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

Neil Olver[1,2], Frans Schalekamp[3], Suzanne van der Ster[5], Leen Stougie[2,1,6], and Anke van Zuylen[4]

[1]Department of Econometrics & Operations Research, Vrije Universiteit Amsterdam
[2]Centrum Wiskunde & Informatica
[3]School of Operations Reseach and Information Engineering, Cornell University
[4]Department of Mathematics, College of William and Mary
[5]Albert Heijn Online
[6]INRIA-Erable

November 15, 2018

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 rooted Subtree Prune-and-Regraft (rSPR) distance between two phylogenetic trees. Our algorithm is combinatorial and its running time is quadratic in the input size. To prove the approximation guarantee, we construct a feasible dual solution for a novel linear programming formulation. In addition, we show this linear program is stronger than previously known formulations, and we give a compact formulation, showing that it can be solved in polynomial time.

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

MSC: 68W25, 90C27, 92D15.

1 Introduction

Evolutionary relationships are often modeled by a rooted tree, where the leaves represent 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 [10], who used them in his notebook to elucidate his thoughts on evolution. For an introduction to phylogenetic trees we refer to [11, 22]

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.

Such partly incompatible trees may actually be unavoidable: 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 [16, 17]. Irrespective of the cause of the conflict, the natural question arises to quantify the dissimilarity between such trees. Especially in the context of reticulation, a particularly meaningful measure of comparing phylogenetic trees is the Subtree Prune-and-Regraft distance for

*This paper is based on the (substantially different) extended abstract [21].

rooted trees (rSPR-distance), which 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 [15].

In the definition of MAF by Hein et al. we are given two rooted binary trees, each having its leaves labeled with the same set of labels $\mathcal{L}$, where in each tree each leaf has one label and each label is assigned to one leaf. The problem is to find a minimum set of edges to be deleted from the two trees, so that the rooted trees in the resulting two forests (where the choice of the root is the natural choice) can be paired up into pairs of *isomorphic* trees. Two rooted trees are isomorphic if their *restrictions* are the same, where the restriction of a tree is obtained by considering the minimal tree spanning the same set of leaves from $\mathcal{L}$, and then contracting nodes with only a single child.

Since the introduction by Hein et al. in [15], in which they also proved NP-hardness, MAF has been extensively studied, mostly in its version of two rooted binary input trees. After Allen and Steel [1] pointed out that the claim by Hein et al. that solving MAF on two rooted directed trees computes the rSPR-distance between the trees is incorrect, Bordewich and Semple [5] presented a subtle redefinition of MAF, whose optimal value does coincide with the rSPR-distance. In this redefinition it is required that the two forests agree on the tree containing the original roots of the input trees. This has now become the standard definition of MAF, for which Bordewich and Semple [5] showed that NP-hardness still holds, and Rodrigues [19] showed that it is in fact APX-hard.

The problem has attracted a lot of attention, and indeed has become a canonical problem in the field of phylogenetic networks. Many variants of MAF have been studied, including versions where the input consists of more than two trees [6, 7], and where the input trees are unrooted [27, 26] or non-binary [20, 25]. We will concentrate on MAF in its classical form with two rooted binary input trees, and we will be concerned with the worst-case approximability of the problem. The literature includes many other approaches to the problem, including fixed-parameter tractable algorithms (e.g., [27, 26]) and integer linear programming [28, 29]. But the quest for better approximation algorithms has become central within the MAF literature.

Our result improves over a sequence spanning 10 years of approximation algorithms, starting with the first correct 5-approximation [3]. This was followed by several 3-approximations [4, 20, 26], each one improving on the running time, and the last one giving a relatively simple and elegant proof. A 2.5-approximation followed [23]. In 2016, Chen et al. [9] described a 7/3-approximation. Independently in the same year, a subset of the present authors [21] gave a factor 2 approximation algorithm. Subsequently, Chen et al. [8] gave a different factor 2 approximation algorithm using very different methods, and with a cubic running time.

The 2-approximation algorithm presented in the current paper may be viewed as the full version of the algorithm in [21]. However, while the algorithm presented here is similar in spirit, it differs in many details, and the exposition is entirely new. Although the algorithm and analysis remain quite subtle, this version is significantly shorter and clearer. Moreover, we show how our algorithm can, with some care, be implemented in quadratic time ([21] discuss only a polynomial time bound). This improves over the cubic running time of Chen et al. [8].

Our 2-approximation algorithm differs from previous works in two key aspects. First of all, our algorithm takes a global approach; choices may depend on large parts of the instance. All the previous algorithms that obtained a worse approximation ratio considered only local, constant-sized, substructures. Secondly, we introduce a novel integer linear programming formulation for the analysis. Our approximation guarantee is proved by constructing a feasible solution to the dual of this linear program, rather than arguing locally about the objective of the optimal solution.

While we provide a new integer linear programming formulation and exploit its linear relaxation in our analysis, we do not need to actually solve the relaxation as part of our algorithm. In fact, the formulation has an exponential number of variables, and so it is not immediately clear that it can be efficiently optimized. We show that it can be reformulated as a compact LP, with only a polynomial number of variables and constraints. We believe that this is interesting for a number of reasons. It implies that the linear relaxation can be solved efficiently (in polynomial time); this may be of future utility in obtaining better approximation guarantees using LP-rounding techniques, which do require an optimal solution to the relaxation. Moreover, the compact formulation is amenable to use in commercial integer programming solvers. There is a previous

formulation due to Wu [28], but our formulation is significantly stronger: the integrality gap of the relaxation of Wu is at least 3.2, whereas for ours we show it is at most 2, and in fact the worst example that we are aware of has integrality gap 1.25 (see the appendix). Finally, we remark that our compact formulation can be easily adapted to handle other variants of MAF—for example, settings with more than two trees.

We have implemented and tested our algorithm, as well as the compact formulation [18]. The implementation has been designed so that it is easy to step through the algorithm and explore its behaviour on a given instance; the reader may find it helpful when examining the technical details of the algorithm.

Outline. We define the problem and introduce necessary notation in Section 2. Section 3 describes the algorithm, and proves that it produces a feasible solution to MAF. In Section 4, we introduce the linear program, and describe a feasible solution to its dual that can be maintained by the algorithm. We then show the objective value of this dual solution is always at least half the objective value of the MAF solution, which proves the approximation ratio of two. In Section 5, we show a compact formulation of the (exponential sized) linear program used for the analysis. In the appendices, we show that our algorithm can be implemented to run in time quadratic in the size of the input, and we give an example that shows that a previously known integer linear program [28] is not as strong as the formulation introduced here.

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 binary 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. We will use $\mathcal{L}$ also to denote the leaves of the trees.

Let V_1 and V_2 denote the node set of T_1 and T_2 respectively, and let $V = V_1 \cup V_2$. We call all nodes in $V \setminus \mathcal{L}$ *internal nodes*. We let $\mathcal{L}(u)$ denote the set of leaves that are descendants of a node $u \in V$.

We will use the following notational conventions: we use u and v to denote arbitrary nodes (including leaves), if the node we refer to is an internal node in V_2, we will use $\hat{u}$ and $\hat{v}$, and we use the letters x, y and w to refer to leaves.

For $A \subset \mathcal{L}$ we use $V_i[A]$ to denote the set of (internal) nodes in T_i that lie on a path between any two leaves in A for $i \in \{1, 2\}$, and define $V[A] := V_1[A] \cup V_2[A]$.

Definition 1. We will say that a set $A \subseteq \mathcal{L}$ *covers* a node $u \in V$ if $u \in V[A]$. We say that $A, A' \subseteq \mathcal{L}$ *overlap* if $V[A] \cap V[A'] \neq \emptyset$; we can also say that A *overlaps* A' in U, for $U \subseteq V$, if $V[A] \cap V[A'] \cap U \neq \emptyset$. We say a partition $\mathcal{P}$ of $\mathcal{L}$ *overlaps* in $U \subseteq V$ if there exist $A, A' \in \mathcal{P}$, $A \neq A'$ such that A and A' overlap in U.

For $A \subseteq \mathcal{L}$, we let $\mathrm{lca}_i(A)$ denote the least common ancestor of A in T_i. We will sometimes omit braces of explicit sets and write, e.g., $\mathrm{lca}_1(x_1, x_2, x_3)$ instead of $\mathrm{lca}_1(\{x_1, x_2, x_3\})$.

For nodes u, v in the same tree, we use $u \prec v$ to indicate that u is a descendant of v and $u \preceq v$ if u is equal to v or a descendant of v.

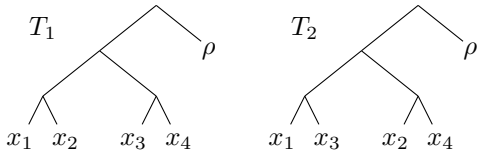
Definition 2. A set $L \subseteq \mathcal{L}$ is *compatible* if for all $x_1, x_2, x_3 \in L$

$$\mathrm{lca}_1(x_1, x_2) \prec \mathrm{lca}_1(x_1, x_2, x_3) \Leftrightarrow \mathrm{lca}_2(x_1, x_2) \prec \mathrm{lca}_2(x_1, x_2, x_3).$$

We call a triple of leaves *incompatible* if it is not a compatible set. Note that $L \subseteq \mathcal{L}$ is compatible precisely if the subtrees induced by L in T_1 and T_2 are isomorphic.

A feasible solution to MAF is a partition $\mathcal{P} = \{A_1, A_2, \ldots, A_k\}$ of $\mathcal{L}$ such that every component A_i is compatible, and A_i does not overlap A_j, for each $i \neq j$. The cost of this solution is defined to be $|\mathcal{P}| - 1$. This cost corresponds to the number of edges that must be deleted from T_1, as well as the same number from T_2, so that in both of the resulting forests, each $A_i \in \mathcal{P}$ is the leaf set of a single tree.

Remark. In order for MAF to correspond to the rSPR distance, it is necessary to add an additional node ρ to $\mathcal{L}$, as a unique sibling to the root in both T_1 and T_2. This is the distinction between the original definition of MAF by Hein [15] and the correction by Bordewich and Semple [5]. We simply assume that this addition is already included in the input instance, after which there is no need to distinguish this additional leaf from the others.



To describe our algorithm, it will be convenient to extend the notion of compatibility.

Definition 3. Given $K \subseteq \mathcal{L}$, we say a set $L \subseteq \mathcal{L}$ is *K-compatible* if $L \cap K$ is compatible. A partition $\mathcal{P} = \{A_1, A_2, \ldots, A_k\}$ of $\mathcal{L}$ is *K-compatible* if A_i is K-compatible for all $i = 1, 2, \ldots, k$.

3 The Red-Blue Algorithm

The algorithm maintains a partition $\mathcal{P}$ of $\mathcal{L}$, which at the end of the algorithm will correspond to a feasible solution to MAF. The algorithm will maintain the invariant that $\mathcal{P}$ does not overlap in V_2. Observe that this is equivalent to defining $\mathcal{P}$ to be the leaf sets of the trees in a forest, obtained by deleting edges from T_2. Initially $\mathcal{P} = \{\mathcal{L}\}$.

The algorithm works towards feasibility by iteratively refining $\mathcal{P}$, focusing each iteration on a set of leaves $\mathcal{L}(u)$ for some $u \in V_1$, for which the current partition is infeasible in some (quite narrowly defined) way. At the end of the iteration the solution is feasible if we restrict our attention to $\mathcal{L}(u)$, and even if we consider $\mathcal{L}(u) \cup \{w\}$ for any arbitrary $w \in \mathcal{L} \setminus \mathcal{L}(u)$.

We use the following definition to specify which sets $\mathcal{L}(u)$ the algorithm considers.

Definition 4. Given an infeasible partition $\mathcal{P}$ that does not overlap in V_2, we call $u \in V_1$ a *root-of-infeasibility* if at least one of the following holds:

(a) $\mathcal{P}$ is not $\mathcal{L}(u)$-compatible;

(b) $\mathcal{P}$ overlaps in $V_1[\mathcal{L}(u)]$;

(c) there exists a component A in $\mathcal{P}$ such that $A \setminus \mathcal{L}(u) \neq \emptyset$, and $A \cap \mathcal{L}(u) \cup \{w\}$ is not compatible for all $w \in A \setminus \mathcal{L}(u)$.

Observe that if $u \in V_1$ is a root-of-infeasibility, then any ancestor of u is a root-of-infeasibility. We will say an internal node u in tree T_i is the "lowest" node with property Γ if property Γ does not hold for any of u's descendants in T_i. The algorithm will identify a lowest node $u \in V_1$ that is a root-of-infeasibility.

Given a root-of-infeasibility $u \in T_1$, we partition $\mathcal{L}$ into R, B, W, where $R = \mathcal{L}(u_r)$ and $B = \mathcal{L}(u_\ell)$ for the two children u_ℓ and u_r of u. We will refer to this partition as a *coloring* of the leaves; we will refer to the leaves in R as red leaves, the leaves in B as blue leaves and the leaves in W as white leaves. We call a component of $\mathcal{P}$ *tricolored* if it has a nonempty intersection with R, B and W, and *bicolored* if it has a nonempty intersection with exactly two of the sets R, B, W. A component is called *multicolored* if it is either tricolored or bicolored, and *unicolored* otherwise.

Observation 1. *Let u be a lowest root-of-infeasibility for $\mathcal{P}$, and consider the coloring R, B, W, where $R = \mathcal{L}(u_r)$ and $B = \mathcal{L}(u_\ell)$ for the two children u_ℓ and u_r of u. Then the set of multicolored components of $\mathcal{P}$ consists of either at most two bicolored components or exactly one tricolored component.*

Proof. If u is a lowest root-of-infeasibility, $\mathcal{P}$ does not overlap in $V_1[R]$ and $V_1[B]$, and so at most one component of $\mathcal{P}$ covers $\mathrm{lca}_1(R)$, and at most one covers $\mathrm{lca}_1(B)$. The observation thus follows immediately, since any bicolored component covers either $\mathrm{lca}_1(R)$ or $\mathrm{lca}_1(B)$, and any tricolored component covers both $\mathrm{lca}_1(R)$ and $\mathrm{lca}_1(B)$. □

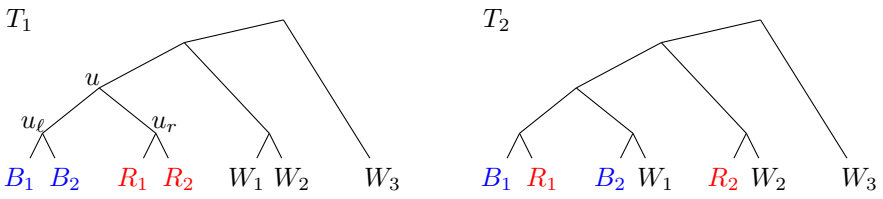


Figure 1: If $\mathcal{P} = \{\mathcal{L}\}$, then node u satisfies case (a) of Definition 4; if $\mathcal{P} = \{\{B_1\}, \{B_2, W_1\}, \{R_1, R_2, W_2, W_3\}\}$, it satisfies case (b) and if $\mathcal{P} = \{\{R_1\}, \{B_1, B_2, W_1, R_2, W_2\}, \{W_3\}\}$, it satisfies (c).

