \sim_F c$, there exists an edge in E that belongs to the path from a to c in F, and we choose f to be such an edge that is closest to c. Since (a,c) is a sibling pair of T_1 and $a \sim_{F-E} b$, $a \nsim_{F-E} c$ implies that c is a singleton in F'. Thus, edges f and $e = e_c$ satisfy the conditions of Lemma 3, and $F - E$ and $F - (E \setminus \{f\} \cup \{e_c\})$ yield the same forest.

If $a \nsim_F c$, then $a \nsim_{F-E} c$ and, as above, c is a singleton in F'. Since c is not a singleton in F, there exists an edge $f \in E$ that belongs to the same connected component of F as c, and $e = e_c$ and f satisfy Lemma 3 if f is chosen so that none of the edges on the path from f to c is in E. Hence, again, $F - E$ and $F - (E \setminus \{f\} \cup \{e_c\})$ yield the same forest. □

Note that Theorem 4 also holds if we replace $e(T_1, \cdot)$ with $\bar{e}(T_1, T_2, \cdot)$. To see this, it suffices to consider a set E in the proof such that $F - E$ yields an MAAF instead of an MAF. The next theorem provides an analogous result for unrooted MAF's.

Theorem 5. *Let T_1 and T_2 be two unrooted X-trees, and let F be a forest of T_2. Let (a,c) be a sibling pair of T_1 that is not a sibling pair of F, and assume that neither a nor c is a singleton in F. Then $e(T_1, F - \{e_x\}) = e(T_1, F) - 1$ for some $x \in \{a,b,c,d\}$.*

Proof. As in the proof of Theorem 4, our goal is to show that there exists a set E such that $F - E$ yields an MAF of T_1 and F and $E \cap \{e_a, e_b, e_c, e_d\} \neq \emptyset$. Again, we show that, if $F - E$ yields an MAF F' of T_1 and F and $E \cap \{e_a, e_b, e_c, e_d\} = \emptyset$, we can find an edge $f \in E$ and an $x \in \{a,b,c,d\}$ such that $F - E$ and $F - (E \setminus \{f\} \cup \{e_x\})$ yield the same forest.

By the same arguments as in the proof of Theorem 4, we can assume that there exist vertices $b' \in B$ and $d' \in D$ such that $b' \sim_{F-E} r_{ab}$ and $d' \sim_{F-E} r_{cd}$ and, hence, $b' \sim_{F-E} a$ and $d' \sim_{F-E} c$. Since (a,c) is a sibling pair of T_1, $ac|b'd'$ is a quartet of T_1, while $c \notin B$ implies that either $ab'|cd$ is a quartet of F or $a \nsim_F c$. In either case, the quartet $ac|b'd'$ is incompatible with F. Since $F - E$ yields an agreement forest of T_1 and F, Observation 1(ii) now implies that $a \nsim_{F-E} c$. Therefore, if $a \sim_F c$, there exists an edge in E that belongs to the path from a to c in F, and we choose f to be such an edge that is closest to c. Since (a,c) is a sibling pair of T_1 and $a \sim_{F-E} b$, $a \nsim_{F-E} c$ implies that c is a singleton in F'. Thus, edges f and $e = e_c$ satisfy the conditions of Lemma 3, and $F - E$ and $F - (E \setminus \{f\} \cup \{e_c\})$ yield the same forest.

If $a \nsim_F c$, then the same argument as in the proof of Theorem 4 shows that $F - E$ and $F - (E \setminus \{f\} \cup \{e_c\})$ yield the same forest, F', for some edge $f \in E$. □

Similar to Theorem 4, Theorem 5 immediately implies that $e(T_1, F - \{e_a, e_b, e_c, e_d\}) \le e(T_1, F) - 1$. However, we can do a little better.

Theorem 6. *Let T_1 and T_2 be two unrooted X-trees, and let F be a forest of T_2. Let (a,c) be a sibling pair of T_1 that is not a sibling pair of F, and assume that neither a nor c is a singleton in F. Then $e(T_1, F - \{e_a, e_b, e_c\}) \le e(T_1, F) - 1$.*

Proof. Let E be an edge set such that $F - E$ yields an MAF F' of T_1 and F. We can again assume that $E \cap \{e_a, e_b, e_c\} = \emptyset$, as otherwise the theorem holds trivially. We show that there exists an edge $f \in E$ such that $F - (E \cup \{e_a, e_b\})$ and $F - (E \setminus \{f\} \cup \{e_a, e_b, e_c\})$ yield the same forest. This forest is an agreement forest of T_1 and F, as it can be obtained by cutting edges e_a and e_b in F'. Hence, $e(T_1, F - \{e_a, e_b, e_c\}) \le |E \setminus \{f\}| = |E| - 1 = e(T_1, F) - 1$. Note that this is not the same as claiming that we can replace an edge $f \in E$ with an edge in $\{e_a, e_b, e_c\}$ without altering the resulting forest. It is crucial that *all three* edges are cut. As in the proof of Theorem 4, we can again assume that there exists a leaf $b' \in B$ such that $b' \sim_{F-E} r_{ab}$ and, hence, $b' \sim_{F-E} a$. For the remainder of the proof, we distinguish three cases.

If $a \sim_{F-E} c$, we observe that (a,c) being a sibling pair in T_1 and $c \notin B$ imply that $ac|b'd'$ is a quartet of T_1 incompatible with F, for all $d' \notin X_B \cup \{a,c\}$. Hence, by Observation 1(ii), $a \sim_{F-E} b'$ and $a \sim_{F-E} c$ imply that $c \nsim_{F-E} d'$, for each such leaf d'. Therefore, $c \nsim_{F-E'} x$, for all $x \in X$, where $E' = E \cup \{e_a, e_c\}$. If we now choose an edge $f \in E$ such that no edge on the path from f to c belongs to E, Lemma 3 implies that $F - (E \cup \{e_a, e_b\})$ and $F - (E \setminus \{f\} \cup \{e_a, e_b, e_c\})$ yield the same forest.

