Available online at www.sciencedirect.com

ScienceDirect

Theoretical Computer Science 374 (2007) 91–110

Theoretical Computer Science

www.elsevier.com/locate/tcs

The maximum agreement forest problem: Approximation algorithms and computational experiments[☆]

Estela M. Rodrigues[a], Marie-France Sagot[b], Yoshiko Wakabayashi[a,*]

[a] *Universidade de São Paulo, Departamento de Ciência da Computação, Rua do Matão, 1010, 05508-090 – São Paulo, Brazil*
[b] *INRIA Rhône-Alpes, Université Claude Bernard, Lyon I, 43, Bd du 11 novembre 1918, 69622 Villeurbanne cedex, France*

Received 22 April 2005; received in revised form 21 August 2006; accepted 12 December 2006

Communicated by M. Crochemore

Abstract

There are various techniques for reconstructing phylogenetic trees from data, and in this context the problem of determining how distant two such trees are from each other arises naturally. Various metrics for measuring the distance between two phylogenies have been defined. Another way of comparing two trees $\mathcal{T}$ and $\mathcal{U}$ is to compute the so called *maximum agreement forest* of these trees. Informally, the number of components of an agreement forest tells how many edges from each of $\mathcal{T}$ and $\mathcal{U}$ need to be cut so that the resulting forests agree, after performing all forced edge contractions. This problem is NP-hard even when the input trees have maximum degree 2. Hein et al. [J. Hein, T. Jiang, L. Wang, K. Zhang, On the complexity of comparing evolutionary trees, Discrete Applied Mathematics 71 (1996) 153–169] presented an approximation algorithm for it, claimed to have performance ratio 3. We show that the performance ratio of the algorithm proposed by Hein et al. is 4, and we also present two new 3-approximation algorithms for this problem. We show how to modify one of the algorithms into a $(d + 1)$-approximation algorithm for trees with bounded degree d, $d \geq 2$. Finally, we report on some computational experiments comparing the performance of the algorithms presented in this paper.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Agreement forest; Approximation algorithm; APX-complete; Computational complexity; Evolutionary process; Phylogenetic tree

1. Introduction

Phylogenetic trees or **phylogenies** are a standard model for representing evolutionary processes, mostly involving biological entities such as species or genes. By a phylogenetic tree, we mean a rooted unordered tree whose leaves are uniquely labeled with elements of some set S, and whose internal nodes are unlabeled and have at least two

[☆] This research was partially supported by CAPES-COFECUB Project 272/99-II. The three authors are members of the INRIA-FAPESP Project 2004/14335-5 and Associate Team HELIX-USP (INRIA). The first author was supported by CNPq Grant Proc. 142307/97-1 and also by CAPES Grant BEX 0650-99/4 during her visit to Institut Pasteur, where part of this research was done. The second author was supported by the Program BioInformatics inter EPST, and the third author is partially supported by CNPq (Procs. 308138/04-0, 490333/04-4, 478329/04-0) and by ProNEx–FAPESP/CNPq Proc. 2003/09925-5.

[*] Corresponding author. Tel.: +55 11 3091 6135; fax: +55 11 3091 6134.
E-mail addresses: emrod2006@gmail.com (E.M. Rodrigues), Marie-France.Sagot@inria.fr (M.-F. Sagot), yw@ime.usp.br (Y. Wakabayashi).

0304-3975/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2006.12.011

children. The **degree** of a node in a rooted tree is the number of children of this node. The elements of S stand for the contemporary taxa whose evolutionary relationships one intends to model. These taxa correspond to the leaves of the tree, whereas the ancestral taxa correspond to its internal nodes, so that for each ancestral taxon, all of its nearest derived taxa are depicted as its children in the tree.

There are various techniques for reconstructing phylogenetic trees from such data, and in this setting the problem of determining how distant two such trees are from each other arises naturally. Various metrics, such as NNI (nearest-neighbor interchange), SPR (subtree prune and regraft) and TBR (tree bisection and reconnection) for measuring the distance between two phylogenies have been proposed [8,5,3]. Some results relating these concepts are presented by Allen and Steel [1]. In particular, they observed that one of the NP-hardness proofs presented by Hein et al. [4] can be adapted to prove that the maximum agreement forest problem is NP-hard. Hein et al. [4] state that this proof can also be adapted to a MAX SNP-hardness proof for the maximum agreement forest problem. On the basis of these results (but using another problem for the reduction) Rodrigues [6] has shown that the maximum agreement forest problem for trees with maximum degree 2 is APX-hard. This result together with the approximation algorithms described in this paper shows that the maximum agreement forest problem for trees with bounded degree d ($d \geq 2$) is APX-complete. Thus, under the hypothesis that P $\neq$ NP, there is no polynomial approximation scheme for this problem.

We are concerned with the problem of finding the size of a maximum agreement forest of two trees with bounded degree d, $d \geq 2$. A formal definition of this problem is given in Section 2.2. A previous algorithm by Hein et al. [4], which we describe in Section 3, has been claimed to find solutions within 3 times the optimum. In Section 4.1 we exhibit a family of instances of the problem which shows that this algorithm has performance ratio at least 4, and in Section 4.5 we prove that its performance ratio is indeed 4.

We also introduce two approximation algorithms for the problem, both based on the same framework as Hein's algorithm but yielding performance ratio 3. These algorithms are quite simple, but the analysis of the approximation ratio of our first algorithm and Hein's is rather long and technical, while our second algorithm admits a more straightforward analysis. All these algorithms are presented and analyzed in Sections 3 and 4. We have implemented and tested the algorithms with randomly generated instances and found that the first of the new algorithms here proposed performs best, constructing solutions closer to the optimum than the other two ones. The implementations and tests are discussed in Section 5. We conclude with a generalization of our first algorithm for trees with bounded degree d ($d \geq 2$), which has performance ratio $d + 1$. This result is presented in Section 6.

Early versions of some of the results shown in this paper were presented (with an outline of the proofs) in [7]. The results on computational experiments with the implementation of the algorithms and the generalization of one of the algorithms for trees with bounded degree d ($d \geq 2$) did not appear in the earlier version. Recently, Chataigner [2] designed an 8-approximation algorithm to find a maximum agreement forest of k binary trees, $k \geq 2$.

2. Basic definitions

2.1. Phylogenetic forests

A **phylogenetic tree** consists of an unordered rooted tree, called its **topology**, such that each internal node has at least two children, and of a set of labels which are mapped one-to-one to the leaves of the tree. If $\mathcal{T}$ is a phylogenetic tree, then $\boldsymbol{T_{\mathcal{T}}}$ denotes its topology, $\boldsymbol{L_{\mathcal{T}}}$ its set of leaves, $\boldsymbol{S_{\mathcal{T}}}$ its set of labels, $\boldsymbol{f_{\mathcal{T}}}$ its one-to-one label-to-leaf mapping, and $\boldsymbol{r_{\mathcal{T}}}$ its root. Since we consider phylogenetic trees as having rooted topologies, these are naturally oriented: we assume arcs are always oriented towards the root.

For each arc e in a phylogenetic tree $\mathcal{T}$, we denote by $\boldsymbol{l(e)}$ its **lower endpoint** (the endpoint of e which is the farthest one from the root) and by $\boldsymbol{u(e)}$ its **upper endpoint** (the other endpoint of e). The **lowest common ancestor** (**lac**) of a set of $m \geq 1$ nodes $\{v_1, \ldots, v_m\}$ of $\mathcal{T}$, written as $\mathbf{lca}_{\mathcal{T}}\,\boldsymbol{(\{v_1, \ldots, v_m\})}$, or simply $\mathrm{lca}_{\mathcal{T}}(v_1, \ldots, v_m)$, is the ancestor of the nodes v_i, $1 \leq i \leq m$, whose distance from the root is maximum, where the **distance** of a node to the root is the length of the (unique) path that connects this node to the root. For each node u in $\mathcal{T}$, we denote by $\boldsymbol{D(u)}$ the set of leaves in $\mathcal{T}$ that are descendants of u, and if $u \neq r_{\mathcal{T}}$, then $\boldsymbol{p(u)}$ is the **parent** of u. For simplicity, for an arc e, instead of writing $D(l(e))$, we write $\boldsymbol{D_e}$ (the set of the descendants of the lower endpoint of e). We observe that a node is also considered a descendant of itself.

The concept of phylogenetic tree can be generalized so as to consider topologies with more than one component. Such phylogenies are called **phylogenetic forests**. Each component in the topology of a phylogenetic forest

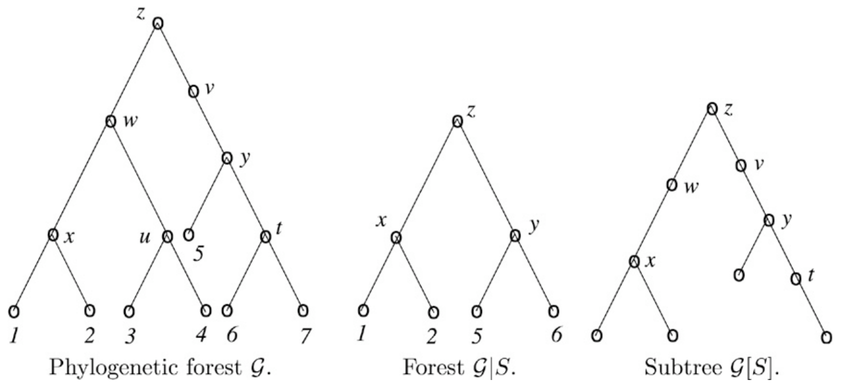


Fig. 1. Forests $\mathcal{G}|S$ and $\mathcal{G}[S]$ for $S := \{1, 2, 5, 6\}$.

corresponds to a **component** of the forest, with its own topology, root, leaf set, label set and label-to-leaf mapping. If $\mathcal{F}$ is a phylogenetic forest, then $\boldsymbol{T_{\mathcal{F}}}$ denotes its topology, $\boldsymbol{\mathcal{L}_{\mathcal{F}}}$ and $\boldsymbol{\mathcal{S}_{\mathcal{F}}}$ (with a calligraphic $\mathcal{L}$ and $\mathcal{S}$) the family of, respectively, the leaf sets and the label sets of its components, $\boldsymbol{f_{\mathcal{F}}}$ its label-to-leaf mapping ($f_{\mathcal{F}}$ is a bijection from $\bigcup \mathcal{S}_{\mathcal{F}} = \bigcup_{S \in \mathcal{S}_{\mathcal{F}}} S$ into $\bigcup \mathcal{L}_{\mathcal{F}} = \bigcup_{L \in \mathcal{L}_{\mathcal{F}}} L$), and $\boldsymbol{R_{\mathcal{F}}}$ the set of roots of its components.

2.2. Restrictions and agreement forests

Two phylogenetic forests $\mathcal{G}$ and $\mathcal{H}$ are said to be **isomorphic** if $\bigcup \mathcal{S}_{\mathcal{G}} = \bigcup \mathcal{S}_{\mathcal{H}}$, their topologies $T_{\mathcal{G}}$ and $T_{\mathcal{H}}$ are isomorphic, and the isomorphism preserves both the roots of the components and the labels of the leaves. If S is a subset of $S_{\mathcal{V}}$ for some component $\mathcal{V}$ of $\mathcal{G}$, then the **simple restriction** of $\mathcal{G}$ to S, denoted by $\boldsymbol{\mathcal{G}|S}$, is the phylogenetic tree $\mathcal{G}'$ such that $S_{\mathcal{G}'} = S$; $L_{\mathcal{G}'} = f_{\mathcal{G}}(S) = \{f_{\mathcal{G}}(a) : a \in S\}$; for each pair of nodes u, v in $\mathcal{G}'$ we have $\text{lca}_{\mathcal{G}'}(u, v) = \text{lca}_{\mathcal{G}}(u, v)$; and $r_{\mathcal{G}'} = \text{lca}_{\mathcal{G}'}(L_{\mathcal{G}'})$. Let $\boldsymbol{\mathcal{G}[S]}$ denote the minimal subtree of $T_{\mathcal{G}}$ which connects all leaves with labels in S. Fig. 1 illustrates the definition of $\mathcal{G}|S$ and $\mathcal{G}[S]$.