We note that the above observation can be refined; it is possible to show that $\mathcal{P}$ contains either one tricolored component or exactly two bicolored components; see Lemma 12 in Section 4.3.

In Figure 1, we give an example of an input T_1, T_2 and a coloring of the leaves, where R_1 can be a single leaf in R, or it can be a tree with leaves labeled by a compatible subset $R_1 \subset R$ with likewise interpretations for the other capital leaves in this and future figures; the caption gives three partitions such that u satisfies exactly one of the three conditions of Definition 4: If $\mathcal{P} = \{\mathcal{L}\}$, u satisfies (a). Note that u is indeed a lowest root-of-infeasibility, since $\{R_1, R_2, W_3\}$ and $\{B_1, B_2, W_3\}$ are compatible sets, so u_ℓ and u_r do not satisfy (c) (nor (a) and (b)). If $\mathcal{P} = \{\{B_1\}, \{B_2, W_1\}, \{R_1, R_2, W_2, W_3\}\}$, node u satisfies (b). Again, u is a lowest root-of-infeasibility(clearly u_ℓ and u_r does not satisfy (a) and (b); they also do not satisfy (c) since $\{B_1, W_1\}$ is compatible, as is $\{R_1, R_2, W_3\}$). Finally, if $\mathcal{P} = \{\{R_1\}, \{B_1, B_2, W_1, R_2, W_2\}, \{W_3\}\}$, node u satisfies (c). Observe that in this case u is again a lowest root-of-infeasibility.

Red-Blue Algorithm

$\mathcal{P} \leftarrow \{\mathcal{L}\}$.
`pairslist` $\leftarrow \emptyset$.
while $\mathcal{P}$ is not feasible **do**
$\star$ Let $u \in T_1$ be a lowest root-of-infeasibility, with children u_ℓ and u_r.
Let $R = \mathcal{L}(u_r)$, $B = \mathcal{L}(u_\ell)$ and $W = \mathcal{L} \setminus (R \cup B)$.
Make-$R \cup B$-compatible$(\mathcal{P}, (R, B, W))$.
Make-Splittable$(\mathcal{P}, (R, B, W))$.
Split$(\mathcal{P}, (R, B, W))$.
Find-Merge-Pair(`pairslist`$, \mathcal{P}, (R, B, W))$.
end while
Merge-Components(`pairslist`$, \mathcal{P}$).

An overview of the algorithm is given above. The procedures will be described in detail in the subsequent subsections, along with with lemmas regarding the properties they ensure.

We will refer to a pass through the main while-loop of the algorithm as an "iteration". In order to simplify the statement of the lemmas, we will make statements like "let $\mathcal{P}'$ be the partition after ProcedureName$(\mathcal{P}, (R, B, W))$". This implicitly assumes that (R, B, W) was a coloring chosen in the beginning of the current iteration of the Red-Blue algorithm (and thus, that $\text{lca}_1(R \cup B)$ was a lowest root-of-infeasibility at that moment), and that $\mathcal{P}'$ is the partition resulting from calling ProcedureName$(\mathcal{P}, (R, B, W))$ in the current iteration.

Finally, the $\star$ in front of certain lines will be used to refer to these lines in the analysis in Section 4.2.

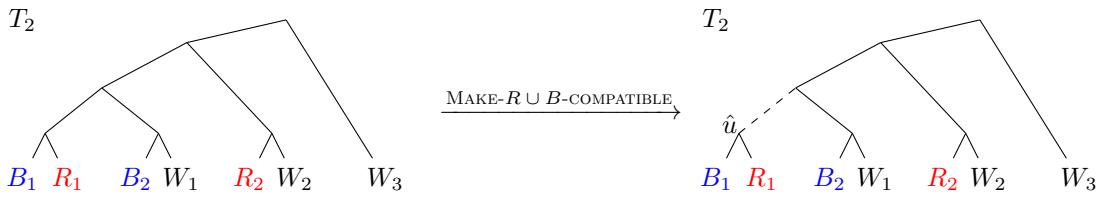


Figure 2: Illustration of MAKE-$R \cup B$-COMPATIBLE$(\mathcal{P}, (R, B, W))$. Because $\mathcal{P}$ and $\mathcal{P}'$ do not overlap in V_2, we can represent these as the leaf sets of trees in a forest obtained by deleting edges from T_2. In this figure and the following figures the dashed edges represent deleted edges.
In this example $\mathcal{P} = \{\mathcal{L}\}$. Then $\hat{u} = \text{lca}_2(R_1, B_1)$, and MAKE-$R \cup B$-COMPATIBLE$(\mathcal{P}, (R, B, W))$ refines the partition to $\{\{B_1, R_1\}, \{B_2, W_1, R_2, W_2, W_3\}\}$, which is $R \cup B$-compatible.

3.1 MAKE-$R \cup B$-COMPATIBLE

procedure MAKE-$R \cup B$-COMPATIBLE$(\mathcal{P}, (R, B, W))$
  while $\exists A \in \mathcal{P}$ that is not $R \cup B$-compatible **do**
  $\star$ Let $\hat{u}$ be a lowest internal node in $V_2[A]$ for which $A \cap \mathcal{L}(\hat{u})$ intersects both R and B.
    $\mathcal{P} \leftarrow \mathcal{P} \setminus \{A\} \cup \{A \cap \mathcal{L}(\hat{u}), A \setminus \mathcal{L}(\hat{u})\}$.
  end while
end procedure

An example is given in Figure 2. We note that in general, the choice of $\hat{u}$ does not have to be unique, and that multiple refinements may be needed to make the partition $R \cup B$-compatible.

As observed above, for any partition $\mathcal{P}$ that does not overlap in V_2, there is a set of edges in T_2, such that $\mathcal{P}$ consists of the leaf sets of the trees in the forest obtained after deleting these edges. Our refinement is equivalent to deleting the edge from $\hat{u}$ towards the root in T_2 and hence the resulting partition does not overlap in V_2 if the original partition did not overlap in T_2.

Lemma 1. *Let $\mathcal{P}'$ be the partition after* MAKE-$R \cup B$-COMPATIBLE$(\mathcal{P}, (R, B, W))$. *Then $\mathcal{P}'$ is a refinement of $\mathcal{P}$ that does not overlap in V_2 and is $R \cup B$-compatible.*

Proof. First, observe $\mathcal{P}$ is R-compatible and B-compatible, since u's children are not roots-of-infeasibility. If $\mathcal{P}$ is $R \cup B$-compatible then $\mathcal{P}$ is not modified by the procedure, and the lemma is vacuously true. Otherwise, the procedure refines $\mathcal{P}$, and we already mentioned above that the resulting partition $\mathcal{P}'$ does not overlap in V_2 provided that $\mathcal{P}$ does not overlap in V_2. The procedure ends when there are no sets in $\mathcal{P}$ that are not $R \cup B$-compatible, so the only thing left to show is that this procedure halts. Because $\hat{u}$ was chosen to be the lowest internal node in $V_2[A]$ such that $A \cap \mathcal{L}(\hat{u})$ intersects both R and B, the children of $\hat{u}$, say $\hat{u}_r$ and $\hat{u}_\ell$, are so that $A \cap \mathcal{L}(\hat{u}_r)$ and $A \cap \mathcal{L}(\hat{u}_\ell)$ can only intersect one of R and B. Therefore $A \cap \mathcal{L}(\hat{u})$ is $R \cup B$-compatible, where A was not, and thus the number of $R \cup B$-compatible components in $\mathcal{P}$ increases, which can only happen at most $|\mathcal{L}|$ times. □

Observe that if $\mathcal{P}$ is $R \cup B$-compatible, then any refinement of $\mathcal{P}$ is also $R \cup B$-compatible, hence we may assume that the partition at any later point in the current iteration of the Red-Blue Algorithm is $R \cup B$-compatible.

3.2 MAKE-SPLITTABLE

The goal of the next two procedures is to further refine the partition so that there is no overlap in $V_1[R \cup B]$. We will do this in two steps, the first of which will achieve the following property.

Figure 3: Illustration of MAKE-SPLITTABLE$(\mathcal{P}, (R, B, W))$. $\mathcal{P} = \{\{R_1\}, \{B_1, B_2, W_1, R_2, W_2\}, \{W_3\}\}$, and the set $A = \{B_1, B_2, W_1, R_2, W_2\}$ is not splittable. MAKE-SPLITTABLE$(\mathcal{P})$ would choose $\hat{u} = \text{lca}_2(B_2, W_1)$ and replace A by $\{B_2, W_1\}$ and $\{B_1, R_2, W_2, W_3\}$.

Definition 5. Given a coloring (R, B, W) of $\mathcal{L}$. A set $A \subseteq \mathcal{L}$ is *splittable* if $A \cap R$, $A \cap B$ and $A \cap W$ do not overlap in V_2.

procedure MAKE-SPLITTABLE$(\mathcal{P}, (R, B, W))$
  while $\exists A \in \mathcal{P}$ that is not splittable **do**
  $\star$ Let $\hat{u}$ be a lowest internal node in $V_2[A]$ such that $A \cap \mathcal{L}(\hat{u})$ is bicolored and $A \setminus \mathcal{L}(\hat{u})$ intersects precisely the same colors as A.
    $\mathcal{P} \leftarrow \mathcal{P} \setminus \{A\} \cup \{A \cap \mathcal{L}(\hat{u}), A \setminus \mathcal{L}(\hat{u})\}$.
  end while
end procedure

First, note that by the same arguments as in the previous subsection, the partition that results from MAKE-SPLITTABLE does not overlap in V_2 if the original partition did not overlap in V_2. It is easy to see that if A is bicolored and not splittable, then there exists $\hat{u} \in V_2[A]$ such that both $A \cap \mathcal{L}(\hat{u})$ and $A \setminus \mathcal{L}(\hat{u})$ are bicolored: just take $\hat{u}$ to be a lowest node in $V_2[A \cap C_1] \cap V_2[A \cap C_2]$ for distinct $C_1, C_2 \in \{R, B, W\}$. We prove below in Lemma 2 that if A is tricolored, we can additionally ensure that $A \setminus \mathcal{L}(\hat{u})$ is tricolored. For this to hold, we need that $\mathcal{P}$ is $R \cup B$-compatible, which by Lemma 1 is indeed true when MAKE-SPLITTABLE is called.

As a first example of MAKE-SPLITTABLE, consider $\mathcal{P} = \{\{B_1, R_1\}, \{B_2, W_1, R_2, W_2, W_3\}\}$ that was the output of MAKE-$R \cup B$-COMPATIBLE depicted in Figure 2. In this example $\mathcal{P}$ is already splittable. In Figure 3 a more interesting example is given.

Lemma 2. MAKE-SPLITTABLE *is well-defined, in that a node $\hat{u}$ satisfying the desired properties in line $\star$ can always be found.*

Proof. As noted above the existence of $\hat{u}$ is clear when A is bicolored. So suppose A is tricolored and not splittable. Note that $V_2[A \cap R]$ and $V_2[A \cap B]$ cannot intersect because A is $R \cup B$-compatible. Assume without loss of generality that $V_2[A \cap R] \cap V_2[A \cap W] \neq \emptyset$, and let $\hat{u}$ be a lowest node in $V_2[A \cap R] \cap V_2[A \cap W]$. Note that both $A \cap \mathcal{L}(\hat{u})$ and $A \setminus \mathcal{L}(\hat{u})$ must intersect W and R, and that $A \cap \mathcal{L}(\hat{u})$ cannot intersect B, since then A is not $R \cup B$-compatible. So $A \cap \mathcal{L}(\hat{u})$ is bicolored, and $A \setminus \mathcal{L}(\hat{u})$ is tricolored. □

Lemma 3. *Let $\mathcal{P}'$ be the partition after* MAKE-SPLITTABLE$(\mathcal{P}, (R, B, W))$. *Then $\mathcal{P}'$ is a refinement of $\mathcal{P}$ that does not overlap in V_2 and in which every component is splittable.*

Proof. By Lemma 2, and since each iteration increases the number of components in $\mathcal{P}$, MAKE-SPLITTABLE must terminate, and by its definition, the final partition $\mathcal{P}'$ contains only splittable components. Clearly $\mathcal{P}'$ is a refinement of $\mathcal{P}$; it does not overlap in V_2 by the same arguments as used in the proof of Lemma 1. □

Before continuing, we summarize the properties of the partition that is the result after MAKE-SPLITTABLE that will be useful in the proof of the approximation guarantee in Section 4. To describe these, we need the notion of a *top component*.

Definition 6. Given the partition $\mathcal{P}$ at the start of the current iteration, and $\mathcal{P}'$ another partition encountered in the current iteration, let $\mathcal{D}$ be the components that were created during the current iteration, i.e., $\mathcal{D} = \mathcal{P}' \setminus \mathcal{P}$. Then $A \in \mathcal{D}$ is a *top component* if there exists no $A' \in \mathcal{D}$ such that $\text{lca}_2(A) \prec \text{lca}_2(A')$.