If $a \sim_F c$ but $a \nsim_{F-E} c$, there exists an edge in E that belongs to the path from a to c in F, and we choose f to be such an edge that is closest to c. Since (a,c) is a sibling pair of T_1 and $a \sim_{F-E} b$, $a \nsim_{F-E} c$ implies that c is a singleton in F'. Thus, edges f and $e = e_c$ satisfy the conditions of Lemma 3, and $F - E$ and $F - (E \setminus \{f\} \cup \{e_c\})$ yield the same forest.

If $a \nsim_F c$, then the same argument as in the proof of Theorem 4 shows that $F - E$ and $F - (E \setminus \{f\} \cup \{e_c\})$ yield the same forest, F', for some edge $f \in E$. □

While Theorem 4 suffices as a basis of an algorithm to compute or approximate an MAF of two rooted trees, a little extra work is required to obtain an MAAF. As observed after the proof of Theorem 4, we can use this theorem to make progress towards an MAAF until we obtain an agreement forest of the two trees. If this forest is in fact acyclic, we are done. Otherwise, we need to continue cutting edges to remove all cycles that may exist. The next theorem identifies candidate edges to cut. In this theorem, we consider two trees, A and B, of the agreement forest whose roots, a and b, form a cycle. We call (a,b) a *cycle pair* and use e_a to denote any of the two edges in A incident to a, and e_b to denote any of the two edges in B incident to b.

Theorem 7. *Let T_1 and T_2 be two rooted X-trees, F an agreement forest of T_1 and T_2, and (a,b) a cycle pair of F. Then $\bar{e}(T_1, T_2, F - \{e_x\}) = \bar{e}(T_1, T_2, F) - 1$ for some $x \in \{a,b\}$. In particular, $\bar{e}(T_1, T_2, F - \{e_a, e_b\}) \le \bar{e}(T_1, T_2, F) - 1$.*

Proof. Once again, our goal is to show that there exists a set E of edges of F such that $F - E$ yields an MAAF of T_1 and F and $E \cap \{e_a, e_b\} \ne \emptyset$. So we choose E to be a set such that $F - E$ yields an MAAF F' of T_1 and F, and assume that $E \cap \{e_a, e_b\} = \emptyset$. Let A_1 and A_2 be the two subtrees of A rooted in a's children, and let B_1 and B_2 be the two subtrees of B rooted in b's children.

First observe that there exists an index i such that either $a' \nsim_{F-E} a$ for all $a' \in X_{A_i}$ or $b' \nsim_{F-E} b$ for all $b' \in X_{B_i}$. Indeed, if this was not the case, there would exist leaves $a_1 \in A_1$, $a_2 \in A_2$, $b_1 \in B_1$, and $b_2 \in B_2$ such that $a_1 \sim_{F-E} a_2$ and $b_1 \sim_{F-E} b_2$, which implies that both a and b exist in F', and F' would not be acyclic.

So assume w.l.o.g. that $a' \nsim_{F-E} a$, for all $a' \in A_1$. In this case, Lemma 3 states that, if we choose a leaf $a' \in A_1$ and the edge $f \in E$ closest to a on the path from a to a', then $F - E$ and $F - (E \setminus \{f\} \cup \{e_a\})$ yield the same forest, which is the MAAF F'. Hence, the edge set $E' = E \setminus \{f\} \cup \{e_a\}$ has the property that $F - E'$ yields an MAAF of T_1 and F and $E' \cap \{e_a, e_b\} \neq \emptyset$. □

4 Approximation Algorithms

4.1 3-Approximation for Rooted MAF (rSPR Distance)

The first algorithm we present is a 3-approximation algorithm for rooted MAF, that is, rooted SPR distance. This algorithm is essentially the one discussed in [25], modified to achieve linear time. We include it here to demonstrate that Theorem 4 establishes its correctness, and also as a reference for the other algorithms. The algorithm starts with a pair of trees (T_1, T_2) and modifies both through a series of transformations. T_1 always remains a tree but shrinks over time; T_2, on the other hand, may become a forest. The algorithm maintains a counter, D, of the number of edges in T_2 it has cut so far. We use $T_1^{(i)}$, $T_2^{(i)}$, and $D^{(i)}$ to denote the two forests obtained from T_1 and T_2 after the ith transformation and the value of D after the ith transformation. The algorithm terminates when the label set of $T_1^{(i)}$ and $T_2^{(i)}$ has size at most 2, including the root label ρ, which is never eliminated. The output is the value of D at the time of termination.

Each iteration applies one of the following cases, illustrated in Figure 4.

1. As long as T_2 contains a singleton $c \neq \rho$, the algorithm removes c from both T_1 and T_2 and performs a forced contraction in T_1 to merge the other two edges incident to c's parent in T_1. D remains unchanged.

For the other two rules, the algorithm chooses a fixed sibling pair (a, c) of T_1.

2. If (a, c) is also a sibling pair of T_2, the algorithm contracts the sibling pair in both trees; that is, it removes a and c from both T_1 and T_2 and assigns the label (a, c) to their parent r_{ac}.
3. If (a, c) is not a sibling pair in T_2, then assume w.l.o.g. that a's distance from the root of T_2 is no less than that of c. Node a must have a sibling b in T_2 because T_2 contains no singletons. In this case, the algorithm cuts edges e_a, e_b, and e_c in T_2 and increases D by three.

