

# Approximating Subtree Distances Between Phylogenies

AU1  
AU2

MARIA LUISA BONET,<sup>1</sup> KATHERINE ST. JOHN,<sup>2,3</sup> RUCHI MAHINDRU,<sup>2,4</sup>  
and NINA AMENTA<sup>5</sup>

## ABSTRACT

We give a 5-approximation algorithm to the rooted Subtree-Prune-and-Regraft (rSPR) distance between two phylogenies, which was recently shown to be NP-complete. This paper presents the first approximation result for this important tree distance. The algorithm follows a standard format for tree distances. The novel ideas are in the analysis. In the analysis, the cost of the algorithm uses a “cascading” scheme that accounts for possible wrong moves. This accounting is missing from previous analysis of tree distance approximation algorithms. Further, we show how all algorithms of this type can be implemented in linear time and give experimental results.

**Key words:**

AU3

## 1. INTRODUCTION

**P**HYLOGENIES, or evolutionary histories, are an important tool in almost all branches of biology. They give a framework for analyzing the interrelationships between species and are indispensable in studying evolution (Hillis et al., 1996). In addition to the direct applications, techniques for building and comparing phylogenies have been used for designing vaccines (Bush et al., 1999; Hillis, 1999; Novitsky et al., 2002), haplotyping (Ding et al., 2005; Gusfield, 2004), and determining the evolution of human language (Gray and Atkinson, 2003; Warnow et al., 1996). Most optimization criteria for creating phylogenies are NP-hard (Roch, 2006; Addario-Berry et al., 2003; Foulds and Graham, 1982). To evaluate the correctness of proposed phylogenies (given by heuristics and approximation algorithms), the distance between the two trees is needed. Efficiency is key since algorithms that approximate the correct phylogeny often produce thousands of candidate trees (Huelsenbeck and Ronquist, 2001; Swofford, 2002) and analyzing and visualizing these requires distances to be computed quickly (Amenta and Klingner, 2002; Hillis et al., 2005). While the Robinson-Foulds distance (Robinson and Foulds, 1981) (basically the sum of the false

---

<sup>1</sup>Lenguajes y Sistemas Informaticos (LSI), Universidad Politecnica de Catalunya (UPC), Barcelona, Spain.

<sup>2</sup>Department of Math & Computer Science, Lehman College–City University of New York (CUNY), Bronx, New York.

<sup>3</sup>Department of Computer Science, CUNY Graduate Center, New York, New York.

<sup>4</sup>Department of Computer Science, Stony Brook University, Stony Brook, New York.

<sup>5</sup>Computer Science Department, University of California, Davis, California.

positives and false negatives) is often used, since it can be calculated in linear time (Day, 1985), it lacks biological motivation. The TBR and rooted SPR metrics have better biological intuition (Hillis et al., 1996). Unfortunately, both are NP-hard to compute (Allen and Steel, 2001; Bordewich and Semple, 2005).

In addition to its value in phylogenetic reconstruction searches, rooted SPR distance is also valuable for more complex evolutionary events. Maddison (1997) showed that the number of reticulation events between two gene trees and rSPR of those two trees are closely related. Baroni et al. (2005) refined this, giving elegant results about the rSPR distance and the number of complex evolutionary events. Building on this equivalence has led to advances in network models of evolution (Bordewich and Semple, 2006; Nakhleh et al., 2005; Page and Charleston, 1998).

Our contributions are a 5-approximation for the SPR distance between rooted trees, counterexamples that point out subtle flaws of past approximation algorithms for TBR, and linear time versions of all these algorithms. With a small modification to the algorithms of Hein et al. (1996) and Rodrigues et al. (2001), we can get a 5-approximation algorithm for the related distance metric, rooted SPR. The proof of the approximation bounds uses a “cascading” accounting scheme to capture the complexity and illustrates the difficult situations to approximate. On the application side, we can improve the running time of these algorithms to linear time in the number of nodes. We end the paper by giving initial experimental results on the effectiveness of these algorithms on both biological and random datasets.

The problem of calculating TBR and SPR distances between trees has generated much interest and past work. Hein et al. (1996) had many good intuitions and defined the maximum agreement forest (MAF), which is the key to the NP-completeness proofs for both TBR and rooted SPR. Unfortunately, subtle details were missed in the proofs. Allen and Steel (2001) gave a counterexample to the argument in Hein et al. (1996) and a correct proof for the NP-completeness of TBR distance. Rodrigues et al. (2001) gave a counterexample to the approximation algorithm for TBR distance in Hein et al. (1996) and provided a new approximation algorithm. Unfortunately, the approximation bounds of Rodrigues et al. (2001) are not correct—we give a counterexample in Section 5 showing that the algorithm gives at best a four-approximation for TBR. Recently, Bordewich and Semple (2005) have shown that the calculation of SPR distance on rooted trees is NP-hard. Their development of a Maximum Agreement Forest for rooted SPR gives a natural way to prove bounds for approximation algorithms for this important distance metric. We use their new work to give the first approximation algorithm with provable bounds for rooted SPR distance.

## 2. BACKGROUND

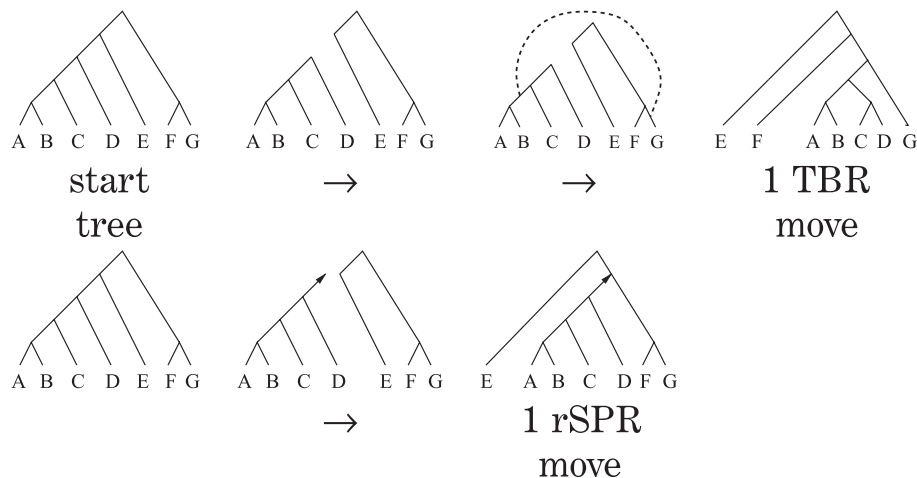
This section contains the basic definitions needed for the paper, which the expert may wish to skip. We follow the standard definitions from Allen and Steel (2001), Bordewich and Semple (2005), and Rodrigues et al. (2001).

**Definition 1 (Bordewich and Semple, 2005).** A rooted binary phylogenetic X-tree (or more briefly a tree) is a rooted tree where the root has degree two, all other interior nodes have degree 3 and the leaves are labeled by elements of the set  $X$ .  $X$  is called the **label set** of the tree.

**Definition 2 (Allen and Steel, 2001).** A subtree prune and regraft (SPR) operation on a binary tree  $T$  is defined as cutting any edge and thereby pruning a subtree  $t$ , then regrafting the subtree by the same cut edge to a new vertex obtained by subdividing a pre-existing edge in  $T - t$ . We apply a forced contraction to maintain the binary property of the resulting tree. The **SPR distance** between two trees  $T_1$  and  $T_2$  is the minimal number of SPR moves needed to transform  $T_1$  into  $T_2$ . When working with rooted trees, we refer to this distance as **rooted SPR** or **rSPR** (Fig. 1).

**F1**

**Definition 3 (Allen and Steel, 2001).** A tree bisection and reconnection (TBR) operation on a binary tree  $T$  is defined as removing any edge, giving two new subtrees  $t_1$  and  $t_2$ , which are then reconnected by creating a new edge between the midpoints of any edge in  $t_1$  and any edge in  $t_2$ . We apply a forced contraction to maintain the binary property of the resulting tree. The **TBR distance** between two trees  $T_1$  and  $T_2$  is the minimal number of TBR moves needed to transform  $T_1$  into  $T_2$ . See Figure 1.



**FIG. 1.** Beginning with the start tree, we show an example of a TBR move at the top and an example of a rSPR move at the bottom. Note that the rSPR move prunes a rooted subtree and must regraft it by the same cut edge. For TBR, an edge is removed entirely and a new edge (the blue dotted edge in the figure) is chosen. Since TBR allows more freedom in reconnecting, there are pairs of trees which take strictly less TBR moves than rSPR moves (e.g., the TBR tree and the start tree).