Lemma 4. *Let $\mathcal{P}^{(0)}$ denote the partition at the start of a given iteration, and (R, B, W) the coloring of the leaves that is selected, let $\mathcal{P}^{(1)}$ denote the partition after* Make-$R \cup B$-compatible$(\mathcal{P}^{(0)}, (R, B, W))$*, and let $\mathcal{P}^{(2)}$ denote the partition after* Make-Splittable$(\mathcal{P}^{(1)}, (R, B, W))$*,*

1. *Only multicolored components are subdivided by the iteration, i.e., if $A \in \mathcal{P}^{(0)} \setminus \mathcal{P}^{(2)}$, then A is multicolored.*
2. *Only multicolored components are created by* Make-$R \cup B$-compatible *and* Make-Splittable*, i.e., if $A \in \mathcal{P}^{(2)} \setminus \mathcal{P}^{(0)}$, then A is multicolored.*
3. *The number of tricolored components in $\mathcal{P}^{(2)}$ is the same as in $\mathcal{P}^{(1)}$.*
4. *Any tricolored component in $\mathcal{P}^{(1)}$ or $\mathcal{P}^{(2)}$ that is not a top component contains no compatible tricolored triple.*
5. *Any bicolored component A in $\mathcal{P}^{(2)}$ that is not a top component satisfies that $\text{lca}_2(A)$ is not overlapped by $A \cap C$ for any color $C \in \{R, B, W\}$. In other words, $\mathcal{L}(\hat{u}_\ell) \cap A$ and $\mathcal{L}(\hat{u}_r) \cap A$ are unicolored where $\hat{u}_\ell$ and $\hat{u}_r$ are the children of $\text{lca}_2(A)$.*
6. *If x_W is in a top component A in $\mathcal{P}^{(0)}$ and x_W is not a descendant of $\text{lca}_2(A \cap (R \cup B))$, then x_W is in a top component in $\mathcal{P}^{(2)}$.*

Proof. Each of the properties is easily verified by inspection of the Make-$R \cup B$-compatible and Make-Splittable procedures. For example, point 4 follows from the fact that a node $\hat{u}$ picked in Make-$R \cup B$-compatible is always chosen as low as possible. This implies that for the newly created component A', and any $r \in A' \cap R, b \in A' \cap B$, $\text{lca}_2(r, b) = \hat{u}$. □

3.3 Split

The next procedure will refine $\mathcal{P}$ so that the resulting partition does not overlap in $V_1[R \cup B]$. Since by Lemma 3, $\mathcal{P}$ is splittable, we can simply intersect each component with R, B and W, to achieve this property. However, we will need to be slightly more careful in order to achieve the approximation guarantee; in particular, we will sometimes need to perform what we call a Special-Split.

```
procedure Split(P, (R, B, W))
    for each multicolored component A do
        if A is tricolored, and there exists a tricolored triple in A that is compatible then
            Special-Split(A, P, (R, B, W))
        else
            P ← P \ {A} ∪ {A ∩ R, A ∩ B, A ∩ W} (where empty sets are not added)
        end if
    end for
end procedure
```

Remark. *Our analysis in Section 4 needs the* Special-Split*,* Find-Merge-Pair *and* Merge-Components *procedures only in one (of three) cases that will be described in Lemma 12. Without these procedures, it is trivial to see that the resulting partition is feasible, and we will see in Section 4 that the proof of the approximation ratio is quite simple in these cases.*

We now describe the property that the outcome partition of Split will have, which goes beyond merely being $R \cup B$-compatible and non-overlapping in $V_2 \cup V_1[R \cup B]$. We first define that property, and give necessary and sufficient conditions for a partition that does not overlap in V_2 to have this property.

Definition 7. Let $K \subseteq \mathcal{L}$. A partition $\mathcal{P}$ is *K-feasible* if for all $w \in \mathcal{L}$, $\mathcal{P}$ is $K \cup \{w\}$-compatible, and no two components in $\mathcal{P}$ overlap in $V_2 \cup V_1[K]$.

We will simply say $\mathcal{P}$ is *feasible* if it is $\mathcal{L}$-feasible, which we note does indeed coincide with the definition of a feasible solution to MAF. We make two additional remarks about the notion of K-feasibility:

- Being K-feasible requires something stronger than simply not overlapping in $V_2 \cup V_1[K]$ and K-compatibility. The stronger compatibility notion will be used in Lemma 7 to show that if $\mathcal{P}$ is $R \cup B$-feasible, then future iterations of the Red-Blue algorithm will not further subdivide (the partition induced on) the leaves in $R \cup B$. This is not necessarily true if $\mathcal{P}$ is only $R \cup B$-compatible and does not overlap in $V_2 \cup V_1[R \cup B]$.[1]
- If $u \in V_1$ is a root-of-infeasibility for $\mathcal{P}$, then $\mathcal{P}$ is not $\mathcal{L}(u)$-feasible. The converse is not true, however: if $\mathcal{P}$ contains a single component containing $\mathcal{L}(u)$ which is $\mathcal{L}(u)$-compatible, but this component contains both $w \in \mathcal{L} \setminus \mathcal{L}(u)$ such that $\mathcal{L}(u) \cup \{w\}$ is compatible, and $w' \in \mathcal{L} \setminus \mathcal{L}(u)$ such that $\mathcal{L}(u) \cup \{w'\}$ is not compatible, then $\mathcal{P}$ is *not* $\mathcal{L}(u)$-feasible, but u is not a root-of-infeasibility. The stronger notion of a u being a root-of-infeasibility versus not being $\mathcal{L}(u)$-feasible is needed when we prove the approximation guarantee in Section 4.

The following technical lemma gives conditions to check if a partition does not overlap in $V_1[R \cup B]$.

Lemma 5. *Let $\mathcal{P}$ be the partition and (R, B, W) be the coloring at the start of an iteration. Let $\mathcal{P}'$ be a refinement of $\mathcal{P}$ that does not overlap in V_2 and that is $R \cup B$-compatible. Then $\mathcal{P}'$ does not overlap in $V_1[R \cup B]$ if*

(i) $\mathcal{P}'$ has at most one multicolored component A^*;

(ii) if A^* exists, and $\mathrm{lca}_2(A^*) \prec \mathrm{lca}_2(R \cup B)$, then any node v' with $\mathrm{lca}_2(A^*) \prec v' \preceq \mathrm{lca}_2(R \cup B)$ is covered only by components in $\mathcal{P}'$ that are subsets of W, or that are also components of $\mathcal{P}$.

Proof. Suppose the conditions of the lemma hold for $\mathcal{P}'$. First, observe that by (i), $\mathcal{P}'$ contains at most one component covering $\mathrm{lca}_1(R \cup B)$. Suppose for a contradiction that $A', A'' \in \mathcal{P}'$ overlap in $V_1[R] \cup V_1[B]$.

Since $\mathrm{lca}_1(R \cup B)$ was chosen as a lowest root-of-infeasibility, $\mathrm{lca}_1(R)$ and $\mathrm{lca}_1(B)$ were not roots-of-infeasibility for $\mathcal{P}$. This implies that no two components of $\mathcal{P}$ overlap in $V_1[R] \cup V_1[B]$, so it must be the case that A' and A'' were both part of a single component in $\mathcal{P}$. Furthermore, $\mathcal{P}$ must have been R-compatible and B-compatible, so $(A' \cup A'') \cap R$ and $(A' \cup A'') \cap B$ are compatible sets. We will show that these facts imply that if A' and A'' overlap in $V_1[R]$ or $V_1[B]$, then they must overlap in $V_2[R]$ or $V_2[B]$ respectively, thus contradicting that $\mathcal{P}'$ does not overlap in V_2.

Let v be a lowest node in $V_1[R \cup B]$ such that $A' \cap \mathcal{L}(v) \neq \emptyset$ and $A'' \cap \mathcal{L}(v) \neq \emptyset$ (where we note that v exists since A', A'' overlap in some node in $V_1[R \cup B]$). Observe that a child of v cannot be in both $V_1[A']$ and $V_1[A'']$, as this contradicts the choice of v. Hence v can be in $V_1[A']$ and $V_1[A'']$ only if A' and A'' also contain leaves in $\mathcal{L} \setminus \mathcal{L}(v)$. Let x', x'' be in $A' \cap \mathcal{L}(v)$ and $A'' \cap \mathcal{L}(v)$ respectively, and choose y', y'' in $A' \setminus \mathcal{L}(v)$ and $A'' \setminus \mathcal{L}(v)$.

First, assume both A' and A'' are unicolored, and thus they are each either red or blue (since otherwise they would not overlap in $V_1[R] \cup V_1[B]$). Note that $\mathrm{lca}_1(x', x'') = v \prec \mathrm{lca}_1(x', x'', y')$ and similarly $\mathrm{lca}_1(x', x'') \prec \mathrm{lca}_1(x', x'', y'')$. Since $\{x', x'', y', y''\}$ is a compatible set, we must also have $\mathrm{lca}_2(x', x'') \prec \mathrm{lca}_2(x', x'', y')$ and $\mathrm{lca}_2(x', x'') \prec \mathrm{lca}_2(x', x'', y'')$. But then $\mathrm{lca}_2(x', x'')$ is on the path from x' to y' as well as on the path from x'' to y''. Hence, A' and A'' overlap in $\mathrm{lca}_2(x', x'') \in V_2$, contradicting that $\mathcal{P}'$ does not overlap in V_2.

Now, suppose A' is unicolored, and A'' is the unique multicolored component A^*, and A', A^* overlap in $V_1[R] \cup V_1[B]$. Without loss of generality $A' \subseteq R$. Then we still know that $\{x', x'', y'\}$ is compatible, and thus that $\mathrm{lca}_2(x', x'') \prec \mathrm{lca}_2(x', x'', y')$, so that $\mathrm{lca}_2(x', x'') \in V_2[A']$. Now, $x'' \in A^*$ is a descendant of $\mathrm{lca}_2(x', x'')$, so if A^* also has a leaf that is not a descendant of $\mathrm{lca}_2(x', x'')$ then A' and A^* overlap in

[1] For example, if $\mathcal{P}$ has only one multicolored component, which is $R \cup B$-compatible but not splittable, and $\mathcal{P}$ does not overlap in $V_2 \cup V_1[R \cup B]$, then the Red-Blue algorithm will further subdivide the partition induced on $R \cup B$.

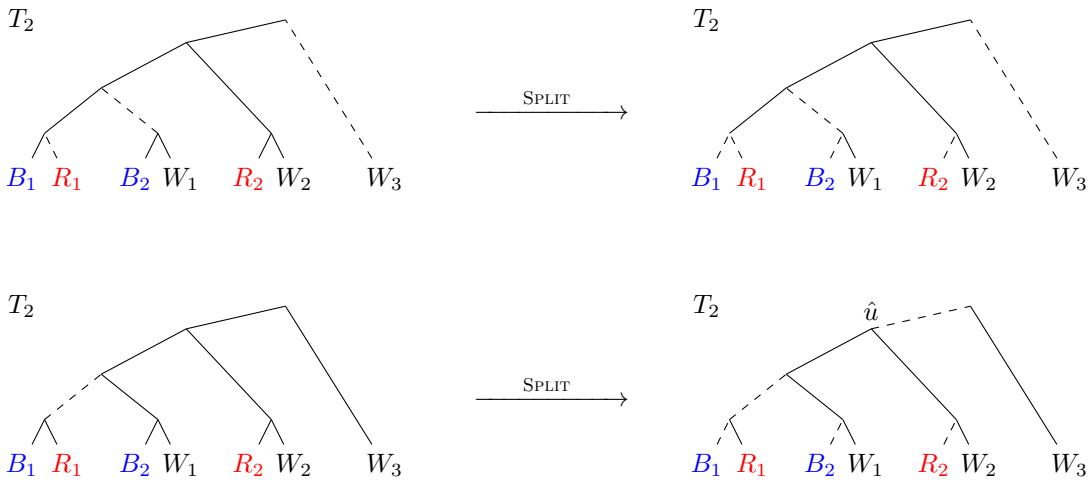


Figure 4: Two illustrations of Split$(\mathcal{P}, (R, B, W))$. In the top example $\mathcal{P} = \{\{R_1\}, \{B_2, W_1\}, \{B_1, R_2, W_2\}, \{W_3\}\}$ and Split$(\mathcal{P})$ would simply refine each set of $\mathcal{P}$ by intersecting it with the three color classes. The result is that every leaf is a singleton in $\mathcal{P}'$.
In the bottom example, $\mathcal{P} = \{\{B_1, R_1\}, \{B_2, W_1, R_2, W_2, W_3\}\}$. The set $A = \{B_2, W_1, R_2, W_2, W_3\}$ is tricolored and contains triple $\{B_2, R_2, W_3\}$ that is tricolored and compatible, but not every tricolored triple in A is compatible, e.g. $\{B_2, R_2, W_2\}$ is not compatible. In this case, the Special-Split replaces A by $\{\{B_2\}, \{R_2\}, \{W_1, W_2\}, \{W_3\}\}$.

$\text{lca}_2(x', x'')$, again contradicting that $\mathcal{P}'$ does not overlap in V_2. So it must be the case that $\text{lca}_2(A^*)$ is a descendant of $\text{lca}_2(x', x'')$. But then $V_2[A']$ intersects the path from $\text{lca}_2(A^*)$ to $\text{lca}_2(R \cup B)$. But we already showed above that $A' \cup A^*$ was part of a single component in $\mathcal{P}$, which implies that A' is not a component of $\mathcal{P}$, contradicting (ii). □

We now describe the Special-Split procedure. Recall that this is only called if A is tricolored, and there is at least one tricolored compatible triple in A.

procedure Special-Split$(A, \mathcal{P}, (R, B, W))$
  if every tricolored triple in A is compatible **then**
    $\mathcal{P} \leftarrow \mathcal{P} \setminus \{A\} \cup \{A \cap R, A \setminus R\}$.
  else
  ⋆ Let $\hat{u} = \text{lca}_2(A \cap (R \cup B))$.
    $\mathcal{P} \leftarrow \mathcal{P} \setminus \{A\} \cup \{A \setminus \mathcal{L}(\hat{u}), A' \cap R, A' \cap B, A' \cap W\}$ where $A' = A \cap \mathcal{L}(\hat{u})$.
  end if
end procedure

The next lemma states that this ensures the partition resulting after Split is $R \cup B$-feasible.

Lemma 6. *Let $\mathcal{P}'$ be the partition after* Split$(\mathcal{P}, (R, B, W))$. *Then $\mathcal{P}'$ is a refinement of $\mathcal{P}$ that is $R \cup B$-feasible. Moreover,* Special-Split *is applied to at most one component in each iteration of the Red-Blue algorithm, implying that $\mathcal{P}'$ has at most one multicolored component.*

Proof. The fact that $\mathcal{P}'$ does not overlap in V_2 follows from the fact that every component of $\mathcal{P}$ was splittable.

It is also easy to see that every component is $R \cup B \cup \{w\}$-compatible for all $w \in \mathcal{L}$: each component is either unicolored (and thus $R \cup B \cup \{w\}$-compatible by the fact that the partition is $R \cup B$-compatible by Lemma 1), or it is the result of a Special-Split on a component that was already $R \cup B \cup \{w\}$-compatible for all $w \in \mathcal{L}$ before the Special-Split.