Let $\mathcal{G}$ be a phylogenetic forest and $\mathcal{S} = \{S_i : 1 \le i \le m\}$ a family of m elements of $\mathcal{S}_{\mathcal{G}}$ such that for each i we have $S_i \subseteq S_{\mathcal{V}}$ for some component $\mathcal{V}$ of $\mathcal{G}$, and the subtrees $\mathcal{G}[S_i]$ are pairwise node-disjoint. Then:

- we say that $\mathcal{S}$ **defines a restriction** of $\mathcal{G}$;
- the **restriction** of $\mathcal{G}$ to $\mathcal{S}$, written as $\boldsymbol{\mathcal{G}|\mathcal{S}}$, is the phylogenetic forest composed of the components $\mathcal{G}|S_i$.

We denote by $\boldsymbol{\mathcal{G}[\mathcal{S}]}$ the forest composed of all components $\mathcal{G}[S_i]$ for $1 \le i \le m$. If $\mathcal{G}' = \mathcal{G}|\mathcal{S}$, then the **size** of $\mathcal{G}'$, denoted by $\boldsymbol{|\mathcal{G}'|}$, is the cardinality of $\mathcal{S}$. If $\bigcup \mathcal{S}_{\mathcal{G}} = \bigcup \mathcal{S}_{\mathcal{G}'}$, then $\mathcal{G}'$ is a **full restriction** of $\mathcal{G}$.

From the above definitions we can deduce the following. If $\mathcal{S} = \{S_i : 1 \le i \le m\}$ defines a restriction of $\mathcal{G}$, then the set of arcs of $\mathcal{G}$ can be partitioned into two classes, one class being the one consisting of those arcs contained in some $\mathcal{G}[S_i]$. If an arc e of $\mathcal{G}$ is in such a class, then $\mathcal{V} := \mathcal{G}|S_i$ is the unique component of $\mathcal{G}|\mathcal{S}$ such that $S_{\mathcal{V}} \cap D_e \neq \emptyset$ and $S_{\mathcal{V}} \setminus D_e \neq \emptyset$ (no other component exists because the subtrees $\mathcal{G}[S_i]$ are all mutually disjoint). If an arc e of $\mathcal{G}$ is not contained in any subtree $\mathcal{G}[S_i]$ then each component $\mathcal{V}$ of $\mathcal{G}|\mathcal{S}$ is either contained in the subtree rooted at $l(e)$ (in which case $S_{\mathcal{V}} \cap D_e \neq \emptyset$ and $S_{\mathcal{V}} \setminus D_e = \emptyset$) or it is not contained in the subtree rooted at $l(e)$ (in which case $S_{\mathcal{V}} \cap D_e = \emptyset$). Thus, the following statement holds.

Lemma 1. *Let $\mathcal{G}$ be a phylogenetic forest, $\mathcal{S}$ a family of subsets of $\bigcup \mathcal{S}_{\mathcal{G}}$ that defines a restriction of $\mathcal{G}$, and e an arc of $\mathcal{G}$. Then there exists at most one component $\mathcal{V}$ of $\mathcal{G}|\mathcal{S}$ such that $S_{\mathcal{V}} \cap D_e \neq \emptyset$ and $S_{\mathcal{V}} \setminus D_e \neq \emptyset$.*

The lemma above will be useful in Sections 4.2 and 4.3 when we define the concept of link. As we will see, the analyses of the performance of the algorithms are all based on this concept.

An **agreement forest** of two given phylogenetic forests $\mathcal{G}$ and $\mathcal{H}$ is a phylogenetic forest $\mathcal{F}$ that is isomorphic to a full restriction of $\mathcal{G}$ and to a full restriction of $\mathcal{H}$. An agreement forest is said to be **maximum** if it has *minimum* size. The problem of computing a maximum agreement forest of two given phylogenetic trees $\mathcal{T}$ and $\mathcal{U}$ is the **Maximum Agreement Forest problem**, for short **MAF**.

We denote by **MAF-*d*** the maximum agreement forest problem for phylogenetic trees with bounded degree $d \ge 2$. Thus, **MAF-2** is the special case of MAF for binary trees. This problem is discussed in Sections 3 and 4, and MAF-d is addressed in Section 6.

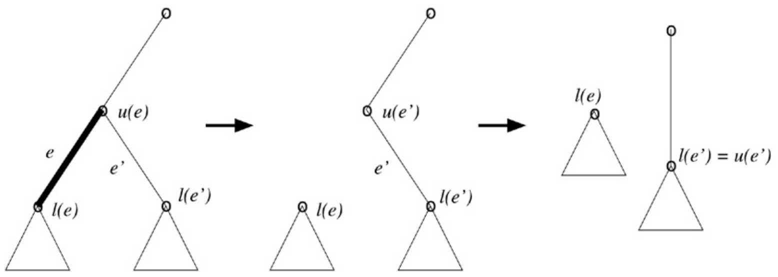


Fig. 2. Elimination of an arc e.

3. The algorithms

In this section we present algorithms for MAF-2. Thus all trees mentioned here are assumed to have maximum degree 2.

3.1. Eliminations

The basic operation used in the forthcoming algorithms is the elimination of an arc of a tree. This operation is defined as follows.

Let $\mathcal{W}$ be a phylogenetic tree. The **elimination** of an arc e of $\mathcal{W}$, denoted by $\mathbf{Elim}(\boldsymbol{\mathcal{W}}, \boldsymbol{l(e)})$, is the operation that yields the phylogenetic forest whose components are $\mathcal{W}|D_e$ and $\mathcal{W}|(S_{\mathcal{W}} \setminus D_e)$. This operation can be implemented by first *removing* the arc e from $\mathcal{W}$ (which splits $\mathcal{W}$ into $\mathcal{W}|D_e$ and $\mathcal{W}[S_{\mathcal{W}} \setminus D_e]$) and by performing a *forced contraction* of the arc e' in $\mathcal{W}[S_{\mathcal{W}} \setminus D_e]$ such that $u(e') = u(e)$ (which yields $\mathcal{W}|(S_{\mathcal{W}} \setminus D_e)$ from $\mathcal{W}[S_{\mathcal{W}} \setminus D_e]$). Fig. 2 illustrates this definition. This operation can be defined for phylogenetic forests as well, by letting all components that do not contain the arc e remain the same and applying an elimination as defined above on the component that contains e.

The lemma below exhibits the equivalence between a full restriction of a phylogenetic forest $\mathcal{H}$ and a sequence of arc eliminations starting from $\mathcal{H}$.

Lemma 2. *Let $\mathcal{H}$ be a phylogenetic forest. Then by performing a sequence of m arc eliminations starting from $\mathcal{H}$, we obtain a full restriction of $\mathcal{H}$ with size $|\mathcal{H}| + m$. Conversely, let $\mathcal{S}$ be a set family that defines a full restriction of $\mathcal{H}$. Then $\mathcal{H}|\mathcal{S}$ can be obtained from $\mathcal{H}$ by performing a sequence of $|\mathcal{S}| - |\mathcal{H}|$ eliminations.*

3.2. The basic method

The algorithm for MAF-2 devised by Hein, Jiang, Wang and Zhang [4] and the two algorithms we propose in this paper proceed within the general framework we describe in this section. The algorithm of Hein et al. will be called Algorithm 1, our two algorithms will be called Algorithm 2 and Algorithm 3. This common framework is introduced as a **basic method**.

The basic method receives as input two phylogenetic trees $\mathcal{T}$ and $\mathcal{U}$ with $S_{\mathcal{T}} = S_{\mathcal{U}}$ and returns an agreement forest of $\mathcal{T}$ and $\mathcal{U}$. The idea is to proceed by eliminating arcs in $\mathcal{T}$ and in $\mathcal{U}$ until two isomorphic restrictions are obtained. Each iteration starts therefore with two restrictions obtained from $\mathcal{T}$ and $\mathcal{U}$ by the eliminations performed up to that moment (see Lemma 2). In the beginning of each iteration the basic method searches one of the restrictions, say the one obtained from $\mathcal{T}$, for a pair of sibling leaves. If it finds such a pair, then it identifies the **case** that is satisfied by this pair in the restriction coming from $\mathcal{U}$. According to this case, the method applies on both restrictions a predefined sequence of **operations** (eliminations, essentially). Such sequences of operations are called **transactions**. The algorithms mentioned above differ in the transactions that are used for each case. In the next subsection we define the operations, cases and transactions that are used in these algorithms.

We introduce now some notation for describing and analyzing the algorithms for MAF-2. The iterations of the basic method are numbered 1, 2, ... and so on. In each iteration $i \geq 1$, the operations are separated into two sequences, one with the operations done on $\mathcal{T}$ and the other with the operations done on $\mathcal{U}$. If the sequences differ in length, the shorter one is padded with identity operations. Let $\bar{j}_i$ be the length of these sequences at iteration i.

We define a notation for the full restrictions obtained from $\mathcal{T}$ and $\mathcal{U}$ during the execution of the method. Let $\boldsymbol{\mathcal{G}_1} := \mathcal{T}$ and for $i \geq 2$ let $\boldsymbol{\mathcal{G}_i}$ be the full restriction we obtain from $\mathcal{G}_{i-1}$ after executing the operations of iteration i.

```
Basic Method
Input: Phylogenetic trees $\mathcal{T}$ and $\mathcal{U}$ such that $S_{\mathcal{T}} = S_{\mathcal{U}}$.
Output: The size of an agreement forest of $\mathcal{T}$ and $\mathcal{U}$.

(1)  $\mathcal{G}_1 := \mathcal{T}$;
(2)  $\mathcal{H}_1 := \mathcal{U}$;
(3)  $i := 1$;
(4)  While there is a pair of sibling leaves $f_{\mathcal{G}_i}(a)$, $f_{\mathcal{G}_i}(b)$ in $\mathcal{G}_i$
(5)      Find out which case is satisfied by $f_{\mathcal{H}_i}(a)$ and $f_{\mathcal{H}_i}(b)$ in $\mathcal{H}_i$;
(6)      Apply the corresponding transaction
         obtaining $\mathcal{G}_{i+1}$ from $\mathcal{G}_i$ and $\mathcal{H}_{i+1}$ from $\mathcal{H}_i$;
(7)      $i := i + 1$;
(8)  Return $|\mathcal{H}_i|$.
```

Fig. 3. An outline of the basic method.

Let $\boldsymbol{\mathcal{G}_i^1} := \mathcal{G}_i$, and for $j \in \{2, \ldots, \bar{j}_i + 1\}$ let $\boldsymbol{\mathcal{G}_i^j}$ be the full restriction we obtain by applying on $\mathcal{G}_i^{j-1}$ the $(j-1)$-th operation of the sequence of operations performed on $\mathcal{T}$ at iteration i. Similarly, we define $\boldsymbol{\mathcal{H}_i}$ and $\boldsymbol{\mathcal{H}_i^j}$ for each $i \geq 1$ and $j \in \{1, \ldots, \bar{j}_i + 1\}$. Observe that $\mathcal{G}_{i+1}^1 = \mathcal{G}_i^{\bar{j}_i+1}$ and $\mathcal{H}_{i+1}^1 = \mathcal{H}_i^{\bar{j}_i+1}$ for each i.

In Fig. 3 we give an outline of the basic method. The algorithms for MAF-2 are derived from this outline by specifying the operations, cases and transactions. Observe that the outline only outputs the size of a solution; we shall see later how to get the solution itself. It also does not guarantee any upper bound on the size of the solution as yet.

3.3. Operations, cases and transactions

In this section, we complete the description of the basic method by defining the operations and cases that may occur, and also state the transactions to be carried out in each case for each algorithm.

Let $\mathcal{W}$ be a phylogenetic tree. The operation of **cutting an arc e of $\mathcal{W}$**, denoted by **Cut($\mathcal{W}$, $l(e)$)**, is the elimination of e as defined in Section 3.1. The operation of **shrinking a pair of sibling leaves u and v to leaf u**, written as **Srk($\mathcal{W}$, u, v)**, corresponds to the elimination of the arc e such that $l(e) = v$, followed by the removal of the isolated node v.

We note that cuts and shrinks are the only operations used directly by the algorithms. Eliminations are used only to implement these operations, and they are never called directly by the algorithms. Though cuts and eliminations produce the same results, they are not interchangeable as a rule, since some eliminations may come from shrinks and not from cuts.

