A Duality Based 2-Approximation Algorithm for Maximum Agreement Forest*†

Frans Schalekamp‡1, Anke van Zuylen§2, and Suzanne van der Ster3

1 **School of Operations Research & Information Engineering, Cornell University, Ithaca, NY, USA**
fms9@cornell.edu

2 **Department of Mathematics, College of William & Mary, Williamsburg, VA,USA**
anke@wm.edu

3 **Institut für Informatik, Technische Universität München, München, Germany**
ster@in.tum.de

Abstract

We give a 2-approximation algorithm for the Maximum Agreement Forest problem on two rooted binary trees. This NP-hard problem has been studied extensively in the past two decades, since it can be used to compute the Subtree Prune-and-Regraft (SPR) distance between two phylogenetic trees. Our result improves on the very recent 2.5-approximation algorithm due to Shi, Feng, You and Wang (2015). Our algorithm is the first approximation algorithm for this problem that uses LP duality in its analysis.

1998 ACM Subject Classification F.2.2 Analysis of algorithms and problem complexity

Keywords and phrases Maximum agreement forest, phylogenetic tree, SPR distance, subtree prune-and-regraft distance, computational biology

Digital Object Identifier 10.4230/LIPIcs.ICALP.2016.70

1 Introduction

Evolutionary relationships are often modeled by a rooted tree, where the leaves are a set of species, and internal nodes are (putative) common ancestors of the leaves below the internal node. Such phylogenetic trees date back to Darwin [5], who used them in his notebook to elucidate his thoughts on evolution.

The topology of phylogenetic trees can be based on different sources of data, e.g., morphological data, behavioral data, genetic data, etc., which can lead to different phylogenetic trees on the same set of species. Different measures have been proposed to measure the similarity of (or distance between) different phylogenetic trees on the same set of species (or individuals). Using the size of a maximum subtree common to both input trees as a similarity measure was proposed by Gordon [8]. The problem of finding such a subtree is now known as the Maximum Agreement Subtree Problem, and has been studied extensively.

* Full version is available at http://arxiv.org/abs/1511.06000.

† This work was initiated when the authors were visitors of Leen Stougie at the Tinbergen Institute.

‡ FS was supported in part by the Simon Prize for Excellence in the Teaching of Mathematics at William & Mary.

§ AvZ was supported in part by a William & Mary Summer Research Award, NSF Prime Award: HRD-1107147, Women in Scientific Education (WISE) and by a grant from the Simons Foundation (#359525, Anke Van Zuylen).

Steel and Warnow [14] are the first to give a polynomial-time algorithm for this problem. Their approach is refined to an $O(n^{1.5} \log n)$-time algorithm by Farach and Thorup [6], who subsequently show their algorithm is optimal, unless unweighted bipartite matching can be solved in near linear time [7].

There exist non-tree-like evolutionary processes that preclude the existence of a phylogenetic tree, so-called reticulation events (such as hybridization, recombination and horizontal gene transfer). In this context, a particularly meaningful measure of comparing phylogenetic trees is the SPR-distance measure (where SPR is short for Subtree Prune-and-Regraft): this measure provides a lower bound on a certain type of these non-tree evolutionary events. The problem of finding the exact value of this measure for a set of species motivated the formulation of the Maximum Agreement Forest Problem (MAF) by Hein, Jian, Wang and Zhang [9]. Since the introduction by Hein et al., MAF has been extensively studied, including several variants, such as the problem where the input consists of more than two trees, whether the input trees are rooted or unrooted, binary or non-binary. In our paper, we concentrate on MAF on two rooted binary trees.

For ease of defining the solutions to MAF on two rooted binary trees, we think of the input trees as being directed, where all edges are directed away from the root. Given two rooted binary trees on the same leaf set $\mathcal{L}$, the MAF problem asks to find a minimum set of edges to delete from the two trees, so that the directed trees in the resulting two forests can be paired up into pairs of "isomorphic" trees. Two trees are isomorphic if they contain the same nodes from $\mathcal{L}$ and recursively removing nodes of out-degree zero that are not in $\mathcal{L}$ and contracting two arcs incident to a node of in-degree and out-degree one, results in the same binary tree. An alternative (but equivalent) definition will be given in Section 2.

The problem of finding a MAF on two rooted binary trees has been extensively studied, although unfortunately some of the published results later turned out to have subtle errors. First of all, Allen and Steel [1] point out that the claim by Hein et al. that solving MAF on two rooted directed trees computes the SPR-distance between the trees is incorrect. Bordewich and Semple [4] show how to redefine MAF for rooted directed trees so that it is indeed the case that the optimal objective value of MAF is equal to the SPR-distance. In the paper in which they introduce MAF, Hein et al. [9] proved NP-hardness and they give an approximation algorithm for the problem, the approximation guarantee of which turned out to be slightly higher than what was claimed. Bordewich and Semple [4] show that, for their corrected definition of MAF, NP-hardness continues to hold. Other approximation algorithms followed [11, 2, 3]. The current best approximation ratio for MAF is 2.5, due to Shi, Feng, You and Wang [13], and Rodrigues [10] has shown that MAF is MAXSNP-hard. In addition, there is a body of work on other approaches, such as Fixed-Parameter Tractable (FPT) algorithms (e.g., [16, 15]) and Integer Programming [17, 18].

In this paper we give an improved approximation algorithm for MAF with an analysis based on linear programming duality. Our 2-approximation algorithm differs from previous works in two aspects. First of all, in terms of bounding the optimal value, we construct a feasible dual solution, rather than arguing more locally about the objective of the optimal solution. Secondly, our algorithm itself also takes a more global approach, whereas the algorithms in previous works mainly consider local substructures of at most four leaves. In particular, we identify a minimal subtree[1] of one of the two input trees for which the leaf set is incompatible, i.e., when deleting all other leaves from both trees, the remaining two trees

[1] By subtree of a rooted tree, we mean a tree containing the leaves that are descendents of some particular node in the rooted tree (including this node itself), and all edges between them.

are not isomorphic (such a minimal subtree can be found efficiently). We then use "local" operations which repeatedly look at two "sibling" leaves in the minimal incompatible subtree, and perform similar operations as those suggested by previous authors.