It remains to show that no two components in $\mathcal{P}'$ overlap in $V_1[R \cup B]$. By Lemma 5, it suffices to show $\mathcal{P}'$ has at most one multicolored component, and that this component, if it exists, is a top component (recall Definition 6). Note that the only possible multicolored components of $\mathcal{P}'$ are bicolored components created by SPECIAL-SPLIT on a component $A \in \mathcal{P}$ that is tricolored and in which every tricolored triple is compatible. By property 4 of Lemma 4, the only tricolored components that have a compatible tricolored triple are top components, and by Observation 1, the partition at the start of the iteration had at most one tricolored component, and thus there is also at most one tricolored top component in $\mathcal{P}$. So $\mathcal{P}'$ has at most one multicolored component, which is a top component, and by Lemma 5, this implies $\mathcal{P}'$ does not overlap in $V_1[R \cup B]$. □

3.4 FIND-MERGE-PAIR and MERGE-COMPONENTS

The astute reader may have noted that the Red-Blue Algorithm sometimes increases the number of components by more than necessary to be $R \cup B$-feasible. For example, it follows from the arguments in the proof of the previous lemma that if there is a tricolored component in which every tricolored triple is compatible, then not further subdividing this component would also leave a partition that is $R \cup B$-feasible. FIND-MERGE-PAIR and MERGE-COMPONENTS aim to merge two components of the partition produced at the end of SPLIT, so that the partition with the merged components is still $R \cup B$-feasible. FIND-MERGE-PAIR thus looks for a pair of components that can be merged, by scanning the components of the current partition, and finding two leaves in $R \cup B$ that are in different sets of the partition now, but that were in the same component at the start of the current iteration.

procedure FIND-MERGE-PAIR(`pairslist`, $\mathcal{P}$, (R, B, W))
  if exists $x_1, x_2 \in R \cup B$ such that
    x_1 and x_2 were in the same component at the start of the current iteration,
    x_1 and x_2 are in distinct components A_1 and A_2 in $\mathcal{P}$, and
    $\mathcal{P} \setminus \{A_1, A_2\} \cup \{A_1 \cup A_2\}$ is $R \cup B$-feasible
  then
    `pairslist` $\leftarrow$ `pairslist` $\cup \{(x_1, x_2)\}$
  end if
end procedure

Although we could simply merge the components containing x_1 and x_2 for the pair found by FIND-MERGE-PAIR, we will not do so until the very end of the algorithm. The reason we keep such "superfluous" splits is because they will increase the objective value of the dual solution we use to prove the approximation guarantee of 2 (see Section 4). We "reverse" these superfluous splits (i.e., we will merge components) at the end of the algorithm; this is reminiscent of a "reverse delete" in approximation algorithms for network design [12].

procedure MERGE-COMPONENTS(`pairslist`, $\mathcal{P}$)
  for each pair (x_1, x_2) in `pairslist` **do**
    Let A_1 and A_2 be the sets in $\mathcal{P}$ containing x_1 and x_2, respectively.
    $\mathcal{P} \leftarrow \mathcal{P} \setminus \{A_1, A_2\} \cup \{A_1 \cup A_2\}$.
  end for
end procedure

The proof that we will be able to merge the components containing the pair of leaves identified by FIND-MERGE-PAIR at the end of the algorithm will rely on the fact that (i) because the partition is $R \cup B \cup \{w\}$-compatible for any $w \in \mathcal{L}$, merging the components containing the identified leaves $x_1, x_2 \in R \cup B$ cannot increase the number of incompatible triples contained in a component, and (ii) because the partition is $R \cup B$-feasible, future iterations of the algorithm will not further refine the partition induced on $R \cup B$. This

is the reason why we do not allow FIND-MERGE-PAIR to choose leaves in W (and only choosing leaves in $R \cup B$ is sufficient to prove the claimed approximation guarantee).

Lemma 7. *Let (R, B, W) be the coloring during some iteration of the Red-Blue algorithm, let $\mathcal{P}$ be the partition at the end of the pass, and let $x, x' \in R \cup B$. If x, x' are in the same component of $\mathcal{P}$, then x and x' are in the same component in any partition at any later point of the algorithm's execution.*

Proof. Let (R', B', W') be the coloring of the leaves in some later iteration of the algorithm, and suppose for a contradiction that the iteration with coloring (R', B', W') separates x and x' in different components. From a brief consideration of the algorithm, it is apparent that there must exist some $\hat{u} \in T_2$ such that $A \cap \mathcal{L}(\hat{u})$ is multicolored with respect to the coloring (R', B', W'), and $A \cap \mathcal{L}(\hat{u})$ contains precisely one of x, x'. By relabeling if needed, assume that $x \in A \cap \mathcal{L}(\hat{u})$ and $x' \in A \setminus \mathcal{L}(\hat{u})$, and let $w \in A \cap \mathcal{L}(\hat{u})$ be any leaf with a color different from x, and note that

$$\mathrm{lca}_2(x, w) \prec \hat{u} \prec \mathrm{lca}_2(x, x', w). \tag{1}$$

Since $\mathcal{P}$ is $R \cup B$-feasible, no $v \in V_1[R \cup B]$ is a root-of-infeasibility, and hence all leaves in $R \cup B$, and in particular x and x', must have the same color in the coloring (R', B', W'). Furthermore, if w has a different color than x and x' in (R', B', W'), then $w \notin R \cup B$, and thus $\mathrm{lca}_2(x, x') \prec \mathrm{lca}_2(x, x', w)$. But, since $\mathcal{P}$ is $R \cup B \cup \{w\}$-compatible, this implies that if w is in the same component as x and x' in (a refinement of) $\mathcal{P}$, then $\mathrm{lca}_2(x, x') \prec \mathrm{lca}_2(x, x', w)$, contradicting (1), because only one of $\mathrm{lca}_2(x, x')$ and $\mathrm{lca}_2(x, w)$ can be strictly below $\mathrm{lca}_2(x, x', w)$. □

3.5 Correctness of the algorithm

Theorem 8. *The Red-Blue Algorithm returns a feasible solution to MAF.*

Proof. Let k be the number of pairs in `pairslist`. We prove the theorem by induction on k. If $k = 0$, then the algorithm returns the partition obtained at the end of the while-loop, which is feasible by the fact that otherwise $\mathrm{lca}_1(\mathcal{L})$ would be a root-of-infeasibility.

If $k > 0$, observe that the final partition is the same irrespective of the order in which the pairs in `pairslist` are considered. We may thus assume without loss of generality that they are considered in the reverse order in which they were added to `pairslist`. Let $\mathcal{P}'$ be the partition after the components have been merged for all pairs on `pairslist`, except the pair (x_1, x_2) that was added to `pairslist` first. Let $\mathcal{P}$ be the partition at the moment when (x_1, x_2) was added to `pairslist`, and let R, B, W be the three color sets at that moment. Observe that $\mathcal{P}'$ is a refinement of $\mathcal{P}$, and that, by Lemma 7, $\mathcal{P}$ and $\mathcal{P}'$ induce the same partition of $R \cup B$.

Let A_1, A_2 be the components in $\mathcal{P}$ containing x_1, x_2 respectively. By the choice of x_1, x_2, $(A_1 \cup A_2)$ is $R \cup B \cup \{w\}$-compatible for any $w \in \mathcal{L}$, and does not overlap any component of $\mathcal{P} \setminus \{A_1, A_2\}$.

If A_1, A_2 are unicolored, they both contain leaves in $R \cup B$ only, and thus by Lemma 7, $\mathcal{P}'$ contains components A_1 and A_2 as well. Furthermore, in this case, the set $A_1 \cup A_2$ is a subset of $R \cup B$ and thus the fact that it is $R \cup B \cup \{w\}$-compatible for any $w \in \mathcal{L}$ implies it is compatible. The fact that $A_1 \cup A_2$ does not overlap any set $A \in \mathcal{P} \setminus \{A_1, A_2\}$ implies it also does not overlap any set $A' \in \mathcal{P}' \setminus \{A_1, A_2\}$, since $\mathcal{P}'$ is a refinement of $\mathcal{P}$.

If A_1 and A_2 are not both unicolored, observe that only one of A_1, A_2 is bicolored and contains leaves in $B \cup W$, since those are the only type of multicolored components after SPLIT, and $\mathcal{P}$ does not overlap in $V_1[R \cup B]$ so it can only have one multicolored component. Suppose without loss of generality that A_1 is unicolored and A_2 contains leaves in $B \cup W$. By Lemma 7, $\mathcal{P}'$ contains component A_1 and a component $A_2' \subseteq A_2$, where $A_2' \cap (R \cup B) = A_2 \cap (R \cup B)$.

We need to show that $A_1 \cup A_2'$ is compatible and does not overlap any component in $\mathcal{P}' \setminus \{A_1, A_2'\}$. For the latter, suppose in order to derive a contradiction that $A_1 \cup A_2'$ overlaps $A' \in \mathcal{P}' \setminus \{A_1, A_2'\}$. Observe that the only nodes in $V[A_1 \cup A_2']$ that are not in $V[A_1] \cup V[A_2']$ are in $V_2 \cup V_1[R \cup B]$, so the overlap must be on a node $v \in V_2 \cup V_1[R \cup B]$. Since $\mathcal{P}'$ is a refinement of $\mathcal{P}$, there must exist $A \in \mathcal{P}$ such that $A' \subset A$, and thus $A_1 \cup A_2'$

overlaps A in v as well. But then $A_1 \cup A_2$ also overlaps A in v contradicting that $\mathcal{P} \setminus \{A_1, A_2\} \cup \{A_1 \cup A_2\}$ is $R \cup B$-feasible.

To show that $A_1 \cup A_2'$ is compatible, note that A_2' is compatible, and that $A_1 \cup A_2' \subset A_1 \cup A_2$ is $R \cup B \cup \{w\}$-compatible for any $w \in \mathcal{L}$. So to show that $A_1 \cup A_2'$ is compatible, it suffices to consider $x, w, w' \in A_1 \cup A_2'$ with $x \in A_1$ and $w, w' \in A_2' \cap W$.

Fix any $x_B \in A_2' \cap B$. Note that $\mathrm{lca}_i(x_B, w) = \mathrm{lca}_i(x, x_B, w) = \mathrm{lca}_i(x, w)$ for $i = 1, 2$, because $\mathrm{lca}_i(x, x_B) \prec \mathrm{lca}_i(x, x_B, w)$ because $A_1 \cup A_2$ is $R \cup B \cup \{w\}$-compatible. Therefore $\{x, w, w'\}$ is compatible exactly when $\{x_B, w, w'\}$ is compatible. We conclude that $A_1 \cup A_2'$ is compatible because A_2' is compatible. □

4 Proof of the approximation guarantee

We showed in the previous section that the Red-Blue algorithm returns a feasible solution $\mathcal{P}$. In order to prove that our algorithm achieves an approximation guarantee of 2, we will use linear programming duality.

4.1 The linear programming relaxation

Introduce a variable x_L for every compatible set $L \in \mathcal{C}$, where in an integral solution, $x_L = 1$ indicates that the tree with leaf set L forms part of the solution to MAF. The constraints ensure that in an integral solution, $\{L : x_L = 1\}$ is a partition, and that $V[L] \cap V[L'] = \emptyset$ for two distinct sets L, L' with $x_L = x_{L'} = 1$. The objective encodes the size of the partition minus 1.

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

In fact, it will be convenient for our analysis to expand the first set of constraints to contain a constraint for every (not necessarily compatible) set of leaves A, stating that every such set must be intersected by at least one tree in the chosen MAF solution. All these constraints are clearly already implied by the constraints for A a singleton, already present in (LP), but they provide us a more expressive dual.

$$\begin{array}{lll} \text{minimize} & \sum_{L \in \mathcal{C}} x_L - 1, & \\ \text{s.t.} & \sum_{L : A \cap L \neq \emptyset} x_L \geq 1 & \forall A \subseteq \mathcal{L}, A \neq \emptyset \\ & \sum_{L : v \in V[L]} x_L \leq 1 & \forall v \in V \setminus \mathcal{L}, \\ & x_L \geq 0 & \forall L \in \mathcal{C}. \end{array} \tag{LP$'$}$$

The dual of (LP$'$) is

$$\begin{array}{lll} \max & \sum_{v \in V \setminus \mathcal{L}} y_v + \sum_{A \subseteq \mathcal{L}} z_A - 1, & \\ \text{s.t.} & \sum_{v \in V[L] \setminus \mathcal{L}} y_v + \sum_{A : A \cap L \neq \emptyset} z_A \leq 1 & \forall L \in \mathcal{C}, \\ & y_v \leq 0 & \forall v \in V \setminus \mathcal{L}, \\ & z_A \geq 0 & \forall A \subseteq \mathcal{L}. \end{array} \tag{D$'$}$$

We will refer to the left-hand side of the first family of constraints, i.e., $\sum_{v \in V[L] \setminus \mathcal{L}} y_v + \sum_{A : A \cap L \neq \emptyset} z_A$, as the *load* on set L, and denote it by $\mathrm{load}_{(y,z)}(L)$. By weak duality, we have that the objective value of any feasible dual solution provides a lower bound on the objective value of any feasible solution to (LP), and hence also on the optimal value of any feasible solution to MAF. Hence, in order to prove that an agreement forest that has $|\mathcal{P}|$ components is a 2-approximation, it suffices to find a feasible dual solution with objective value $\frac{1}{2}(|\mathcal{P}| - 1)$, i.e., for every new component created by the algorithm, the dual objective value should increase by $\frac{1}{2}$ (on average).

4.2 The dual solution

The dual solution maintained is as follows. Throughout the main loop of the algorithm, $z_A = 1$ if and only if A is a component in $\mathcal{P}$. In the last part of the algorithm, when we merge components according to `pairslist`, we do not update the dual solution; these operations affect the primal solution (i.e., $\mathcal{P}$) only.

Initially, $y_v = 0$ for all $v \in V_1 \cup V_2$. At the start of each iteration, we decrease y_u by 1, where $u = \text{lca}_1(R \cup B)$. Whenever in the algorithm we choose a component A and a node $\hat{u} \in V_2[A]$, and separate the component A into $A \cap \mathcal{L}(\hat{u})$ and $A \setminus \mathcal{L}(\hat{u})$, we decrease $y_{\hat{u}}$ by 1. To be precise this happens in MAKE-$R \cup B$-COMPATIBLE, MAKE-SPLITTABLE and in one case in SPECIAL-SPLIT (where we actually further refine $A \cup \mathcal{L}(\hat{u})$). The lines where such nodes are chosen are indicated by $\star$ in the description of the algorithm and the procedures it contains.

Lemma 9. *The dual solution maintained by the algorithm is feasible.*

Proof. We prove the lemma by induction on the number of iterations. Initially, $z_A = 0$ for all $A \neq \mathcal{L}$ and $z_{\mathcal{L}} = 1$ and hence every compatible set L has a load of 1.

At the start of an iteration, we decrease $y_{\text{lca}_1(R \cup B)}$ by 1, thus decreasing the load by 1 on any multicolored compatible set L. We show that the remainder of the iteration increases the load by at most 1 on a multicolored compatible set and that it does not increase the load on any unicolored compatible set.