We now define the cases. If, at some iteration i, the basic method does not halt, then $\mathcal{G}_i$ admits a pair of sibling leaves. Let a and b be its labels.

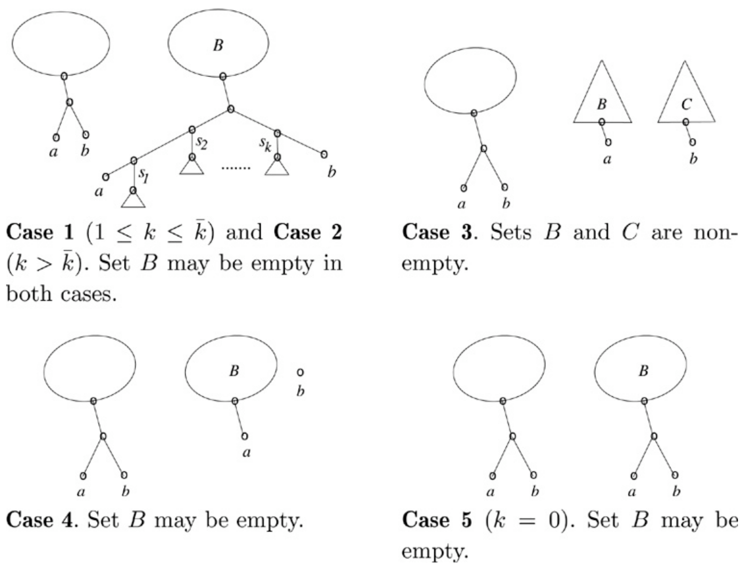
If $f_{\mathcal{H}_i}(a)$ and $f_{\mathcal{H}_i}(b)$ belong to the same component in $\mathcal{H}_i$, then the **(a, b)-axis** in $\mathcal{H}_i$ is the (unique) path in $\mathcal{H}_i$ connecting $f_{\mathcal{H}_i}(a)$ and $f_{\mathcal{H}_i}(b)$, and the **stems** of the (a, b)-axis are the arcs e in $\mathcal{H}_i$ such that $u(e)$ belongs to the axis but e does not. The stems are labeled $s_1, s_2, \ldots, s_k$, according to the order in which they appear along the axis, with stem s_1 being the nearest one to $f_{\mathcal{H}_i}(a)$. The cases are shown in Fig. 4.

The forests $\mathcal{G}_i$ and $\mathcal{H}_i$ may have other components (we indicate in the figure only those containing the leaves a or b). The constant $\bar{k}$ specified in Cases 1 and 2 depends on the algorithm: we have $\bar{k} = 1$ for Algorithm 1 and $\bar{k} = 2$ for Algorithm 2 (see Table 1).

Table 1 describes, for each algorithm, the transactions to be performed in each case. Note that the transactions for Cases 1 and 2 are different for each of these algorithms. In these cases, Algorithm 1 and Algorithm 2 behave equally, except when $k = 2$ (that is, there are two stems on the (a, b)-axis); and Algorithm 3 uses a somewhat hybrid transaction. The transactions for Cases 3, 4 and 5 are identical for all three algorithms.

It will be proved in the analysis that these transactions satisfy the following properties:

- for each transaction performed by the algorithms, an arc that connects two blocks of a refinement $\mathcal{H}'$ of some fixed maximum agreement forest $\mathcal{F}$ of $\mathcal{T}$ and $\mathcal{U}$ is eliminated; and



Case 1 ($1 \leq k \leq \bar{k}$) and **Case 2** ($k > \bar{k}$). Set B may be empty in both cases.

Case 3. Sets B and C are non-empty.

Case 4. Set B may be empty.

Case 5 ($k = 0$). Set B may be empty.

Fig. 4. Cases of the basic method (with respect to the sibling leaves a and b). In each figure, the leftmost restriction is $\mathcal{G}_i$ and the rightmost restriction is $\mathcal{H}_i$.

Table 1
Transactions for Algorithms 1, 2 and 3

Algorithm 1 (Hein's algorithm): $\bar{k} = 1$	
Case 1 ($k = 1$)	Cut($\mathcal{H}_i^j, l(s_j)$), for $j = 1, \dots, k$
Case 2 ($k > 1$)	Cut($\mathcal{G}_i^1, f(a)$); Cut($\mathcal{G}_i^2, f(b)$); Cut($\mathcal{H}_i^1, f(a)$); Cut($\mathcal{H}_i^2, f(b)$)
Case 3	Cut($\mathcal{G}_i^1, f(a)$); Cut($\mathcal{G}_i^2, f(b)$); Cut($\mathcal{H}_i^1, f(a)$); Cut($\mathcal{H}_i^2, f(b)$)
Case 4	Cut($\mathcal{G}_i^1, f(b)$)
Case 5	Srk($\mathcal{G}_i^1, f(a), f(b)$); Srk($\mathcal{H}_i^1, f(a), f(b)$)
Algorithm 2: $\bar{k} = 2$	
Case 1 ($1 \leq k \leq 2$)	Cut($\mathcal{H}_i^j, l(s_j)$), for $j = 1, \dots, k$
Case 2 ($k > 2$)	Cut($\mathcal{G}_i^1, f(a)$); Cut($\mathcal{G}_i^2, f(b)$); Cut($\mathcal{H}_i^1, f(a)$); Cut($\mathcal{H}_i^2, f(b)$)
Case 3	Cut($\mathcal{G}_i^1, f(a)$); Cut($\mathcal{G}_i^2, f(b)$); Cut($\mathcal{H}_i^1, f(a)$); Cut($\mathcal{H}_i^2, f(b)$)
Case 4	Cut($\mathcal{G}_i^1, f(b)$)
Case 5	Srk($\mathcal{G}_i^1, f(a), f(b)$); Srk($\mathcal{H}_i^1, f(a), f(b)$)
Algorithm 3	
Cases 1 and 2	Cut($\mathcal{G}_i^1, f(a)$); Cut($\mathcal{G}_i^2, f(b)$); Cut($\mathcal{H}_i^1, f(a)$); Cut($\mathcal{H}_i^2, f(b)$); Cut($\mathcal{H}_i^3, l(s_1)$)
Case 3	Cut($\mathcal{G}_i^1, f(a)$); Cut($\mathcal{G}_i^2, f(b)$); Cut($\mathcal{H}_i^1, f(a)$); Cut($\mathcal{H}_i^2, f(b)$)
Case 4	Cut($\mathcal{G}_i^1, f(b)$)
Case 5	Srk($\mathcal{G}_i^1, f(a), f(b)$); Srk($\mathcal{H}_i^1, f(a), f(b)$)

- at each iteration, $\mathcal{H}'$ is an agreement forest of $\mathcal{G}$ and $\mathcal{H}$ ($\mathcal{H}'$ is obtained from $\mathcal{F}$ by performing on $\mathcal{F}$'s edges the same operations done on $\mathcal{U}$'s edges).

Fig. 8 shows a sequence of iterations performed by Algorithm 2.

3.4. *Correctness and solution output*

The next lemma guarantees that the algorithms produce an agreement forest of $\mathcal{T}$ and $\mathcal{U}$:

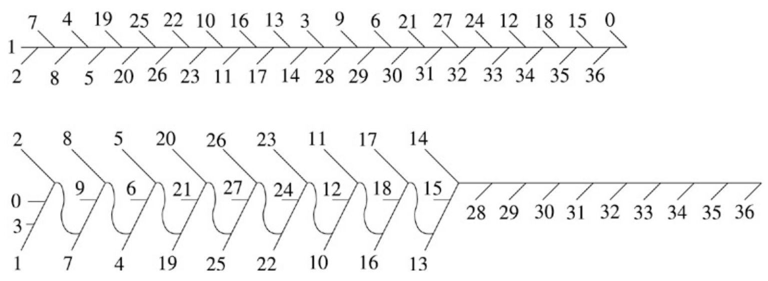


Fig. 5. Instance $\mathcal{I}_9$: An example of the family of instances $\mathcal{I}_m$ which shows that the approximation ratio of Algorithm 1 converges to 4 as m increases.

Lemma 3. *For each iteration $i \geq 1$ of any of the Algorithms 1, 2 and 3, we have:*
(a) $\bigcup \mathcal{S}_{\mathcal{G}_i} = \bigcup \mathcal{S}_{\mathcal{H}_i}$;
(b) *for each component $\mathcal{W}$ of $\mathcal{H}_i$, there exists a component $\mathcal{V}$ of $\mathcal{G}_i$ such that $S_\mathcal{W} \subseteq S_\mathcal{V}$.*

The halting condition of the basic method is the non-existence of a pair of sibling leaves in $\mathcal{G}_i$. Thus, when the algorithm halts, $\mathcal{G}_i$ contains only isolated nodes. The same holds for $\mathcal{H}_i$ because of Lemma 3(b); and since by (a) $\bigcup \mathcal{S}_{\mathcal{G}_i} = \bigcup \mathcal{S}_{\mathcal{H}_i}$, we have that $\mathcal{G}_i$ and $\mathcal{H}_i$ are isomorphic. By re-attaching to $\mathcal{G}_i$ and $\mathcal{H}_i$ all the leaves that were removed from their former location in a shrinking operation, we obtain from $\mathcal{G}_i$ a full restriction of $\mathcal{T}$ and from $\mathcal{H}_i$ a full restriction of $\mathcal{U}$ such that these restrictions are isomorphic. We refer to this set of re-attaching steps as the **re-assembling** task. Any of these two full restrictions can be output by the algorithm if a solution is required, but if we want only the size of an agreement forest then re-assembling is not necessary.

3.5. Time complexity

It is not difficult to conclude that all these algorithms can be implemented to run in polynomial time in the number n of leaves of the phylogenetic trees given as input.

In each iteration, it takes time $O(n)$ to test whether $\mathcal{G}_i$ has a sibling pair. Let m be the number of arcs in $\mathcal{T}$ and $\mathcal{U}$. If $\mathcal{G}_i$ has a sibling pair, then the case must be identified, which takes time $O(m)$ (for testing if the (a, b)-axis exists and for counting the stems), and the respective transaction must be performed, which takes time $O(n)$. Finally, there are $O(m)$ iterations since in each one at least an arc of $\mathcal{T}$ is eliminated. So the time complexity of each algorithm amounts to $O(m^2)$. Re-assembling (if required) just adds another $O(m)$ term to this amount. As our input trees do not admit internal nodes of degree 1, each algorithm has time complexity $O(n^2)$.

4. Approximation ratio

4.1. Lower bounds

We start the approximation ratio analysis by giving lower bounds for the approximation ratios of the algorithms presented in Section 3.

First of all, we exhibit a family of instances $\mathcal{I}_m = (\mathcal{T}_m, \mathcal{U}_m)$, $m \geq 7$, which shows that the approximation ratio of Algorithm 1 is larger than 3. Fig. 5 shows the instance $\mathcal{I}_9$ of this family: the tree $\mathcal{T}_9$ is the one consisting of the long strip (the root is on the right-hand side). In the general case, the tree $\mathcal{T}_m$ is a generalization of $\mathcal{T}_9$: it has the same structure and $4m+1$ leaves labelled from 0 to $4m$. The tree $\mathcal{U}_m$ has two parts: one part consisting of m subtrees (which will be recognized as Cases 2 with $k = 2$) laid each on top of the other; and the other part consisting of m single lined up leaves. The labels of these leaves have to follow the pattern shown in the example shown in Fig. 5 for $m = 9$. The tree $\mathcal{T}_9$ is constructed so as to force Algorithm 1 to handle at first all Cases 2 (cutting leaves 1–2–7–8–$\cdots$–13–14 in $\mathcal{U}_9$ in the example) and then alternate cuts between the stringed leaves (28–29–$\cdots$–35–36) and the remaining leaves from the Cases 2 (0–3–9–$\cdots$–18–15). In the general case, the transactions are similar (the set of transactions is determined unambiguously by the pairs of sibling leaves the algorithm is forced to select in the tree $\mathcal{T}_m$).

