Approximating Subtree Distances Between Phylogenies

Maria Luisa Bonet* Katherine St. John†‡ Ruchi Mahindru†§ Nina Amenta¶

September 23, 2005

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 by Bordewich and Semple [5]. This paper presents the first approximation result for this important tree distance. The algorithm follows a standard format for tree distances such as Rodigues *et al.* [22] and Hein *et al.* [11]. 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.

*Lenguajes y Sistemas Informaticos (LSI), Universidad Politecnica de Catalunya (UPC), 08034 Barcelona, Spain bonet@lsi.upc.edu.

†Department of Math & Computer Science, Lehman College– City University of New York (CUNY), Bronx, NY 12581, {stjohn,ruchi.mahindru}@lehman.cuny.edu.

‡Department of Computer Science, CUNY Graduate Center, New York, NY 10016

§Computer Science Department, Stony Brook University (SUNYSB), Stony Brook, NY 11794-4400

¶Computer Science Department, University of California,Davis, CA 95616, amenta@cs.ucdavis.edu.

1 Introduction

Phylogenies, 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 [12]. In addition to the direct applications, techniques for building and comparing phylogenies have been used for designing vaccines [6, 14, 19], haplotyping [8, 10], and determining the evolution of human language [24]. Most optimization criteria for creating phylogenies are NP-hard [1, 9]. 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 (for example [15, 23]) and analyzing and visualizing these requires distances to be computed quickly [3, 13]. While the Robinson-Foulds distance [21] (basically the sum of the false positives and false negatives) is often used since it can be calculated in linear time [7], it lacks biological motivation. The TBR and rooted SPR metrics have better biological intuition [12]. Unfortunately, both are NP-hard to compute [2, 5].

In addition to its value in phylogenetic reconstruction searches, rooted SPR distance is also valuable for more complex evolutionary events. Maddison [16] showed that the number of reticulation events between two gene trees and rSPR of those two trees are closely related. Building on this equivalence has led to advances in network models of evolution [4, 18, 20].

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 [11, 22], 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 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.* [11] 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 their proofs. Allen and Steel [2] gave a counterexample to the argument in [11] and a correct proof for the NP-completeness of TBR distance. Rodriguez *et al.* [22] gave a counterexample to the approximation algorithm for TBR distance in [11] and provide a new approximation algorithm. Unfortunately, the approximation bounds of [22] are not correct– we give a counterexample in Section 5. Recently, Bordewich and Semple [5] 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.

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 [2, 5, 22].

Definition 1. *[5]: 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. *[2]: A* **subtree prune and regraft** *(SPR) operation on a binary tree T is defined as cutting any edge and thereby pruning a*

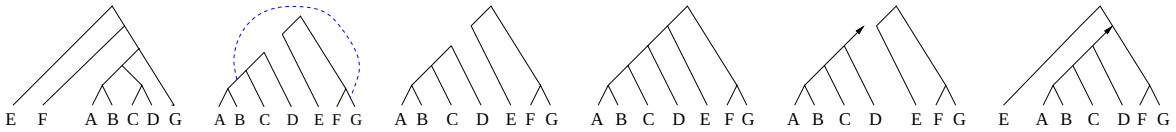


Figure 1: Beginning with the start tree, we show an example of a TBR move towards the left and an example of a rSPR move towards the right. Note that the rSPR move prunes a rooted subtree and must regraft it by the same cut edge. For TBR, an edge is removed entirely and a new edge (the blue dotted edge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

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**. *See Figure 2.*

Definition 3. *[2]: 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 betweeen 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 2.*

Allen and Steel [2] showed that the TBR distance between two trees is one less than the size of the maximum agreement forest of the two trees. Bordewich and Semple [5] 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 rooted trees, and since we focus only on rooted trees, we will call it simply maximum agreement forest or MAF.

Definition 4. *[5]: Let T be a rooted binary phylogenetic X-tree (or more briefly a tree) and V a subset of the vertex set of T. $\mathbf{T}(\mathbf{V})$ is the minimal rooted subtree of T that connects the elements of V. $\mathbf{T} \mid \mathbf{V}$ 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 5. *[5]: 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 ρ at the end of an edge adjoined to the original root. Furthermore, we regard ρ 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 $\{\mathcal{T}_\rho, \mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k\}$ where $\mathcal{T}_\rho$ is a rooted tree and $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$ are roted binary phylogenetic trees with label sets $\mathcal{L}_\rho, \mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_k$ 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 $i \in \{\rho, 1, 2, \ldots, k\}$,*

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

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

Definition 6. *A (rSPR)* **maximum agreement forest** *(MAF) for T_1 and T_2 is the agreement forest for T_1 and T_2 with the minimal number of components.*

The MAF need not be unique, so in the subsequent proof, we will select an arbitrary MAF F and use it consistently.