Allen and Steel (2001) showed that the TBR distance between two trees is one less than the size of the TBR maximum agreement forest,  $MAF_{TBR}$  of the two trees.

**Definition 4 (Allen and Steel, 2001).** Suppose we have two binary trees  $T_1$  and  $T_2$  on the same leaf set,  $\mathcal{L}$ . Then

- An **agreement forest (AF)** for  $T_1$  and  $T_2$  is a collection  $\mathcal{F} = \{t_1, \dots, t_k\}$  of binary trees such that, if we let  $\mathcal{L}_j$  be the leaf set of  $t_j$  for  $j \in \{1, \dots, k\}$ , then the following are satisfied:
  1.  $\mathcal{L}_1, \dots, \mathcal{L}_k$  partitions  $\mathcal{L}$ .
  2.  $t_j = T_1|_{\mathcal{L}_j} = T_2|_{\mathcal{L}_j}$  for all  $j \in \{1, \dots, k\}$ ; and
  3. for both  $i = 1$  and  $i = 2$  the trees  $\{T_i(\mathcal{L}_j) : j = 1, \dots, k\}$  are vertex-disjoint subtrees of  $T_i$ .
- A **maximum agreement forest ( $MAF_{TBR}$ )** for  $T_1$  and  $T_2$  is an agreement forest  $\mathcal{F}$  for  $T_1$  and  $T_2$  for which  $|\mathcal{F}|$  is minimal. Let

$$m(T_1, T_2) := \min\{|\mathcal{F}| - 1 : \mathcal{F} \text{ is an AF for } T_1 \text{ and } T_2\}.$$

Bordewich and Semple (2005) showed that the rSPR distance of two trees is the same as the size of the maximum agreement forest for rooted trees of the two trees. Below we give the definition of maximum agreement forest for SPR on rooted trees, and since we focus only on rooted trees, we will call it simply maximum agreement forest for SPR or  $MAF_{SPR}$ .

**Definition 5 (Bordewich and Semple, 2005).** Let  $T$  be a rooted binary phylogenetic X-tree (or more briefly a tree) and  $V$  a subset of the vertex set of  $T$ .  $\mathbf{T}(V)$  is the minimal rooted subtree of  $T$  that connects the elements of  $V$ .  $\mathbf{T} \mid V$  is obtained from  $T(V)$  by splicing out all non-root vertices of degree two (that is, replacing the vertex and its two adjacent edges with a single edge).

**Definition 6 (Bordewich and Semple, 2005).** Let  $T_1$  and  $T_2$  be two rooted binary phylogenetic X-trees. For the purposes of definition, we regard the root of both  $T_1$  and  $T_2$  as a vertex  $\rho$  at the end of an edge adjoined to the original root. Furthermore, we regard  $\rho$  as part of the label set of  $T_1$  and  $T_2$ . An (rSPR) **agreement forest** for  $T_1$  and  $T_2$  is a collection  $\{\mathcal{T}_\rho, \mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_k\}$  where  $\mathcal{T}_\rho$  is a rooted tree and  $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_k$  are rooted binary phylogenetic trees with label sets  $\{\mathcal{L}_\rho, \mathcal{L}_1, \mathcal{L}_2, \dots, \mathcal{L}_k\}$  such that the following properties are satisfied:

1. The label sets partition  $X \cup \rho$  and, in particular,  $\rho \in \mathcal{L}_\rho$ .
2. For all  $i \in \{\rho, 1, 2, \dots, k\}$ ,

$$\mathcal{T}_i \simeq T_1 \mid \mathcal{L}_i \simeq T_2 \mid \mathcal{L}_i.$$

3. The trees in  $\{T_1(\mathcal{L}_i) \mid i \in \{\rho, 1, 2, \dots, k\}\}$ , and the trees in  $\{T_2(\mathcal{L}_i) \mid i \in \{\rho, 1, 2, \dots, k\}\}$  are vertex disjoint rooted subtrees of  $T_1$  and  $T_2$ , respectively.

**Definition 7.** A (rSPR) **maximum agreement forest** ( $MAF_{\text{SPR}}$ ) for  $T_1$  and  $T_2$  is the agreement forest for  $T_1$  and  $T_2$  with the minimal number of components.

The  $MAF_{\text{SPR}}$  need not be unique, so in the subsequent proof, we will select an arbitrary  $MAF_{\text{SPR}}$   $F$  and use it consistently.

**Definition 8.** Let  $F$  be an agreement forest for  $T_1$  and  $T_2$ . We say that a set  $E$  of edges in  $T_1$  ( $T_2$ , respectively) are **links** with respect to  $F$  if removing  $E$  and then splicing out all vertices of degree two yields the agreement forest  $F$ .

Two algorithms (Hein et al., 1996; Rodrigues et al., 2001) have been proposed to approximate  $MAF_{\text{TBR}}$  size (for the TBR distance) and therefore the TBR distance. In this paper, we adapt these two algorithms to get an algorithm for the rooted version of maximum agreement forest for SPR, and analyze its performance.

### 3. THE GENERAL ALGORITHM

The general algorithm proceeds by examining the sibling pairs in the first tree,  $T_1$  and finding the corresponding leaves in the second tree,  $T_2$ . At each step, the algorithm eliminates an edge or contracts identical sibling pairs, creating, in the end, an agreement forest for  $T_1$  and  $T_2$ . To distinguish between the original trees and the forests that are created after each step of the algorithms, we add a superscript with the step number.  $T_1^0$  and  $T_2^0$  are the initial trees given to the algorithm, and  $T_1^i$  and  $T_2^i$  are the forests that result after the  $i$ th step of the algorithm. We let  $N$  be the count of the number of components in the agreement forest that are created from  $T_2$  by the algorithm.  $N$  provides an upper bound on the distance between the trees. At the start  $N = 1$ . At each step  $i$ , for each sibling pair  $a, b$  in  $T_1^i$ , we look at  $a$  and  $b$  in  $T_2^i$  and follow the cases in Figure 2. We repeat, until all components are of size 1 and then output the number of cuts,  $N$ , and the resulting agreement forest. F2

Figure 2 gives the Hein et al. variant of the algorithm. The Rodrigues et al. variant differs only by a small change in Cases 1 and 2. Namely, Case 1 occurs when there is either one or two subtrees between  $a$  and  $b$  in  $T_2^i$ , and we cut them. Case 2 occurs when there are three or more subtrees between  $a$  and  $b$ , and we cut  $a$  and  $b$ . Due to Case 5, which contracts identical sibling pairs, the leaves of the forests,  $T_1^i$  and  $T_2^i$ , can differ from one step of the algorithm to another.

A slight modification is needed for the rSPR distance. Since you could have the component that contains the root in the forest for one tree that does not occur in the forest for the other tree, as in Bordewich and Semple (2005), we view the root as a node  $\rho$  hanging off the pendant edge where the root is located in the tree. We then apply the algorithm above to trees on the extended leaf set of  $X \cup \rho$ .

**Definition 9.** Consider a Case 5 step in which nodes  $a$  and  $b$  are contracted into a new node  $(a, b)$ . We call  $(a, b)$  a **contraction** and we say  $a$  and  $b$  are **contracted into**  $(a, b)$ . If  $a$  is contracted into  $b$  and  $b$  is contracted into  $c$ , we also say  $a$  is contracted into  $c$ . We also say that  $a$  is (trivially) contracted into itself.