First, observe that MAKE-$R \cup B$-COMPATIBLE and MAKE-SPLITTABLE do not increase the load on any set: Separating A into $A \cap \mathcal{L}(\hat{u})$ and $A \setminus \mathcal{L}(\hat{u})$ increases the load on sets L that intersect both $A \cap \mathcal{L}(\hat{u})$ and $A \setminus \mathcal{L}(\hat{u})$, since z_A gets decreased from 1 to 0, and $z_{A \cap \mathcal{L}(\hat{u})}$ and $z_{A \setminus \mathcal{L}(\hat{u})}$ increase from 0 to 1. However, in this case $\hat{u} \in V[L]$, and thus decreasing $y_{\hat{u}}$ by 1 ensures that the load on L does not increase.

To analyze the effect of SPLIT, we use the following two claims.

Claim 10. *In the procedure* SPLIT$(\mathcal{P}, (R, B, W))$ *the load on any compatible set L is increased by at most the number of components $A \in \mathcal{P}$ such that $L \cap A$ is multicolored.*

Proof of Claim: If the load on L is increased because SPLIT splits a bicolored component A into two unicolored components, then L must intersect both new components, so $L \cap A$ is bicolored (and thus multicolored).

Consider the case where the load on L is increased because a tricolored component A is split into $A \cap R$, $A \cap B$ and $A \cap W$. This split happens when *all* tricolored triples in A are incompatible. Therefore $L \cap A$ cannot be tricolored, and the load is increased by 1. And again $L \cap A$ is multicolored.

Suppose the load on L is increased because SPECIAL-SPLIT$(A, \mathcal{P}, (R, B, W))$ is executed for a component A. We consider the two cases. Either A is split into two components, one of which contains all red leaves in A. The load on a set L thus increases by 1 if $L \cap A$ is multicolored and $L \cap A \cap R \neq \emptyset$ and by 0 otherwise. If A is split into four components; we think of this as first splitting A into $A \cap \mathcal{L}(\hat{u})$ and $A \setminus \mathcal{L}(\hat{u})$, and then splitting $A \cap \mathcal{L}(\hat{u})$ by intersecting with R, B and W. Since $y_{\hat{u}}$ is decreased by 1, splitting A into $A \cap \mathcal{L}(\hat{u})$ and $A \setminus \mathcal{L}(\hat{u})$ does not affect the load on any set L. Splitting $A \cap \mathcal{L}(\hat{u})$ by intersecting with R, B, W increases the load on L by 1 if $L \cap A \cap \mathcal{L}(\hat{u})$ is bicolored and by 2 if it is tricolored; note however that the latter is impossible, since $\hat{u} = \text{lca}_2(A \cap (R \cup B))$, so any tricolored triple in $A \cap \mathcal{L}(\hat{u})$ must be incompatible. So the load on L again increases by at most 1 if $A \cap L$ is multicolored. ◇

Claim 11. *If L is compatible, and A and A' do not overlap in V_2, then $L \cap A$ and $L \cap A'$ cannot both be multicolored.*

Proof of Claim: Since $V_2[A]$ and $V_2[A']$ are disjoint, it must be the case that $\text{lca}_2(x, y) \prec \text{lca}_2(x, y, x')$ for all $x, y \in A$ and $x' \in A'$, or $\text{lca}_2(x', y') \prec \text{lca}_2(x, x', y')$ for all $x', y' \in A'$ and $x \in A$ (or both). Hence, if $L \cap A$ and $L \cap A'$ are both multicolored sets, then there exists $x, y, x', y' \in L$ where x, y have different colors, x', y' have different colors, $\text{lca}_2(x, y) \prec \text{lca}_2(x, y, x')$, and $\text{lca}_2(x, y) \prec \text{lca}_2(x, y, y')$. We claim this implies $\{x, y, x', y'\}$ is incompatible.

Clearly one of x, y has the same color as one of x', y'. Suppose first that either red or blue is a shared color. Without loss of generality, we may assume that x and x' are both red; y is then either blue or white. x and x' being red implies $\text{lca}_1(x, x') \prec \text{lca}_1(x, y, x')$, which, since $\text{lca}_2(x, y) \prec \text{lca}_2(x, y, x')$, shows that $\{x, x', y\}$ is an incompatible triple.

So suppose that white is the only shared color, and that x and x' are both white. Then either y is red and y' is blue, or vice versa. This implies $\mathrm{lca}_1(y, y') \prec \mathrm{lca}_1(x, y, y')$, and so, since $\mathrm{lca}_2(x, y) \prec \mathrm{lca}_2(x, y, y')$, this implies $\{x, y, y'\}$ is an incompatible triple. $\diamond$

It follows immediately from the two claims that Split increases the load by at most 1 on any multicolored compatible set and that it does not increase the load on any unicolored set, which completes the proof of the lemma. $\square$

4.3 The primal and dual objective values

Let $\mathcal{P}$, `pairslist` be the partition and `pairslist` at the end of an iteration, and let $D = \sum_{v \in V \setminus \mathcal{L}} y_v + |\mathcal{P}| - 1$ be the objective value of the dual solution at this time. In this section, we show that our algorithm maintains the invariant that

$$2D \geq (|\mathcal{P}| - 1 - |\texttt{pairslist}|)\,. \tag{2}$$

Observe that the approximation guarantee immediately follows from this inequality, since the objective value of the algorithm's solution is $\mathcal{P} - 1 - |\texttt{pairslist}|$ (where $\mathcal{P}$, `pairslist` are the partition and `pairslist` at the end of the final iteration), and by weak duality D gives a lower bound on the optimal value of the MAF instance.

To prove that the algorithm maintains the invariant, we will show that a given iteration increases the left-hand side of (2) by at least as much as the right-hand side. We let ΔD be the change in the dual objective during the iteration and ΔP be the increase in the number of components less the number of pairs added to `pairslist` (either 0 or 1) during the current iteration.

Since at the start of the algorithm, the partition consists of exactly one component, and $y_v = 0$ for all $v \in V \setminus \mathcal{L}$, (2) holds before the first iteration. So to show (2), it suffices to show that

$$2\Delta D \geq \Delta P \tag{3}$$

for any iteration.

In what follows, we use the following to refer to the state of the partition at various points in the current iteration: $\mathcal{P}^{(0)}$ at the start; $\mathcal{P}^{(1)}$ after Make-$R \cup B$-compatible; $\mathcal{P}^{(2)}$ after Make-Splittable; and $\mathcal{P}^{(3)}$ after Split.

We begin by showing that the coloring (R, B, W) and the partition $\mathcal{P}^{(0)}$ satisfies the conditions of one of three cases.

Lemma 12. *Given an infeasible partition $\mathcal{P}^{(0)}$ that does not overlap in T_2, let $u \in V_1$ be a lowest root-of-infeasibility, and let u_ℓ and u_r be u's children in T_1. Let $R = \mathcal{L}(u_r), B = \mathcal{L}(u_\ell)$, and $W = \mathcal{L} \setminus (R \cup B)$. Then $\mathcal{P}^{(0)}$ is R-compatible and B-compatible and satisfies exactly one of the following three additional properties:*

Case 1. *$\mathcal{P}^{(0)}$ has exactly one multicolored component, say A_0, where A_0 is tricolored, not $R \cup B$-compatible, and there exists $x_W \in A_0 \setminus \mathcal{L}(\mathrm{lca}_2(A_0 \cap (R \cup B)))$.*

Case 2. *$\mathcal{P}^{(0)}$ has exactly two multicolored components, say A_B, A_R, where $A_B \cap R = \emptyset$ and $A_R \cap B = \emptyset$.*

Case 3. *$\mathcal{P}^{(0)}$ has exactly one multicolored component, say A_0, where A_0 is tricolored, $R \cup B$-compatible and A_0 contains no compatible tricolored triple.*

We will see in the proof below that Case 1, 2 and 3 correspond to a lowest root-of-infeasibility satisfying (a), (b) and (c) respectively in Definition 4. We refer the reader to Figure 1 for an illustration of the three cases.

Proof. Observe that if $\mathcal{P}^{(0)}$ is infeasible, then the root of T_1, i.e., $\mathrm{lca}_1(\mathcal{L})$ is a root-of-infeasibility, and that any $v \in \mathcal{L}$ is not a root-of-infeasibility. Hence, u is well-defined and R and B are non-empty. Note that $\mathcal{P}^{(0)}$ is R-compatible and B-compatible, since u's children are not root-of-infeasibility.

We will show that if u satisfies condition (a) in the definition of root-of-infeasibility, then the conditions of Case 1 are satisfied, if (b) holds, the conditions of Case 2 are satisfied, and if (c) holds, but not (a), then the conditions of Case 3 are satisfied.

We start with (b). Observe that, because $\mathcal{P}^{(0)}$ is R-compatible and B-compatible, there must be at least two multicolored components if (b) holds. If there are two multicolored components, both containing, say, red leaves, then they overlap in $u_r = \mathrm{lca}_1(R)$, which implies u_r is a root-of-infeasibility, contradicting the choice of u. Similarly, there is at most one multicolored component containing blue leaves. Hence, the conditions of Case 2 are satisfied.

If (b) does not hold, then there is at most one multicolored component; the conditions in (a) and (c) both imply there must be at least one (and thus there is exactly one), which we will call A_0. We let $R_0 = R \cap A, B_0 = B \cap A$ and $\hat{u} = \mathrm{lca}_2(R_0 \cup B_0)$ (where we stress that $\hat{u}$ is a node in V_2, whereas u is a node in V_1).

If (a) holds, then A_0 is not $R \cup B$-compatible, and thus $R_0 \neq \emptyset, B_0 \neq \emptyset$. Assume, in order to derive a contradiction, that $A_0 \subseteq \mathcal{L}(\hat{u})$. Observe that, because A_0 is not $R \cup B$-compatible, $\mathrm{lca}_2(R_0) = \hat{u}$ or $\mathrm{lca}_2(B_0) = \hat{u}$. Suppose the former without loss of generality. But then $\mathrm{lca}_1(R_0)$ is a root-of-infeasibility satisfying (c) which is a descendant of u thus contradicting the choice of u: $A_0 \setminus \mathcal{L}(\mathrm{lca}_1(R_0)) = A_0 \setminus R_0 \supseteq B_0 \neq \emptyset$, and $R_0 \cup \{w\}$ is incompatible for any $w \in A_0 \setminus R_0$.

Suppose now (c) holds, but not (a), i.e., $\mathcal{P}^{(0)}$ is $R \cup B$-compatible. Thus A_0 must be $R \cup B$-compatible and $A_0 \setminus (R_0 \cup B_0) \neq \emptyset$. Assume without loss of generality that $R_0 \neq \emptyset$, and note that $\mathrm{lca}_1(R_0)$ is a descendant of u, and that, if $B_0 = \emptyset$, then (c) holds for $\mathrm{lca}_1(R_0)$, contradicting the choice of u. Hence, A_0 is tricolored. Since A_0 is $R \cup B$-compatible, $\mathrm{lca}_2(R_0)$ and $\mathrm{lca}_2(B_0)$ must be descendants of the distinct children of $\hat{u}$, or the children itself. Furthermore, the fact that $\mathcal{P}^{(0)}$ is not $R \cup B \cup \{w\}$-compatible for *any* $w \in A_0 \setminus (R \cup B)$ implies that all white leaves are descendants of $\hat{u}$ as well, and thus any tricolored triple of leaves in A_0 is incompatible. Thus, if (c) holds but not (a), the conditions for Case 3 are satisfied. □

Recall that the coloring is defined only at the start of the iteration. The lemma ensures that the partitions during the iteration always have either one (in Case 1 and 3) or two (in Case 2) top components.

For Cases 2 and 3, the analysis is quite simple.

Proposition 13. *Let the initial partition $\mathcal{P}^{(0)}$ and coloring (R, B, W) satisfy the conditions of Case 2 or 3 in Lemma 12. Then $2\Delta D \geq \Delta P$.*

Proof. We first make two observations that apply in Case 2 and 3: (i) $\mathcal{P}^{(0)}$ is already $R \cup B$-compatible, so $\mathcal{P}^{(1)} = \mathcal{P}^{(0)}$, and (ii) Split$(\mathcal{P}^{(2)}, (R, B, W))$ will not perform any Special-Split, by property 4 in Lemma 4 and because the top component has no tricolored triple that is compatible (since we are in Case 2 or 3).

These two observations imply that

$$|\mathcal{P}^{(3)}| - |\mathcal{P}^{(2)}| = |\mathcal{P}^{(2)}| - |\mathcal{P}^{(0)}| + 2. \tag{4}$$

To see this, note that, since no Special-Split is performed, $|\mathcal{P}^{(3)}| - |\mathcal{P}^{(2)}|$ is equal to the number of bicolored components in $\mathcal{P}^{(2)}$ plus twice the number of tricolored components in $\mathcal{P}^{(2)}$. Since $\mathcal{P}^{(1)} = \mathcal{P}^{(0)}$, and using properties 2 and 3 of Lemma 4, $\mathcal{P}^{(2)}$ has $|\mathcal{P}^{(2)}| - |\mathcal{P}^{(0)}|$ more multicolored components than $\mathcal{P}^{(0)}$, and the same number of tricolored components as $\mathcal{P}^{(0)}$. So in Case 2, $\mathcal{P}^{(2)}$ has $|\mathcal{P}^{(2)}| - |\mathcal{P}^{(0)}| + 2$ bicolored components and zero tricolored components, and in Case 3, $\mathcal{P}^{(2)}$ has $|\mathcal{P}^{(2)}| - |\mathcal{P}^{(0)}|$ bicolored components plus one tricolored component, thus indeed (4) holds.

In addition, we note that

$$\Delta D = |\mathcal{P}^{(3)}| - |\mathcal{P}^{(2)}| - 1. \tag{5}$$

To see this, note that at the start of the iteration, the dual objective value is reduced by 1 when y_u is decreased by 1 for $u = \mathrm{lca}_1(R \cup B)$. Make-splittable does not change the dual objective value, because, even though $|\mathcal{P}|$ increases by 1 every time the number of components increases by 1, $\sum_v y_v$ decreases by 1 as well. Finally, since Split will not perform any Special-Split, the increase in the dual objective value due to Split is equal to the increase in the number of components due to Split, which is $|\mathcal{P}^{(3)}| - |\mathcal{P}^{(2)}|$.

Note that the size of `pairslist` may increase but will never decrease, and thus

$$\begin{aligned}\Delta P &\le |\mathcal{P}^{(3)}| - |\mathcal{P}^{(2)}| + |\mathcal{P}^{(2)}| - |\mathcal{P}^{(0)}| \\ &= 2\left(|\mathcal{P}^{(3)}| - |\mathcal{P}^{(2)}|\right) - 2 && \text{by (4)} \\ &= 2\Delta D && \text{by (5).}\end{aligned}$$

□

We now prove a similar proposition for Case 1, the proof of which is more involved.