Definition 7. *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 [11, 22] have been proposed to approximate MAF 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, 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 ith 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.

Figure 2 gives the Hein *et al.* variant of the algorithm. The Rodriguez *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.

Definition 8. *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.*

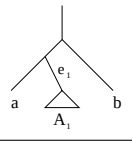
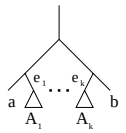
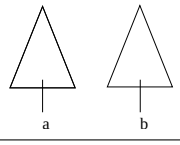
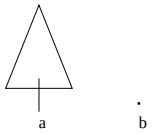
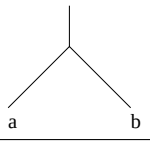
Case 1: There's 1 subtree between a & b in T_2^i: Cut the subtree in T_2^i. $N := N + 1$	
Case 2: There are ≥ 2 subtrees between a & b in T_2^i: Cut a & b in T_1^i & T_2^i. $N := N + 2$.	
Case 3: a & b are in separate components of T_2^i: Cut a & b in T_1^i & T_2^i. $N := N + 2$.	
Case 4: b is a singleton in T_2^i: Cut b in T_1^i. N doesn't change.	
Case 5: The sibling pair occurs in both: Replace it by "(a, b)," in T_1^i & T_2^i. N doesn't change.	

Figure 2: The cases of Hein *et al.*'s variant of the algorithm at step i.

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 doesn't occur in the forest for the other tree, as in [5], we view the root as a node ρ 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 $X \cup \rho$.

An edge cut in T_2^i at step i corresponds to a particular edge of T_2. We call this a **cut edge** in T_2. Recall that the links are defined in terms of the initial trees, T_1 and T_2 and the MAF F. Next we will define a related concept, that of virtual link, in terms of T_2, the forest T_2^i, and F.

Definition 9. *An edge e in T_2^i is a* **virtual link** *of T_2^i (or at step i) if every path between it and a leaf contains a link. If the leaf is a contraction of nodes due to Case 5, we undo the contractions.*

Since virtual links are defined in terms of the forests T_2^i for some step i, an edge may not be a virtual link at stage j but could become one at some stage $j + s$. 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 MAF 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 10. *Say that F is the MAF of T_1 and T_2, and a and b are 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 (the "A" cases), and those where a and b don't share any component of the MAF (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. The following lemma deals with Cases 1A and 2A.

Lemma 11. *Let T_1 and T_2 be rooted phylogenetic trees on the same leaf set. Let F be a MAF for T_1 and T_2, and T_1^i and T_2^i be the result after the ith 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 $A_1, A_2, \ldots, A_k$, $k \geq 1$. Call the edges above the subtrees $e_1, e_2, \ldots, e_k$ (see Figure 2). If a and b share a component t of F (see Definition 10), then*

i) For all j, none of the leaves of A_j participate in the MAF component t.

ii) $e_1, e_2, \ldots, e_k$ are all virtual links at stage i of the algorithm.

Proof of Lemma 11:

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 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, t is an induced rooted subtree of T_1^0 and of T_2^0, and therefore l' should appear in T_1^0 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 doesn't participate in t.

Proof of ii: Recall that a and b share a rooted component t of the MAF 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, that means that there is a path down from e_i to a leaf l of T_2 without a link, and hence a link-free path from a' and b' to l'. So l' would participate in t, which contradicts part i) of this lemma. □

Lemma 11 shows that in Cases 1A and 2A, the edges e_i might not be links, but virtual links. 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 *el al.* and Rodriguez *et al.*, this subtlety was missed (Theorem 9 of [11] and Lemma 4 of [22]) and is the main reason their approximation bounds are incorrect. This is illustrated in the example in Appendix 5.

The following lemma will help us analyze Case 3A.

Lemma 12. *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.*

The proof of this Lemma can be found in Appendix B.

The following definition and lemmas will help us deal with 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. The definition of virtual link can be trivially adapted to the tree T_1^i.