Preliminary tests were conducted using a proof-of-concept implementation of our algorithm in Java. Our preliminary results indicate that our algorithm finds a dual solution that in 44% of the 1000 runs is equal to the optimal dual solution, and in 37% of the runs is 1 less than the optimal solution. The observed average approximation ratio is about 1.92; following our algorithm with a simple greedy search algorithm decreases this to less than 1.28.

The remainder of our paper is organized as follows. In Section 2, we formally define the problem. In Section 3, we give a 3-approximation algorithm for MAF that introduces the dual linear program that is used throughout the remainder of the paper and gives a flavor of the arguments used to prove the approximation ratio of two. In Section 4, we give an overview of the 2-approximation algorithm and the key ideas in its analysis. In Section 5 we give more details on the randomly generated instances that we used to obtain our preliminary experimental results, and we conclude in Section 6 with some directions for future research.

The full version of this paper [12] contains further details on the algorithm and a complete analysis that shows that its approximation ratio is 2.

2 Preliminaries

The input to the Maximum Agreement Forest problem (MAF) consists of two rooted binary trees T_1 and T_2, where the leaves in each tree are labeled with the same label set $\mathcal{L}$. Each leaf has exactly one label, and each label in $\mathcal{L}$ is assigned to exactly one leaf in T_1, and one leaf in T_2. For ease of exposition, we sometimes think of the edges in the trees as being directed, so that there is a directed path from the root to each of the leaves.

We call the non-leaf nodes the internal nodes of the trees, and we let V denote the set of all nodes (internal nodes and leaves) in $T_1 \cup T_2$. Given a tree containing u and v, we let $\text{lca}(u, v)$ denote the lowest (closest to the leaves, furthest from the root) common ancestor of u and v. We let $\text{lca}_1(u, v)$ and $\text{lca}_2(u, v)$ denote $\text{lca}(u, v)$ in tree T_1, respectively, T_2. We extend this notation to $\text{lca}(U)$ which will denote the lowest common ancestor of a set of leaves U. For three leaves u, v, w and a rooted tree T, we use the notation $uv|w$ in T to denote that $\text{lca}(u, v)$ is a descendent of $\text{lca}(\{u, v, w\})$. A triplet $\{u, v, w\}$ of labeled leaves is *consistent* if $uv|w$ in $T_1 \Leftrightarrow uv|w$ in T_2. The triplet is called *inconsistent* otherwise. We call a set of leaves $L \subseteq \mathcal{L}$ a *compatible set*, if it does not contain an inconsistent triplet.

For a compatible set $L \subseteq \mathcal{L}$, define $V[L] := \{v \in V$: there exists a pair of leaves u, u' in L so that v is on the path between u and u' in T_1 or $T_2\}$. Then, a partitioning $L_1, L_2, \ldots, L_p$ of $\mathcal{L}$ corresponds to a feasible solution to MAF with objective value $p - 1$, if the sets $L_1, L_2, \ldots, L_p$ are compatible, and the sets $V[L_j]$ for $j = 1, \ldots, p$ are node disjoint. Using this definition, we can write the following Integer Linear Program[2] for MAF: Let $\mathcal{C}$ be the collection of all compatible sets of leaves, and introduce a binary variable x_L for every compatible set $L \in \mathcal{C}$, where the variable takes value 1 if the optimal solution to MAF has a tree with leaf set L. The constraints ensure that each leaf $v \in \mathcal{L}$ is in some tree in the optimal forest, and each internal node $v \in V \setminus \mathcal{L}$ is in at most one tree in the optimal forest. The objective encodes the fact that we need to delete $\sum_{L \in \mathcal{C}} x_L - 1$ edges from each of T_1

[2] This ILP was obtained in discussions with Neil Olver and Leen Stougie.

and T_2 to obtain forests with $\sum_{L\in\mathcal{C}} x_L$ trees.

$$\begin{array}{lll} \text{minimize} & \sum_{L\in\mathcal{C}} x_L - 1, & \\ \text{s.t.} & \sum_{L:v\in L} x_L = 1 & \forall v \in \mathcal{L}, \\ & \sum_{L:v\in V[L]} x_L \leq 1 & \forall v \in V \setminus \mathcal{L}, \\ & x_L \in \{0,1\} & \forall L \in \mathcal{C}. \end{array}$$

Remark

The definition of MAF we use is *not* the definition that is now standard in the literature, but any (approximation) algorithm for our version can be used to get the same (approximation) result for the standard formulation: The standard formulation was introduced by Bordewich and Semple [4] in order to ensure that the objective value of MAF is equal to the rooted SPR distance. They note that for this to hold, we need the additional requirement that the two forests must also agree on the tree containing the original root; in other words, the original roots of T_1 and T_2 should be contained in a tree with the same (compatible) subset of leaves. An easy reduction shows that we can solve this problem using our definition of MAF: given two rooted binary trees for which we want to compute the SPR distance, we can simply add one new label ρ, and for each of the two input trees, we add a new root which has an edge to the original root and an edge to a new node with label ρ.[3] A solution to "our" MAF problem on this modified input defines a solution to Bordewich and Semple's problem on the original input with the same objective value and vice versa.