It turns out that for any instance $\mathcal{I}_m$ of this family, Algorithm 1 produces an agreement forest with $4m$ components, while there is an agreement forest with at most $m + 2$ components. Indeed, for the instance $\mathcal{I}_9$ shown in Fig. 5, consider the agreement forest which contains leaves 1–2–7–8–$\cdots$–13–14 together with leaves 28–29–$\cdots$–35–36 as a

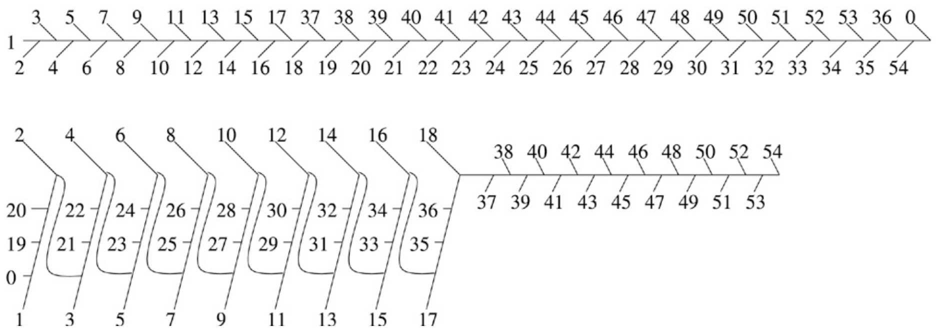


Fig. 6. Instance $\mathcal{I}'_9$: An example of family of instances $\mathcal{I}'_m$ which shows that the approximation ratio of Algorithm 2 converges to 3 as m increases.

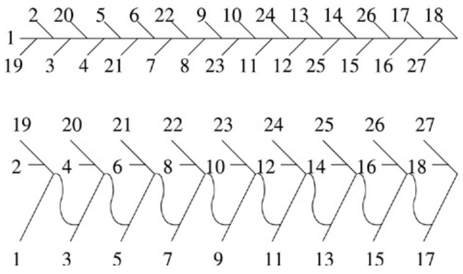


Fig. 7. Instance $\mathcal{I}''_9$: An example of family of instances $\mathcal{I}''_m$ which shows that Algorithm 3 converges to 3 as m increases.

single component and the other leaves as isolated nodes; this agreement forest has 11 components. For this instance, Algorithm 1 produces an agreement forest with 36 components: one consisting of a tree with leaves 15 and 36 and the remaining ones consisting of precisely one leaf. Thus, for $m = 4$ the ratio is 36/11, but as m increases the ratio $4m/(m + 2)$ converges to 4. In Section 5 we show some computational experiments obtained for this instance, from which we can conclude that an optimum solution for this instance has indeed $m + 2$ components.

The instance $\mathcal{I}_9$ can be modified to an instance $\mathcal{I}'_9$ to prove that the approximation ratio 3 of Algorithm 2 is tight. It suffices to use three stems (instead of two) for a sequence of nine Cases 2, as shown in Fig. 6. For this instance, Algorithm 2 outputs an agreement forest with 53 components: one containing leaves 35–53–54, and the remaining ones consisting of precisely one leaf. However, for this instance, there exists an agreement forest with 20 components: nodes 0–19–20–21–22–$\cdots$–35–36 are separated, while the remaining nodes form a single component with 36 leaves. Similarly, a more general instance $\mathcal{I}'_m$ with $6m+1$ leaves can be constructed, for which Algorithm 2 outputs a solution with size $6m - 1$ while the optimum is at most $2m + 2$.

Finally, in Fig. 7 we give an instance $\mathcal{I}''_9$ to show that the approximation ratio 3 of Algorithm 3 is tight. This instance can be generalized in the obvious way to an instance $\mathcal{I}''_m = (\mathcal{T}_m, \mathcal{U}_m)$, with $3m + 1$ leaves, by piling m Cases 2 for $\mathcal{U}_m$, and building $\mathcal{T}_m$ so that these cases are identified, and the leaves cut, in due order.

For the instance $\mathcal{I}''_9$, Algorithm 3 outputs an agreement forest with 27 components, each of them consisting of precisely one leaf; however, there exists a solution for this instance with 10 components which can be obtained by separating nodes 2–4–6–$\cdots$–18 and keeping the remaining nodes in a single component. Similarly, for the instance $\mathcal{I}''_m$ there is an optimal solution with $m + 1$ components, while Algorithm 3 finds a solution with $3m$ components.

In Section 5, we show some computational results for these families of instances.

4.2. *Links*

The analysis of the approximation ratio of the algorithms is based on the following idea. We fix an arbitrary maximum agreement forest, say $\mathcal{F}$, of the input trees $\mathcal{T}$ and $\mathcal{U}$; we then compare the "ideal cuts" induced by $\mathcal{F}$ on $\mathcal{T}$ and $\mathcal{U}$ with the cuts performed by the algorithms. We introduce some definitions in order to get hold of these "ideal cuts".

Let $\mathcal{G}$ be a phylogenetic forest and $\mathcal{S}$ a family of sets that defines a full restriction of $\mathcal{G}$. An arc e of $\mathcal{G}$ is said to be a **link** of $\mathcal{G}$ with respect to $\mathcal{S}$ if e is not in $\mathcal{G}[\mathcal{S}]$. Alternatively, we may say that e is a link with respect to $\mathcal{G}|\mathcal{S}$. Observe

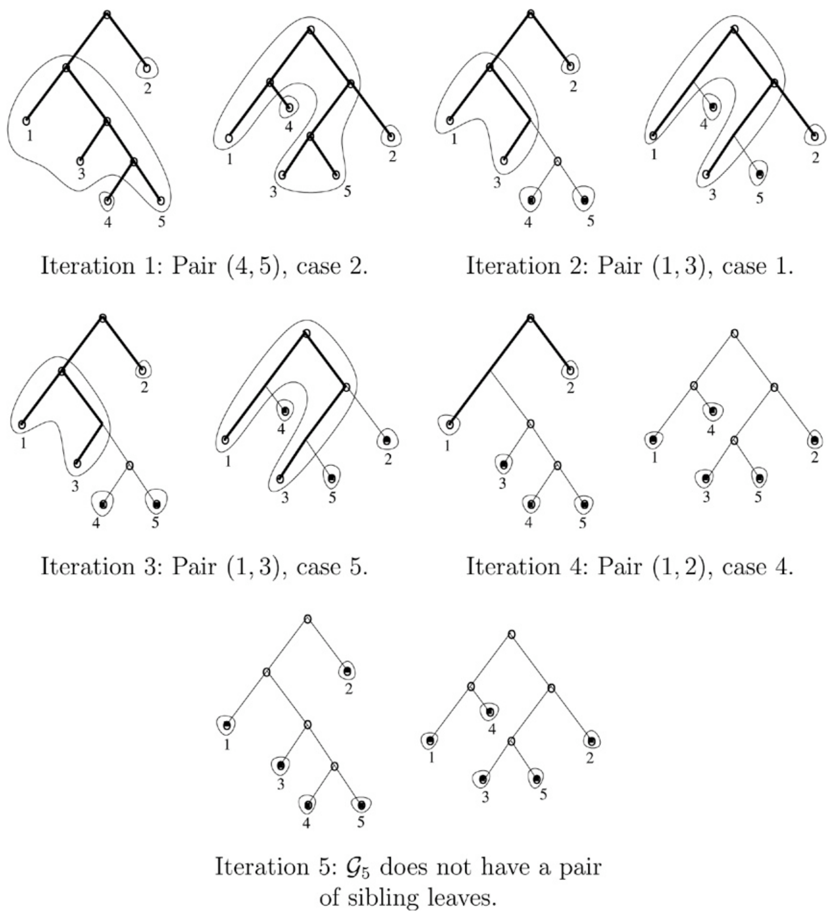

Iteration 1: Pair (4, 5), case 2.
Iteration 2: Pair (1, 3), case 1.
Iteration 3: Pair (1, 3), case 5.
Iteration 4: Pair (1, 2), case 4.
Iteration 5: $\mathcal{G}_5$ does not have a pair of sibling leaves.

Fig. 8. An execution of Algorithm 2.

that the definition of a link depends on the full restriction one is considering; hereafter, these full restrictions will be clear from the context and therefore not explicitly written out.

The following two lemmas give further characterizations of links (the second lemma was used as the definition for links in a previous version of this paper [7]).

Lemma 4. *Let $\mathcal{G}$ be a phylogenetic forest, $\mathcal{S}$ a family of subsets of $\bigcup \mathcal{S}_\mathcal{G}$ that defines a full restriction of $\mathcal{G}$, and e an arc of $\mathcal{G}$. Then e is a link with respect to $\mathcal{S}$ if and only if for every component $\mathcal{V}$ of $\mathcal{G}|\mathcal{S}$ we have $S_\mathcal{V} \cap D_e = \emptyset$ or $S_\mathcal{V} \setminus D_e = \emptyset$.*

Lemma 5 ([7]). *Let $\mathcal{G}$ be a phylogenetic forest, $\mathcal{S}$ a family of subsets of $\bigcup \mathcal{S}_\mathcal{G}$ that defines a full restriction of $\mathcal{G}$ and e an arc of $\mathcal{G}$. Then e is a link with respect to $\mathcal{S}$ if and only if there are $m \geq 1$ components $\mathcal{V}_1, \ldots, \mathcal{V}_m$ of $\mathcal{G}|\mathcal{S}$ such that $\bigcup_{1 \leq i \leq m} L_{\mathcal{V}_i} = D_e$.*

Another property of links, which is important for the analysis, is stated in the following:

Lemma 6. *Let $\mathcal{H}$ be a phylogenetic forest and $\mathcal{S}$ a family of sets that defines a full restriction of $\mathcal{H}$. If a sequence of $|\bigcup \mathcal{S}_\mathcal{H}| - |\mathcal{H}|$ eliminations is performed on $\mathcal{H}$, then $|\mathcal{S}| - |\mathcal{H}|$ links with respect to $\mathcal{S}$ are eliminated.*

Proof. The proof is by induction on $|\bigcup \mathcal{S}_\mathcal{H}| - |\mathcal{H}|$. If this is zero, then no elimination is performed on $\mathcal{H}$ and $|\mathcal{S}| - |\mathcal{H}| = 0$. Otherwise, at least one elimination is performed. In this case, let $\mathcal{H}'$ be the phylogenetic forest obtained after the first elimination. If the eliminated arc e is a link then let $\mathcal{S}' := \mathcal{S}$; otherwise the family $\mathcal{S}$ has a set that can be split into two parts in order to get a set family $\mathcal{S}'$ that defines a full restriction on $\mathcal{H}$ and preserves the links with respect to $\mathcal{S}$ (see Lemmas 1 and 4). We now have $|\bigcup \mathcal{S}_{\mathcal{H}'}| - |\mathcal{H}'| = |\bigcup \mathcal{S}_\mathcal{H}| - |\mathcal{H}| - 1$, so by the induction hypothesis $|\mathcal{S}'| - |\mathcal{H}'|$ links are eliminated on $\mathcal{H}'$ after performing the remaining eliminations on it. If e is a link with respect to $\mathcal{S}$, then there are $|\mathcal{S}'| - |\mathcal{H}'| + 1 = |\mathcal{S}'| - (|\mathcal{H}'| - 1) = |\mathcal{S}| - |\mathcal{H}|$ links eliminated in $\mathcal{H}$; otherwise there are $|\mathcal{S}'| - |\mathcal{H}'| = (|\mathcal{S}| + 1) - (|\mathcal{H}| + 1) = |\mathcal{S}| - |\mathcal{H}|$ links eliminated in $\mathcal{H}$. □

The result mentioned above shows the relationship between the arcs eliminated by the algorithms and the aforementioned "ideal cuts" (eliminated links). Thus, for a given pair $(\mathcal{T}, \mathcal{U})$ of trees, if $\mathcal{F}$ is a maximum agreement forest of $\mathcal{T}$ and $\mathcal{U}$, then Lemma 6 guarantees that the algorithms eliminate exactly $|\mathcal{F}| - 1 = |\mathcal{S}_{\mathcal{F}}| - 1$ links in $\mathcal{U}$. The next section addresses the question of how to "recognize" these links in the analysis of the performance of the algorithms.

4.3. A property of the basic method

In order to make use of Lemma 6, we need first to provide a full restriction of $\mathcal{G}_i^j$ and $\mathcal{H}_i^j$ for each $i \geq 1$ and $j \in \{1, \ldots, \bar{\jmath}_i + 1\}$. A convenient way of doing this is to consider a maximum agreement forest $\mathcal{F}$ of the trees $\mathcal{T}$ and $\mathcal{U}$ at the beginning of the execution of the basic method and to perform suitable eliminations on its arcs along with those done on $\mathcal{T}$ and $\mathcal{U}$ by the transactions.