Proposition 14. *Suppose the initial partition $\mathcal{P}^{(0)}$ and coloring (R, B, W) satisfy the conditions of Case 1 in Lemma 12. Then $2\Delta D \ge \Delta P$.*

Proof. In Case 1, we start with one tricolored component A_0, which is not $R \cup B$-compatible. Let x_W be a white leaf in A_0 that is not a descendant of $\mathrm{lca}_2(A_0 \cap (R \cup B))$, which exists by the definition of Case 1. By property 6 in Lemma 4, x_W is also contained in the top component of $\mathcal{P}^{(3)}$, and by property 2 in Lemma 4, the top component is multicolored. Therefore, the top component of $\mathcal{P}^{(2)}$ is either bicolored, or it is tricolored and a Special-Split is performed on the top component.

Let χ be an indicator variable that is 1 if the top component in $\mathcal{P}^{(2)}$ is tricolored and has a tricolored triple that is incompatible. Let t be the number of tricolored components in $\mathcal{P}^{(2)}$ that are not top components. We claim that

$$|\mathcal{P}^{(3)}| - |\mathcal{P}^{(2)}| = |\mathcal{P}^{(2)}| - |\mathcal{P}^{(0)}| + 1 + 2\chi + t. \tag{6}$$

Indeed, if $\chi = 0$, the top component is divided into two components by Split, and if $\chi = 1$ it is subdivided into four components. Thus splitting the top component increases the number of components by $1 + 2\chi$. By property 2 of Lemma 4, $\mathcal{P}^{(2)}$ has $|\mathcal{P}^{(2)}| - |\mathcal{P}^{(0)}|$ multicolored components that are not top components, and by property 4, each of the tricolored components that are not top components do not require a Special-Split and are thus subdivided into three components by Split. Hence, splitting the components that are not top components increases the number of components by $|\mathcal{P}^{(2)}| - |\mathcal{P}^{(0)}| + t$.

Next, we analyze the increase in the dual objective. We claim that

$$\Delta D = |\mathcal{P}^{(3)}| - |\mathcal{P}^{(2)}| - 1 - \chi. \tag{7}$$

To see this, note that the dual objective is decreased by 1 when we decrease $y_{\mathrm{lca}_1(R\cup B)}$ by 1 at the start of the iteration. The dual objective is not affected by Make-$R \cup B$-compatible and Make-Splittable. Finally, if $\chi = 0$, the increase in the dual objective due to Split is equal to the increase in the number of components $|\mathcal{P}^{(3)}| - |\mathcal{P}^{(2)}|$. If $\chi = 1$, the same holds, but Special-Split on the top component also decreases $y_{\hat{u}_0}$ by 1.

So we get that

$$\begin{aligned}|\mathcal{P}^{(3)}| - |\mathcal{P}^{(0)}| &= |\mathcal{P}^{(3)}| - |\mathcal{P}^{(2)}| + |\mathcal{P}^{(2)}| - |\mathcal{P}^{(0)}| \\ &= 2\left(|\mathcal{P}^{(3)}| - |\mathcal{P}^{(2)}|\right) - 1 - 2\chi - t && \text{by (6)} \\ &= 2\Delta D + 1 - t && \text{by (7).}\end{aligned}$$

Recall that ΔP is equal to $|\mathcal{P}^{(3)}| - |\mathcal{P}^{(0)}|$ minus the number of pairs added to `pairslist` in the current iteration. Hence, to conclude that $\Delta P \le 2\Delta D$, we need to show that if $t = 0$, then a pair is added to `pairslist` by Find-Merge-Pair.

We will say that a component A is able to *reach* $\hat{u}$ if $\hat{u} \in V_2[A]$ or if $\mathrm{lca}_2(A) \prec \hat{u}$ and all intermediate nodes on the path from $\mathrm{lca}_2(A)$ to $\hat{u}$ are not covered by any component in $\mathcal{P}^{(3)}$. The following lemma (which is actually valid in general, and not only for Case 1) enumerates precisely the situations when a merge is possible.

Lemma 15. *Let $A_0 \in \mathcal{P}^{(0)}$, and let $\mathcal{Q}$ denote the set of components in $\mathcal{P}^{(3)}$ that are contained in A_0. Then there exist a pair of elements in A_0 that can be added to* `pairslist` *if and only if at least one of the following is true:*

(a) *$\mathcal{Q}$ contains a bicolored component.*

(b) *There is a node $\hat{u} \in V_2$ that can be reached by two red components or two blue components in $\mathcal{Q}$.*

(c) *There is a node $\hat{u} \in V_2$ that can be reached by a red and a blue component in $\mathcal{Q}$, but is not covered by these components. Furthermore, the node $\hat{u}$ must satisfy that the nodes on the path from $\hat{u}$ to $\mathrm{lca}_2(A_0)$ are not covered by any red or blue component in $\mathcal{Q}$.*

Proof. By the definitions of Split and Special-Split, a multicolored component in $\mathcal{P}^{(3)}$ must be a top component with blue and white leaves created by applying Special-Split to a tricolored component. By Lemma 6, there is at most one such multicolored component; if it exists, call it A^*.

(a) If A^* exists, then let $A \cup A^*$ be the tricolored component from which Special-Split formed a red component A and the bicolored component A^*. Then we can merge A and A^* to obtain a new partition that is $R \cup B$-feasible: it is clear that undoing the Special-Split yields a partition that does not overlap any other component of $\mathcal{P}$ in V_2. The new component $A \cup A^*$ is $R \cup B \cup \{w\}$-compatible since $A \cup A^*$ is $R \cup B$-compatible and every tricolored triple in $A \cup A^*$ is compatible. Since the new unique bicolored component is a top component, by Lemma 5, the new partition also does not overlap in $V_1[R \cup B]$.

(b) If $\mathcal{Q}$ does not contain a bicolored component A^*, suppose $A, A' \in \mathcal{Q}$ are distinct red components in $\mathcal{Q}$ so that A and A' can both reach the same node $\hat{u}$ in V_2. Then merging A and A' gives a new partition that does not overlap in V_2, and which has no multicolored components. Since $A_0 \cap R$ is compatible, so is $A \cup A'$. By Lemma 5 the new partition does not overlap in $V_1[R \cup B]$. Hence, merging A and A' gives a new partition that is $R \cup B$-feasible.

The same applies if A and A' are both blue components in $\mathcal{Q}$.

(c) If $\mathcal{Q}$ does not contain a bicolored component A^*, suppose there exist $A, A' \in \mathcal{Q}$ with A red and A' blue such that (i) there exists $\hat{u} \in V_2 \setminus (V_2[A] \cup V_2[A'])$ that can be reached by both A and A'; and (ii) the nodes on the path from $\hat{u}$ to $\mathrm{lca}_2(A_0)$ are not in $V_2[A'']$ for any $A'' \not\subseteq W$. Then merging A and A' gives a new partition that does not overlap in V_2 and the new component $A \cup A'$, is $R \cup B$-compatible by (i). Thus the new partition is $R \cup B \cup \{w\}$-compatible for any $w \in \mathcal{L}$. The unique bicolored component $A \cup A'$ in this new partition satisfies that any node on the path from $\hat{u} = \mathrm{lca}_2(A \cup A')$ to $\mathrm{lca}_2(A_0)$ is not covered by a component that is not white. The components of the partition that overlap a node on the path from $\mathrm{lca}_2(A_0)$ to $\mathrm{lca}_2(R \cup B)$ were not changed in the current iteration. Therefore, by Lemma 5, the new partition does not overlap in $V_1[R \cup B]$. Hence, merging A and A' gives a new partition that is $R \cup B$-feasible.

We note that the above three cases encompass all possible merge opportunities within $\mathcal{Q}$. If two components cannot reach the same node $\hat{u} \in V_2$, then merging them gives a partition that overlaps in V_2. If a red and blue component A and A' can reach the same node $\hat{u} \in V_2$ and this node is covered by either A or A', then $A \cup A'$ is not $R \cup B$-compatible. And if a red and blue component A and A' can reach a node $\hat{u} \in V_2$ that is not in $V_2[A] \cup V_2[A']$, but some node on the path from $\hat{u}$ to $\mathrm{lca}_2(A_0)$ is covered by a component $A'' \in \mathcal{Q}$ that is red or blue, then $A \cup A'$ will overlap A'' in $V_1[R]$ or $V_1[B]$. To see this, assume A'' is red (the blue case is analogous) and let $\hat{v}$ be the node in $V_2[A'']$ closest to $\hat{u}$ on the path from $\hat{u}$ to $\mathrm{lca}_2(A_0)$. Then $\hat{v} = \mathrm{lca}_2(A \cup (A'' \cap \mathcal{L}(\hat{v})) \prec \mathrm{lca}_2(A'')$, and since $A \cup A''$ are compatible in R, we should also have $\mathrm{lca}_1(A \cup (A'' \cap \mathcal{L}(\hat{v})) \prec \mathrm{lca}_1(A'')$. Thus A'' and A overlap on a node on the path from $\mathrm{lca}_1(A)$ to $\mathrm{lca}_1(R \cup B)$. □

We are now ready to complete the proof of Proposition 14, by showing that if $t = 0$, then at least one of (a), (b) and (c) in Lemma 15 holds for $\mathcal{P}^{(3)}$.

Suppose (a) does not hold, i.e., $\mathcal{P}^{(3)}$ has only unicolored components. Let $\hat{u}$ be the last node chosen in Make-$R \cup B$-compatible to subdivide the top component. The existence of $\hat{u}$ follows since $\mathcal{P}^{(1)} \neq \mathcal{P}^{(0)}$ as we observed above. Let $A \subset \mathcal{L}(\hat{u})$ be the non-top component added to the partition at this point. Since $t = 0$, we have that A is bicolored by property 2 in Lemma 4. Split will split A into $A \cap R$ and $A \cap B$, and

by property 5 in Lemma 4, node $\hat{u}$ itself is not covered by any component of $\mathcal{P}^{(3)}$ and it can be reached by red component $A \cap R$ and blue component $A \cap B$. So if there is no node on the path in T_2 from $\hat{u}$ to $\mathrm{lca}_2(A_0)$ that is covered by a red or blue component in $\mathcal{P}^{(3)}$, then $A \cap R$ and $A \cap B$ give a pair that can be added to `pairslist` according to (c).

So suppose that there is a node on the path from $\hat{u}$ to $\mathrm{lca}_2(A_0)$ that is covered by a (say) red component A_R in $\mathcal{P}^{(3)}$; without loss of generality $\hat{v}$ is the node closest to $\hat{u}$ such that $\hat{v} \in V_2[A_R]$ for some red component A_R. If $A \cap R$ can reach $\hat{v}$ then $\hat{v}$ can be reached by two red components, so we can add a pair to `pairslist` according to (b).

Otherwise, let $\hat{w}$ be the node closest to $\hat{v}$ on the path from $\hat{u}$ to $\hat{v}$ that is covered by a white component A_W. By definition of $\hat{w}$, all nodes on the path from $\hat{w}$ to $\hat{v}$ are not covered by any component in $\mathcal{P}^{(3)}$. Observe that

- A_W and A_R must have been part of the top component of $\mathcal{P}^{(1)}$, by definition of $\hat{u}$.
- A_W and A_R cannot have been part of the top component of $\mathcal{P}^{(2)}$: as we observed above, the top component of $\mathcal{P}^{(2)}$ contains a white leaf x_W that is not a descendant of $\mathrm{lca}_2(A_0 \cap (R \cup B))$, so $A_W \cup \{x_W\}$ overlaps $\hat{v}$, and since A_R also overlaps $\hat{v}$, the top component of $\mathcal{P}^{(2)}$ would not be splittable if it contained $A_R \cup A_W$, contradicting Lemma 3.
- A_W was part of non-top component A'' in $\mathcal{P}^{(2)}$, which contained red and white leaves: by the second observation, A_W must have been part of a non-top component A'', which was multicolored by property 2 of Lemma 4. Since A'' and A_R were part of the same component of $\mathcal{P}^{(1)}$ by the first observation, $A'' \cup A_R$ must be $R \cup B$-compatible by Lemma 1. This implies A'' does not contain any blue leaves, since otherwise such a blue leaf x_B, and a red leaf $x_R \in A_R \cap \mathcal{L}(\hat{v})$ and a red leaf $y_R \in A_R \setminus \mathcal{L}(\hat{v})$ would have $\mathrm{lca}_2(x_B, x_R) = \hat{v} \prec \mathrm{lca}_2(x_B, x_R, y_R)$, and thus the triple would be incompatible, contradicting that $A'' \cup A_R$ is $R \cup B$-compatible.

Thus, $A'' = A_W \cup (A'' \cap R)$, and by property 5 in Lemma 4, the component $A'' \cap R$ in $\mathcal{P}^{(3)}$ can reach $\mathrm{lca}_2(A'')$. Since $\hat{w} \prec \mathrm{lca}_2(A'')$, $\mathrm{lca}_2(A'')$ is on the path from $\hat{w}$ to $\hat{v}$. But then $A'' \cap R$ can reach $\hat{v}$, so A_R and $A'' \cap R$ give a pair to be added to `pairslist` according to (b). □

5 A compact formulation of the LP

Here we give a compact formulation for (LP). This shows that it can be optimized efficiently. While this is not needed in our algorithm, it is possible that an LP-rounding based algorithm could achieve a better approximation guarantee, in which case this formulation will be of use. Moreover, the LP explicitly encodes the structure of compatible sets in a way that (LP) does not; we believe this may provide additional structural insights in the future.

We remark that (LP) can also be shown to be polynomially solvable by providing a separation oracle for the dual. The dual of (LP) is similar to (5), the dual of (LP′), except that z is indexed only by singletons and not arbitrary subsets of $\mathcal{L}$. This dual has a polynomial number of variables, but an exponential number of constraints. By the equivalence of separation and optimization, it suffices to provide a separation oracle for this dual. The following problem subsumes this separation problem: given some (positive or negative) weights y on the nodes of V, find a compatible subset $L \in \mathcal{C}$ which maximizes $\sum_{v \in V[L]} y_v$. This is a weighted variant of the *maximum agreement subtree problem*. Similar to the usual (unweighted) version [24], this can be solved in polynomial time via dynamic programming.

Assume for convenience that $\mathcal{L} = \{1, 2, \ldots, n\}$. We will deviate from the notational conventions in the previous sections, and use i and j to denote leaves, and we will use $t \in \{1, 2\}$ to denote the indices of the two input trees.