3 Duality Based 3-Approximation Algorithm

3.1 Algorithm

The algorithm we describe in this section is a variant of the algorithm of Rodrigues et al. [11] (see also Whidden and Zeh [16]). The algorithm maintains two forests, T_1' and T_2' on the same leaf set $\mathcal{L}'$, and iteratively deletes edges from these forests. At the start, T_1' is set equal to T_1, T_2' to T_2 and $\mathcal{L}'$ to $\mathcal{L}$. The leaves in $\mathcal{L}'$ are called the *active* leaves. The algorithm will ensure that the leaves that are not active, will have been resolved in one of the two following ways: (1) they are part of a tree that contains only inactive leaves in both T_1' and T_2'; these two trees then have the same leaf set, which is compatible, and they will be part of the final solution; or (2) an inactive leaf is *merged* with another leaf which is active, and in the final solution this inactive leaf will be in the same tree as the leaf it was merged with.

A tree is called active if it contains a leaf in $\mathcal{L}'$, and the tree is called inactive otherwise. An invariant of the algorithm is that there is a single active tree in T_1'.

We define the parent of a set of active leaves W in a tree of a forest, denoted by $p(W)$, as the lowest node in the tree that is a common ancestor of W and at least one other active node. (That is, $p(W)$ is undefined if there are no other active leaves in the tree that contains the leaves in W.) Note that the parent of a node is defined with respect to the current state of the algorithm, and not with respect to the initial input tree. If $W = \{u\}$ is a singleton, we will also use the notation $p(u) = p(\{u\})$. For a given tree or forest T_i', for $i \in \{1,2\}$, we use the notation $p_i(W)$ to denote $p(W)$ in T_i'.

[3] This is essentially the formulation that is common in the literature, except that in order to ensure that *only leaves have labels*, we give the label ρ to a new leaf that is an immediate descendent of the new root in both trees, rather than to the new root itself.

The operation in the algorithm that deletes edges from forest T_i' is *cut off a subset of leaves* W in T_i'. The edge that is deleted by this operation is the edge directly below $p_i(W)$ towards W (provided $p_i(W)$ is defined). Note that this means that the algorithm maintains the property that each internal node has a path to at least one leaf in $\mathcal{L}$. This ensures that the number of trees with leaves in $\mathcal{L}$ in T_i' is equal to the number of edges cut in T_i' plus 1. It also ensures that the only leaves (nodes with outdegree 0) are the nodes in $\mathcal{L}$.

We will call two leaves u and v *a sibling pair* or *siblings* in a forest, if they belong to the same tree in the forest, and they are the only two leaves in the subtree rooted at the lowest common ancestor $\text{lca}(u, v)$. Similarly, u and v are an *active* sibling pair in a forest, if they belong to the same tree in the forest, and are the only two *active* leaves in the subtree rooted at the lowest common ancestor $\text{lca}(u, v)$ (an equivalent definition is that $p(u) = p(v)$ in the forest).

If leaves u and v are an active sibling pair in both T_1' and T_2', we *merge* one of the leaves (say u) with the other (v). This means that from that point on v represents the subtree containing both u and v, instead of just the leaf v itself. This is accomplished by just making u inactive. Note that this merge operation can be performed recursively, where one or both of the leaves involved in the operation can already be leaves that represent subtrees. It is not hard to see that the subtree that is represented by an active leaf v is one of the two subtrees rooted at the child of $p(v)$, namely the subtree that contains v.

If leaves u and v are not active siblings in T_2' (and they are active siblings in T_1'), we can choose to *cut off an active subtree* between leaves u and v. To describe this operation, let $W_1, W_2, \ldots, W_k$ be the active leaves of the active trees that would be created by deleting the path between u and v (both the nodes and the edges) in T_2'. Note that $p_2(W_\ell)$ is on the path between u and v for all $\ell \in \{1, 2, \ldots, k\}$, because u and v are not active siblings. Cutting off an active subtree between leaves u and v now means cutting off any such a set W_ℓ.

The algorithm is given in Algorithm 1. The boxed expressions refer to updates of the dual solution which will be discussed in Section 3.2.2. These expressions are only necessary for the analysis of the algorithm.

▶ **Theorem 1.** *Algorithm 1 is a 3-approximation algorithm for the Maximum Agreement Forest problem.*

The proof of this theorem is given in the next subsection. It is clear the algorithm can be implemented to run in polynomial time. In Section 3.2.1, we show that the algorithm returns an agreement forest and we show that the number of edges deleted from T_2 by the algorithm can be upper bounded by three times the objective value of a feasible solution to the dual of a linear programming (LP) relaxation of MAF.

3.2 Analysis of the 3-Approximation Algorithm

3.2.1 Correctness

We need to show that the algorithm outputs an agreement forest. The trees of T_1' and T_2' each give a partitioning of $\mathcal{L}$, and clearly any internal node v belongs to $V[L]$ for at most one set in the partitioning. It remains to show that the two forests give the *same* partitioning of $\mathcal{L}$ and that each set in the partitioning is compatible.

The algorithm ends with all trees in T_1' and T_2' being inactive, and the algorithm maintains that the set of leaves represented by an active leaf u (i.e., the leaves that were merged with u (recursively), and u itself) form the leaf set of a subtree in both T_1' and T_2'. To be precise, it is the subtree rooted at one of the children of $p(u)$, namely the subtree that contains u.