Because of contractions, the leaves of the input trees  $T_1$  and  $T_2$  are generally different from the leaves of the intermediate trees  $T_i^j$ ,  $i \in \{1, 2\}$ . Nonetheless, we can identify any node  $a$  of  $T_i^j$ , with a node  $a'$  of  $T_i$  as follows. Consider the subtree  $A$  of  $T_i^j$  rooted at  $a$ . We identify  $a$  with the least common ancestor  $a'$  of

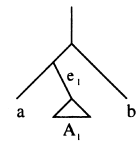
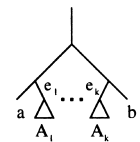
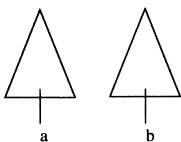
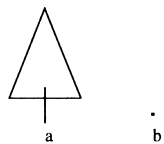
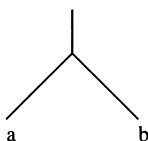
<p><b>Case 1:</b> There's 1 subtree between <math>a</math> &amp; <math>b</math> in <math>T_2^i</math>: Cut the subtree <math>l</math> in <math>T_2^i</math>.  <math>N := N + 1</math></p>	
<p><b>Case 2:</b> There are <math>\geq 2</math> subtrees between <math>a</math> &amp; <math>b</math> in <math>T_2^i</math>: Cut <math>a</math> &amp; <math>b</math> in <math>T_1^i</math> &amp; <math>T_2^i</math>.  <math>N := N + 2</math>.</p>	
<p><b>Case 3:</b> <math>a</math> &amp; <math>b</math> are in separate components of <math>T_2^i</math>: Cut <math>a</math> &amp; <math>b</math> in <math>T_1^i</math> &amp; <math>T_2^i</math>.  <math>N := N + 2</math>.</p>	
<p><b>Case 4:</b> <math>b</math> is a singleton in <math>T_2^i</math>: Cut <math>b</math> in <math>T_1^i</math>.  <math>N</math> does not change.</p>	
<p><b>Case 5:</b> The sibling pair occurs in both: Replace it by "<math>(a, b)</math>," in <math>T_1^i</math> &amp; <math>T_2^i</math>.  <math>N</math> does not change.</p>	

FIG. 2. The cases of Hein et al. variant of the algorithm at step  $i$ .

all of the leaves of  $T_i$  which are contracted into the leaves of  $A$ . Using this identification, we can identify an edge cut in  $T_i^j$  at step  $j$  with a particular edge of  $T_i$ . We call this a **cut edge** in  $T_i$ .

Recall that the links are defined in terms of the initial trees,  $T_1$  and  $T_2$  and the  $\text{MAF}_{\text{SPR}} F$ . Next we will define a related concept, that of virtual link, which is defined with respect to an intermediate tree  $T_i^j$ .

**Definition 10.** Let  $e$  be an edge in  $T_i^j$ , let  $a$  be the node adjacent to  $e$  nearer the root, and let  $a'$  be the node identified with  $a$  in  $T_i$ . Let  $L$  be the set of leaves in  $T_i$  which have been contracted into leaves of the subtree rooted at  $a$  in  $T_i^j$ . Edge  $e$  is a **virtual link** in  $T_i^j$  if, for every leaf  $l \in L$ , the path between  $a'$  and  $l$  contains a link.

This definition implies that if  $e$  is a link, it is also a virtual link. Notice that since virtual links are defined in terms of the forests  $T_i^j$  for some step  $j$ , an edge may not be a virtual link at step  $j$  but could become one at some later step  $j + s$ . But once an edge becomes a virtual link, it remains a virtual link.

#### 4. 5-APPROXIMATION FOR rSPR

The basic idea of our proof that the algorithm gives a 5-approximation for rSPR is to look at an arbitrarily chosen  $\text{MAF}_{\text{SPR}} F$  for trees  $T_1$  and  $T_2$ . For each link of  $T_2$  with respect to  $F$ , we will “charge” a cost for the edges the algorithm cuts. The overall charges on the links will equal the number of cuts made by the

algorithm, and we show that the charges do not exceed 5 for any link of  $T_2$ . Thus, the algorithm produces a result that is at most a multiplicative factor of 5 of the true distance.

**Definition 11.** Let  $F$  be an  $MAF_{SPR}$  for  $T_1$  and  $T_2$ , and  $a$  and  $b$  be two nodes of  $T_1^i$  and  $T_2^i$ .

- i. We say that  $a$  **participates in component**  $t$  of  $F$  if either  $a$  is a leaf of  $t$ , or  $a$  is a contraction containing a leaf  $a'$  of  $t$ .
- ii. We say that  $a$  and  $b$  **share a component**  $t$  of  $F$  if both  $a$  and  $b$  participate in  $t$ .

Notice that a contraction  $a$  may participate in multiple components of  $F$ .

We split the analysis of the cases of the algorithm into two sub-cases: those in which the nodes  $a$  and  $b$  of the sibling pair we are processing share a component of the  $MAF_{SPR}$  (**the “A” cases**), and those where  $a$  and  $b$  do not share any component of the  $MAF_{SPR}$  (**the “B” cases**). Thus, we consider Case 1A, Case 1B, etc.

#### 4.1. Agreement forest lemmas

For the cost analysis of the algorithm, we prove several lemmas about the trees with respect to the underlying maximum agreement forest. Lemma 12 shows that in Cases 1A and 2A, the edges  $e_i$  might not be links, but virtual links. That is, it is possible after repeated applications of Case 2 to have links “buried” in the subtrees  $A_i$ , with  $e_i$  not being a link. In both Hein et al. and Rodrigues et al., this subtlety was missed (Theorem 9 of Hein et al. [1996] and Lemma 4 of Rodrigues et al. [2001]) and is the main reason their approximation bounds are incorrect. This is illustrated in the example in Section 5.

**Lemma 12.** Let  $T_1$  and  $T_2$  be rooted phylogenetic trees on the same leaf set. Let  $F$  be a  $MAF_{SPR}$  for  $T_1$  and  $T_2$ , and  $T_1^i$  and  $T_2^i$  be the result after the  $i$ th step of the algorithm. Assume that  $a$  and  $b$  are a sibling pair in  $T_1^i$ , and  $a$  and  $b$  are in the same component of  $T_2^i$ , but are not siblings, so that between  $a$  and  $b$ , there are subtrees  $A_1, A_2, \dots, A_k$ ,  $k \geq 1$ . Call the edges above the subtrees  $e_1, e_2, \dots, e_k$  (Fig. 2). If  $a$  and  $b$  share a component  $t$  of  $F$  (see Definition 11), then

- i. For all  $j$ , none of the leaves of  $A_j$  participate in the  $MAF_{SPR}$  component  $t$ .
- ii.  $e_1, e_2, \dots, e_k$  are all virtual links at stage  $i$  of the algorithm.

#### Proof.

**Proof of i:** Towards a contradiction, assume that there exists  $j$ , such that some leaf  $l$  in  $A_j$  participates in  $t$ . Since  $a, b$  and  $l$  all share the  $MAF_{SPR}$  component  $t$ , there are nodes  $a', b'$  and  $l'$  of the original trees such that  $a', b'$  and  $l'$  are in  $t$ , and  $a'$  is contracted into  $a$ ,  $b'$  into  $b$  and  $l'$  into  $l$ . By the definition of  $MAF_{SPR}$ ,  $t$  is an induced rooted subtree of  $T_1$  and of  $T_2$ , and therefore  $l'$  should appear in  $T_1$  between  $a'$  and  $b'$ . Since  $a$  and  $b$  are a sibling pair in  $T_1^i$ ,  $l'$  (or a node into which  $l'$  has been contracted) has been detached from  $T_1$  at some step of the algorithm prior to step  $i$ . Only Cases 2, 3, or 4 cut edges in  $T_1$ . Case 2 and 3 cut the same nodes in both trees. Case 4 only cuts off a node  $b$  if  $b$  is a singleton already in  $T_2$ . Therefore if  $l'$  (or a node into which  $l'$  is contracted) was detached at some earlier step in the forest for  $T_1$ , it must also have been detached in the forest for  $T_2$  as well. But this is not the case, and we get a contradiction. Thus  $A_j$  does not participate in  $t$ .

**Proof of ii:** Recall that  $a$  and  $b$  share a rooted component  $t$  of the  $MAF_{SPR}$   $F$ . Then there are leaves  $a'$  and  $b'$  in  $T_2$ , contracted into  $a$  and  $b$  of  $T_2^i$  respectively, such that  $a'$  and  $b'$  belong to component  $t$ . Therefore we know that there are no links nor virtual links in the path from  $a$  to  $b$  in  $T_2^i$ . Now, if say  $e_i$  is not a virtual link, then there must be a leaf  $l'$  of  $T_2$  contracted into a leaf  $l$  in subtree  $A_i$  of  $T_2^i$ , such that there a link-free path from both  $a'$  and  $b'$  to  $l'$ . So  $l$  would participate in  $t$ , which contradicts part i of this lemma. ■

The following lemma will help us analyze Case 3A.