We define the notation to access the status of $\mathcal{F}$ at any time of an execution in the same way as we have proceeded with $\mathcal{T}$ and $\mathcal{U}$. Let $\boldsymbol{\mathcal{H}'_1}$ be the restriction of $\mathcal{U}$ isomorphic to $\mathcal{F}$ and for $i \geq 2$, let $\boldsymbol{\mathcal{H}'_i}$ be the restriction we get from $\mathcal{H}'_{i-1}$ after executing on it some "suitable" eliminations at iteration i. Let $\boldsymbol{\mathcal{H}'^1_i} := \mathcal{H}'_i$ and for $j \in \{2, \ldots, \bar{\jmath}_i + 1\}$, let $\boldsymbol{\mathcal{H}'^j_i}$ be $\mathcal{H}'^{j-1}_i$ if the arc e which is eliminated when transforming $\mathcal{H}_i^{j-1}$ into $\mathcal{H}_i^j$ is a link with respect to $\mathcal{H}'^j_i$; otherwise let $\mathcal{H}'^j_i$ be the result of $\mathrm{Elim}(\mathcal{H}'^{j-1}_i, l(e))$. If the respective operation performed in $\mathcal{U}$ is a shrinking, the resultant isolated node in $\mathcal{H}'^j_i$ is also removed.

Fig. 8 exemplifies how the forests $\mathcal{H}'_i$ evolve as Algorithm 2 proceeds. In this figure, $\mathcal{G}_i$ and $\mathcal{H}_i$ are indicated by thick lines and the components of $\mathcal{H}'_i$ are indicated by curve lines. Note that in the second iteration the arc that is cut is a link.

It is not difficult to prove the following lemma.

Lemma 7. *For each $i \geq 1$ and $j \in \{1, \ldots, \bar{\jmath}_i + 1\}$, $\mathcal{H}'^j_i$ is a full restriction of $\mathcal{H}_i^j$ and is isomorphic to a full restriction of $\mathcal{G}_i^j$.*

Proof. The proof is by induction on (i, j). If $i = j = 1$, then $\mathcal{H}'^1_1 = \mathcal{F}$, which is an agreement forest of $\mathcal{G}_1^1 = \mathcal{T}$ and $\mathcal{H}_1^1 = \mathcal{U}$. If $i > 1$ or $j > 1$, it suffices to prove for $j \geq 2$. If the arc e eliminated by the operation that produces $\mathcal{H}_i^j$ from $\mathcal{H}_i^{j-1}$ is a link, then $\mathcal{H}'^j_i$ is a full restriction of $\mathcal{H}_i^{j-1}$ by the induction hypothesis ($\mathcal{H}'^{j-1}_i$ is a full restriction of $\mathcal{H}_i^{j-1}$) and by the definition of $\mathcal{H}'^j_i$. Since $\mathcal{H}_i^j$ is obtained by eliminating the arc e in $\mathcal{H}_i^{j-1}$ and this arc is a link with respect to $\mathcal{H}'^j_i$, we have that $\mathcal{H}'^j_i$ is a full restriction of $\mathcal{H}_i^j$.

If the arc e eliminated by the operation that produces $\mathcal{H}_i^j$ from $\mathcal{H}_i^{j-1}$ is not a link, then $\mathcal{H}'^j_i$ is produced by the operation $\mathrm{Elim}(\mathcal{H}'^{j-1}_i, l(e))$. In this case, by Lemma 2 we have that $\mathcal{H}'^j_i$ is a full restriction of $\mathcal{H}'^{j-1}_i$. By the induction hypothesis, $\mathcal{H}'^{j-1}_i$ is a full restriction of $\mathcal{H}_i^{j-1}$, so $\mathcal{H}'^j_i$ is a full restriction of $\mathcal{H}_i^{j-1}$. As in the other case, since the arc e in $\mathcal{H}_i^{j-1}$ is a link with respect to $\mathcal{H}'^j_i$ and $\mathcal{H}_i^j$ is the result of the operation $\mathrm{Elim}(\mathcal{H}_i^{j-1}, l(e))$, then $\mathcal{H}'^j_i$ is a full restriction of $\mathcal{H}_i^j$. □

The next result relates the links with the cuts made by the algorithms.

Lemma 8. *Let $\mathcal{G}$ and $\mathcal{H}$ be two phylogenetic forests with $\bigcup \mathcal{S}_{\mathcal{G}} = \bigcup \mathcal{S}_{\mathcal{H}}$, and let $\mathcal{H}'$ be a full restriction of $\mathcal{H}$ that is isomorphic to a full restriction of $\mathcal{G}$. Suppose that $f_{\mathcal{G}}(a)$ and $f_{\mathcal{G}}(b)$ form a pair of sibling leaves in $\mathcal{G}$.*

- *If $f_{\mathcal{H}'}(a)$ and $f_{\mathcal{H}'}(b)$ are in different components of $\mathcal{H}'$ and $f_{\mathcal{H}}(a)$ and $f_{\mathcal{H}}(b)$ are not isolated, then at least one of the arcs incident to $f_{\mathcal{H}}(a)$ and $f_{\mathcal{H}}(b)$ is a link.*
- *If $f_{\mathcal{H}'}(a)$ and $f_{\mathcal{H}'}(b)$ are in the same component of $\mathcal{H}'$ and the (a, b)-axis in $\mathcal{H}$ admits at least one stem, then all of its stems are links.*

Proof. The first item is a consequence of Lemma 1 and of the definition of link. Let us prove the second item.

If $f_{\mathcal{H}'}(a)$ and $f_{\mathcal{H}'}(b)$ are in the same component of $\mathcal{H}'$, then $f_{\mathcal{H}}(a)$ and $f_{\mathcal{H}}(b)$ are in the same component of $\mathcal{H}$ and thus the (a, b)-axis exists. Let e be a stem of this axis.

If e is not a link, then by Lemmas 1 and 4, there exists a unique component $\mathcal{W}$ of $\mathcal{H}'$ such that $S_{\mathcal{W}} \cap D_e \neq \emptyset$ and $S_{\mathcal{W}} \setminus D_e \neq \emptyset$. Let c be the label of a leaf in $S_{\mathcal{W}} \cap D_e$. This component must cover nodes $f_{\mathcal{H}'}(a)$ and $f_{\mathcal{H}'}(b)$. Thus in $\mathcal{G}$ we have $\mathrm{lca}_{\mathcal{G}}(f_{\mathcal{G}}(a), f_{\mathcal{G}}(b))$ as a strict descendent of $\mathrm{lca}_{\mathcal{G}}(f_{\mathcal{G}}(a), f_{\mathcal{G}}(c))$ and $\mathrm{lca}_{\mathcal{G}}(f_{\mathcal{G}}(b), f_{\mathcal{G}}(c))$, while in $\mathcal{H}$ we have $\mathrm{lca}_{\mathcal{H}}(f_{\mathcal{H}}(a), f_{\mathcal{H}}(b))$ as a strict ancestor of $\mathrm{lca}_{\mathcal{H}}(f_{\mathcal{H}}(a), f_{\mathcal{H}}(c))$ or $\mathrm{lca}_{\mathcal{H}}(f_{\mathcal{H}}(b), f_{\mathcal{H}}(c))$. This however is absurd since $\mathcal{H}'$ is an agreement forest of $\mathcal{G}$ and $\mathcal{H}$. $\square$

Lemma 7 implies that, at each iteration i of the basic method, $\mathcal{H}'_i$ is an agreement forest of $\mathcal{G}_i$ and $\mathcal{H}_i$. So we can apply Lemma 8 for $\mathcal{G}_i$, $\mathcal{H}_i$ and $\mathcal{H}'_i$ and conclude that, at each iteration i of the basic method, either at least one of the arcs incident to $f_{\mathcal{H}_i}(a)$ and $f_{\mathcal{H}_i}(b)$ is a link, or all stems (if the (a, b)-axis admits any) are links.

Given that Lemma 6 relates the size of an optimum solution to the number of links that are eliminated, and that the size of the solutions yielded by the algorithms is related to the number of cuts they perform (it is in fact that number plus 1), it remains to analyze the relation between cuts and links at each iteration.

4.4. Performance ratio analysis of Algorithm 3

A simple observation is that any algorithm for MAF-2 derived from the basic method whose transactions *cut at most p arcs among which at least $q \geq 1$ of them are links* is a p/q-approximation algorithm. This observation has the next result as a corollary:

Theorem 9. *Algorithm* 3 *is a 3-approximation algorithm for MAF-2.*

Proof. It suffices to verify that for $p = 3$ and $q = 1$ the fact stated above is true for Algorithm 3. $\square$

4.5. The charging protocol

The common framework we adopted to describe Algorithms 1 and 2, where the constant $\bar{k}$ is used to distinguish them, allows us to analyze them together. With that purpose, we henceforth refer to them together as Algorithm 1–2 and consider $\bar{k}$ as a parameter of this algorithm.

For Algorithm 1–2, the observation stated in Section 4.4 does not hold: there may be iterations in which cuts are performed but no link is eliminated. However, Lemma 8 guarantees that in such situations there still exists a set of links next to each arc that is cut. In our analysis these link sets make for the lack of eliminated links.

To implement this idea, we use a charging protocol. We suppose that there is a fee to be paid each time the algorithm performs a cut, while shrinks are performed for free. Each fee is paid with a **credit unit**, and at the beginning of any execution an initial amount of credit units is allotted to the algorithm, so that it can pay for the cuts it performs. In order to allow for the momentary lack of credit units, the algorithm is permitted to issue **debt units** and to use them to pay fees. At the end of the execution, the algorithm must have all debt units paid. We consider that one credit unit pays one debt unit. It turns out that the initial amount of credit is related to the size of an optimum solution for MAF while the total fee is related to the size of the algorithm's solution. The goal of the analysis is therefore to ensure that this protocol is consistent and that the total amount of credit units suffices to pay the total fee. This analysis technique elaborates the sketch given in [4], but here the control over debt units is improved in order to prevent their clustering, a feature that is lacking in [4].

The following sections state the charging protocol in full. This protocol has two parts, a credit protocol and a debt protocol. Let α be the approximation ratio of Algorithm 1–2 and $\mathcal{F}$ be a maximum agreement forest of $\mathcal{T}$ and $\mathcal{U}$.

Credit protocol

At the beginning of the first iteration, $\alpha(|\mathcal{F}| - 1)$ credit units are **placed** on the unique component of $\mathcal{H}^1_1$, that is $\mathcal{U}$. Throughout the execution, these units are **kept** by the components of $\mathcal{H}^j_i$. When there is an elimination in a component of $\mathcal{H}^j_i$, some of the credit units allotted to that component are **released** and the remaining is **redistributed** between the two resultant components and placed on them. Some of the released units may be used to pay debt units or fees, and the remainder (if any) **expire**. The releasing and redistribution is done according to the following rules.

For each $i \geq 1$ and $j \in \{1, \ldots, \bar{j}_i\}$, let $\mathcal{W}$ be the component of $\mathcal{H}_i^j$ that is split at the j-th operation of the i-th iteration, and let e be the arc which is eliminated thereat. Consider the following definitions:

$m^{\mathcal{W}} :=$ number of components $\mathcal{W}'$ of $\mathcal{H'}_i^j$ such that $S_{\mathcal{W}'} \subseteq S_{\mathcal{W}}$;

$$m^{\cap} := \begin{cases} 1 \text{ if the operation performed in } \mathcal{U} \text{ is a shrinking;} \\ \text{number of components } \mathcal{W}' \text{ of } \mathcal{H'}_i^{j+1} \text{ such that } S_{\mathcal{W}'} \subseteq D_e \\ \text{if the operation performed in } \mathcal{U} \text{ is a cut.} \end{cases}$$

$m^{\setminus} :=$ number of components $\mathcal{W}'$ of $\mathcal{H'}_i^{j+1}$ such that $S_{\mathcal{W}'} \subseteq S_{\mathcal{W}} \setminus D_e$.