Theorem 8. *Given two rooted X-trees T_1 and T_2, a 3-approximation of $e(T_1, T_2) = d_{rSPR}(T_1, T_2)$ can be computed in linear time.*

Proof. We use the algorithm above and output the final value of D as the approximation of $e(T_1, T_2)$. We argue below that the algorithm terminates, in linear time. If the algorithm terminates after k iterations, k' of which change D, then its output is $D^{(k)} = 3k'$. We prove that $e(T_1, T_2) \leq 3k' \leq 3e(T_1, T_2)$, thereby proving that the value $D^{(k)}$ output by the algorithm is a 3-approximation of $e(T_1, T_2) = d_{\mathrm{rSPR}}(T_1, T_2)$.

For every iteration that leaves D unchanged, we have $e(T_1^{(i)}, T_2^{(i)}) = e(T_1^{(i-1)}, T_2^{(i-1)})$ because only Cases 1 and 2 leave D unchanged and Lemmas 1 and 2 show that the transformations applied in these cases do not alter $e(T_1, T_2)$. Every iteration that changes D applies Case 3. Since (a, c) is not a sibling pair of T_2 in this case, and neither a nor c is a singleton in T_2, Theorem 4 implies that $e(T_1^{(i)}, T_2^{(i)}) \leq e(T_1^{(i-1)}, T_2^{(i-1)}) - 1$. Hence, we have $e(T_1, T_2) \geq k'$, that is, $D^{(k)} = 3k' \leq 3e(T_1, T_2)$. Conversely, since the $3k'$ edges we cut in T_2 yield an agreement forest of T_1 and T_2, we have $e(T_1, T_2) \leq 3k'$.

To bound the running time of the algorithm, we observe that it terminates after $O(n)$ iterations. Indeed, each iteration removes at least one of the $O(n)$ vertices and edges from T_1 or T_2. Moreover, T_1 contains a sibling pair as long as the label set of T_1 has size greater than 2, which implies that one of the transformation rules is applicable. Thus, it suffices to prove that each iteration can be implemented in constant time, which we do in Appendix B. (The argument is similar to the one presented by Bonet et al. [7].) □

4.2 3-Approximation for Unrooted MAF (TBR Distance)

The 3-approximation algorithm for unrooted MAF and, hence, for TBR distance is the same as for the rooted case, except that edges e_a, e_b, and e_c in Case 3 are used in their unrooted meaning, and the algorithm may end up cutting not e_b but the third edge apart from e_a and e_b incident to r_{ab}:

3. If (a,c) is not a sibling pair in T_2, let b^* be one of the two nodes at distance two from a in T_2. Then the algorithm cuts edges e_a, e_{b^*}, and e_c in T_2 and increases D by three.

The different cases of the algorithm are illustrated in Figure 5 in Appendix A.

Theorem 9. *Given two unrooted X-trees T_1 and T_2, a 3-approximation of $e(T_1,T_2) = d_{TBR}(T_1,T_2)$ can be computed in linear time.*

Proof. The linear running time of the algorithm is established as in the rooted case. We need to prove that, if the algorithm runs for k iterations, the final value of D, $D^{(k)}$, is a 3-approximation of $e(T_1,T_2)$.

Again, let k' be the number of applications of Case 3. Then $D^{(k)} = 3k'$. As in the proof of Theorem 8, each application of Cases 1 and 2 satisfies $e(T_1^{(i)},T_2^{(i)}) = e(T_1^{(i-1)},T_2^{(i-1)})$. Hence, it suffices to prove that $e(T_1^{(i)},T_2^{(i)}) \leq e(T_1^{(i-1)},T_2^{(i-1)}) - 1$ for each application of Case 3. If Case 3 cut edges e_a, e_b, and e_c, this would follow from Theorem 6 similarly to the proof of Theorem 8. As pointed out, however, we may cut the third edge incident to r_{ab} instead of e_b. The reason we do this is that it would be costly to check which of the two edges apart from e_a incident to r_{ab} does not belong to the path from a to c. To establish that $e(T_1^{(i)},T_2^{(i)}) = e(T_1^{(i-1)},T_2^{(i-1)} - \{e_a,e_{b^*},e_c\}) \leq e(T_1^{(i-1)},T_2^{(i-1)}) - 1$, we observe that, if $b^* \neq b$, then edges $e = e_{b^*}$ and $f = e_b$ satisfy Lemma 3 with respect to the edge set $E = \{e_a,e_b,e_c\}$. Hence, $T_2^{(i-1)} - \{e_a,e_b,e_c\}$ and $T_2^{(i-1)} - \{e_a,e_{b^*},e_c\}$ yield the same forest, $T_2^{(i)}$. □