**Lemma 13.** *Let  $a$  and  $b$  be a sibling pair in  $T_1^i$  but contained in different components of  $T_2^i$ , and assume  $a$  and  $b$  share a component  $t$  of  $F$ . Then the step  $j$  at which  $a$  and  $b$  were first separated into two components was a Case 1B.*

**Proof.** Let  $(a', b')$  be the sibling pair in  $T_1^j$  handled in step  $j$ . Step  $j$  has to be a Case 1, since neither  $a$  nor  $b$  is a singleton and Case 1 is the only one in which two non-singleton components are created. Then in  $T_2^j$ , the subtree induced by the four nodes  $a, b, a', b'$  has sibling pairs either  $(a', a)$  and  $(b, b')$ , or  $(a, b')$  and  $(b, a')$ , while in  $T_1^j$ , the induced subtree has to have sibling pairs  $(a', b')$  and  $(a, b)$ . Thus the four nodes cannot all share a component of  $F$ . Similarly,  $a$  and  $b$  cannot share one component  $t_1$  of  $F$  while  $a'$  and  $b'$  share another component  $t_2$ , since these two components would have to intersect in  $T_2$ . Since we assume  $a$  and  $b$  share a component of  $F$ ,  $a'$  and  $b'$  cannot share a component of  $F$ , and step  $j$  is a Case 1B. ■

The following definition and lemmas will help us deal with the analysis of the first three cases when the sibling pair does not share a component, i.e., Cases 1B, 2B, and 3B. We show here that there is a link for every “B” case. We use the links in tree  $T_1$  since that makes the argument simpler.

**Definition 14.** *Let  $a$  be a node of  $T_1^i$  or  $T_2^i$ , and then let  $e_a$  be the edge connecting  $a$  with its parent.*

**Lemma 15.** *Suppose  $a$  and  $b$  are a sibling pair in  $T_1^i$ , but not in  $T_2^i$ , and that  $a$  and  $b$  do not share a component of  $F$ . Then at least one of  $e_a$  or  $e_b$  is a virtual link in  $T_1^i$ .*

**Proof.** Since  $a$  and  $b$  do not share any component of  $F$ , then for all  $a'$  contracted into  $a$  and all  $b'$  contracted into  $b$ , there is a link in the path between  $a'$  and  $b'$ . This implies that at least one of  $e_a$  or  $e_b$  is a virtual link in  $T_1^i$ . ■

Let  $(a, b)$  be a sibling pair in  $T_1^i$ . We define the *least common ancestor* of  $(a, b)$  in  $T_1$ ,  $\text{lca}(a, b)$ , to be the least common ancestor of any leaf contracted into  $a$  and any leaf contracted into  $b$ , or, equivalently, the node in  $T_1$  identified with the parent of  $a$  and  $b$  in  $T_1^i$ .

**Lemma 16.** *Let  $(a_1, b_1) \dots (a_k, b_k)$  be the sibling pairs in  $T_1^{i_1} \dots T_1^{i_k}$  (respectively) corresponding to the Cases 1B, 2B, and 3B of the algorithm. We can assign a link  $u_i$  in  $T_1$  to each sibling pair  $(a_i, b_i)$ , such that*

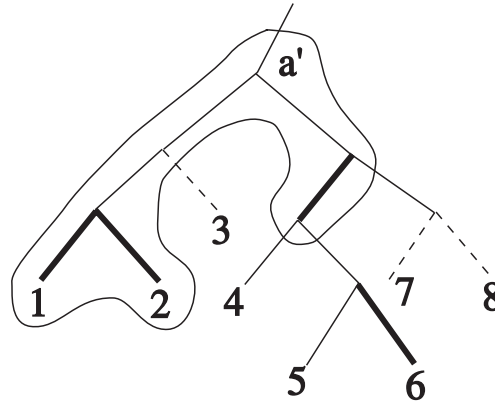
1. link  $u_i$  belongs to the subtree rooted at  $\text{lca}((a_i, b_i))$ , and
2. each link in  $T_1$  is assigned at to most one sibling pair.

**Proof.** We consider the situation when processing a Case 1B, 2B, or 3B sibling pair  $(a, b)$  in  $T_1^i$ , and we assume inductively that at all prior steps  $j < i$ , the lemma holds.

Lemma 15 established that either  $e_a$  or  $e_b$  is a virtual link; assume without loss of generality that  $e_a$  is a virtual link, and let  $a'$  be the node identified with  $a$  in  $T_1$ . If  $e_a$  is also a link, we assign it to  $(a, b)$ . By our inductive assumption, it cannot have been assigned to any sibling pair  $(c, d)$  in any prior step since it lies in the subtree rooted at  $\text{lca}(a, b)$  but not in the subtree rooted at any prior  $\text{lca}(c, d)$ .

If  $e_a$  is a virtual link but not a link, we will assign  $(a, b)$  to a link in the subtree of  $T_1$  rooted at  $a'$ , which again belongs to the subtree rooted at  $\text{lca}(a, b)$ . Any path in  $T_1$  from  $a'$  to one of the leaves contracted into  $a$  must contain at least one link. Let  $u$  be the link in the path nearest to  $a'$ , that is, there is no other link in the path between  $u$  and  $a'$ . Consider the sub-path connecting  $a'$  and  $u$ . We form a subtree  $A$  of  $T_1$  by taking the union of all of these initial sub-paths, for every leaf contracted into  $a$ . The edge adjacent to every leaf of  $A$  (which might not be a leaf of  $T_1$ ) is a link, so that the number of links in  $A$  is a equal to the number of leaves in  $A$ . (Fig. 3).

We will show that one of these links was not assigned to a sibling pair by a prior Case 1B, 2B, or 3B step of the algorithm and can therefore be assigned to  $(a, b)$ . First we show that none of the links assigned to a 2B or 3B step can belong to  $A$ . Assume step  $j < i$  of the algorithm was a Case 2B or 3B step on



**FIG. 3.** Figure for Proof of Theorem 16. Heavy edges are links, and dashed edges are cut edges with respect to  $T_1^i$ . The subtree  $A$  of  $T_1$  rooted at node  $a'$  is circled. Leaves 1, 2, 4, 5 and 6 are contracted into leaves of the subtree rooted at  $a$  in  $T_1^i$ .

a sibling pair  $(c, d)$ , so that  $(c, d)$  was assigned to a link in the subtree rooted at  $\text{lca}(c, d)$ . But both  $c, d$  were cut at step  $j$ , so that no leaf of  $T_1$  contracted into  $c$  or  $d$  is contracted into  $a$ . Hence no edge of the subtree rooted at  $\text{lca}(c, d)$  (including the relevant link) can belong to  $A$ .

Therefore if step  $j < i$  assigns a link  $u$  in  $A$  to a sibling pair  $(c, d)$ , it must be a Case 1B step. Since  $u$  belongs to  $A$ ,  $\text{lca}(c, d)$  and at least one child of  $\text{lca}(c, d)$  belong to  $A$ , and there is no link between  $\text{lca}(c, d)$  and the root  $a'$  of  $A$ .

A Case 1B step on  $(c, d)$  is always followed immediately by a Case 5B step on  $(c, d)$ , which creates a new node  $f$  in  $T_1^{j+1}$ , identified with  $\text{lca}(c, d)$  in  $T_1$ . Every leaf contracted into  $c$  or  $d$  is contracted into  $f$ , and, since  $u$  belongs to  $A$ , it must be the case that  $f$  is contracted into  $a$ . Since there is at least one leaf of  $T_1$  contracted into each of  $c, d$ , and there is no link between  $\text{lca}(c, d)$  and  $a'$ ,  $\text{lca}(c, d)$  must have degree two in  $A$ .

Thus, every step 1B which associates a link  $u$  with a sibling pair  $(c, d)$  can also be associated with the degree-two node  $\text{lca}(c, d)$  in  $A$ . Since all the leaves contracted into  $c, d$  are contracted into  $f$  at step  $j$ , no later step 1B can be associated with  $\text{lca}(c, d)$ . So the number of sibling pairs to which a link  $u$  has already been assigned at step  $i$  is at most the number of degree-two nodes in  $A$ .

Since  $A$  is a binary tree, the number of leaves in  $A$  is one greater than the number of degree two nodes in  $A$ , and therefore at least one link remains to be assigned to  $(a, b)$ . ■

#### 4.2. Assigning charges and analysis

To be able to analyze the algorithm, we will describe how to charge a cost to the links for each edge that the algorithm cuts. The overall charges on the links equals the number of cuts, and we will prove that on each link the charge will be at most 5. To simplify the accounting, we will keep track of the links in the original tree  $T_2$ .

We now describe how to assign the charges for the different cases. Suppose, without loss of generality, that we are at stage  $i$  of the algorithm, and that we are processing the sibling pair  $(a, b)$  of  $T_1^i$ .