Definition 13. *Let a be a node of T_1^i or T_2^i, and let then u_a be the edge that has a as its lower endpoint.*

Definition 14. *Let a be a leaf of T_1^i and let $(a_1, b_1) \ldots (a_k, b_k)$ be sibling pairs handled in previous steps $i_1, \ldots, i_k$ of the algorithm. Suppose that $\forall j, 1 \leq j \leq k$, a_j doesn't share any MAF components with b_j and that $\forall j, 1 \leq j \leq k$, a_j and b_j are contracted into a, that is, steps $i_1, \ldots, i_k$ of the were all Case 1B (which is always followed by a Case 5). Define* **nol(a)** *to be the number of links among $\{u_{a_1}, u_{b_1}, \ldots, u_{a_k}, u_{b_k}\}$, and define* **nop(a)** *to be the number of pairs k.*

Lemma 15. *Let a be a leaf of T_1^i. Then if u_a is a virtual link at stage i which is not a link, then $nop(a) < nol(a)$. Otherwise, $nop(a) \leq nol(a)$.*

The proof of this Lemma can be found in Appendix B.

Lemma 16. *Let T_1 and T_2 be rooted phylogenetic trees on the same leaf set, and let F be a MAF for T_1 and T_2.*

i) Suppose a and b are a sibling pair in T_1^i, but not in T_2^i, and that a and b don't share a component of F. Then at least one of u_a or u_b is a virtual link in T_1^i.

ii) Suppose $(a_1, b_1) \ldots (a_k, b_k)$ are sibling pairs in $T_1^{i_1} \ldots T_1^{i_k}$ respectively, and assume that a_i and b_i don't share any components of the MAF. Then the number of different links for the k paths between a_i and b_i is greater than or equal to k.

Proof of Lemma 16:

Proof of i): Since a and b don't 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 u_a or u_b is a virtual link in T_1^i.

Proof of ii): We will show by induction on the number of pairs that correspond to "B" cases, that for each sibling pair (a, b) there is a link in T_1 that is either u_a or u_b or inside a subtree of T_1 contracted into a or b.

Suppose that the statement is true up to $<$ s pairs, and we are dealing with pair (a_s, b_s). By the i) part of this lemma, then either u_{a_s} or u_{b_s} or both are virtual links. If u_{a_s} or u_{b_s} is a link, we are done. This link cannot have been used by one of the prior $s-1$ pairs. If say u_{a_s}

is a virtual link, then by lemma 15, $nop(a_s) < nol(a_s)$ and $nop(b_s) \leq nol(b_s)$. This means that inside a_s there is a link that hasn't been used by the previous sibling pairs contracted into a_s and we assign that link to the pair (a_s, b_s). □

Finally, observe that T_2 contains the same number of links as T_1. So we can choose some arbitrary matching of the links of T_1 with the links of T_2, to assign the charges associated with the "B" cases to links of T_2.

4.2 Assigning Charges & Analysis

To be able to analyze the algorithm, we will describe how to charge a cost of 1 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. To be able to do this, we need to identify a node, say a of T_2^i, with a node of T_2 the following way. If a is a leaf in T_2 as well as in T_2^i, then we identify the leaves. If a is a contraction of several leaves of T_2, then we identify it with the internal node in T_2 that is the least common ancestor of all the nodes contracted into a.

Next we describe how to assign the charges for the different cases. Suppose wlog 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 MAF component. We identify the nodes of T_2 corresponding to a and b, and a path from a to b. Say $A_1, \ldots, A_k$ are the trees that hang directly from this path, and let $e_1, \ldots, e_k$ 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 any nodes. Wlog 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. Wlog say they are $A_1, \ldots, A_s$, where $2 \leq s \leq k$. Then we charge $2/s$ to each of the s edges $e_1, \ldots, e_s$. Recall that in this case we detach two nodes, so the cost is 2, and we are distributing it through s edges. Now, we asigned 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_2^0. 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 11). 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 12). 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 that 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 of Lemma 17:

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 12, 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 MAF 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 $lca(a, b)$ doesn't have c as a descendant, while in T_2 it does. But this cannot happen by the definition of MAF, giving a contradiction.

Part iv): 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 MAF, and c and d also must share a component of the MAF. Let e' be the link that is charged twice. Since e' is below $lca(a, b)$ and below $lca(c, d)$, there are three posibilities for the relative positions of a, b, c, d in T_2: c and d are below the $lca(a, b)$, one between c and d is below and the other is above, and finally, c and d are above $lca(a, b)$.