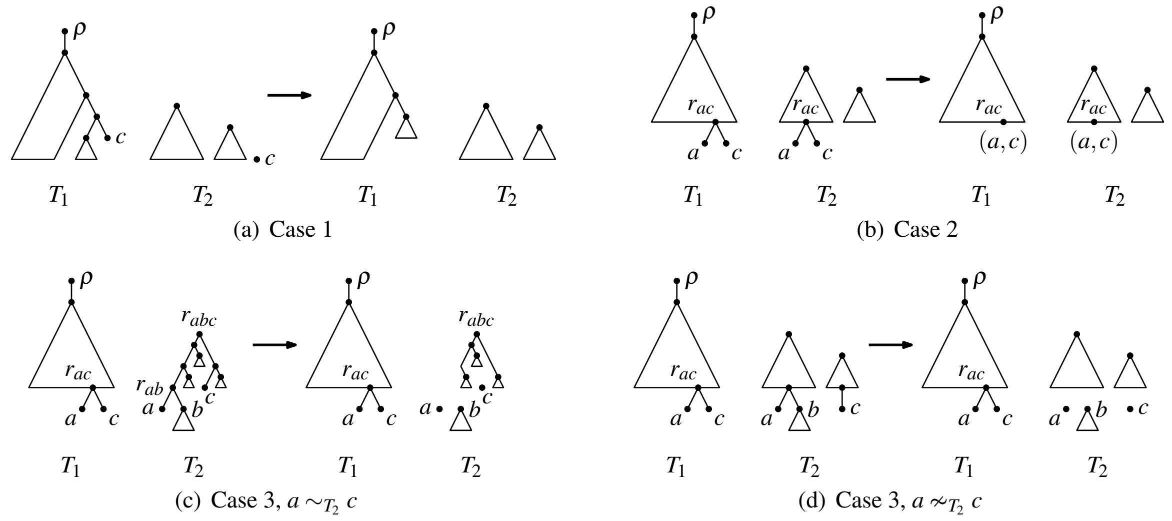


Fig. 4. The three cases of the approximation algorithm for rooted MAF.

4.3 3-Approximation for Rooted MAAF (Hybridization Number)

The algorithm for approximating rooted MAAF consists of two stages. First we run the algorithm for rooted MAF to obtain an approximation F of an MAF of T_1 and T_2. That algorithm, as described in Section 4.1, computes only an approximation of the number of edges that need to be cut to obtain an MAF. However, it is easy to extend this algorithm so that every edge in $T_2^{(i)}$ stores a pointer to one of the edges of T_2 that have been contracted into it. Thus, whenever an application of Case 3 cuts an edge in $T_2^{(i-1)}$, we can also cut the corresponding edge in T_2. The forced contractions necessary to obtain F are then easily performed in linear time once the algorithm terminates.

Given an agreement forest F of T_1 and T_2, it remains to identify and break cycles. Whenever we cut an edge in the process, we increase D by one, starting with the value of D at the end of the MAF algorithm. Our algorithm maintains two sets, R_d and R_t, of roots of trees in F. R_d contains a set of roots that do not form any cycles with each other. R_t contains roots that may be involved in cycles. Initially, $R_d = \emptyset$ and R_t contains all roots of trees in F. Each iteration of the algorithm removes a root a from R_t and tests whether a forms a cycle with a root $b \in R_d$. If not, we add a to R_d and move on to the next root in R_t. If there is a root $b \in R_d$ such that (a,b) is a cycle pair, we remove one of the two edges incident to each of a and b and increase D by two. This breaks the two trees in F with roots a and b into two subtrees each; their roots are the children of a and b in F. We add these children to R_t and then move on to the next iteration. The algorithm terminates when $R_t = \emptyset$, at which point F has been refined to an acyclic agreement forest of T_1 and T_2.

After spending linear time to label every node in T_1 and T_2 with its preorder number and with the interval of preorder numbers of its descendants, checking whether two nodes a and b form a cycle takes constant time, as (a,b) is a cycle pair if and only if a's preorder interval in T_1 contains b's, and b's preorder interval in T_2 contains a's. The total number of nodes added to R_t throughout the algorithm is at most the number of nodes in F, which is $O(n)$. Hence, R_d also never has size greater than $O(n)$, and checking every node in R_t against every node in R_d takes $O(n^2)$ time in total. In Appendix C, we discuss how to reduce the running time to $O(n \log n)$ by taking a geometric view of the problem. Thus, we have the following result.

Theorem 10. *Given two unrooted X-trees T_1 and T_2, a 3-approximation of $\bar{e}(T_1,T_2,T_2) = hyb(T_1,T_2)$ can be computed in $O(n \log n)$ time.*

Proof. We have already discussed the running time of the algorithm. To prove the approximation bound, consider all iterations over the two phases of the algorithm. As in the proof of Theorem 8, an iteration that leaves D unchanged also leaves $\bar{e}(T_1,T_2,F)$ unchanged, by Lemmas 1 and 2. By Theorems 4 and 7, every iteration that changes D decreases $\bar{e}(T_1,T_2,F)$ by at least one, as each such iteration is an invocation of Case 3 of the MAF algorithm or cuts two edges e_a and e_b in the current agreement forest F, where (a,b) is a cycle pair. Hence, the number k' of such iterations is at most $\bar{e}(T_1,T_2,T_2)$. Each such iteration increases D by at most three. Thus, $D \leq 3k' \leq 3\bar{e}(T_1,T_2,T_2)$ at the end of the algorithm. On the other hand, once the algorithm terminates, R_t is empty, and the roots in R_d do not form cycles. The resulting agreement forest is therefore acyclic, and we cut D edges to obtain it. Thus, $\bar{e}(T_1,T_2,T_2) \leq D$. Together with the upper bound, this shows that the final value of D is a 3-approximation of $\bar{e}(T_1,T_2,T_2)$. □

5 Fixed-Parameter Algorithms

The approximation algorithms discussed in the previous section are easily modified to obtain fixed-parameter algorithms for the respective problems. As is customary when discussing such algorithms, we focus on the decision version: "Given two X-trees T_1 and T_2, a distance measure $d(\cdot,\cdot)$, and a parameter k, is $d(T_1,T_2) \leq k$?" If this decision version can be solved in $O(c^k \text{poly}(n))$ time, then the exact distance $d = d(T_1,T_2)$ can be found in $O(c^d \text{poly}(n))$ time by iteratively trying larger guesses of k until we obtain the first positive answer.