**Cases 1A and 2A:**  $a$  and  $b$  are in the same  $\text{MAF}_{\text{SPR}}$  corresponding to  $a$  and  $b$ , and there is a path from  $a$  to  $b$ . Say  $A_1, \dots, A_k$  are the trees that hang directly from this path, and let  $e_1, \dots, e_k$  be the edges connecting the trees to the path. If we are in Case 1A, at stage  $i$  only nodes from one of these trees appears in  $T_2^i$ . Without loss of generality, say the tree is  $A_1$ . Then we put a charge of 1 on  $e_1$ . If we are in Case 2A, then at least two of these trees keeps some nodes at stage  $i$ . Without loss of generality, say they are  $A_1, \dots, A_s$ , where  $2 \leq s \leq k$ . Then we charge  $2/s$  to each of the  $s$  edges  $e_1, \dots, e_s$ . Recall that in this case we detach two nodes, so the cost is 2, and we are distributing it through  $s$  edges. Now, we assigned charges to edges that might not be links. Now, for each  $e$  which is not a link, we pass the charge down to the links below in the following way: Say  $e$  has two child edges,  $e_1, e_2$  in  $T_2$ . If both are links, or have links below, then we charge each one with half the cost. If only one is a link or has links below, then we



charge it the full cost. Note that at least one must be a link or have a link below to have a virtual link at stage  $i$  (by Lemma 12). Since our trees are finite in size, this recursive charging scheme is well-defined.

**Cases 1B, 2B, and 3B:** The cost of a Case 1B operation is 1. The cost of a Case 2B or Case 3B operation is 2. We charge the cost to the link assigned to the “B” case.

**Case 3A:** The cost of this operation is 2. The step in which  $a$  and  $b$  were separated into different components was a Case 1B step (Lemma 13). We charge the cost of the Case 3A operation to the link assigned to the Case 1B step.

**Cases 4 and 5:** The cost of these operations is zero, since in Case 4 we cut an edge in  $T_1^i$  rather than in  $T_2^i$ , and in Case 5 we just make contractions.

The following lemma gives extra information on how the charges are assigned to links.

**Lemma 17.**

- i. We can only charge a link once for Cases 1B + 2B + 3B.*
- ii. Similarly, we can only charge a link once for Case 3A.*
- iii. We can only charge 2 on a link for Cases 1A and 2A.*

**Proof.** In the following proofs we use the labels on trees and edges for the different cases of the algorithm in Figure 2.

Part i: By Lemma 16, we have at least one link for each application of a “B” case of the algorithm.

Part ii: By Lemma 13, any Case 3A is created by a Case 1B. If we charge the same link twice from a Case 3A it must be because both 3A cases have been created by the same Case 1B step. Suppose the first Case 3A happens at step  $i$ , and the second at step  $j$ . Then, at step  $i$ ,  $T_1^i$  contains a sibling pair  $(a, b)$  with  $a$  and  $b$  in different  $T_2^i$ -components, and at step  $j$ ,  $T_1^j$  has a sibling pair  $(c, d)$  with  $c$  and  $d$  in different  $T_2^j$ -components. Without loss of generality, say  $a$  and  $c$  are in the subtree cut off by the Case 1B step. Note that  $a, b, c, d$  must all share a component of the  $\text{MAF}_{\text{SPR}} F$ ; the edge cut by the Case 1B step has to be contained in any component of  $F$  shared by  $a$  and  $b$  and also in any component of  $F$  shared by  $c$  and  $d$ . Thus, in tree  $T_1$ , the  $\text{lca}(a, b)$  does not have  $c$  as a descendant, while in  $T_2$  it does. But this cannot happen by the definition of  $\text{MAF}_{\text{SPR}}$ , giving a contradiction.

Part iii: Say we charge twice a link for two applications of Cases 1A and 2A. Say the first time is at step  $i$ , and the second at step  $j$ . Let  $(a, b)$  be the sibling pair considered at step  $i$ , and  $(c, d)$  be the pair considered at step  $j$ . Since we’re considering Cases 1A and 2A,  $a$  and  $b$  must share a component of the  $\text{MAF}_{\text{SPR}}$ , and  $c$  and  $d$  also must share a component of the  $\text{MAF}_{\text{SPR}}$ . Let  $e'$  be the link that is charged twice. Since  $e'$  is below  $\text{lca}(a, b)$  and below  $\text{lca}(c, d)$ , there are three possibilities for the relative positions of  $a, b, c, d$  in  $T_2$ :  $c$  and  $d$  are below the  $\text{lca}(a, b)$ , one of  $c$  and  $d$  is below and the other is above, and finally,  $c$  and  $d$  are above  $\text{lca}(a, b)$ .

If  $c$  and  $d$  are below the  $\text{lca}(a, b)$ , then the least common ancestor of  $c$  and  $d$  is in one of the intermediate subtrees  $A_i$  (or if it is the same as  $\text{lca}(a, b)$  the proof is the same as in the second case).  $e_i$  is a virtual link, so there is a link between  $c$  and  $e_i$ , and also between  $d$  and  $e_i$ . But since  $c$  and  $d$  are in the same  $\text{MAF}_{\text{SPR}}$  component, the link must be above  $\text{lca}(c, d)$ . Therefore, the charge of  $e_i$  as a virtual link will not reach  $e'$ . So this case will not happen.

In the second case,  $a, b, c, d$  must all share a  $\text{MAF}_{\text{SPR}}$  component. This gives a contradiction with the definition of  $\text{MAF}_{\text{SPR}}$ , since in  $T_1$ ,  $a$  and  $b$  are more closely related than  $c$  and  $d$ , but in  $T_2$ ,  $a$  and  $c$  are more closely related than  $b$  and  $d$ . So this second case cannot happen either.

We have to conclude that  $c$  and  $d$  must be above  $\text{lca}(a, b)$ . This means that when we charge a link repeatedly for Cases 1A or 2A, the process occurs bottom up towards the root of  $T_2$ . We will discuss now the first two charges on  $e'$  from the bottom up. We distinguish two cases:

**Case I:** The lower case is a Case 2A. At step  $i$ , we process the pair  $(a, b)$ , and  $a$  and  $b$  disappear. To assign a charge of 2 in  $T_2$ , we first identify in  $T_2$  the nodes corresponding to  $a$  and  $b$ . Let  $A_1 \dots A_k$  be the trees that hang from the path from  $a$  to  $b$  in  $T_2$ , and  $e_1 \dots e_k$  the connecting edges. Some  $s \geq 2$  of these trees haven’t completely disappeared by step  $i$ . Without loss of generality, say the one containing  $e'$

is  $A_1$ . Since there are at least two subtrees,  $e'$  gets a charge  $\leq 1$  from this step. When we process the pair  $(c, d)$ , a charge  $\leq 1$  will be passed to  $\text{lca}(e_1 \dots e_s)$ . Since  $s \geq 2$ , the charge passed down to each one of the  $e_i$ 's will be divided by two. So at most a  $1/2$  charge will be passed to  $e'$ .

**Case II:** The lower case is a Case 1A. So, in  $T_2^i$ , the intermediate subtree gets detached, and immediately afterwards  $a$  and  $b$  get contracted. Looking at  $T_2$ , we assign to  $e'$  a charge of one at this step. Later, in step  $j$ , we process the sibling pair  $(c, d)$ . In  $T_2$ , we identify the nodes corresponding to  $c$  and  $d$  as explained before. From the path between  $c$  and  $d$ , hang trees  $A_1, \dots, A_k$ , with connecting edges  $e_1, \dots, e_k$ . Without loss of generality,  $e'$  is in  $A_1$ . Now, if at step  $j$ , the whole of  $A_1$  has disappeared, then we do not charge  $e_1$  and as a consequence we do not charge  $e'$ . So we must conclude that some part of  $A_1$  stays in  $T_2^j$ . The connecting edge in  $T_2^i$  is a virtual link, and  $e'$  at step  $j$  has disappeared. So there must be another link in  $A_1$ . So, in the case the charge gets split among  $e'$  and the other link (or other links), and the charge on  $e'$  is  $\leq 1/2$ .