Consider a set $L \subseteq \mathcal{L}$, and the tree $T_t[L]$ it induces in T_t for $t \in \{1, 2\}$. We can identify each internal node u of T_t that has two children in $T_t[L]$ by a pair consisting of the smallest leaf in L in the subtree below

each of the two children. If these two leaves are i and j, where $i \leq j$, we will label u with (i, j). Note that $u = \text{lca}_t(i, j)$. We extend this labeling to the leaves in L, and label $i \in L$ with the pair (i, i).

The set L is compatible precisely if the set of labels of the labelled nodes of $T_1[L]$ and $T_2[L]$ are the same, and the ancestry relationship between the labelled nodes are also the same. Observe that if an internal node u in $T_t[L]$ is labelled with (i_1, i_2), then it must have two descendants u_1 and u_2 where u_1 is labelled (i_1, j) and u_2 is labelled (i_2, k) for some $j, k \in L$.

Using this observation, we can use the labelling to describe compatibility constraints. To do this, we begin by constructing a directed acyclic graph $D = (Z, U_1 \cup U_2)$ as follows. The node set Z consists of all pairs $(i_1, i_2) \in \mathcal{L}^2$ for which $i_1 \leq i_2$. With a slight abuse of notation, we define $\text{lca}_t(r)$ for $r = (i_1, i_2) \in Z$ as $\text{lca}_t(i_1, i_2)$. Given two nodes $r = (i_1, i_2)$ and $s = (j_1, j_2)$ in Z:

- if $\text{lca}_t(s) \prec \text{lca}_t(r)$ for all $t \in \{1, 2\}$ and $i_1 = j_1$, then $(r, s) \in U_1$;
- if $\text{lca}_t(s) \prec \text{lca}_t(r)$ for all $t \in \{1, 2\}$ and $i_2 = j_1$, then $(r, s) \in U_2$.

Suppose $(r, r_1) \in U_1, (r, r_2) \in U_2$. Observe that then $\mathcal{L}(\text{lca}_t(r_1))$ and $\mathcal{L}(\text{lca}_t(r_2))$ are disjoint subsets of $\mathcal{L}(\text{lca}_t(r))$ for $t \in \{1, 2\}$. This implies that r_1 and r_2 cannot both have a directed path to the same node $s = (j_1, j_2)$, since that would imply that $j_1, j_2 \in \mathcal{L}(\text{lca}_t(r_1)) \cap \mathcal{L}(\text{lca}_t(r_2))$.

Define $Z_{\mathcal{L}} = \{(i, i) : i \in \mathcal{L}\}$. Let $\mathcal{F}$ denote the set of out-arborescences in D with leaf set contained in $Z_{\mathcal{L}}$ and where each internal node has one outgoing arc in U_1 and one outgoing arc in U_2. Then the above implies that $L \in \mathcal{C}$ if and only if there is an $F(L) \in \mathcal{F}$ with leaf set corresponding to L. (If $|L| = 1$, $F(L)$ is empty.) Let $\chi_F \in \{0, 1\}^{U_1 \cup U_2}$ be the characteristic vector of the arc set of F, for any $F \in \mathcal{F}$. Let $C_{\mathcal{F}}$ denote the cone generated by $\{\chi_F : F \in \mathcal{F}\}$, i.e., $y \in C_{\mathcal{F}}$, if and only if there exists $x_L \geq 0$ for $L \in \mathcal{C}$ such that $y = \sum_{L \in \mathcal{C}: |L| \geq 2} x_L \chi_{F(L)}$.

We begin by giving a description of $C_{\mathcal{F}}$. For $r \in Z$, let $\delta^+(r)$ denote the arcs in D leaving r, and $\delta^-(r)$ the arcs entering r. For $S \subseteq U_1 \cup U_2$, let $y(S) = \sum_{a \in S} y_a$.

Lemma 16.

$$C_{\mathcal{F}} = \{y \in \mathbb{R}_+^{U_1 \cup U_2} : y(\delta^+(r) \cap U_1) = y(\delta^+(r) \cap U_2) \qquad \forall r \in Z \setminus Z_{\mathcal{L}}$$
$$y(\delta^+(r) \cap U_1) \geq y(\delta^-(r)) \qquad \forall r \in Z \setminus Z_{\mathcal{L}}\}.$$

Proof. Let Y denote the cone described by the right hand side of the claimed equality. It is clear that $\chi_F \in Y$ for any $F \in \mathcal{F}$, and hence that $C_{\mathcal{F}} \subseteq Y$. It remains to show that $Y \subseteq C_{\mathcal{F}}$.

Suppose $y \in Y$; we prove that $y \in C_{\mathcal{F}}$, proceeding by induction on the size of the support of y. The claim trivially holds if $y = 0$. So suppose $y \neq 0$. Choose $r = (i_1, i_2) \in Z$ such that $y(\delta^-(r)) = 0$ but $y(\delta^+(r)) > 0$. We now find an arborescence $F \in \mathcal{F}$ rooted at r and contained in the support of y. This is trivial if $i_1 = i_2$; if not, we proceed as follows. Choose any $(r, r_1) \in U_1 \cap \delta^+(r)$ and $(r, r_2) \in U_2 \cap \delta^+(r)$ that are both in the support of y. Arguing inductively, we obtain arborescences F_1 and F_R in the support of y rooted at r_1 and r_2 respectively. We have already noted that there is no node that both r_1 and r_2 can reach; thus F_1 and F_R are disjoint. We obtain F by combining F_1, F_R and the arcs from r.

Now set $y' = y - \epsilon \chi_F$, where ϵ is chosen maximally so that $y' \geq 0$. So y' has smaller support, and so by induction, $y' \in C_{\mathcal{F}}$. Hence $y = y' + \epsilon \chi_F$ is too. □

Using Lemma 16, we now describe our compact formulation. For $t \in \{1, 2\}$ and $v \in V_t$, let $\text{lca}^{-1}(v) =$

$\{r \in Z : \mathrm{lca}_t(r) = v\}$.

$$\begin{aligned}
\min \quad & \sum_{r \in Z \setminus Z_{\mathcal{L}}} \big(y(\delta^+(r) \cap U_1) - y(\delta^-(r))\big) + \sum_{i \in \mathcal{L}} x_{\{i\}} - 1 && && \text{(LP}^\star\text{-1)}\\
\text{s.t.} \quad & y(\delta^+(r) \cap U_1) = y(\delta^+(r) \cap U_2) && \forall r \in Z && \text{(LP}^\star\text{-2)}\\
& y(\delta^+(r) \cap U_1) \geq y(\delta^-(r)) && \forall r \in Z && \text{(LP}^\star\text{-3)}\\
& x_{\{i\}} = 1 - y(\delta^-(i,i)) && \forall i \in Z && \text{(LP}^\star\text{-4)}\\
& \sum_{r \in \mathrm{lca}^{-1}(v)} y(\delta^+(r) \cap U_1) \leq 1 && \forall v \in V \setminus \mathcal{L} && \text{(LP}^\star\text{-5)}\\
& x_{\{i\}} \geq 0 && \forall i \in \mathcal{L}\\
& y_a \geq 0 && \forall a \in U_1 \cup U_2
\end{aligned}$$

Lemma 17. *This LP is equivalent to* (LP).

Proof. By Lemma 16, (LP*-2) and (LP*-3) ensure $y \in C_{\mathcal{F}}$ and thus we can expand $y = \sum_{L \in \mathcal{C}: |L| \geq 2} x_L \chi_{F(L)}$ for $x \geq 0$. Then taking x_L for $|L| = 1$ as defined by (LP*-4), we ensure that $\sum_{L \in \mathcal{C}: i \in L} x_L = 1$ for all $i \in \mathcal{L}$. Constraint (LP*-5) becomes $\sum_{L \in \mathcal{C}: v \in L} x_L \leq 1$ for all non-leaves v. And for $r \in Z \setminus Z_{\mathcal{L}}$,

$$y(\delta^+(r) \cap U_1) - y(\delta^-(r)) = \sum_{L \in \mathcal{C}: \mathrm{rt}(F(L)) = r} x_L,$$

where $\mathrm{rt}(F)$ denotes the root of F. So the objective becomes $\sum_{L \in \mathcal{C}} x_L - 1$. Thus, this formulation is equivalent to (LP). □

Acknowledgements. We acknowledge the support of the Tinbergen Institute and the Hausdorff Research Institute for Mathematics, where portions of this research was pursued.

N.O. was supported in part by NWO Veni grant 639.071.307. F.S. was supported in part by NSF grants CCF-1526067 and CCF-1522054. A.Z. was supported in part by grant #359525 from the Simons Foundation.

References

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

[2] M. A. Bender and M. Farach-Colton. The LCA problem revisited. In *Proceedings of the 4th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 88–94, 2000.

[3] M. L. Bonet, K. S. John, R. Mahindru, and N. Amenta. Approximating subtree distances between phylogenies. *Journal of Computational Biology*, 13(8):1419–1434, 2006.

[4] 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.

[5] M. Bordewich and C. Semple. On the computational complexity of the rooted subtree prune and regraft distance. *Annals of Combinatorics*, 8(4):409–423, 2004.

[6] F. Chataigner. Approximating the maximum agreement forest on k trees. *Information processing letters*, 93(5):239–244, 2005.

[7] J. Chen, F. Shi, and J. Wang. Approximating maximum agreement forest on multiple binary trees. *Algorithmica*, 76(4):867–889, 2016.

[8] Z.-Z. Chen, Y. Harada, and L. Wang. A new 2-approximation algorithm for rSPR distance. In Z. Cai, O. Daescu, and M. Li, editors, *Bioinformatics Research and Applications*, pages 128–139, Cham, 2017. Springer International Publishing.

[9] Z.-Z. Chen, E. Machida, and L. Wang. An improved approximation algorithm for rSPR distance. In *International Computing and Combinatorics Conference*, pages 468–479. Springer, 2016.

[10] C. Darwin. *Notebook B: Transmutation of species (1837?-1838)*. In: John van Wyhe: The Complete Work of Charles Darwin Online, 2002. http://darwin-online.org.uk/.

[11] O. Gascuel, editor. *Mathematics of Evolution and Phylogeny*. Oxford University Press, Inc., 2005.

[12] M. X. Goemans and D. P. Williamson. The primal-dual method for approximation algorithms and its application to network design problems. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*, pages 144–191. PWS Publishing Co., Boston, 1997.

[13] D. Harel. A linear time algorithm for the lowest common ancestors problem. In *Proceedings of the 21st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 308–319, 1980.

[14] D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

[15] J. Hein, T. Jiang, L. Wang, and K. Zhang. On the complexity of comparing evolutionary trees. *Discrete Applied Mathematics. The Journal of Combinatorial Algorithms, Informatics and Computational Sciences*, 71(1-3):153–169, 1996.

[16] D. Huson, R. Rupp, and C. Scornavacca. *Phylogenetic Networks: Concepts, Algorithms and Applications*. Cambridge University Press, 2010.

[17] L. Nakhleh. Evolutionary phylogenetic networks: models and issues. In L. Heath and N. Ramakrishnan, editors, *The Problem Solving Handbook for Computational Biology and Bioinformatics*. Springer, 2009.

[18] N. Olver, F. Schalekamp, L. Stougie, and A. van Zuylen. Implementation of the MAF algorithm and compact formulation. Available at http://nolver.net/maf and http://frans.us.MAF, 2018.

[19] E. 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.

[20] 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.

[21] F. Schalekamp, A. van Zuylen, and S. van der Ster. A duality based 2-approximation algorithm for maximum agreement forest. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 55 of *LIPIcs*, pages 70:1–70:14. Leibniz-Zentrum für Informatik, 2016.

[22] C. Semple and M. Steel. *Phylogenetics*. Oxford University Press, 2003.

[23] F. Shi, Q. Feng, J. You, and J. Wang. Improved approximation algorithm for maximum agreement forest of two rooted binary phylogenetic trees. *Journal of Combinatorial Optimization*, 2015.

[24] M. Steel and T. Warnow. Kaikoura tree theorems: Computing the maximum agreement subtree. *Information Processing Letters*, 48(2):77–82, Nov. 1993.

[25] L. van Iersel, S. Kelk, N. Lekic, and L. Stougie. Approximation algorithms for nonbinary agreement forests. *SIAM Journal on Discrete Mathematics*, 28(1):49–66, 2014.

[26] C. Whidden, R. G. Beiko, and N. Zeh. Fixed-parameter algorithms for maximum agreement forests. *SIAM Journal on Computing*, 42(4):1431–1466, 2013.

[27] C. Whidden and N. 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.

[28] Y. Wu. A practical method for exact computation of subtree prune and regraft distance. *Bioinformatics*, 25(2):190–196, 2009.

[29] Y. Wu and J. 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.

A The running time

It is quite clear from the definition of the Red-Blue Algorithm that it runs in polynomial time. In this section we show that it can be implementated to run in $O(n^2)$ time, where n denotes the number of leaves. (We work in the random access machine model of computation, and assume a word size of $\Omega(\log n)$.)

We note that our presentation is focused on showing the bound on the running time as straightforwardly as possible, and there are some places where a more careful implementation is more efficient. However, we have not been able to find an implementation with an overall running time of $o(n^2)$.

We assume $\mathcal{P}$ is a given partition (not overlapping in V_2), that is stored such that we can query the size of any component in constant time, and that for each node $\hat{u} \in V_2$, we can query $A_{\hat{u}}$, the component in $\mathcal{P}$ that covers $\hat{u}$ (which will be equal to $\emptyset$ if $\mathcal{P}$ does not cover $\hat{u}$), and $s(\hat{u}) = |\mathcal{L}(\hat{u}) \cap A_{\hat{u}}|$. Note that we can determine this information by a bottom-up pass of T_2 in $O(n)$ time. We will recompute it whenever we refine $\mathcal{P}$; since there can only be at most $n-1$ refinement operations, the total time to maintain this information is $O(n^2)$.

By Harel [13] (see also [14, 2]), we furthermore may assume that the computation of $\mathrm{lca}_i(u, v)$ for given nodes $u, v \in V_i$ takes constant time (after a linear preprocessing time). It immediately follows from this that we can determine whether or not $u \preceq v$ in tree T_i in constant time as well.

We will show that the time between subsequent refinements of $\mathcal{P}$ is $O(n)$. This bounds the time of the main loop of the algorithm by $O(n^2)$. The only remaining part of the algorithm is the Merge-Components step, which will perform at most $n-1$ merges, each of which can clearly be done in $O(n)$ time.