1 $\boxed{y_v \leftarrow 0 \text{ for all } v \in V.}$
2 **while** *there exist at least 2 active leaves* **do**
3 Find an active sibling pair u, v in the active tree in T_1'.
4 **if** *u or v is the only active leaf in its tree in T_2'* **then**
5 Cut off this node (say u) in T_1' as well and make it inactive. $\boxed{y_u \leftarrow 1.}$
6 **else**
7 **if** *u and v are active siblings in T_2'* **then**
8 Merge u and v (i.e., make u inactive to "merge" it with v).
9 **else**
10 **if** *u and v are in the same tree in T_2', and this tree contains an active leaf w such that $uv|w$ in T_2* **then**
11 Cut off an active subtree W between u and v in T_2'. $\boxed{\text{Decrease } y_{p_2(W)} \text{ by 1.}}$
12 **end if**
13 Cut off u and cut off v in T_1' and T_2' and make them inactive.
$\boxed{y_u \leftarrow 1,\ y_v \leftarrow 1, \text{ decrease } y_{\mathrm{lca}_1(u,v)} \text{ by 1.}}$
14 **end if**
15 **end if**
16 **end while**
17 Make the remaining leaf (say v) inactive. $\boxed{y_v \leftarrow 1.}$

Algorithm 1: A 3-Approximation for Maximum Agreement Forest. The boxed text refers to updates of the dual solution as discussed in Section 3.2.2.

Furthermore note that this leaf set is compatible. This is easily verified by induction on the number of merges.

When u is the only active leaf in its tree in both forests, then the trees containing u in the two forests are thus guaranteed to have the same, compatible, set of leaves. Now, an inactive tree is created exactly when both T_1' and T_2' have an active tree in which some u is the only active leaf (lines 5, 13 and 17), and thus the two forests indeed induce the same partition of $\mathcal{L}$ into compatible sets.

3.2.2 Approximation Ratio

In order to prove the claimed approximation ratio, we will construct a feasible dual solution to the dual of the relaxation of the ILP given in Section 2. The dual LP is given in Figure 1(a). The dual LP has an optimal solution in which $0 \leq y_v \leq 1$ for all $v \in \mathcal{L}$. The fact that $\{v\}$ is a compatible set implies that $y_v \leq 1$ must hold for every $v \in \mathcal{L}$. Furthermore, note that changing the equality constraints of the primal LP to $\geq$-inequalities does not change the optimal value, and hence we may assume $y_v \geq 0$ for $v \in \mathcal{L}$.

It will be convenient for our analysis to rewrite this dual by introducing additional variables for every (not necessarily compatible) set of labeled leaves. We will adopt the convention to use the letter A to denote a set of leaves that is not necessarily compatible, and the letter L to denote a set of leaves that is compatible (i.e., $L \in \mathcal{C}$). The dual LP can then be written as in Figure 1(b). Any solution to this new LP can be transformed into a solution to the original dual LP by, for each A such that $z_A > 0$, taking some leaf $v \in A$ and setting $y_v = y_v + z_A$ and $z_A = 0$. This is feasible because the left-hand side of the first family of inequalities will not increase for any compatible set L, and it will decrease for L such that $A \cap L \neq \emptyset$ and $v \notin L$. Conversely, a solution to the original dual LP is feasible for this new LP by setting $z_A = 0$, for every set of labeled leaves A.

$$\begin{array}{lll} \max & \sum_v y_v - 1, & \\ \text{s.t.} & \sum_{v \in V[L]} y_v \leq 1 & \forall L \in \mathcal{C}, \\ & y_v \leq 0 & \forall v \in V \setminus \mathcal{L}. \end{array}$$

(a) Dual LP

$$\begin{array}{lll} \max & \sum_v y_v + \sum_{A \subseteq \mathcal{L}} z_A - 1, & \\ \text{s.t.} & \sum_{v \in V[L]} y_v + \sum_{A : A \cap L \neq \emptyset} z_A \leq 1 & \forall L \in \mathcal{C}, \\ & y_v \leq 0 & \forall v \in V \setminus \mathcal{L}, \\ & y_v \geq 0 & \forall v \in \mathcal{L}, \\ & z_A \geq 0 & \forall A \subseteq \mathcal{L}. \end{array}$$

(b) Reformulated dual LP

Figure 1 The dual of the LP relaxation for the ILP given in Section 2. The reformulated dual LP will be referred to as (D).

We will refer to the left-hand side of the first family of constraints, i.e., $\sum_{v \in V[L]} y_v + \sum_{A : A \cap L \neq \emptyset} z_A$, as the *load* on set L.

▶ **Definition 2.** The dual solution associated with a forest T_2', obtained from T_2 by edge deletions, active leaf set $\mathcal{L}'$, and variables $y = \{y_v\}_{v \in V}$ is defined as (y, z) where $z_A = 1$ exactly when A is the active leaf set of a tree in T_2', and 0 otherwise.

We will sometimes use "the dual solution" to refer to the dual solution associated with $T_2', \mathcal{L}'$ and y when the forest, active leaf set and y-values are clear from the context.

▶ **Lemma 3.** *After every execution of the while-loop of Algorithm 1, the dual solution associated with T_2', $\mathcal{L}'$, and y is a feasible solution to (D) and the objective value of this solution is at least $\frac{1}{3}|E(T_2) \setminus E(T_2')|$.*

Proof. We prove the lemma by induction on the number of iterations. Initially, $z_{\mathcal{L}} = 1$, and all other variables are equal to 0. Clearly, this is a feasible solution with objective value 0.

Observe that the dual solution maintained by the algorithm satisfies that $y_u = 0$ while u is active. Therefore, if there is a single active leaf u in a tree in T_2', then making this leaf u inactive and setting $y_u = 1$ does not affect dual feasibility and the dual objective value, since making u inactive decreases $z_{\{u\}}$ from 1 to 0. Also note that merging two active leaves (and thus making one of the two leaves inactive), replaces the set of active leaves A in an active tree in T_2' by a smaller set $A' \subset A$, with $A' \neq \emptyset$. Hence, the dual solution changes from having $z_A = 1$ to having $z_{A'} = 1$, which clearly does not affect dual feasibility or the dual objective value. Hence, we only need to verify that the dual solution remains feasible and its objective increases sufficiently for operations of the algorithm that cut edges from T_2', i.e., lines 11 and 13.