If c and d are below the $lca(a, b)$, 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 $lca(a, b)$ 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 MAF component, the link must be above $lca(c, d)$. 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 MAF component. This gives a contradiction with the definition of MAF, 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 $lca(a, b)$. 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_2^0, we first identify in T_2^0 the nodes 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. Say the first s trees haven't completely dessapeared by step i. Say e' is in A_1. Since $2 \leq s$, e' gets a charge ≤ 1 from this step. When we process the pair (c, d) a charge ≤ 1 will be passed to $lca(e_1 \dots e_s)$. 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 dettached, and later a and b get contracted. Looking back at T_2^0 we assign to e' a charge ≤ 1 from this step. Later, in step j, we process the sibling pair (c, d). In T_2^0 we identify the nodes c and d as explained before. Say from the path between c and d, hang trees $A_1, \dots A_k$, and $e_1, \dots e_k$ are the connecting edges. Wlog e' is in A_1. Now, if at step j, the whole A_1 has dessapeared, then we don't charge e_1 and as a consequence we don't 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 dessapeared. 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 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 17i, l can only be charged once for any of the B cases.

Suppose l is charged by Case 1B. This will be 1 point, and it can also be charged with 2 points 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 ≤ 5.

Suppose l is charged by Case 2B. This charge is 2. By Lemma 17i, l cannot be charged by other 1B or 2B or 3B cases. Also it cannot be charged by a Case 3A, since then the link would also have a charge for a 1B case. 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 doesn't 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 Rodigues variant of the algorithm. The proof is included in Appendix C.

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

5 Counterexample for TBR

Hein *et al.* [11] claim a 3-approximation to TBR. Rodriguez *et al.* [22] 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. Using TBR moves that are not SPR moves (i.e. take at least 2 SPR moves to emulate), we show that the algorithm of Rodigues *et al.* [22] is at best a 4-approximation. See Figure 5 in Appendix. A similar example shows that the algorithm of Hein *et al* is not a 4-approximation, as claimed, but at best a 5-approximation.

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 stated polynomial time implementation of Rodriguez *et al.* [22]. 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 leaves 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 determing 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 experiments compare the two variants of the algorithm, Rodriguez *et al.* [22] and Hein *et al.* [11], 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 [17] (freely available at `//olduvai.sourceforge.net`). We chose both Java and the code base to make inclusion into tree visualization programs such as TreeJuxtaposer and TreeSet Visualization [17, 3] 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 . 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 variant, the average distance between trees was 14.8 versus 16.1 for the Rodriguez 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 variant, the average distance between trees was 7.6 versus 8.1 for the Rodriguez variant.

On the biological datasets, the Hein variant algorithm performs better experimentally. This is somewhat surprising due to the the stronger counterexample for Hein versus Sagot: we have a counterexample for Hein that take almost 4 times the optimal answer for rSPR versus the best known counterexample for Sagot that takes only 3 times. Our analysis of the algorithms in Section 4 suggests another possibility. The worst situation for the Hein variant is many cascading 2A charges. The Rodigues 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 **??** and its corresponding version in Appendix C). 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 (the 1B and 3A cases). For trees with a different bias, we could expect to see the Rodigues 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 & Future Work

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

		Animal RNA	Chloroplast DNA	Uniform				Yule-Harding			
number of taxa:		128	28	10	50	100	500	10	50	100	500
Hein	ave	14.8	7.6	6.28	43.30	90.70	476.80	6.27	44.77	93.73	489.16
	std	1.17	0.53	0.15	2.74	7.42	92.04	0.15	2.45	7.55	88.67
Rodigues	ave	16.1	8.1	6.83	43.82	91.40	478.53	6.44	45.19	94.31	490.19
	std	1.05	0.49	0.17	2.88	9.37	101.70	0.19	2.52	7.61	89.21

Figure 3: The experimental results of the Hein and Rodigues variants of the algorithm run on datasets of 100 trees. The table reports the average distance and standard deviation between distinct trees in the dataset.

further plan are larger experimental study focusing on the effects of heuristics on the bounds achieved.