The $\alpha(m^{\mathcal{W}} - 1)$ credit units kept by $\mathcal{W}$ are divided between $\mathcal{W}|D_e$ and $\mathcal{W}|S_{\mathcal{W}} \setminus D_e$ according to the following rules:

- if e is a link, then $\alpha(m^{\cap} - 1)$ units are placed on $\mathcal{W}|D_e$, $\alpha(m^{\setminus} - 1)$ units are placed on $\mathcal{W}|S_{\mathcal{W}} \setminus D_e$ and α units are released;
- if e is not a link, then $\alpha(m^{\cap} - 1)$ units are placed on $\mathcal{W}|D_e$, $\alpha(m^{\setminus} - 1)$ units are placed on $\mathcal{W}|S_{\mathcal{W}} \setminus D_e$ and no units are released.

It can be easily verified that this protocol for releasing and redistributing credit units preserves such units throughout the execution of the algorithm.

Debt protocol

The credit protocol ties credit units to the components of $\mathcal{U}$ that are yielded along the execution of the algorithm. Debt units are likewise tied to another element of the algorithm's data structure by the debt protocol. In the charging protocol, debt units are issued whenever the algorithm lacks credit units to pay fees. According to the protocol (provided it is correct and the initial credit amount covers the total fee), this can only happen at iterations with no link elimination. However, we have observed before that in such situations, there is a set of links next to each arc that is cut. We need some definitions in order to specify these sets.

Let $\mathcal{W}$ be a phylogenetic tree, and let B be a set of nodes of $\mathcal{W}$ with at least two nodes. Denote by **$\mathrm{UP}_{\mathcal{W}}$ (B)** the arc set whose upper endpoints are the lcas in $\mathcal{W}$ of at least two nodes of B. If for some i with $i \geq 2$, iteration $i - 1$ is case 2, then let the **residue of the stems** be the arc set $\mathrm{UP}_{\mathcal{H}_i}(\{l(s_h) : 1 \leq h \leq k\})$ where $s_1, s_2, \ldots, s_k$ are the stems of the (a, b)-axis. Fig. 9 shows an example.

If for some iteration $i \geq 1$ there are cuts but no link elimination, then one of the following two situations applies:

(1) The iteration is case 1 and s_h is not a link in $\mathcal{H}_{i-1}^h$ for $h \in \{1, \ldots, \bar{k}\}$. In this case, Lemma 8 guarantees that at least one of the arcs incident to $f_{\mathcal{H}_{i-1}}(a)$ and $f_{\mathcal{H}_{i-1}}(b)$ is a link, and the algorithm **associates** the at most $\bar{k}$ debt units issued in this iteration (see Table 1) with this link.

(2) The iteration is case 2 and $f_{\mathcal{H}_{i-1}}(a)$, $f_{\mathcal{H}_{i-1}}(b)$ are in a component of $\mathcal{H}'_{i-1}$ which has at least three leaves. In this situation Lemma 8 guarantees that all arcs of the stem residues are links, and the algorithm **associates** the two issued debt units (see Table 1) with this stem residue.

It remains to specify how debt units are handled at eliminations. In order to do this and to see that the protocol is fully consistent, we need to give a few properties kept by stem residues as they are split by further eliminations.

Barriers

Take some $\mathcal{H}_i^j$ with $(i, j) \neq (1, 1)$. Consider arc e in Fig. 2 for the elimination performed to produce $\mathcal{H}_i^j$. A node set B in $\mathcal{H}_i^j$ with at least two nodes is a **barrier** if:

(1) B is a **primary barrier**; that is, $j = 1$; iteration $i - 1$ is case 2 with $f_{\mathcal{H}_{i-1}}(a)$ and $f_{\mathcal{H}_{i-1}}(b)$ in a component of $\mathcal{H}'_{i-1}$ which has at least three leaves; $B = \{l(s_h) : 1 \leq h \leq k\}$; or

(2) B admits an **antecessor barrier** B'; that is, $j \geq 2$ and there exists a barrier B' in $\mathcal{H}_i^{j-1}$ such that one of the following conditions holds:

(a) $e \in \mathrm{UP}_{\mathcal{H}_i^{j-1}}(B')$ and $B = B' \cap D_e$ or $B = B' \setminus D_e$;

(b) $e \notin \mathrm{UP}_{\mathcal{H}_i^{j-1}}(B')$ and $B = B'$.

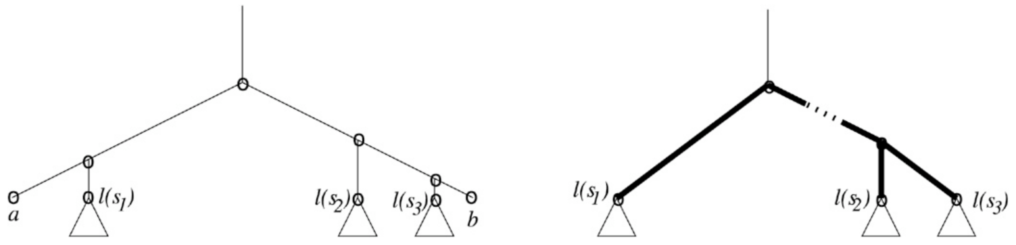


Fig. 9. A primary barrier.

Fig. 9 has an example of a primary barrier. On the left, we have $\mathcal{H}_{i-1}$, with the case 2 situation depicted, and on the right we can see the residue of the stems and the primary barrier.

We have observed before that all arcs of primary barriers (seen as arc sets rather than node sets, and thus corresponding to the stem residues) are links. This is also true for barriers with antecessors, according to the following property.

Lemma 10. *Let $\mathcal{H}_i^j$ be such that $(i, j) \neq (1, 1)$, and let B be a barrier of $\mathcal{H}_i^j$. Then all arcs in $\mathrm{UP}_{\mathcal{H}_i^j}(B)$ are links.*

According to the debt protocol, each primary barrier has some debt units associated with it. When an arc belonging to a barrier is eliminated, this barrier is split in two halves that become new barriers (the barrier which has been split being their antecessor). The released credit units are used to pay any fees and also a positive **quota** of the debt associated with the antecessor barrier, and the remaining quotas are further associated with the two newly created barriers. Since barriers shrink to singletons as Algorithm 1–2 proceeds and since all quotas must be paid when the algorithm terminates, it is necessary to reach a balance between the size and number of quotas and the minimum size of a primary barrier. Since each primary barrier of size k undergoes $k - 1$ credit releases, and the primary barriers for Algorithm 1–2 have size greater than or equal to $\bar{k} + 1$ and at most two associated debt units, then the size of each quota is at most $2/\bar{k}$.

Now the charge protocol is fully stated. To assert its consistency, we must still guarantee that barriers do not merge (at the forced contractions), so that debt units do not cluster. The next lemma asserts this.

Lemma 11. *In any $\mathcal{H}_i^j$ with $(i, j) \neq (1, 1)$, consider two distinct barriers B and C. Then $\mathrm{UP}_{\mathcal{H}_i^j}(B)$ and $\mathrm{UP}_{\mathcal{H}_i^j}(C)$ are arc-disjoint.*

Proof. The proof is by induction on the number of operations performed since the most recent barrier, B or C, was created. The basis of the induction is when this number is zero (that is, B is primary or C is primary).

Suppose without loss of generality that B is the most recent barrier and that it is primary. Then:

- according to the definition of primary barrier, two operations were performed starting from $\mathcal{H}_{i-1}$ in the iteration $i - 1$ (that is, $\bar{j}_{i-1} = 2$);
- C admits an antecessor barrier C' in $\mathcal{H}_{i-1}^2$, and C' admits an antecessor barrier C'' in $\mathcal{H}_{i-1}^1 = \mathcal{H}_{i-1}$.

Fig. 10 sketches the barrier $\mathcal{C}''$ in $\mathcal{H}_{i-1}$, showing the two possible cases in which B is a primary barrier in $\mathcal{H}_i$. Set $\mathrm{UP}_{\mathcal{H}_{i-1}}(C'')$ is shown in thick lines.

Consider the (a, b)-axis in $\mathcal{H}_{i-1}$, and suppose that the path connecting $\mathrm{lca}(a, b)$ and $r_{\mathcal{H}_{i-1}}$ does not contain the node $\mathrm{lca}(C'')$ (Fig. 10, left). According to the definition of primary barrier, all the arcs of the (a, b)-axis are in the same component in $\mathcal{H}'_{i-1}$, and therefore none of the arcs in $\mathrm{UP}_{\mathcal{H}_{i-1}}(C'')$ (which are links; see Lemma 10) is in the axis. Since the arc whose lower endpoint is $\mathrm{lca}(C'')$ is a link, then $\mathrm{UP}_{\mathcal{H}_i^j}(B)$ and $\mathrm{UP}_{\mathcal{H}_i^j}(C)$ are arc-disjoint. The proof is similar if the path connecting $\mathrm{lca}(a, b)$ and $r_{\mathcal{H}_{i-1}}$ contains $\mathrm{lca}(C'')$ (Fig. 10, right).

In the induction step one has to analyse when both B and C admit antecessor barriers. The lemma can be easily verified for all the cases of the definition of antecessor barrier. □

4.6. *Performance ratio analysis of Algorithms 1 and 2*

The analysis of Algorithm 1–2 considers four possible situations that can occur at each iteration:

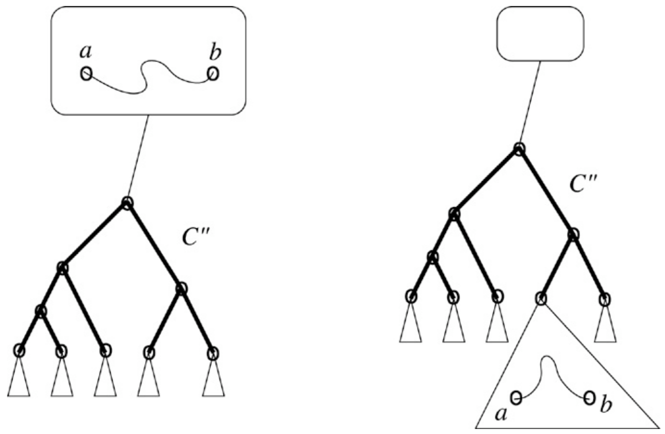


Fig. 10. Restriction $\mathcal{H}_{i-1}$ and antecessor barrier C'' in the proof of Lemma 11.

- case 5 and no link elimination;
- case 1 and no link elimination;
- case 2 and no link elimination;
- at least one link elimination.

In the first case, there are no new debt units since shrinks require no fee to be paid.

In the second case, according to the debt protocol, at most $\bar{k}$ debt units are associated with a single link. Without loss of generality, this link is eliminated at an iteration case 5.

In the third case, the debt protocol states that a primary barrier holds the two debt units issued thereat, so that this debt is paid as the barrier is split. We have seen that the size of the quota is at most $2/\bar{k}$.

Let us now consider the fourth case. Each link elimination should release enough credit to pay at most $\max(2, \bar{k})$ fees plus a quota of at most $2/\bar{k}$ debt units. Thus,

$$\alpha \geq \max(2, \bar{k}) + 2/\bar{k},$$

and Algorithm 1–2 is a $(\max(2, \bar{k}) + 2/\bar{k})$-approximation algorithm for MAF-2. The minimum ratio is reached for $\bar{k} = 2$, which yields approximation ratio 3 for Algorithm 2 while Algorithm 1 (Hein's algorithm) has approximation ratio 4. Thus combining these results with the lower bounds (family of instances) for algorithms 1 and 2 shown in Section 4.1, we have the following:

Theorem 12. *Algorithm* 1 *is a 4-approximation algorithm for MAF-*2*. Furthermore, the performance ratio* 4 *of this algorithm is tight.* □

Theorem 13. *Algorithm* 2 *is a 3-approximation algorithm for MAF-*2*. Furthermore, the performance ratio* 3 *of this algorithm is tight.* □

5. Computational experiments

We have implemented and tested four algorithms for MAF-2: Algorithm 1, Algorithm 2, Algorithm 3, and a variation of Algorithm 3 which we call here Algorithm 4.