Finding a lowest root-of-infeasibility

We make a single pass through T_1, in bottom-up order (starting from the leaves), until we find a root-of-infeasibility. We will spend constant time per node, thus showing that the time to find a lowest root-of-infeasibility is $O(n)$.

For each node $u \in V_1$ that we have already considered, A_u references the component $A \in \mathcal{P}$ which covers u, with $A_u = \emptyset$ if there are no such components. (If there are multiple such components, u is a root-of-infeasibility.) Furthermore, $\hat{p}_u$ is equal to $\mathrm{lca}_2(A_u \cap \mathcal{L}(u))$, and $s(u)$ is the size of $A_u \cap \mathcal{L}(u)$. Observe that for any $x \in \mathcal{L}$, we know A_x, the component containing x, and $s(x) = 1$ and $\hat{p}_x = x$.

Given a non-leaf node $u \in V_1$, with children u_1 and u_2 that have already been considered, we can determine whether u is a root-of-infeasibility, and if not determine $A_u, \hat{p}_u$ and $s(u)$, in constant time: If either (or both) of A_{u_1} and A_{u_2} do not cover u (which can be determined by checking if $A_{u_i} = \emptyset$ or $s(u_i) = |A_{u_i}|$), set all the values according to which child (if any) does cover u, and end the consideration of node u. So assume from now on that both do cover u.

If $A_{u_1} \neq A_{u_2}$, then u satisfies the second condition of a root-of-infeasibility, and we are done. Otherwise, $A_u = A_{u_1} = A_{u_2}$. Set $\hat{p}_u = \mathrm{lca}_2(\hat{p}_{u_1}, \hat{p}_{u_2})$ and $s(u) = s(u_1) + s(u_2)$. If $\hat{p}_{u_1} \not\preceq \hat{p}_u$ or $\hat{p}_{u_2} \not\preceq \hat{p}_u$, then $\mathcal{L}(u)$ is incompatible, and u satisfies the first condition of a root-of-infeasibility. If $s(\hat{p}_u) = |A_u|$ and $s(u) < |A_u|$ then u satisfies the third condition for being a root-of-infeasibility: by $s(u) < |A_u|$, we know $A_u \setminus \mathcal{L}(u) \neq \emptyset$.

For any $w \in A_u \setminus \mathcal{L}(u)$, $\text{lca}_1(A_u \cap \mathcal{L}(u)) \preceq u \preceq \text{lca}_1(A_u \cap \mathcal{L}(u) \cup \{w\})$, while by $s(\hat{p}_u) = |A_u|$ we know that $\text{lca}_2(A_u \cap \mathcal{L}(u)) = \text{lca}_2(A_u)$. So $A_u \cap \mathcal{L}(u) \cup \{w\}$ is incompatible for any $w \in A_u \setminus \mathcal{L}(u)$. Otherwise, u is not a root-of-infeasibility, and we finish our consideration of u.

Once we have determined the coloring (R, B, W), we compute $|A \cap C|$ for each component $A \in \mathcal{P}$ and $C \in \{R, B, W\}$. We also compute three additional labels for each node $\hat{u} \in V_2$: $s_C(\hat{u}) = |A_{\hat{u}} \cap \mathcal{L}(\hat{u}) \cap C|$ for $C \in \{R, B, W\}$. This information can be determined by a bottom-up traversal of T_2 in $O(n)$ time. We assume this information is updated whenever the partition is refined.

Make-$R \cup B$-compatible

Consider the nodes of T_2 in bottom-up order, until we find a node $\hat{u}$ such that both $s_B(\hat{u}) > 1$ and $s_R(\hat{u}) > 1$. Since $\hat{u}$ is a lowest such node, if $|A_{\hat{u}} \cap R| = s_R(\hat{u})$ and $|A_{\hat{u}} \cap B| = s_B(\hat{u})$, then $A_{\hat{u}}$ is $R \cup B$-compatible; otherwise $\hat{u}$ is precisely as indicated in Make-$R \cup B$-compatible.

Make-Splittable

We again consider the nodes in T_2 in bottom-up order. For any node $\hat{u}$ with $A_{\hat{u}} \neq \emptyset$, using $s_C(\hat{u})$ for $C \in \{R, B, W\}$, we can check in $O(1)$ time whether $A_{\hat{u}} \cap \mathcal{L}(\hat{u})$ is bicolored, and that for any $C \in \{R, B, W\}$ with $A_{\hat{u}} \cap C \neq \emptyset$, that $s_C(\hat{u}) < |A_{\hat{u}} \cap C|$ (and hence $(A_{\hat{u}} \setminus \mathcal{L}(\hat{u})) \cap C \neq \emptyset$).

Split

Note that a regular split of a component A can be done in $O(n)$ time, by simply checking the color of each leaf in A and partitioning A accordingly. We now show how to check if A needs a Special-Split (and if so which of the two possible refinements is applied) by considering the nodes in $V_2[A]$ in bottom-up order.

If A is tricolored, then the fact that A is $R \cup B$-compatible and splittable implies that there exist $\hat{u}_R$ and $\hat{u}_B$ that are covered by A and for which $A \cap \mathcal{L}(\hat{u}_R) = A \cap R$ and $A \cap \mathcal{L}(\hat{u}_B) = A \cap B$. Using a bottom-up traversal of V_2 will find $\hat{u}_R$ and $\hat{u}_B$ (they are the first nodes $\hat{v}$ encountered such that $A_{\hat{v}} = A$ and $s_C(\hat{v}) = |A \cap C|$ for $C = R$ and B respectively).

Given $\hat{u}_R$ and $\hat{u}_B$, we check if a Special-Split is required, by considering $\hat{u} = \text{lca}_2(A \cap (R \cup B)) = \text{lca}_2(\hat{u}_R, \hat{u}_B)$; a Special-Split is required exactly if $s(\text{lca}_2(\hat{u}_R, \hat{u}_B)) < |A|$, since in that case any $x_W \in A \setminus \mathcal{L}(\text{lca}_2(\hat{u}_R, \hat{u}_B))$ forms a compatible triple with any $x_R \in A \cap R, x_B \in A \cap B$. If, in addition, $s_W(\text{lca}_2(\hat{u}_R, \hat{u}_B)) = 0$, we know that every tricolored triple in A is compatible.

Find-Merge-Pair

We need to determine if there exist two components (both intersecting $R \cup B$) that can be merged, in time $O(n)$. If such components are found, then we can take a non-white leaf in each component and add this pair to `pairslist`. Recall Lemma 15, which enumerates all possible situations where a potential merge may exist.

(a) *$\mathcal{P}$ has a bicolored component.* This component must have been created by an application of Special-Split, splitting some component $A \cup A^* \in \mathcal{P}^{(2)}$ into A and A^*. As discussed in the proof of Lemma 15, simply undoing this split is a valid merge. Since Special-Split is invoked at most once per iteration, we can simply add a pair to `pairslist` during Special-Split.

(b) *There is a node $\hat{u} \in V_2$ that can be reached by two red or two blue components that were part of the same component at the start of the current iteration.* By Lemma 12 (and property 1 of Lemma 4), any two components that are not white that were created in the current iteration must have been part of the same partition at the start of the iteration. We may assume that we can check for each component in constant time whether it was created in the current iteration.

We work bottom-up in T_2, and set $\mathcal{B}_{\hat{u}}$ to be the set of red and blue components that were created in the current iteration, and that can reach $\hat{u}$ for every $\hat{u} \in V_2$. If $\mathcal{B}_{\hat{u}}$ contains two components of the same color, these two components can be merged, and we terminate.

Note that if $\mathcal{B}_{\hat{u}}$ ever contains three components, we will have found a merge and the algorithm will terminate. This ensures that we can compute this for $\hat{u}$ in constant time, given the values of $A_{\hat{u}}$ and $\mathcal{B}_{\hat{u}_1}$ and $\mathcal{B}_{\hat{u}_2}$ for the children of $\hat{u}$.

(c) *There is a node $\hat{u} \in V_2$ that can be reached by a red and a blue component that were part of the same component A_0 at the start of the current iteration, but is not covered by these components. Furthermore, the node $\hat{u}$ must satisfy that the nodes on the path from $\hat{u}$ to $\mathrm{lca}_2(A_0)$ are not covered by any red or blue component.* Note that by Lemma 12, A_0 is the only component that was modified in the current iteration, so for this second condition we can simply check that no node on the path from $\hat{u}$ to the root of V_2 is covered by a red or blue component that was created in the current iteration.

If we did not find two components of the same color that can be merged, we have found $\mathcal{B}_{\hat{u}}$ for every $\hat{u} \in V_2$, where $|\mathcal{B}_{\hat{u}}| \leq 2$. We now work top-down in T_2. If we encounter a node $\hat{u}$ that is covered by a red or blue component that was created in the current iteration, we stop and do not consider the descendants of $\hat{u}$ (since for any such a descendant, $\hat{u}$ is on its path to $\mathrm{lca}_2(A_0)$). If we encounter a node $\hat{u}$ such that $|\mathcal{B}_{\hat{u}}| = 2$ and $\hat{u}$ is not covered by any component, the two components in $\mathcal{B}_{\hat{u}}$ can be merged, and we may terminate.

B Integrality gap lower bounds

We show a lower bound on the integrality gap of $\frac{16}{5}$ for the integer linear program formulation of Wu [28]. Recall that a solution to MAF can be viewed as the leaf sets of the trees in a forest, obtained by deleting edges from the input trees. The formulation has binary variables x_e for every edge $e \in T_1$, indicating whether e is deleted from T_1. We use $P(i,j)$ to denote the set of edges in T_1 on the path between leaves i and j, and $P_2(i,j)$ to denote the set of edges in T_2 on the path between leaves i and j. Wu's integer linear program [28] is given by:

$$\begin{aligned}
\text{minimize} \quad & \sum_e x_e \\
\text{s.t.} \quad & \sum_{e \in P(i,j) \cup P(i,k) \cup P(j,k)} x_e \geq 1 && \text{for all incompatible triples } i,j,k \\
& \sum_{e \in P(i,j)} x_e + \sum_{e \in P(k,\ell)} x_e \geq 1 && \text{for all two pairs } (i,j) \text{ and } (k,\ell) \text{ for which } P(i,j) \cap P(k,\ell) = \emptyset, \text{ and } P_2(i,j) \cap P_2(k,\ell) \neq \emptyset
\end{aligned}$$

The first family of constraints ensures that at least one edge of the paths between i and j, i and k, and j and k has to be deleted for each inconsistent triple i, j and k. The second family of constraints ensures that at least one edge is deleted for every pair of paths between i and j, and k and ℓ that are disjoint in T_1, but for which the corresponding paths in T_2 are not disjoint.

Lemma 18. *The integrality gap of the integer linear program of Wu [28] is at least $\frac{16}{5}$.*

Proof. Let $n = 2^k$ for some k even. We label each internal node in both T_1 and T_2 with a binary string: the roots get the empty string as label, and given an internal node u its left child gets u's label with a "0" appended, and its right child gets u's label with a "1" appended. In T_1, the leaves are labeled in the same way as the internal nodes, with a binary string of length k. In T_2, the binary string is *reversed* to give the label of the leaf. For example, the leftmost leaf (of both trees) has label $00\cdots0$, and the leaf to the right of it has label $0\cdots01$ in T_1, and $10\cdots0$ in T_2.

Consider the internal nodes whose label is a string of length strictly less than $k/2$; there are exactly $2^{k/2} - 1 = \sqrt{n} - 1$ such nodes in each tree. We claim that any component A must cover at least $|A| - 1$ of these internal nodes. To see this, consider the set of internal nodes in V_t that have two children in the subtree of T_t induced by A for $t = 1, 2$. We will call such nodes *bifurcating*. Observe that there are $2(|A| - 1)$ such

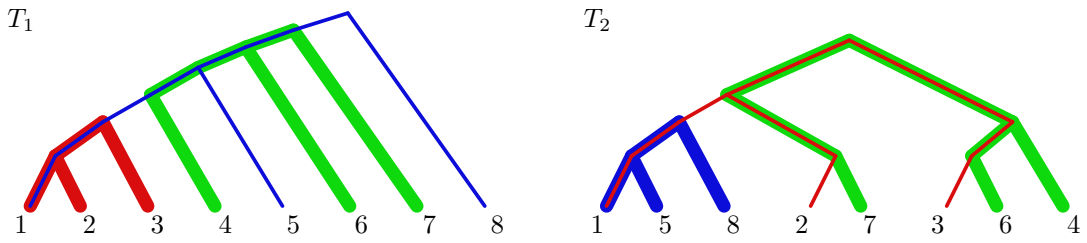


Figure 5: Example with integrality gap of $\frac{5}{4}$ for the new ILP introduced in Section 4. An optimal solution to the LP-relaxation is indicated by the colors: the compatible sets corresponding to each of the components $\{1,2,3\}, \{1,5,8\}$ and $\{4,6,7\}$ (indicated by the colors red, blue and green respectively) have an x-value of $\frac{1}{2}$, as well as every singleton leaf set, except leaf 1; all other x-values are 0. The objective value of this solution is $\frac{1}{2}(3+7)-1=4$. An optimal solution to the ILP has 6 components, i.e., an objective value of 5.

nodes. Furthermore, since A is compatible, there is a 1-1 mapping f from the bifurcating nodes in V_1 to the bifurcating nodes in V_2, where, $\mathcal{L}(u) \cap A = \mathcal{L}(f(u)) \cap A$. Now, the label for a bifurcating node $u \in V_1$ is the maximum length prefix that the binary strings for the leaves in $\mathcal{L}(u) \cap A$ have in common, and the label for $f(u)$ is the reverse of the maximum length suffix the leaves in $\mathcal{L}(u) \cap A$ have in common. Hence, at least one of u and $f(u)$'s labels has length less than $k/2$.

The fact that any component A must cover at least $|A|-1$ of the $2\sqrt{n}-2$ internal nodes with labels of length less than $k/2$ implies that any partition that does not overlap must have at least $n-2\sqrt{n}+2$ components. Thus the optimal value of the integer program is at least $n-2\sqrt{n}+1$.

On the other hand, the LP relaxation of the integer program has a feasible solution with objective value $\frac{5}{16}n$: set a value of $\frac{1}{4}$ on the edges to each leaf in the tree (i.e., from an internal node with a label of length $k-1$ to a node with a label length k), and a value of $\frac{1}{8}$ on all edges between nodes with labels of length $k-2$ to nodes with labels of length $k-1$. This implies a lower bound of $\lim_{n\to\infty} \frac{n-2\sqrt{n}+1}{\frac{5}{16}n} = \frac{16}{5}$ on the integrality gap. □

As remarked in the introduction, the largest integrality gap for our formulation that we are aware of is 5/4. The instance is described in Figure 5.