In line 11, one edge is cut in T_2', $y_{p_2(W)}$ decreases by 1. Let A be the set of active leaves in the tree containing W in T_2' before cutting off W. z_A decreases by 1, $z_{A \setminus W}$ increases by 1, z_W increases by 1. The only sets L for which the left-hand side potentially increases are sets L so that $W \cap L \neq \emptyset$ and $(A \setminus W) \cap L \neq \emptyset$. However, $p_2(W) \in V[L]$ for such sets L, and since $y_{p_2(W)}$ is decreased by 1, the load is not increased for any compatible set L. The dual objective is unchanged, but will change in line 13 of the algorithm, as we will show next.

In line 13, let A_u be the set of active leaves in the tree in T_2' containing u at the start of line 13 in the algorithm, and A_v be the set of active leaves in the tree in T_2' containing v. Note that $A_u \setminus \{u, v\} \neq \emptyset$: if $v \notin A_u$, then this holds because otherwise we would execute line 5, and if $v \in A_u$, then this holds because u, v are not active siblings at the start of line 11, and if u, v became active siblings after executing line 11, then the condition for line 11 implies that there exists $w \in A_u$ such that $uv|w$ in T_2.

The fact that $A_u \setminus \{u, v\} \neq \emptyset$ (and, by symmetry $A_v \setminus \{u, v\} \neq \emptyset$) implies that the total value of $\sum_A z_A + y_u + y_v$ increases by 2. Since we also decrease $y_{\mathrm{lca}_1(u,v)}$ by 1 the total increase in the objective of the dual solution by line 13 is 1. Also, in lines 11 and 13, a total of at most three edges are cut in T_2'.

It remains to show that executing line 13 does not make the dual solution (y, z) infeasible. Note that for each active set A' with $z_{A'} = 1$ before cutting off u and v, there is exactly one unique active subset $A \subseteq A'$ with $z_A = 1$ after cutting off u and v. Therefore the total value of $\sum_{A:A\cap L\neq\emptyset} z_A$ does not increase after cutting off u and v for any $L \subseteq \mathcal{L}$.

For any L that includes one of u, v, and at least one other active leaf, it must be the case that $\mathrm{lca}_1(u, v) \in V[L]$, because all active leaves are in one tree in T_1', and u and v were active siblings in T_1' at the start. Hence the only compatible sets L for which the load on L potentially increases by 1 because of an increase in $\sum_{x\in L} y_x$ are sets L that include both u and v. We discern two cases.

Case 1: An active subtree W was cut off in line 11. In this case, the load on L was decreased by 1 in line 11, compensating for the increase in line 13: $V[L]$ contains all nodes on the path between u and v in T_2, and hence also $p_2(W)$. It cannot contain a leaf $x \in W$, because $\{u, v, x\}$ form an inconsistent triplet (because $uv|x$ in T_1).

Case 2: No active subtree W was cut off in line 11. In this case, the value of $\sum_{A:A\cap L\neq\emptyset} z_A$ is decreased by at least 1: If u and v are in the same tree in T_2' before cutting off u and v, then this tree contains no leaves x such that $uv|x$ in T_2 since otherwise an active subtree W would have been cut off. Hence, L does not contain any active leaf x in the active tree that remains after cutting off u and v in T_2', since any such leaf x does not have $uv|x$ in T_2 and therefore forms an inconsistent triplet with u and v. Since L does contain active leaves in the tree containing u and v in T_2' before cutting off u and v (namely, u and v themselves), the value of $\sum_{A:A\cap L\neq\emptyset} z_A$ indeed decreases by 1.

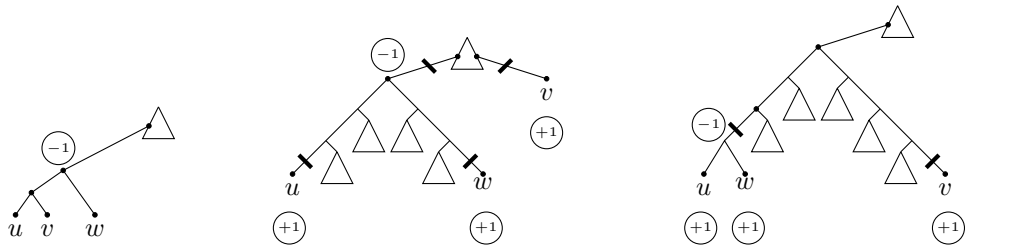
If u and v are not in the same tree in T_2' before cutting off u and v, then a similar argument holds. Since T_2' is obtained from T_2 by deleting edges, at least one of the two active trees containing u and v contains no leaves x such that $uv|x$ in T_2. Without loss of generality, suppose that this holds for the tree containing u. Then, L does not contain any active leaves in the active tree remaining after u is cut off, and hence $\sum_{A:A\cap L\neq\emptyset} z_A$ decreases by at least 1. ◀

By weak duality, we have that the objective value of any feasible solution to (D) provides a lower bound on the objective value of any feasible solution to the LP relaxation of our ILP for MAF, and hence also on the optimal value of the ILP itself. Theorem 1 thus follows from Lemma 3 and the correctness shown in Section 3.2.1.

4 Overview of the 2-Approximation Algorithm

In this section, we begin by giving an outline of the key ideas of our 2-approximation algorithm. We then give an overview of the complete algorithm that we call the "Red-Blue Algorithm".