To obtain such a decision algorithm for (rooted or unrooted) MAF, we modify the approximation algorithm from Section 4.1. We denote an invocation of the algorithm on tree T and forest F and with distance bound k by $\mathscr{A}(T,F,k)$. If $k \geq 0$ and T and F have at most two nodes each, the algorithm returns "yes". Otherwise, whenever the approximation algorithm applies Case 1 or 2, so does $\mathscr{A}(T,F,k)$. When the approximation algorithm would apply Case 3, $\mathscr{A}(T,F,k)$ recurses. In the rooted case, the algorithm makes three recursive calls: $\mathscr{A}(T,F-\{e_a\},k-1)$, $\mathscr{A}(T,F-\{e_b\},k-1)$, and $\mathscr{A}(T,F-\{e_c\},k-1)$. $\mathscr{A}(T,F,k)$ returns "yes" if and only if one of these recursive calls does. In the rooted case, the algorithm makes four recursive calls: $\mathscr{A}(T,F-\{e_a\},k-1)$, $\mathscr{A}(T,F-\{e_b\},k-1)$, $\mathscr{A}(T,F-\{e_c\},k-1)$, and $\mathscr{A}(T,F-\{e_d\},k-1)$ and again returns "yes" if and only if one of these recursive calls does.

Theorem 11. *For two rooted X-trees T_1 and T_2 and a parameter k, it takes $\mathrm{O}(3^k n)$ time to decide whether $e(T_1,T_2) \leq k$. In the unrooted case, it takes $\mathrm{O}(4^k n)$ time.*

Proof. First we prove the algorithm's correctness. By Lemmas 1 and 2, whenever we remove a singleton or contract a sibling pair without recursing, $e(T_1,T_2)$ does not change. For two rooted X-trees T_1 and T_2 and a sibling pair (a,c) as in Case 3, Theorem 4 states that $e(T,F-\{e_x\}) = e(T,F)-1$, for at least one $x \in \{a,b,c\}$, while $e(T,F-\{e_x\}) \geq e(T,F)-1$, for all $x \in \{a,b,c\}$. Hence, $e(T,F) \leq k$ if and only if $e(T,F-\{e_x\}) \leq k-1$ for at least one $x \in \{a,b,c\}$. Thus, the algorithm gives the correct answer. In the unrooted case, the correctness of the algorithm follows from a similar argument using Theorem 5.

As for the running time of the algorithm, we can view each recursive call as a truncated invocation of the approximation algorithm from Section 4.1 that recurses as soon as it would invoke Case 3. Hence, each recursive call takes $\mathrm{O}(n)$ time. Since each recursive call decreases k by one, and the recursion stops no later than when $k=0$, the recursion tree has height at most k. In the rooted case, each non-leaf node has three children; in the unrooted case, four. Hence, the number of recursive calls is $\mathrm{O}(3^k)$ in the rooted case and $\mathrm{O}(4^k)$ in the unrooted case. This gives the claimed running times of $\mathrm{O}(3^k n)$ and $\mathrm{O}(4^k n)$, respectively. □

In order to obtain an FPT algorithm for MAAF, we augment the above algorithm for rooted MAF as follows. For every recursive call $\mathscr{A}(T,F,k)$ that would output "yes", we compute the corresponding agreement forest F' of the two original trees T_1 and T_2. Note that F' is not necessarily an MAF, as k may be greater than $e(T_1,T_2)$. If F' is acyclic, the algorithm answers "yes". Otherwise, it invokes a second recursive algorithm $\mathscr{B}(F',k)$. This invocation returns "yes" if $k \geq 0$ and F' contains no cycle, and "no" if $k < 0$. If $k \geq 0$ and F' contains a cycle pair (a,b), $\mathscr{B}(F',k)$ makes two recursive calls $\mathscr{B}(F'-\{e_a\},k-1)$ and $\mathscr{B}(F'-\{e_b\},k-1)$ and returns "yes" if and only if one of the two calls does.

Theorem 12. *For two rooted X-trees T_1 and T_2 and a parameter k, it takes $\mathrm{O}(3^k n \log n)$ time to decide whether $\bar{e}(T_1,T_2,T_2) \leq k$.*

Proof. We show the correctness for each of the two types of recursive calls. As for rooted MAF, whenever we eliminate a singleton or contract a sibling pair in an invocation $\mathscr{A}(T,F,k)$ without recursing, Lemmas 1 and 2 show that this does not alter $\bar{e}(T_1,T_2,F)$. When an invocation $\mathscr{A}(T,F,k)$ makes recursive calls $\mathscr{A}(T,F-\{e_a\},k-1)$, $\mathscr{A}(T,F-\{e_b\},k-1)$, and $\mathscr{A}(T,F-\{e_c\},k-1)$, Theorem 4 implies that $\bar{e}(T_1,T_2,F) \leq k$ if and only if $\bar{e}(T_1,T_2,F-\{e_x\}) \leq k-1$, for some $x \in \{a,b,c\}$. Hence, we output the right answer in this case. If $\mathscr{A}(T,F,k)$ makes a recursive call $\mathscr{B}(F',k)$, F already yields an agreement forest F' of T and F. Hence, $\bar{e}(T_1,T_2,F) \leq k$ if and only if we can find k edges in F' such that cutting them yields an acyclic forest. As we show next, $\mathscr{B}(F',k)$ returns "yes" if and only if this is the case.

For an invocation $\mathscr{B}(F',k)$ such that F' contains a cycle pair (a,b), Theorem 7 states that F' contains a set E of k edges such that $F'-E$ yields an acyclic forest if and only if $F'-\{e_a\}$ or $F'-\{e_b\}$ contains such a set of of size $k-1$. Since the latter is equivalent to $\mathscr{B}(F'-\{e_a\},k-1)$ or $\mathscr{B}(F'-\{e_b\},k-1)$ returning "yes", $\mathscr{B}(F',k)$ gives the right answer.