Algorithm 4 is built using the basic method, and its transactions are listed in Table 2. For cases 1, 2, 4 and 5 of the basic method, Algorithm 4 uses the same transactions as Algorithm 3. For case 3, after cutting off leaves $f(a)$ in $\mathcal{H}_i^1$ and $f(b)$ in $\mathcal{H}_i^2$, a third non-isolated leaf (if any), labeled c, is cut off in $\mathcal{H}_i^3$. It is not difficult to see that Algorithm 4 is a 3-approximation algorithm for MAF-2: the same reasoning used to prove Theorem 9 proves this. It is also clear that the solutions output by Algorithm 4 must have size greater than or equal to that of the solutions output by Algorithm 3 (it suffices to note that Algorithm 4 performs an additional cut in case 3).

Table 2
Transactions for Algorithm 4

Algorithm 4	
Cases 1 and 2	$\mathrm{Cut}(\mathcal{G}_i^1, f(a))$; $\mathrm{Cut}(\mathcal{G}_i^2, f(b))$; $\mathrm{Cut}(\mathcal{H}_i^1, f(a))$; $\mathrm{Cut}(\mathcal{H}_i^2, f(b))$; $\mathrm{Cut}(\mathcal{H}_i^3, l(s_1))$
Case 3	$\mathrm{Cut}(\mathcal{G}_i^1, f(a))$; $\mathrm{Cut}(\mathcal{G}_i^2, f(b))$; $\mathrm{Cut}(\mathcal{H}_i^1, f(a))$; $\mathrm{Cut}(\mathcal{H}_i^2, f(b))$; $\mathrm{Cut}(\mathcal{H}_i^3, f(c))$
Case 4	$\mathrm{Cut}(\mathcal{G}_i^1, f(b))$
Case 5	$\mathrm{Srk}(\mathcal{G}_i^1, f(a), f(b))$; $\mathrm{Srk}(\mathcal{H}_i^1, f(a), f(b))$

Our motivation for implementing Algorithm 4 was to obtain tighter lower bounds for the optimal solutions of the instances we have tested. As we noted, since the number of cuts performed by Algorithm 4 is at most 3 times the number of cuts an optimal solution requires, we have that

$$|\,\text{solution of Algorithm 4}\,| - 1 \leq 3\,(|\,\text{optimal solution}\,| - 1).$$

The -1 in the left-hand side of the above inequality is needed because the algorithm outputs the number of components of the agreement forest that it finds, which is one unit larger than the number of cuts this algorithm performs; a similar argument explains the -1 in the right-hand side of the inequality.

Thus, we can conclude that

$$\mathrm{lb} = \lceil (|\,\text{solution of Algorithm 4}\,| + 2)/3 \rceil$$

is a lower bound for the optimal solutions. The same argument also holds for Algorithm 3, but since $|\,\text{solution of Algorithm 4}\,| \geq |\,\text{solution of Algorithm 3}\,|$, Algorithm 4 provides better lower bounds (than those that we could obtain with Algorithm 3). This lower bound, lb, is shown in Tables 3 and 4 for the instances we considered in the experiments.

We have run all four algorithms for randomly generated trees and also for the instances $\mathcal{I}_m$, $\mathcal{I}'_m$ and $\mathcal{I}''_m$, which we constructed to show that the performance ratios we have shown for these algorithms are tight.

The random trees in the instances we have tested were generated with the following algorithm. The user enters the number n of labels of the instance and the number m of **subtree transfers** that are to be performed in the generation algorithm. At the beginning of each iteration $k \geq 1$ of the tree generation algorithm, we have a random tree W_k with k leaves. To obtain a random tree with $k+1$ leaves at the end of iteration k, the algorithm chooses randomly a node x of W_k and appends to W_k a new leaf above x, as seen in Fig. 11. After $n-1$ iterations, the algorithm produces a tree with n leaves. It then proceeds by making a copy $\mathcal{V}$ of this tree and performing m subtree transfers on $\mathcal{V}$. Each subtree transfer is defined by a pair (x, y) of randomly chosen nodes, satisfying the following restrictions:

$$x \neq r_{\mathcal{V}} \text{ and } y \neq p(x).$$

The tree W_n plus its copy $\mathcal{V}$, to which a number of subtree transfers has been applied, constitute an instance of MAF-2 for which an upper bound for the size of the optimal solutions $(m+1)$ is known beforehand.

We have generated 10 such instances with 1000 labels and 399 subtree transfers each, numbered 001 through 010, taking snapshots of each tree pair after the 9th, 49th, 99th, 199th and 399th subtree transfers. In this way, we have obtained 50 instances partitioned into 5 groups with 10 instances each, such that an upper bound for the optimal solution of, respectively, 10, 50, 100, 200 and 400 is known beforehand for each group. Tables 3 and 4 list some results obtained for such instances.

We point out the following observations with respect to the performance of the algorithms for randomly generated trees (Tables 3 and 4).

- Algorithm 2 performs no worse than Algorithm 1 in all cases. Algorithm 2 also performs systematically better than Algorithm 1 (meaning that it produces approximate solutions closer to the optimum) when the number of subtree transfers is greater than or equal to 99 for trees with 1000 leaves.
- Although the performance ratio of Algorithm 3 is better than the performance ratio of Algorithm 1, the solutions found by Algorithm 1 – for all instances we have tested – were systematically better than the solutions found by Algorithm 3.

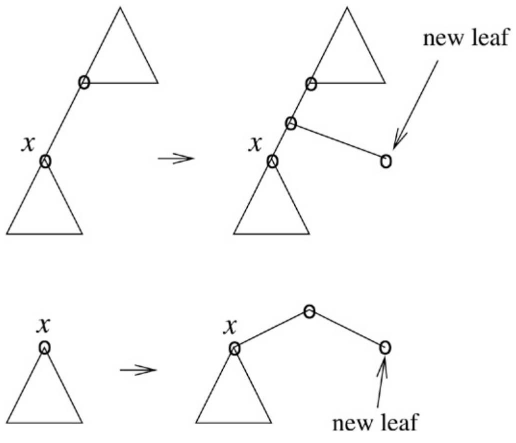


Fig. 11. Adding a new node to a tree.

Table 3
Test results for randomly generated trees (part 1)

File	Alg.2	Alg.1	Alg.3	Alg.4	lb	(Alg.2)/lb
Optimum $\leq$ 10 (9 subtree transfers)						
001–010	15	15	28	28	10	1.50
002–010	15	15	28	28	10	1.50
003–010	16	16	28	28	10	1.60
004–010	15	15	27	28	10	1.50
005–010	13	13	28	28	10	1.30
006–010	16	16	28	28	10	1.60
007–010	16	16	28	28	10	1.60
008–010	14	14	25	25	9	1.56
009–010	13	13	24	25	9	1.45
010–010	13	13	28	28	10	1.30
Optimum $\leq$ 50 (49 subtree transfers)						
001–050	75	78	134	139	47	1.60
002–050	77	78	137	141	48	1.61
003–050	75	81	138	145	49	1.54
004–050	70	70	134	139	47	1.49
005–050	69	69	130	135	46	1.50
006–050	70	70	126	136	46	1.53
007–050	75	75	137	146	50	1.50
008–050	70	73	134	139	47	1.49
009–050	73	73	129	139	47	1.56
010–050	74	77	135	145	49	1.52

- It can be seen from the seventh column of these tables ((Alg.2)/lb) that Algorithm 2 produces solutions not greater than 2 times the optimum when up to 199 subtree transfers are performed on trees with 1000 leaves each, and that this ratio gets close to 1.5 as the number of subtree transfers decreases.
- For a very small subtree transfer number per leaf number ratio, the lower bound of Algorithm 4 comes close to the size of the optimal solution.

In what follows we show the computational results obtained with the instances $\mathcal{I}_m$, $\mathcal{I}'_m$ and $\mathcal{I}''_m$, which we mentioned in Section 4.1.

The results exhibited in Table 5 for the instance $\mathcal{I}_m$ (see Fig. 5) show that, in fact, the ratio 4 of Algorithm 1 (of Hein et al.) is tight. We note that the lower bounds (lb) obtained with the solution of Algorithm 4 combined with the

Table 4
Test results for randomly generated trees (part 2)

File	Alg.2	Alg.1	Alg.3	Alg.4	lb	(Alg.2)/lb
Optimum $\leq$ 100 (99 subtree transfers)						
001–100	139	150	244	265	89	1.57
002–100	150	153	253	270	91	1.65
003–100	152	156	259	277	93	1.65
004–100	137	141	239	255	86	1.60
005–100	148	151	253	267	90	1.65
006–100	142	148	228	250	84	1.70
007–100	142	148	253	265	89	1.60
008–100	144	147	249	266	90	1.60
009–100	146	146	242	264	89	1.65
010–100	147	153	233	259	87	1.69
Optimum $\leq$ 200 (199 subtree transfers)						
001–200	291	332	430	477	160	1.82
002–200	291	307	449	483	162	1.80
003–200	284	300	426	473	159	1.79
004–200	274	303	433	462	155	1.77
005–200	275	313	434	485	163	1.69
006–200	269	303	452	470	158	1.71
007–200	278	298	445	495	166	1.68
008–200	281	295	434	474	159	1.77
009–200	285	314	425	478	160	1.79
010–200	288	324	437	490	164	1.76
Optimum $\leq$ 400 (399 subtree transfers)						
001–400	515	556	683	744	249	2.07
002–400	526	562	688	731	245	2.15
003–400	513	553	690	731	245	2.10
004–400	492	546	675	723	242	2.04
005–400	514	562	680	733	245	2.10
006–400	526	569	676	708	237	2.22
007–400	508	558	688	758	254	2.00
008–400	501	551	673	711	238	2.11
009–400	529	570	673	747	250	2.12
010–400	539	582	703	751	251	2.15

Table 5
Solutions output by Algorithms 1, 2, 3 and 4 for the instance $\mathcal{I}_m$, constructed to show that Algorithm 1 has performance ratio 4

Instance	# leaves	(opt $\leq$)	lb	Alg.2	**Alg.1**	Alg.3	Alg.4
$\mathcal{I}_m$	$4m+1$	$m+2$			$4m$		
$m = 20$	81	22	22	22	80	64	64
$m = 40$	161	42	42	42	160	124	124
$m = 60$	241	62	62	62	240	184	184
$m = 80$	321	82	82	82	320	244	244
$m = 100$	401	102	102	102	400	304	304
$m = 120$	481	122	122	122	480	364	364
$m = 140$	561	142	142	142	560	424	424
$m = 160$	641	162	162	162	640	484	484
$m = 180$	721	182	182	182	720	544	544
$m = 200$	801	202	202	202	800	604	604

fact that opt $\leq m + 2$ indicate that these are in fact the optimum values (here opt denotes the optimum value). It is interesting to see that Algorithm 2 found optimum solutions for this family of instances.

Table 6
Solutions output by Algorithms 1, 2, 3 and 4 for the instance $\mathcal{I}'_m$, constructed to show that the performance ratio 3 of Algorithm 2 is tight

Instance	# leaves	(opt $\le$)	(ratio $\ge$)	**Alg.2**	Alg.1	Alg.3	Alg.4
$\mathcal{I}'_m$	$6m+1$	$2m+2$	$\frac{6m-1}{2m+2}$	$6m-1$			
$m = 20$	121	42	2.833	119	120	104	106
$m = 40$	241	82	2.914	239	240	204	205
$m = 60$	361	122	2.942	359	360	304	307
$m = 80$	481	162	2.956	479	480	404	406
$m = 100$	601	202	2.965	599	600	504	505
$m = 120$	721	242	2.971	719	720	604	607
$m = 140$	841	282	2.975	839	840	704	706
$m = 160$	961	322	2.978	959	960	804	805
$m = 180$	1081	362	2.980	1079	1080	904	907
$m = 200$	1201	402	2.982	1199	1200	1004	1006

Table 7
Solutions output by Algorithms 1, 2, 3 and 4 for the instance $\mathcal{I}''_m$, constructed to show that the ratio 3 of Algorithm 3 is tight

Instance	# leaves	(opt $\le$)	lb	Alg.2	Alg.1	**Alg.3**	Alg.4
$\mathcal{I}''_m$	$3m+1$	$m+1$				$3m$	
$m = 20$	61	21	21	21	21	60	60
$m = 40$	121	41	41	41	41	120	120
$m = 60$	181	61	61	61	61	180	180
$m = 80$	241	81	81	81	81	240	240
$m = 100$	301	101	101	101	101	300	300
$m = 120$	361	121	121	121	121	360	360
$m = 140$	421	141	141	141	141	420	420
$m = 160$	481	161	161	161	161	480	480
$m = 180$	541	181	181	181	181	540	540
$m = 200$	601	201	201	201	201	600	600

In Table 6 we show the results obtained for the instance $\mathcal{I}'_m$ (see Fig. 6), which we constructed to show that Algorithm 2 has performance ratio 3. We note that the ratio $(6m-1)/(2m+2)$ is a lower bound for the performance ratio of Algorithm 2: in Section 4.1 we observed that the instance $\mathcal{I}'_m$ has an optimum solution with value at most $2m+2$. It is interesting to note that, this is the only example we have for which Algorithm 3 performed better than Algorithm 2.