One of the main ideas behind our 2-approximation is the consideration of the following two "essential" cases. The first "essential" case is the case where we have an active sibling pair u, v in T_1' that are (i) active siblings in T_2', or (ii) in different trees in T_2', or (iii) the tree in T_2' containing u, v does *not* contain an active leaf w such that $uv|w$ in T_2. Then, it is easy to verify, using the arguments in the proof of Lemma 3, that Algorithm 1 "works": it



(a) Subtree in T_1' **(b)** Case (I)(a): Subtree in T_2' **(c)** Case (I)(b): Subtree in T_2'

Figure 2 Dealing with an inconsistent triplet: Circled values denote the y-variables that are set by the algorithm, and bold diagonal lines denote edges that are cut (deleted from T_2'). Triangles denote subtrees with active leaves (that may be empty). Note that there is a distinction between edges that are incident to the root of a subtree represented by a triangle, and edges that are incident to some internal node of the subtree. The latter edges are connected to a dot on the triangle.

increases the dual objective value by at least half of the increase in $|E(T_2) \setminus E(T_2')|$. We will say such a sibling pair u, v is a "success".

The second "essential" case is the case where, in our current forest T_1', there is a subtree containing exactly three active leaves, say u, v, w, where $uv|w$ in T_1, and $\{u, v, w\}$ is an inconsistent triplet; assume without loss of generality that $uw|v$ in T_2, and that the first "essential" case does not apply; in particular, this implies that u, v are in the same tree in T_2'. It turns out that such an inconsistent triplet can be "processed" in a way that allows us to increase the objective value of the dual solution in such a way that it "pays for" half the increase in the number of edges cut from T_2'. There are a number of different cases to consider depending on whether all three leaves u, v, w are in the same tree in T_2' (case I) or not (case II), and whether the tree in T_2' containing w has an active leaf x such that $xw|u$ in T_2' (case (a)) or not (case (b)).

Figure 2 gives an illustration of T_1' and some possible configurations for T_2'. Consider for example case (I)(b). Since $\{u, v, w\}$ is an inconsistent triplet, it is not hard to see that any solution to MAF either has v as a singleton component, or either u or w must be a singleton component. Indeed, we can increase the dual objective by 1, by updating the y-values as indicated by the circled values in Figure 2 (a) and (b). The bold diagonal lines denote two edges that are cut (deleted from T_2'). Similar arguments can be made for the other cases.

Unfortunately, neither of the essential cases may be present in the forests T_1', T_2', and therefore the ideas given above may not be applicable. However, they do work if we generalize our notions. First, we generalize the notion of "active sibling pair in T_1'".

▶ **Definition 4.** A set of active leaves U is an *active sibling set* in T_1' if the leaves in U are the only active leaves in the subtree of T_1' rooted at $\text{lca}_1(U)$. U is a *compatible active sibling set* in T_1' if U is an active sibling set in T_1' that contains no inconsistent triplets.

Note that we will only use the term compatible active sibling set for T_1', and never for T_2'. We will therefore sometimes omit the reference to T_1', and simply talk about a "compatible active sibling set".

We similarly generalize the notion of a subtree in T_1' containing exactly three active leaves that form an inconsistent triplet.

▶ **Definition 5.** A set of active leaves $R \cup B$ is a *minimal incompatible active sibling set* in T_1' if $R \cup B$ is incompatible, R and B are compatible active sibling sets in T_1', and $p_1(R) = p_1(B)$.

The Red-Blue Algorithm now proceeds as follows: it begins by identifying a minimal incompatible active sibling set $R \cup B$ in T_1'. Such a set can be found by checking if the active leaf sets of the left and right subtrees of the root are compatible sets. If yes, then either all active leaves are compatible, or we have found a minimal incompatible active set. If not, then the active leaf set of one of the subtrees is incompatible, and we recurse on this subtree until we find a node in T_1' for which the active leaf sets of the left and right subtrees form a minimal incompatible set $R \cup B$. Note that we can assume $\text{lca}_2(R) = \text{lca}_2(R \cup B)$.

The algorithm will then "distill" R by repeatedly considering sibling pairs u, v in R, and executing operations similar to those in Algorithm 1, except that only one of u and v becomes inactive (and a bit more care has to be taken in certain cases). Procedure 1 gives the procedure ResolvePair the algorithm uses for handling a sibling pair u, v.

```
1  if u and v are in different trees in T_2' then
2      Relabel u and v if necessary so that lca_2(u,v) is not in the tree containing u in T_2'.
3      if the tree containing u in T_2' has other active leaves not in U then
4          FinalCut: Cut off u in T_1' and T_2' and make it inactive. [y_u ← 1.]
5      else
6          Cut off u in T_1' and make it inactive. [y_u ← 1.]
7      end if
8  else
9      if u and v are active siblings in T_2' then
10         Merge u and v (i.e., make u inactive to "merge" it with v).
11     else
12         Relabel u and v if necessary so that p_2(u) ≠ lca_2(u,v).
13         Cut off an active subtree W between u and v by cutting the edge below p_2(u) that is
               not on the path from u to v. [Decrease y_{p_2(u)} by 1.]
14         if u and v are now active siblings in T_2' then
15             Merge-After-Cut: Merge u and v (i.e., make u inactive to "merge" it with v).
                   [y_u ← 1.]
16         else
17             Cut off u in T_1' and T_2' and make u inactive. [y_u ← 1.]
18         end if
19     end if
20 end if
```

Procedure 1: ResolvePair(u, v)

Arguments similar to those in Section 3 show that ResolvePair maintains dual feasibility, *provided that we initially reduce* $y_{\text{lca}_1(R)}$ *by* 1. It is also not hard to verify that ResolvePair increases the dual objective by at least half the increase in the primal objective, and the only thing that is therefore needed to show that the algorithm is a 2-approximation is that we can "make up for" the initial decrease of the dual objective caused by decreasing $y_{\text{lca}_1(R)}$. Let us define the operation of "distilling" R as starting by reducing $y_{\text{lca}_1(R)}$ by 1, and then repeatedly finding a pair of active leaves u, v in R which are siblings in T_1' and executing ResolvePair(u, v) until only two active leaves $\hat{u}, \hat{v}$ in R remain. Since all other leaves in R are inactive, $\hat{u}$ and $\hat{v}$ form an active sibling pair in T_1'.