To bound the running time, we again observe that every recursive call of the algorithm can be seen as a truncated version of the approximation algorithm for MAF or MAAF and, hence, takes $\mathrm{O}(n \log n)$ time. The depth of the recursion is at most k again, and every node has at most three children. Hence, there are $\mathrm{O}(3^k)$ recursive calls overall, and the running time of the algorithm is $\mathrm{O}(3^k n \log n)$. $\square$

Using known kernelizations [2,9,10], we can reduce the two input trees T_1 and T_2 to trees T_1' and T_2' such that $e(T_1,T_2) \leq k$ or $\bar{e}(T_1,T_2,T_2) \leq k$ if and only if $e(T_1',T_2') \leq k$ or $\bar{e}(T_1',T_2',T_2') \leq k$, respectively, and trees T_1' and T_2' have size $\mathrm{O}(k)$. These kernelizations take $\mathrm{O}(n^3)$ time. Hence, we obtain the following corollary.

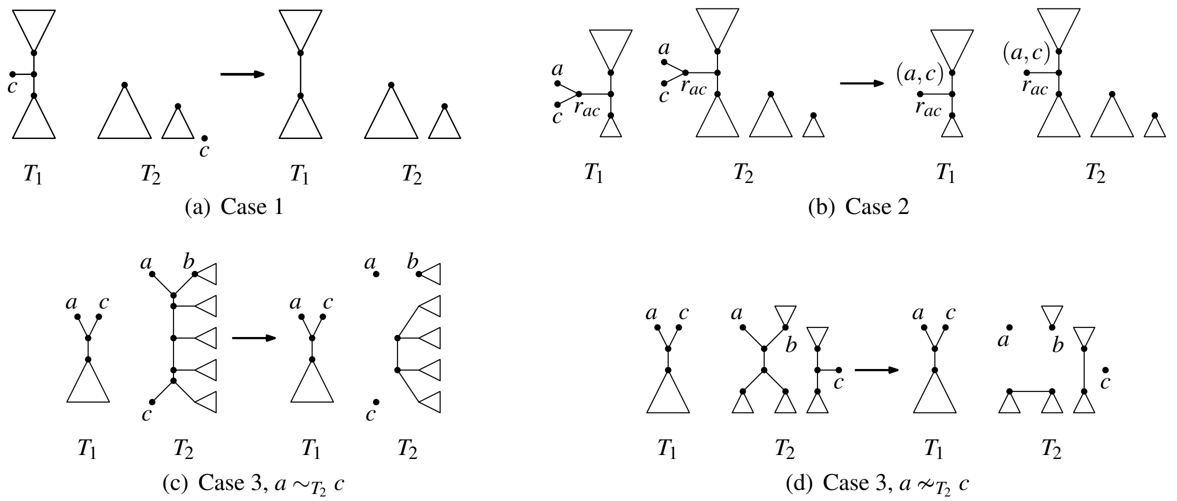
Corollary 1. *For two rooted X-trees T_1 and T_2 and a parameter k, it takes $\mathrm{O}(3^k k + n^3)$ time to decide whether $e(T_1,T_2) \leq k$ and $\mathrm{O}(3^k k \log k + n^3)$ time to decide whether $\bar{e}(T_1,T_2,T_2) \leq k$. In the unrooted case, it takes $\mathrm{O}(4^k k + n^3)$ time to decide whether $e(T_1,T_2) \leq k$.*

References

1. L. Addario-Berry, B. Chor, M. T. Hallett, J. Lagergren, A. Panconesi, and T. Wareham. Ancestral maximum likelihood of evolutionary trees is hard. *Journal of Bioninformatics and Computational Biology*, 2(2):257–272, 2004.
2. B. L. Allen and M. Steel. Subtree transfer operations and their induced metrics on evolutionary trees. *Annals of Combinatorics*, 5(1):1–15, 2001.
3. N. Amenta and J. Klingner. Case study: Visualizing sets of evolutionary trees. In *Proceedings of the IEEE Symposium on Information Visualization*, pages 71–74, 2002.
4. 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.
5. R. G. Beiko and N. Hamilton. Phylogenetic identification of lateral genetic transfer events. *BMC Evolutionary Biology*, 6(1):15, 2006.
6. M. A. Bender and M. Farach-Colton. The LCA problem revisited. In *Proceedings of the 4th Latin American Symposium on Theoretical Informatics*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer-Verlag, 2000.
7. 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.
8. 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.
9. 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.
10. 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.
11. 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.
12. F. Chataigner. Approximating the maximum agreement forest on k trees. *Information Processing Letters*, 93:239–244, 2005.
13. W. H. E. Day. Optimal algorithms for comparing trees with labeled leaves. *Journal of Classification*, 2(1):7–28, 1985.
14. L. R. Foulds and R. L. Graham. The Steiner problem in phylogeny is NP-complete. *Advances in Applied Mathematics*, 3:43–49, 1982.
15. M. Hallett and C. McCartin. A faster FPT algorithm for the maximum agreement forest problem. *Theory of Computing Systems*, 41:539–550, 2007.
16. 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.
17. G. Hickey, F. Dehne, A. Rau-Chaplin, and C. Blouin. The computational complexity of the unrooted subtree prune and regraft distance. Technical Report CS-2006-06, Faculty of Computer Science, Dalhousie University, 2006.
18. G. Hickey, F. Dehne, A. Rau-Chaplin, and C. Blouin. SPR distance computation for unrooted trees. *Evolutionary Bioinformatics*, 4:17–27, 2008.
19. D. M. Hillis, T. A. Heath, and K. St. John. Analysis and visualization of tree space. *Systematic Biology*, 54(3):471–482, 2005.
20. D. M. Hillis, C. Moritz, and B. K. Mable, editors. *Molecular Systematics*. Sinauer Associates, 1996.
21. W. P. Maddison. Gene trees in species trees. *Systematic Biology*, 46(3):523–536, 1997.
22. 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.
23. R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.
24. D. F. Robinson and L. R. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53(1-2):131–147, 1981.