If we consider charges on  $e'$  further up on the tree, it is clear that the charges get halved at each iteration. So, the largest possible charge for the 1A and 2A steps is  $1 + 1/2 + 1/4 + \dots < 2$ . ■

The following theorem sums up the final accounting for the analysis:

**Theorem 18.** *The Hein et al. variant of the algorithm is a 5-approximation for rSPR.*

**Proof.** We will show that each link can only be given a total charge of at most 5. Let  $l$  be a link. By Lemma 17 part i),  $l$  can only be charged once for any of the B cases. ■

Suppose  $l$  is charged by Case 1B. The cost is 1, with a possible additional cost of 2 by at most one Case 3A (see Lemma 17ii). Also, the sum of charges on an edge by Cases 1A + 2A is at most 2 by Lemma 17iii. The total is  $\leq 5$ .

Suppose  $l$  is charged by Case 2B. This charge is 2. By Lemma 17i,  $l$  cannot be charged by other 1B, 2B or 3B cases. Also it cannot be charged by a Case 3A, since then the link would also have a charge for a Case 1B. Also the sum of charges of Cases 1A and 2A is at most 2. So the total is at most 4.

The equivalent argument holds if  $l$  is charged by a Case 3B.

Finally if  $l$  does not have any B charges, then the total charge can only be given by 1A and 2A cases, and the total is at most 2. ■

A similar analysis can be applied to achieve a 5-approximation of rSPR for the Rodrigues et al. variant of the algorithm.

**Theorem 19.** *The Rodrigues et al. variant of the algorithm is a 5-approximation for rSPR.*

**Proof.** Lemmas 12, 13, and 16 all hold for this version of the algorithm. There are differences in the way we assign the charges only in Cases 1 and 2. Suppose, without loss of generality, that we are at stage  $i$  of the algorithm, and that we are processing the sibling pair  $(a, b)$  of  $T_1^i$ .

**Cases 1A and 2A:**  $a$  and  $b$  are in the same  $\text{MAF}_{\text{SPR}}$  component. We identify the nodes of  $T_2$  corresponding to  $a$  and  $b$ , and a path from  $a$  to  $b$ . Say  $A_1, \dots, A_k$  are the trees that hang directly from this path, and let  $e_1, \dots, e_k$  be the edges connecting the trees to the path. If we are in Case 1A, in tree  $T_2^i$  at most two of these trees will keep any nodes. Then we put a charge of 1 on the corresponding  $e_i$  edges. Also we have to mark the edges  $e_i$  so that we do not pass down charges through them in the future. If we are in Case 2A, then at least three of these trees keeps some nodes at stage  $i$ . Without loss of generality, say they are  $A_1, \dots, A_s$ , where  $3 \leq s \leq k$ . Then we charge  $2/s$  to each of the  $s$  edges  $e_1, \dots, e_s$ . Recall that in this case we detach two nodes, so the cost is 2, and we are distributing it through  $s$  edges. Since  $3 \leq s$ ,  $2/s \leq 2/3$ . Now, we assigned charges to edges that might not be links. Now, for each  $e$  which is not a link, we pass the charge down as we did for the Hein et al. algorithm to the links below except the ones that have been marked.

**Cases 1B and 2B:** The cost of a Case 1B operation is now 1 or 2. The cost of a Case 2B operation is 2. We charge the cost to the link assigned to the “B” case.

Parts i and ii of Lemma 17 are the same. Part iii now says that the maximum charge on a link for Cases 1A + 2A is 1. This is because charges on links for Cases 1A are now final due to the marking

of edges that we just described. On the other hand cascading down charges for Cases 2A are at most  $2/3 + 2/9 + 2/27 \dots$

As for the previous algorithm, the worst situation for charging a link is when it gets assigned a Case 1B. Now this can be a charge of 2. In this case, it can also get a charge of 2 by a Case 3A. Finally by Cases 1A + 2A, it can get a charge of 1. This could be a total of 5.

If the link gets assigned a Case 2B or 3B, then the maximum charge on that link will be 3. ■

### 5. COUNTEREXAMPLE FOR TBR

Hein et al. (1996) claim a 3-approximation to TBR. Rodrigues et al. (2001) give a counterexample to that by providing a pair of trees for which the algorithm gives nearly a 4-approximation. Their counterexample consists of moves that do not take advantage of the full power of TBR. Instead, every move is in the proper subset of SPR moves. So, the counterexample also shows that Hein’s algorithm is at best a 4-approximation for rSPR. A related counterexample mentioned in Rodrigues et al. (2001) shows that the Rodrigues et al. variant is at best a 3-approximation for rSPR. Using TBR moves that are not SPR moves (i.e., take at least 2 SPR moves to emulate), we show that the algorithm of Rodrigues et al. (2001) is at best a 4-approximation. The example in Figure 5 is constructed to take full advantage of TBR (and thus does not work for SPR). In TBR, each cut edge yields an *unrooted* tree in the agreement forest (unlike the rooted trees in the SPR agreement forest). The tree in Figure 5 has subtrees (with bold edges) that are identical after the removal of an edge under TBR, but for which the algorithm (and rSPR) makes 3 cuts. A similar example shows that the algorithm of Hein et al. is not a 4-approximation, as claimed, but at best a 5-approximation.

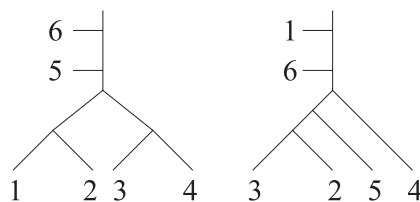
We have also isolated where the proofs break down (Hein et al., 1996; Rodrigues et al., 2001). For Hein et al. (1996), they claim that

Let  $T_1$  and  $T_2$  be rooted phylogenetic trees on the same leaf set. Then there is  $F$  a  $MAF_{SPR}$  for  $T_1$  and  $T_2$  such that the edges cut by the algorithm in Case 1 are links (Lemma 10 of Hein et al. [1996]).

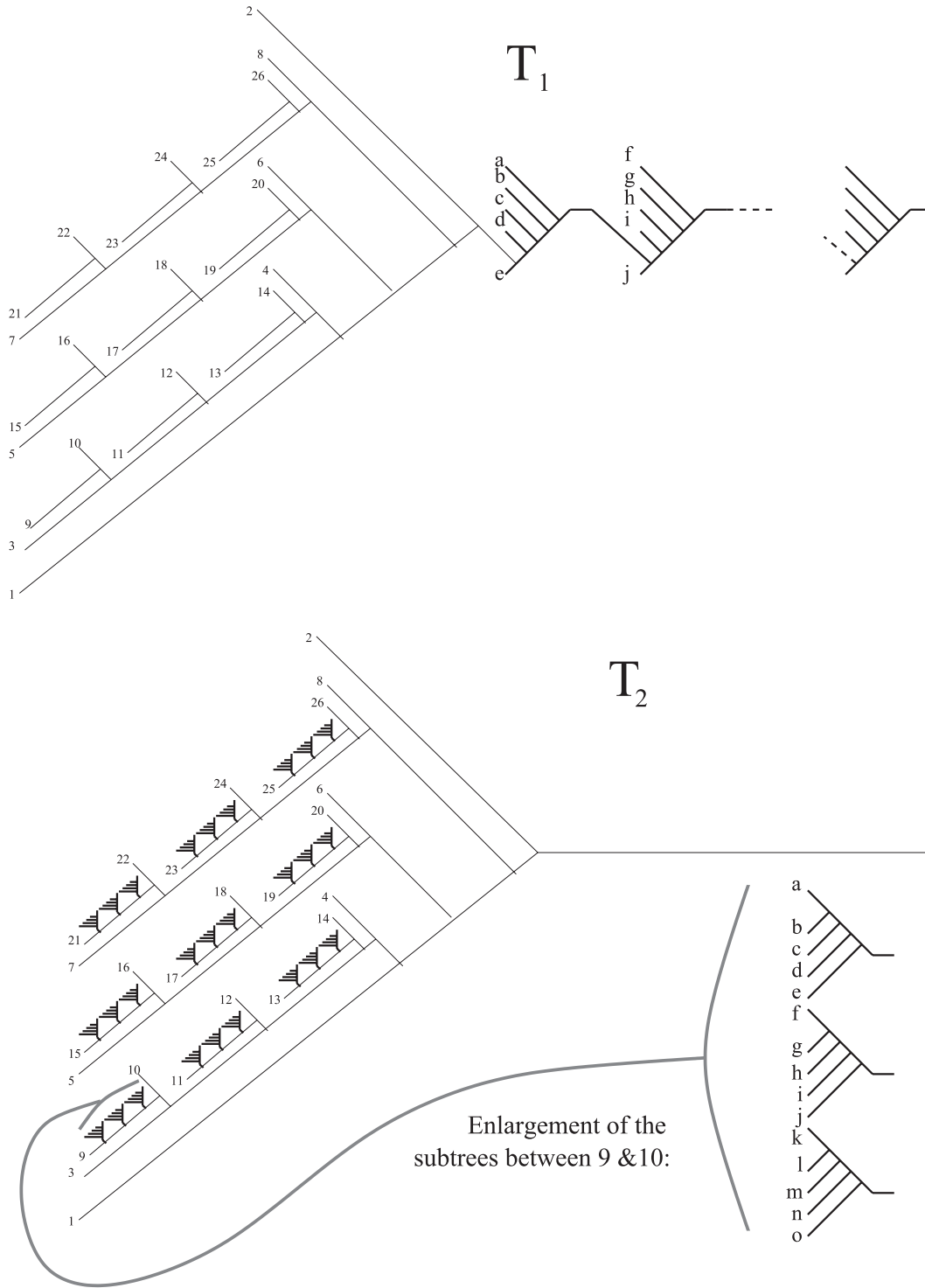
However, there are pairs of trees for which no choice of the  $MAF_{TBR}$  will give that the edges cut in Case 1 are links (Fig. 4). F4