If pair $\hat{u}, \hat{v}$ is a "success" or if line 4 or 15 was executed at least once during the distilling of R, then there exists an operation that makes at least one of $\hat{u}, \hat{v}$ inactive and updates the dual, so that the total increase in the primal objective is at most twice the total increase in the dual objective caused by the processing of pairs in R. Procedure 2 gives the complete description of the procedure that, if successful, "resolves" set R (and will return "Success"):

▶ **Lemma 6.** *If* ResolveSet(R) *returns* Success *then at least one leaf in* R *became inactive, and the increase in the primal objective* $|E(T_2) \setminus E(T_2')|$ *caused by the procedure is at most twice the increase in the dual objective.*

Lemma 12 of the full version [12] contains a more precise formulation of this lemma.

```
21 Decrease y_{lca_1(R)} by 1.
22 while there exist at least three active leaves in R do
23 |  Find u, v in R that form an active sibling pair in T_1'.
24 |  ResolvePair(u, v).
25 end while
26 Let û, v̂ be the remaining active leaves in R.
27 if û and v̂ are active siblings in T_2' then
28 |  Merge û and v̂ (i.e., make û inactive to "merge" it with v̂). y_û ← 1.
29 |  Return Success.
30 else if (û and v̂ are in different trees in T_2') or (the tree containing û and v̂ does not contain
     an active leaf w such that ûv̂|w in T_2) then
31 |  Cut off û in T_2' (if û's tree contains at least one other active leaf) and in T_1' and make û
   |    inactive. y_û ← 1.
32 |  Cut off v̂ in T_2' (if v̂'s tree contains at least one other active leaf) and in T_1' and make v̂
   |    inactive. y_v̂ ← 1.
33 |  Return Success.
34 else if (At least one FinalCut or Merge-After-Cut was executed in some call to ResolvePair
     in the course of the current ResolveSet procedure) then
35 |  ResolvePair(û, v̂).
36 |  Cut off the last active leaf v̂ in U in T_2' and in T_1' and make v̂ inactive. y_v̂ ← 1.
37 |  Return Success.
38 else
39 |  Return Fail.
40 end if
```

Procedure 2: ResolveSet(R)

If Lemma 6 applies, we have made progress (since we have made at least one leaf inactive), and we will have paid for the increase in the primal objective $|E(T_2) \setminus E(T_2')|$ caused by the procedure by twice the increase in the dual objective.

Otherwise, the last active pair of leaves $\hat{u}, \hat{v}$ in R remain active, and we will have a "deficit" in the sense that the increase in the dual objective is at most half the increase in the primal objective *plus* 1. In this case, we similarly distill B by repeatedly calling ResolvePair(u, v) for pairs u, v in B that are active siblings in T_1' until only a single active leaf in B remains. However, we will show that in order to retain dual feasibility, we do not need to start the distilling of B by decreasing $y_{\text{lca}_1(B)}$ (which would give a total "deficit" of 2), but that we can "move" the initial decrease of $y_{\text{lca}_1(R)}$ to instead decrease $y_{\text{lca}_1(R \cup B)}$. Lemma 13 in the full version [12] shows that this indeed preserves dual feasibility.

Once R and B have both been "distilled", we are left with $\hat{u}, \hat{v}, \hat{w}$ that are an inconsistent triplet and form the active leaf set of a subtree in T_1'. We show how to deal with the triplet $\{\hat{u}, \hat{v}, \hat{w}\}$ (in ways similar to those in Figure 2) and we prove that in the entire processing of $R \cup B$, we have increased the dual objective by half of the number of edges we cut from T_2'.

Algorithm 2 gives an overview of the "Red-Blue Algorithm". It first calls a procedure Preprocess, which executes simple operations that do not affect the primal or dual objective:

merging two leaves if they are active siblings in both forests, and cutting off and deactivating a leaf in T_1' if it is the only active leaf in its tree in T_2'. At the end of an iteration, the Red-Blue algorithm needs to consider different cases for the final triplet. The description of these subroutines can be found in [12].

Set $T_1' \leftarrow T_1$, $T_2' \leftarrow T_2$, $\mathcal{L}' \leftarrow \mathcal{L}$. $y_u \leftarrow 0$ for all $u \in \mathcal{L}$.
PREPROCESS.
while $\mathcal{L}' \neq \emptyset$ **do**
  Find a minimal incompatible active sibling set $R \cup B$, with $\mathrm{lca}_2(R) = \mathrm{lca}_2(R \cup B)$.
  if RESOLVESET(R) *returns* FAIL **then**
    Decrease $y_{\mathrm{lca}_1(R \cup B)}$ by 1, and increase $y_{\mathrm{lca}_1(R)}$ by 1.
    while *there exist at least two active leaves in* B **do**
      Find u, v in B that form an active sibling pair in T_1'.
      RESOLVEPAIR(u, v).
    end while
    Let $\hat{r}_1, \hat{r}_2 \in R$ and $\hat{b} \in B$ be the remaining active leaves.
    Consider three different cases depending on whether $\hat{r}_1, \hat{r}_2$ and $\hat{b}$ are in one, two or three different trees in T_2' (see Section 6.1 and 6.2 in [12] for details).
  end if
  PREPROCESS.
end while

Algorithm 2: Red-Blue Algorithm for Maximum Agreement Forest

▶ **Theorem 7.** *The Red-Blue Algorithm is a 2-approximation algorithm for the Maximum Agreement Forest problem.*

5 Implementation Details

We implemented the Red-Blue approximation algorithm in Java, and tested it on instances with $|\mathcal{L}| = 2000$ leaves that were generated as follows: the number of leaves in the left subtree is set equal to a number between 1 and $|\mathcal{L}| - 1$ drawn uniformly at random, and a subset of this size is chosen uniformly at random from the label set. Then this procedure recurses until it arrives at a subtree with only 1 leaf – this will be the whole subtree.