25. 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.
26. K. St. John. Comparing phylogenetic trees. In *EMBO Workshop on Current Challenges and Problems in Phylogenetics*, Isaac Newton Institute, Cambridge, UK, 2007.
27. V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.

A Supplementary Figures



(a) Case 1

(b) Case 2

(c) Case 3, $a \sim_{T_2} c$

(d) Case 3, $a \nsim_{T_2} c$

Fig. 5. The three cases of the approximation algorithm for unrooted MAF.

B Linear-Time Implementation of Rooted MAF Approximation

In this appendix, we discuss how to represent the two trees in the rooted MAF approximation algorithm from Section 4.1 to ensure that each of the $O(n)$ iterations of the algorithm takes constant time, resulting in a total running time of $O(n)$ for the algorithm.

We represent each tree as a collection of nodes, each of which points to its parent, left child, and right child. In addition, every labelled node stores a pointer to its counterpart in the other tree. For T_1, we maintain a list of sibling pairs, and every labelled node of T_1 stores a pointer to the pair it belongs to, if any. For T_2, we maintain a list of singletons.

While the singleton list is non-empty, we remove its next element c. Given the above pointers, it takes constant time to identify node c in T_1 and its parent, to remove c from T_1 and splice out its parent. Thus, we can check in constant time whether Case 1 applies and, if so, apply it. If c belonged to a sibling pair in T_1, we can eliminate it from the list of sibling pairs. Moreover, c's sibling a in T_1 may now be the sibling of another leaf c' in T_1, and we may need to add the pair (a,c') to the list of sibling pairs. This is the case if a's new sibling c' has no children, which is also easily checked in constant time. This concludes the discussion of Case 1.

To implement Cases 2 and 3, we retrieve the next sibling pair (a,c) of T_1. The pair (a,c) is a sibling pair of T_2 if and only if a and c have the same parent in T_2. This allows us to distinguish between Cases 2 and 3 in constant time.

In Case 2, we remove a and c from both trees and set up pointers between the two parents in T_1 and T_2 (which become labelled as a result of the contraction). If r_{ac} has no parent in T_2, it is now a singleton and is added to the singleton list. Additionally, if r_{ac} has a leaf a' for a sibling in T_1, then (a',r_{ac}) is a sibling pair in T_1, which must be added to the list of sibling pairs.

In Case 3, we identify the sibling b of a in T_2 by following two pointers. Then edges e_a, e_b, and e_c can be removed in constant time. We need to add a and c to the list of singletons, and also node b if it is a leaf. Finally, we need to remove r_{ab} from T_2 and splice out its parent, if any. The parent r_c of node c also needs to be removed or spliced out, depending on whether it has a parent. Moreover, if r_c is removed and the sibling d of c is a leaf, then d is now a singleton and needs to be added to the singleton list. Again, all these are local pointer manipulations that can be carried out in constant time.

The last remaining issue is that, in Case 3, we are required to choose a and c so that the distance from a to the root r of its component of $T_2^{(i)}$ is no less than that from c to r when $a \sim_{T_2^{(i)}} c$. Observe that this condition is used only to ensure that $c \notin B$. The latter, however, is true also if a has the greater depth than c in the initial input tree $T_2^{(0)}$. Hence, after using a standard linear-time depth-first traversal of T_2 to label all its nodes with their depths, we can choose a to be the node with the greater depth label in constant time in each application of Case 3.

C An $O(n \log n)$ Time Implementation of the MAAF Approximation

This appendix provides the details of how to implement the MAAF approximation algorithm described in Section 4.3 in $O(n \log n)$ time. Both the $O(n^2)$-time and the $O(n \log n)$-time implementations require a method to test quickly whether a pair of roots, (a,b), is a cycle pair of the current forest. We discuss this in Section C.1. In Section C.2, we discuss how to avoid testing all pairs of roots by reducing the elimination of cycle pairs to a geometric problem very similar to orthogonal line segment intersection. Throughout this section, we consider an agreement forest F of two rooted X-trees T_1 and T_2.

C.1 Testing Cycles in Constant Time

In order to decide whether a pair of roots (a,b) in F is a cycle pair, we need to be able to identify the nodes corresponding to a and b in T_1 and T_2 and be able to test for any two nodes in T_1 or T_2 whether one is an ancestor of the other.

Testing for ancestry. Given a tree T, we can test whether a node $x \in T$ is an ancestor of a node $y \in T$ by computing a preorder numbering of T and labelling every node of T with the interval of preorder numbers of its descendants. We denote the interval associated with node x by $I(x)$. Such an assignment of intervals can be computed in $O(n)$ time using a DFS-traversal of T. Given two nodes $x, y \in T$, x is an ancestor of y if and only if $I(y) \subset I(x)$, which can be tested in constant time.

Mapping nodes from F to T_1 and T_2. We discuss how to identify the nodes in T_1 corresponding to the nodes in F. The same procedure can be used to identify the corresponding nodes in T_2.

First we use the linear-time algorithm of Bender and Farach-Colton [6] to preprocess T_1 for lowest-common ancestor (LCA) queries, where the lowest common ancestor of two nodes x and y is the node farthest away from the root that is an ancestor of both x and y. The structure constructed by the algorithm of [6] supports such queries in constant time.