The analysis of Rodrigues et al. (2001) also fails in a subtle way. In Lemma 4, it is claimed, without proof, that if in Case 2 of the algorithm incorrect cuts are made, that there is always a link connecting the remaining subtrees that can be charged the cost of the incorrect move. However, if you have repeated applications of Case 2, you can have remaining subtrees whose connecting edges are not links, and you will charge an edge that is not a link. Since only links can be charged, this subtle oversight undercounts the charges and makes their analysis not hold.

For example, let us run the algorithm on the trees in Figure 5. Assume you start with the sibling pair (9, 10) in the first tree. In the second tree,  $T_2$ , 9 and 10 have 3 subtrees (call them  $S_1, S_2,$  and  $S_3$ ) in between. The algorithm applies Case 2 which says to cut off the sibling pair nodes in both trees. The resulting trees after this first set of moves are missing the leaves 9 and 10. Above each of the subtrees,  $S_1, S_2,$  and  $S_3$  is a link in the  $MAF_{TBR}$ . So, the edges directly above the subtrees  $S_1, S_2,$  and  $S_3,$  are now F5



**FIG. 4.** The pair of trees give a counterexample to Lemma 10 of Hein et al. Any  $MAF_{TBR}$  for the pair of trees has  $\{2, 3, 5, 6\}$  as a component, but when the algorithm is run, the leaf 5 will be cut by a Case 1.



**FIG. 5.** A counterexample to the approximation to TBR claims of Rodrigues et al. (2001). Note that the trees are constructed so that the removal of the bold subtrees yields a maximum agreement forest. Recall that for TBR, the resulting  $MAF_{TBR}$  consists of unrooted trees. The algorithm starts by choosing a sibling pair in  $T_1$ , say (9, 10). By Case 2, 9 and 10 are cut in both trees, leaving behind the bold subtrees in  $T_1$ , listed at the bottom of the figure. This is repeated for (11, 12), (13, 14), ..., (25, 26). Note that all of these are bad cuts. The same process is repeated for (3, 4), (5, 6), (7, 8) as well as (1, 2), leaving only the bold subtrees. The bold subtrees are then cut apart starting at their sibling pair, using 3 cuts, when a single cut at the root of the bold subtrees would suffice. This example takes  $\frac{3 \times 27 + 26}{27} = 3 + \frac{26}{27}$  times the optimal.

virtual links. Note that none of these were virtual links when we started, since 9 and 10 do not have links directly above them. But after 9 and 10 are deleted, they become a virtual link.

Continuing the analysis of the algorithm on the trees in Figure 5, the algorithm examines the sibling pairs: (9, 10), . . . , (25, 26). For each, Case 2 is applied (where you delete the pair in both trees). This gets repeated up one level with (3, 4), (5, 6), and (7, 8), and again with (1, 2). In the analysis in Rodrigues et al. (2001), it is assumed that there is a link above the remaining subtrees between 1 and 2 which we charge for removing 1 and 2. However, due to the “cascading” of the Case 2 applications, none of these remaining edges is a link, so, a non-link is incorrectly charged.

The optimal edges to cut would have been the ones above all the bold subtrees. But instead, the algorithm cuts off 1, 2, . . . , 26, plus three edges for each of the bold subtrees (the three edges are needed due to the way the bold subtrees are arranged in  $T_1$  and  $T_2$ ), using  $26 + 3 * 27$  when 27 edges would be enough.

## 6. LINEAR RUNNING TIME

We can show that all variants of the algorithm can be implemented in linear time. This is a significant improvement over the polynomial time implementation stated (but not shown) in Rodrigues et al. (2001). Our data structure for a tree contains links from the parent to its two children and a link from the child to the parent. The queue of sibling pairs to be processed is stored a linked list.

All of the algorithms look at sibling pairs in  $T_1$  and find the corresponding leaves in  $T_2$ . There are a linear number of sibling pairs examined by the algorithm (this is bounded above by the number of internal nodes of  $T_1$ , which is linear in the number of leaves). We can find the initial sibling pairs by scanning the tree  $T_1$  and placing the sibling pairs in a queue. We process each sibling pair in the queue according to the case of the algorithm that applies. The trees are preprocessed in linear time to construct a lookup table of leaf nodes. This allows leaves to be located in each tree in constant time.

We need to show that each of the linear number of sibling pairs can be processed in constant time. Each case is a local operation on the trees that can be done in constant time, so, we need only show that determining the case to be applied can also be done in constant time. The number of subtrees between those leaves in  $T_2$  determines which step of the algorithm is applied. Note that Cases 2 and 3 of all the variants of the algorithm perform the same action, so, we do not need to distinguish between them in our checks. To decide which case to apply, we perform several simple (and local) operations. Each check takes constant time:

- if parent(a) does not exist or parent(b) does not exist, then either  $a$  or  $b$  is a singleton in  $T_2^i$  and we must be in Case 4.
- else if parent(a) = parent(b), then  $a$  and  $b$  are a sibling pair in  $T_2^i$  and we must be in Case 5.
- else if parent(a) = grandparent(b) or grandparent(a) = parent(b), then there’s a single subtree between  $a$  and  $b$  and we must be in Case 1.
- otherwise, the distance between  $a$  and  $b$  in  $T_2^i$  is larger than 2. This could occur because there’s many subtrees on the path between  $a$  and  $b$  (Case 2) or because  $a$  and  $b$  are in different components (Case 3).

## 7. EXPERIMENTAL RESULTS

Our initial experiments compare the two variants of the algorithm (Rodrigues et al., 2001; Hein et al., 1996) applied to rSPR distances on sets of real data and also on randomly generated trees. The biology trees were generated by heuristic searches on DNA and RNA sequence data and tend to be very similar in topology. The random trees were generated for varying number of taxa under a uniform and Yule-Harding distribution.

We implemented the algorithms in Java, using the code base of TreeJuxtaposer software package (Munzner et al., 2003) (freely available at [//olduvai.sourceforge.net](http://olduvai.sourceforge.net)). We chose both Java and the code base to make inclusion into tree visualization programs such as TreeJuxtaposer and TreeSet Visualization (Munzner et al., 2003; Amenta and Klingner, 2002) easier in the future.

Each dataset consists of 100 trees. We did a pairwise comparison of all distinct trees in the dataset and report the average distance and standard deviation for each dataset under the two variants of the algorithms.

number of taxa:	Uniform				Yule-Harding				
	10	50	100	500	10	50	100	500	
Hein	ave	6.28	43.30	90.70	476.80	6.27	44.77	93.73	489.16
	std	0.15	2.74	7.42	92.04	0.15	2.45	7.55	88.67
Rodrigues	ave	6.83	43.82	91.40	478.53	6.44	45.19	94.31	490.19
	std	0.17	2.88	9.37	101.70	0.19	2.52	7.61	89.21

		Animal RNA	Chloroplast DNA
number of taxa:		128	28
Hein	ave	14.8	7.6
	std	1.17	0.53
Rodrigues	ave	16.1	8.1
	std	1.05	0.49

**FIG. 6.** The experimental results of the Hein et al. and Rodrigues et al. variants of the algorithm run on datasets of 100 trees. The data reports the average distance and standard deviation between distinct trees in the dataset.