After generating T_1 as described above, the tree T_2 was created by doing 50 random Subtree Prune-and-Regraft operations (where random means that the root of the subtree that is pruned was chosen uniformly at random, as well as the edge which is split into two edges, so that the new node created can be the parent of the pruned subtree, under the conditions that this is a valid SPR-operation). This construction allows us to deduce an upper bound of 50 on the optimal value. Our algorithm finds a dual solution that in 44% of the 1000 runs is equal to the optimal dual solution, and in 37% of the runs is 1 less than the optimal solution. The observed average approximation ratio is about 1.92. After running our algorithm, we run a simple greedy search algorithm which repeatedly looks for two trees in the agreement forest that can be merged (i.e., such that the resulting forest is still a feasible solution to MAF). The solution obtained after executing the greedy algorithm decreases the observed approximation ratio to less than 1.28. The code is available at `http://frans.us/MAF`.

6 Conclusion

Our algorithm and analysis raise a number of questions. First of all, although we believe that, conceptually, our algorithm is quite natural, the actual algorithm is complicated, and it would be interesting to find a simpler 2-approximation algorithm. Secondly, it is clear that our algorithm can be implemented in polynomial time, but the exact order of the running time is not clear. The bottleneck seems to be the finding of a minimal incompatible active sibling set, although it may be possible to implement the algorithm in a way that simultaneously processes sibling pairs as in ResolvePair, while it is looking for a minimal incompatible active sibling set.

Acknowledgements. We thank Neil Olver and Leen Stougie for fruitful discussions.

References

1 Benjamin L. Allen and Mike Steel. Subtree transfer operations and their induced metrics on evolutionary trees. *Annals of Combinatorics*, 5(1):1–15, 2001.

2 Maria Luisa Bonet, Katherine St John, Ruchi Mahindru, and Nina Amenta. Approximating subtree distances between phylogenies. *Journal of Computational Biology*, 13(8):1419–1434, 2006.

3 Magnus Bordewich, Catherine McCartin, and Charles Semple. A 3-approximation algorithm for the subtree distance between phylogenies. *Journal of Discrete Algorithms*, 6(3):458–471, 2008. doi:10.1016/j.jda.2007.10.002.

4 Magnus Bordewich and Charles Semple. On the computational complexity of the rooted subtree prune and regraft distance. *Ann. Comb.*, 8(4):409–423, 2004. doi:10.1007/s00026-004-0229-z.

5 Charles Darwin. *Notebook B: Transmutation of species (1837–1838)*. In: John van Wyhe: The Complete Work of Charles Darwin Online, 2002. URL: http://darwin-online.org.uk/.

6 Martin Farach and Mikkel Thorup. Optimal evolutionary tree comparison by sparse dynamic programming. In *FOCS'94: Proceedings of 35th Annual Symposium on Foundations of Computer Science*, pages 770–779. IEEE, 1994.

7 Martin Farach and Mikkel Thorup. Sparse dynamic programming for evolutionary-tree comparison. *SIAM Journal on Computing*, 26(1):210–230, 1997.

8 A. D. Gordon. A measure of the agreement between rankings. *Biometrika*, 66(1):7–15, 1979.

9 Jotun Hein, Tao Jiang, Lusheng Wang, and Kaizhong Zhang. On the complexity of comparing evolutionary trees. *Discrete Applied Mathematics*, 71(1-3):153–169, 1996. doi:10.1016/S0166-218X(96)00062-5.

10 Estela M. Rodrigues. *Algoritmos para Comparação de Árvores Filogenéticas e o Problema dos Pontos de Recombinação*. PhD thesis, University of São Paulo, Brazil, 2003. Chapter 7, available at http://www.ime.usp.br/~estela/studies/tese-traducao-cp7.ps.gz.

11 Estela M. Rodrigues, Marie-France Sagot, and Yoshiko Wakabayashi. The maximum agreement forest problem: approximation algorithms and computational experiments. *Theoretical Computer Science*, 374(1-3):91–110, 2007. doi:10.1016/j.tcs.2006.12.011.

12 Frans Schalekamp, Anke van Zuylen, and Suzanne van der Ster. A duality based 2-approximation algorithm for maximum agreement forest. *CoRR*, abs/1511.06000, 2015. URL: http://arxiv.org/abs/1511.06000.

13 Feng Shi, Qilong Feng, Jie You, and Jianxin Wang. Improved approximation algorithm for maximum agreement forest of two rooted binary phylogenetic trees. *Journal of Combinatorial Optimization*, 2015. doi:10.1007/s10878-015-9921-7.

14 Mike Steel and Tandy Warnow. Kaikoura tree theorems: Computing the maximum agreement subtree. *Information Processing Letters*, 48(2):77–82, November 1993. doi:10.1016/0020-0190(93)90181-8.

15 Chris Whidden, Robert G. Beiko, and Norbert Zeh. Fixed-parameter algorithms for maximum agreement forests. *SIAM Journal on Computing*, 42(4):1431–1466, 2013. doi:10.1137/110845045.

16 Chris Whidden and Norbert Zeh. A unifying view on approximation and FPT of agreement forests. In *Algorithms in Bioinformatics*, volume 5724 of *Lecture Notes in Computer Science*, pages 390–402. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-04241-6_32.

17 Yufeng Wu. A practical method for exact computation of subtree prune and regraft distance. *Bioinformatics*, 25(2):190–196, 2009. doi:10.1093/bioinformatics/btn606.

18 Yufeng Wu and Jiayin Wang. Fast computation of the exact hybridization number of two phylogenetic trees. In *Bioinformatics Research and Applications*, volume 6053 of *Lecture Notes in Computer Science*, pages 203–214. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-13078-6_23.