In Table 7 we exhibit the results obtained for the instance $\mathcal{I}''_m$ (see Fig. 7), constructed to show that the performance ratio 3 we have proved for Algorithm 3 is tight.

We conclude from the computational tests that the performance of Algorithm 2 is generally better than the performance of the other algorithms. We note that, with the exception of the "bad" instances ($\mathcal{I}'_m$) we have constructed for Algorithm 2, in all other instances the solutions it provides are at least as good as (mostly better than) the solutions provided by the other algorithms, and also far better than 3 times the optimum.

6. A generalization of Algorithm 2

This section discusses how to generalize the main result in this paper – a 3-approximation algorithm for MAF-2 – for trees with bounded degree $d \ge 2$, by providing an approximation algorithm for MAF-d with performance ratio $d+1$.

Elimination is defined for trees with bounded degree as before: if $\mathcal{W}$ is a phylogenetic tree and e is an arc of $\mathcal{W}$, then $\text{Elim}(\mathcal{W}, l(e))$ is the phylogenetic tree whose components are $\mathcal{W}|D_e$ and $\mathcal{W}|(S_{\mathcal{W}} \setminus D_e)$. The only difference is that in the implementation there is no need to perform a forced contraction in $\mathcal{W}[S_{\mathcal{W}} \setminus D_e]$ to produce $\mathcal{W}|(S_{\mathcal{W}} \setminus D_e)$ unless $u(e)$ has degree 2 in $\mathcal{W}$.

In each iteration of the new algorithm – referred to as Algorithm 5 – $\mathcal{G}_i$ is searched for a maximal set of sibling labels with at least two elements. Such a set is called a **tuple**. If $\mathcal{G}_i$ has no tuple, the algorithm outputs $|\mathcal{H}_i|$ and terminates; otherwise it proceeds by finding the lowest of all lcas in $\mathcal{H}_i$ of at least two leaves with labels in the tuple.

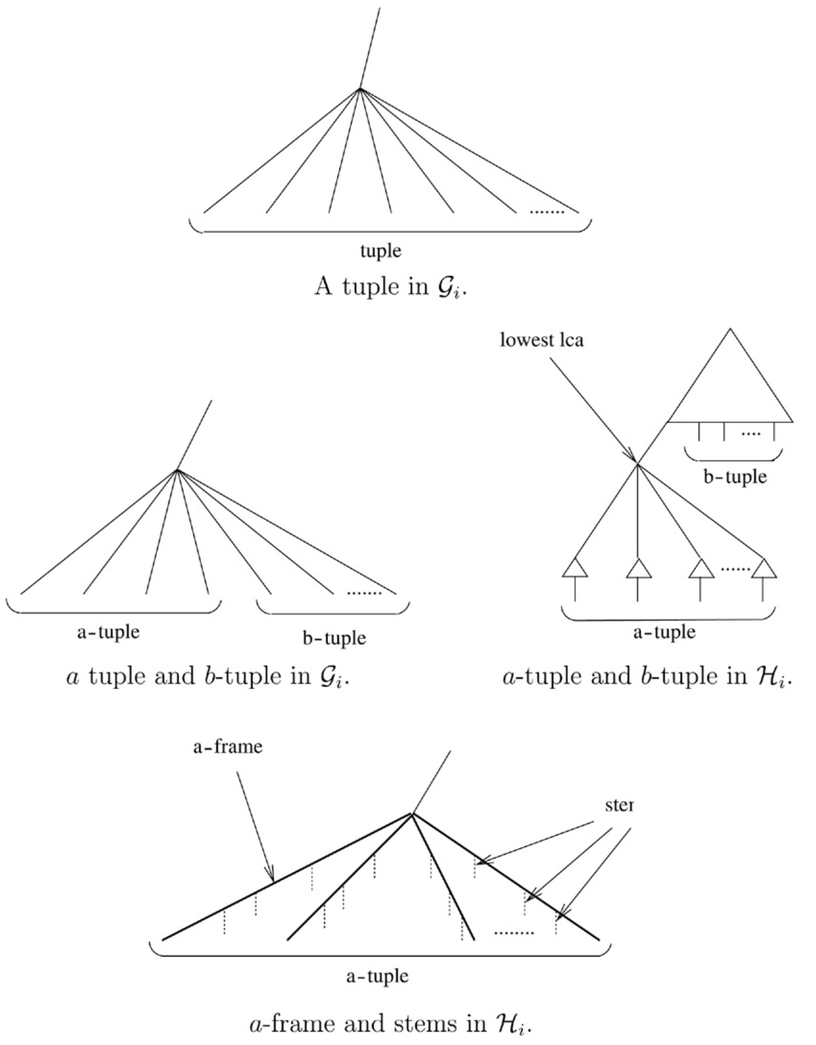

tuple
A tuple in $\mathcal{G}_i$.
lowest lca
b-tuple
a-tuple
b-tuple
a-tuple
a tuple and b-tuple in $\mathcal{G}_i$.
a-tuple and b-tuple in $\mathcal{H}_i$.
a-frame
stems
a-tuple
a-frame and stems in $\mathcal{H}_i$.

Fig. 12. Tuple searching and splitting in Algorithm 5.

Table 8
Cases for Algorithm 5

Case 1	$q + r > 0$ and $r \leq p$.
Case 2	$q + r > 0$ and $r > p$.
Case 5	$q + r = 0$.
Case 3	No label in the tuple is isolated in $\mathcal{H}_i$.
Case 4	At least one label in the tuple is isolated in $\mathcal{H}_i$.

If this lowest lca exists, the tuple is split into an ***a*-tuple** and a ***b*-tuple**, the first one comprising all labels descending from the lowest lca in $\mathcal{H}_i$ and the second comprising the other labels in the tuple. If the lowest lca does not exist, then the a-tuple is the whole original tuple and the b-tuple is empty. Let the ***a*-frame** be the set of paths in $\mathcal{H}_i$ connecting each leaf whose label is in the a-tuple with the lowest lca, and let the **stems** be, as before, those arcs not in the a-frame whose upper endpoints are in the a-frame. Let p be the size of the a-tuple ($p \geq 2$), q be the size of the b-tuple ($q \geq 0$) and r be the number of stems ($r \geq 0$) (see Fig. 12).

We define five cases for Algorithm 5. If there is no lowest lca in $\mathcal{H}_i$ for any two-element subset of the tuple, then we have cases 3 and 4; otherwise we have cases 1, 2 or 5. Table 8 lists all cases and Table 9 lists the corresponding transactions.

Table 9
Transactions for Algorithm 5

Case 1	Cut in $\mathcal{H}_i$ all stems and all leaves in the b-tuple.
Case 2	Cut in $\mathcal{G}_i$ and $\mathcal{H}_i$ all leaves in the a-tuple.
Case 3	Cut in $\mathcal{H}_i$ all leaves in the tuple.
Case 4	Cut in $\mathcal{G}_i$ all leaves in the tuple which are isolated in $\mathcal{H}_i$.
Case 5	Shrink in $\mathcal{G}_i$ and $\mathcal{H}_i$ all leaves in the tuple.

Algorithm 5 has polynomial time complexity, like Algorithms 1 and 2.

Tuple search can be performed in time $O(n)$, tuple splitting and stem counting can be performed in time $O(nd^2)$, and each transaction takes time $O(nd)$. As before, there are $O(n)$ iterations, since at each elimination at least an arc of $\mathcal{T}$ and $\mathcal{U}$ is eliminated. Algorithm 5 has therefore time complexity $O(n^2d^2)$.

The same technique as was applied to Algorithm 1–2 can be used to prove that Algorithm 5 has approximation ratio $d + 1$. In particular, the following version of Lemma 8 is true:

Lemma 14. *Let $\mathcal{G}$ and $\mathcal{H}$ be two phylogenetic forests with $\bigcup \mathcal{S}_\mathcal{G} = \bigcup \mathcal{S}_\mathcal{H}$, and let $\mathcal{H}'$ be a full restriction of $\mathcal{H}$ that is isomorphic to a full restriction of $\mathcal{G}$. Suppose that $\{f_\mathcal{G}(a_i) : 1 \le i \le m\}$ form a tuple in $\mathcal{G}$.*

- *If there exist i_1 and i_2 such that $f_{\mathcal{H}'}(a_{i_1})$ and $f_{\mathcal{H}'}(a_{i_2})$ are in different components of $\mathcal{H}'$ and $f_\mathcal{H}(a_{i_1})$ and $f_\mathcal{H}(a_{i_2})$ are not isolated, then at least one of the arcs incident to $f_{\mathcal{H}'}(a_{i_1})$ and $f_{\mathcal{H}'}(a_{i_2})$ is a link.*
- *If all $f_{\mathcal{H}'}(a_1), \ldots, f_{\mathcal{H}'}(a_m)$ are in the same component of $\mathcal{H}'$ and the a-frame admits at least one stem, then all of its stems are links.*

The proof of Lemma 14 is similar to the proof of Lemma 8. This lemma is needed to guarantee that, in the debt protocol for phylogenetic trees with bounded degree, there always exists a link set to which debt units can be associated when there is an iteration with no link elimination. For bounded degree trees, the debt protocol must be reformulated as follows:

- If an iteration is case 1 and no eliminated arc is a link, then it can be proven (using Lemma 14) that for some i such that a_i is in the a-tuple, $f_{\mathcal{H}'}(a_i)$ is a link. Then the $q + r \le q + p \le d$ newly issued debt units are associated with this link.
- If an iteration is case 2 and there is no link elimination, then Lemma 14 asserts that all arcs of the stem residue (suitably defined) are links. The $p \le d$ issued debt units are then associated with this residue.

Stem residues yield barriers as the algorithm proceeds with arc eliminations. The balance between debt and primary barriers is attained by using quotas of size one, since each primary barrier has size at least $r \ge p + 1$. In this way, the protocols can be proven to be correct by setting $\alpha = d + 1$.

Acknowledgements

We thank Cristina G. Fernandes for helpful discussions, and we are also grateful to an anonymous referee for valuable suggestions.

References

[1] B. Allen, M. Steel, Subtree transfer operations and their induced metrics on evolutionary trees, Annals of Combinatorics 5 (2001) 1–13.
[2] F. Chataigner, Approximating the maximum agreement forest on k trees, Information Processing Letters 93 (5) (2005) 239–244.
[3] B. dasGupta, X. He, T. Jiang, M. Li, J. Tromp, L. Zhang, On distances between phylogenetic trees, in: Proceedings of the 8th ACM–SIAM Symposium of Discrete Algorithms, 1997, pp. 427–436.
[4] J. Hein, T. Jiang, L. Wang, K. Zhang, On the complexity of comparing evolutionary trees, Discrete Applied Mathematics 71 (1996) 153–169.
[5] M. Li, J. Tromp, L. Zhang, On the nearest neighbour interchange distance between evolutionary trees, Journal on Theoretical Biology 182 (4) (1996) 463–467.
[6] E.M. Rodrigues, Algoritmos para Comparação de Árvores Filogenéticas e o Problema dos Pontos de Recombinação. Ph.D. Thesis, Computer Science Department, University of São Paulo, Brazil, February 2003.
[7] E.M. Rodrigues, M.-F. Sagot, Y. Wakabayashi, Some approximation results for the maximum agreement forest problem, in: Proceedings of the 4th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, in: Lecture Notes in Computer Science, vol. 2129, Springer-Verlag, 2001.
[8] D.L. Swofford, G.J. Olsen, P.J. Waddell, D.H. Hillis, Phylogenetic inference, in: D. Hillis, C. Moritz, B. Mable (Eds.), Molecular Systematics, Sinauer Associates, 1996, pp. 407–513.