**AU5**

We looked at two sets of biological trees: first, trees generated by parsimony search on animal RNA data (provided by the Hillis laboratory at UT Austin). Each tree has 128 leaves. For the Hein et al. variant, the average distance between trees was 14.8 versus 16.1 for the Rodrigues et al. variant. The second set of trees was also generated by parsimony search on chloroplast DNA (provided by the Jansen laboratory at UT Austin). each tree has 28 leaves. We did a pairwise comparison of all distinct trees in the dataset. For the Hein et al. variant, the average distance between trees was 7.6 versus 8.1 for the Rodrigues et al. variant.

On the biological datasets, the Hein et al. variant algorithm performs better experimentally. This is somewhat surprising due to the the stronger counterexample for Hein et al. versus Rodrigues et al.: we have a counterexample for Hein et al. that take almost 4 times the optimal answer for rSPR versus the best known counterexample for Rodrigues et al. that takes only 3 times. Our analysis of the algorithms in Section 4 suggests another possibility. The worst situation for the Hein et al. variant is many cascading 2A charges. The Rodrigues et al. variant of the algorithm is designed to minimize the problems with the cascading 2A charges, and instead performs worst when there are 1B and 3A charges (see Lemma 17 and its corresponding version in Theorem 19). This suggests that biological trees (that we analyzed) have less nesting of sibling pairs inside one another (the cascading 2A cases) and more occurrences of sibling pairs occurring in different components of the  $MAF_{SPR}$  (the 1B and 3A cases). For trees with a different bias, we could expect to see the Rodrigues et al. variant doing well.

We then applied both variants of the algorithm to randomly generated trees under both the uniform and Yule-Hardy distributions. We created datasets of 100 trees with taxa of 10, 50, 100, and 500. We then ran both variants of the algorithm on the pairwise comparison of the trees and compared the average distance. In contrast to the biological data sets, we saw no statistically significant difference between the two variants of the algorithm, except at 10 taxa for the uniform model. For the other 7 datasets of random trees, the average distances reported were within the standard deviations. Interestingly, the average distance for random trees scales linearly with the number of taxa under both distributions of random trees.

## 8. CONCLUSION

We have given the first approximation algorithm for the important rSPR tree distance metric. We hope to improve the algorithm and give a tighter analysis of the running time, since we currently have a

5-approximation but have a counterexample that only requires a 4-approximation. We further plan a larger experimental study focusing on the effects of heuristics on the bounds achieved.

### ACKNOWLEDGMENTS

M.L.B. was partially supported by the Spanish grant TIN2004-04343. Also this project was partially supported by NSF grants ITR 0121651 and 0121682 and computational support by MRI 0215942 and the computational facility at the CUNY Graduate Center. K.S.J. would like to thank the Centre de Recerca Matemàtica at the Barcelona for hosting her visit for Spring 2005. We would also like to thank the Hillis and Jansen laboratories at the University of Texas, Austin, for the biological tree datasets and the Munzner group at the University of British Columbia for the TreeJuxtaposer code base used in implementing the algorithms.

### REFERENCES

- Addario-Berry, L., Chor, B., Hallett, M., et al. 2003. Ancestral maximum likelihood of phylogenetic trees is hard. *Lect. Notes Comput. Sci.* 2812, 202–215.
- Allen, B., and Steel, M. 2001. Subtree transfer operations and their induced metrics on evolutionary trees. *Ann. Combinatorics* 5, 1–13.
- Amenta, N., and Klingner, J. 2002. Case study: visualizing sets of evolutionary trees. *8th IEEE Symp. Inform. Visualization (InfoVis 2002)* 71–74. Software available at: [comet.lehman.cuny.edu/treeviz](http://comet.lehman.cuny.edu/treeviz).
- Baroni, M., Grunewald, S., Moulton, V., et al. 2005. Bounding the number of hybridisation events for a consistent evolutionary history. *TJ. Math. Biol.* 51, 171–182.
- Bordewich, M., and Semple, C. 2005. On the computational complexity of the rooted subtree prune and regraft distance. *Ann. Combinatorics* 8, 409–423.
- Bordewich, M., and Semple, C. 2006. Computing the minimum number of hybridisation events for a consistent evolutionary history (submitted).
- Bush, R., Bender, C., Subbarao, K., et al. 1999. Predicting the evolution of human influenza a. *Science* 286, 1921–1925.
- Day, W.H.E. 1985. Optimal algorithms for comparing trees with labeled leaves. *J. Classific.* 2, 7–28.
- Ding, Z., Filkov, V., and Gusfield, D. 2005. A linear-time algorithm for the perfect phylogeny haplotyping (pph) problem. *RECOMB 2005* 585–600.
- Foulds, L.R., and Graham, R.L. 1982. The Steiner tree problem in phylogeny is np-complete. *Adv. Appl. Math.* 3, 43–49.
- Gray, R., and Atkinson, Q. 2003. Language-tree divergence times support the Anatolian theory of Indo-European origin. *Nature* 426, 435–439.
- Gusfield, D. 2004. An overview of combinatorial methods for haplotype inference. *Lect. Notes Comput. Sci.* 2983, 9–25.
- Hein, J., Jiang, T., Wang, L., et al. 1996. On the complexity of comparing evolutionary trees. *Discrete Appl. Math.* 71, 153–169.
- Hillis, D., Heath, T., and John, K.S. 2005. Analysis and visualization of tree space. *J. System. Biol.* 3, 471–482.
- Hillis, D.M. 1999. Predictive evolution. *Science* 286, 1866–1867.
- Hillis, D.M., Mable, B.K., and Moritz, C. 1996. *Molecular Systematics*. Sinauer Associates, Sunderland, MA.
- Huelsenbeck, J.P., and Ronquist, F. 2001. Mrbayes: Bayesian inference of phylogeny. AU7
- Maddison, W. 1997. Gene trees in species trees. *Syst. Biol.*
- Munzner, T., Guimbrètière, F., Tasiran, S., et al. 2003. TreeJuxtaposer: scalable tree comparison using Focus+Context with guaranteed visibility. *SIGGRAPH 2003 Proc. (Trans. Graphics)* 453–462.
- Nakhleh, L.K., Warnow, T., Linder, C.R., et al. 2005. Reconstructing reticulate evolution in species—theory and practice. *J. Comput. Biol.* 12, 6–7.
- Novitsky, V., Smith, U.R., Gilbert, P., et al. 2002. Human immunodeficiency virus type 1 subtype c molecular phylogeny: consensus sequence for an aids vaccine design? *J. Virol.* 76, 5435–5451.
- Page, R., and Charleston, M. 1998. Trees within trees: phylogeny and historical associations. *Trends Ecol. Evol.* 13, 356–359.
- Robinson, D., and Foulds, L. 1981. Comparison of phylogenetic trees. *Math. Biosci.* 53, 131–147.
- Roch, S. 2006. A short proof that phylogenetic tree reconstruction by maximum likelihood is hard. *IEEE/ACM Trans. Comput. Biol. Bioinform.* 3, 92–94. AU8

- Rodrigues, E.M., Sagot, M.-F., and Wakabayashi, Y. 2001. Some approximation results for the maximum agreement forest problem. *Lect. Notes Comput. Sci.* 2129, 159–169.
- Swofford, D. 2002. *PAUP\*. Phylogenetic Analysis Using Parsimony (\*and Other Methods). Version 4.* Sinauer Associates, Sunderland, MA.
- Warnow, T., Ringe, D., and Taylor, A. 1996. Reconstructing the evolutionary history of natural languages. *Proc. ACM-SIAM Symp. Discrete Algorithms (SODA)* 314–322.

Address reprint requests to:  
 PLEASE PROVIDE  
 PLEASE PROVIDE  
 PLEASE PROVIDE

**AU9**

*E-mail:* PLEASE PROVIDE

### AU1

**Please review your article as a whole for correctness. There were discrepancies between the hardcopy (copy-edited) manuscript and electronic file (including an additional figure/new Figure 3). Please verify that the resubmitted version of this paper appears correctly as shown in these proofs. Also, please ensure that all artwork, symbols, and special characters are intact/correct as shown and that there were no software interpretation problems. Thank you!**

### AU2

**Right running head (short title) okay as shown?**

### AU3

**Please provide Key words.**

### AU4

**“Lemma 17” correct as shown?**

### AU5

**Please cite Figure 6 in text.**

### AU6

**Update available?**

### AU7

**Please provide journal, volume number, and page range.**

### AU8

**Please provide volume number and page range.**

### AU9

**Please provide address and email information.**