For a node $x \in F$, let X_x be the set of labels of all descendant leaves of x in F. Then the node in T_1 corresponding to x is the lowest node that is an ancestor of all leaves in X_x. Assuming the children, l and r, of x have already been mapped to nodes in T_1, it is easy to see that the node corresponding to x is nothing else than the LCA of the nodes corresponding to l and r. Hence, we can compute all nodes in T_1 corresponding to nodes in F by traversing the trees of F in postorder, that is, bottom-up; for every visited node x, we can find the node in T_1 corresponding to x in constant time by answering an LCA query for the two nodes corresponding to its children in F.

Preprocessing T_1 for LCA queries takes linear time. There are $O(n)$ nodes in F, and each gives rise to a constant-time LCA query. Hence, the nodes in T_1 corresponding to the nodes in F can be found in linear time.

Putting it together. To be able to test for cycle pairs in F, we compute intervals $I(x)$ for all nodes $x \in T_1$ and $x \in T_2$, and we use the method just described to find the nodes in T_1 and T_2 corresponding to each node in F. This preprocessing takes linear time.

Given a pair of nodes (a,b) in F, we can now find their corresponding nodes in T_1 and T_2 in constant time, including their preorder intervals. It now takes constant time to decide whether $I_{T_1}(a) \subset I_{T_1}(b)$ and $I_{T_2}(b) \subset I_{T_2}(a)$. If so, (a,b) is a cycle pair. Otherwise, it is not.

C.2 A Geometric View of Cycle Elimination

The constant-time cycle testing method from the previous section suffices to obtain an $O(n^2)$-time implementation of the MAAF approximation algorithm, as there are only $O(n^2)$ node pairs to test. By taking a geometric view, however, we can reduce the running time to $O(n \log n)$ using the plane sweep paradigm.

Recall the definition of sets R_d and R_t in Section 4.3. We represent each node $a \in R_d$ by a horizontal line segment $h_a := (x_1,y)$–(x_2,y), where y is the preorder number of a in T_1 and $I_{T_2}(a) = [x_1,x_2]$. Each node $b \in R_t$ is represented by a vertical line segment $v_b := (x,y_1)$–(x,y_2), where x is b's preorder number in T_2 and $I_{T_1}(b) = [y_1,y_2]$. Then (a,b) is a cycle pair if and only if h_a and v_b intersect.

We start by generating the vertical line segments for the nodes in R_t and insert them into a priority queue Q, in order to be able to process them by increasing x-coordinates to simulate a left-to-right sweep. The line segments corresponding to the nodes in R_d are represented by a balanced binary search tree T storing the segments that intersect the sweep line, sorted by their y-coordinates. The left and right endpoints of the segments in R_d are also added to Q, in order to process them as event points during the sweep. Initially, T is empty and Q only stores the segments corresponding to nodes in R_t.

To perform the sweep, we now process the elements in Q by increasing x-coordinates. If the next element is the left endpoint of a horizontal segment h_a, we insert h_a into T. If it is h_a's right endpoint, we remove h_a from T. This ensures that at all times T stores the horizontal segments intersecting the sweep line. Now, if the next element to be processed is a vertical segment $v_b = (x,y_1)$–(x,y_2), v_b intersects a horizontal segment corresponding to a node in R_d if and only if the interval $[y_1,y_2]$ contains the y-coordinate of a segment h_a currently in T. Deciding this and finding a segment h_a that intersects v_b requires a search in T to find the largest y-coordinate no greater than y_2. If v_b does not intersect a segment h_a in R_d, we add b to R_d by adding the endpoints of h_b to Q. If v_b intersects a segment h_a, (a,b) is a cycle pair and we delete edges e_a and e_b from F, thereby removing nodes a and b from F. To reflect this change, we perform the following updates on Q and T.

1. First we insert the vertical segments v_{b_1} and v_{b_2} corresponding to the children of b into Q. Note that they are to the right of the sweep line, as the segment coordinates correspond to preorder numbers and the sweep line is currently at the x-coordinate corresponding to b's preorder number.
2. Second, we inspect the horizontal segments h_{a_1} and h_{a_2} corresponding to the children of a. If h_{a_i} is to the left of the sweep line, it is discarded. If it is to the right of the sweep line, its endpoints are added to Q. If it intersects the sweep line, we insert it into T and add its right endpoint to Q. This corresponds to inserting a_1 and a_2 into R_d.

The algorithm terminates when Q is empty.

Lemma 4. *The approximation algorithm for MAAF from Section 4.3 can be implemented in* $O(n \log n)$ *time.*

Proof. The running time is pretty obvious. The total number of segments added to R_d and R_t over the course of the algorithm is at most equal to the number of nodes in F and is, hence, $O(n)$. Therefore, we process at most $O(n)$ event points in Q. Processing each event point entails a constant number of updates and queries on Q and T, each of which takes $O(\log n)$ time if T is balanced and Q is implemented using any standard efficient priority queue implementation. (A binary heap is good enough.) Therefore, the running time of the algorithm is $\mathrm{O}(n \log n)$.

Ignoring the insertions of a_1 and a_2 into R_d without testing for intersections, the correctness of the algorithm follows from the fact that we transform a vertical segment into a horizontal one—and, hence, insert its corresponding node into R_d—only if it intersects no horizontal segments corresponding to nodes currently in R_d. Hence, the correctness of the algorithm follows if we can argue that neither a_1 nor a_2 forms a cycle pair with a node $b \in R_d$ when inserting them into R_d as a result of removing their parent a from R_d. Note, however, that the nodes in R_d are roots of trees in the current forest F. Any node $b \in R_d$ that forms a cycle pair with a_i is w.l.o.g. an ancestor of a_i in T_1 and a descendant of a_i in T_2. Since b cannot belong to the tree with root a before the removal of a, this implies that b is also an ancestor of a in T_1 and a descendant of a in T_2, making (a, b) a cycle pair. Since $a, b \in R_d$ before replacing a with its children, this is a contradiction. $\square$