9 Acknowledgments

The first author was partially supported by the Spanish grant TIN2004-04343. Also this project was partially supported by USA NSF grants ITR 0121651 and 0121682 and computational support by MRI 0215942 and the computational facility at the CUNY Graduate Center. The second author would like to thank the Centre de Recerca Matematica 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

[1] L. Addario-Berry, B. Chor, M. Hallett, J. Lagergren, A. Panconesi, and T. Wareham. Ancestral maximum likelihood of phylogenetic trees is hard. In *Proc. 3rd International Workshop Algs. in Bioinformatics (WABI03)*, volume 2812 of *Lecture Notes in Computer Science*, pages 202–215, 2003.

[2] B. Allen and M. Steel. Subtree transfer operations and their induced metrics on evolutionary trees. *Annals of Combinatorics*, 5:1–13, 2001.

[3] Nina Amenta and Jeff Klingner. Case study: Visualizing sets of evolutionary trees. In *8th IEEE Symposium on Information Visualization (InfoVis 2002)*, pages 71–74, 2002. Software available at `comet.lehman.cuny.edu/treeviz`.

[4] M. Bordewich and C. Semple. Computing the minimum number of hybridisation events for a consistent evolutionary history. submitted.

[5] M. Bordewich and C. Semple. On the computational complexity of the rooted subtree prune and regraft distance. *Annals of Combinatorics*, 8:409–423, 2005.

[6] RM Bush, CA Bender, K Subbarao, NJ Cox, and WM Fitch. Predicting the evolution of human influenza a. *Science*, 286(5446):1921–5, 3 December 1999.

[7] W. H. E. Day. Optimal algorithms for comparing trees with labeled leaves. *Journal of Classification*, 2:7–28, 1985.

[8] Zhihong Ding, Vladimir Filkov, and Dan Gusfield. A linear-time algorithm for the perfect phylogeny haplotyping (pph) problem. In *RECOMB 2005*, pages 585–600, 2005.

[9] L. R. Foulds and R. L. Graham. The Steiner tree problem in phylogeny is np-complete. *Advances in Appl. Math.*, 3:43–49, 1982.

[10] Dan Gusfield. An overview of combinatorial methods for haplotype inference. In S. Istrail, M. Waterman, and A. Clark, editors, *Computational Methods for SNPs and Haplotype Inference*, volume 2983 of *Lecture Notes in Computer Science*, pages 9–25. Springer, 2004.

[11] J. Hein, T. Jiang, L. Wang, and K. Zhang. On the complexity of comparing evolutionary trees. *Discrete Applied Mathematics*, 71:153–169, 1996.

[12] D. M. Hillis, B. K. Mable, and C. Moritz. *Molecular Systematics*. Sinauer Assoc., Sunderland, Mass., 1996.

[13] David Hillis, Tracy Heath, and Katherine St. John. Analysis and visualization of tree space. *J. of Systematic Biology*, 3:471–482, 2005.

[14] David M. Hillis. Predictive evolution. *Science*, 286(5446):1866–1867, 3 December 1999.

[15] John P. Huelsenbeck and Fredrik Ronquist. Mrbayes: Bayesian inference of phylogeny, 2001.

[16] W.P. Maddison. Gene trees in species trees. *Syst. Biol.*, 1997.

[17] Tamara Munzner, François Guimbrètiere, Serdar Tasiran, Li Zhang, and Yunhong Zhou. TreeJuxtaposer: Scalable tree comparison using Focus+Context with guaranteed visibility. *SIGGRAPH 2003 Proceedings, published as special issue of Transactions on Graphics*, pages 453–462, 2003.

[18] Luay K. Nakhleh, Tandy Warnow, C. Randal Linder, and Katherine St. John. Reconstructing reticulate evolution in species–theory and practice. To appear in Journal of Computational Biology.

[19] V. Novitsky, U. R. Smith, P. Gilbert, M. F. McLane, P. Chigwedere, C. Williamson, T. Ndung'u, I. Klein, S. Y. Chang, T. Peter, I. Thior, B. T. Foley, S. Gaolekwe, N. Rybak, S. Gaseitsiwe, F. Vannberg, R. Marlink, T. H. Lee, and M. Essex. Human immunodeficiency virus type 1 subtype c molecular phylogeny: Consensus sequence for an aids vaccine design? *Journal of Virology*, 76(11):5435–5451, June 2002.

[20] R.D.M. Page and M.A. Charleston. Trees within trees: phylogeny and historical associations. *Trends in Ecology and Evolution*, 13(9):356–359, 1998.

[21] D.R. Robinson and L.R. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53:131–147, 1981.

[22] E. M. Rodrigues, M.-F. Sagot, and Y. Wakabayashi. Some approximation results for the maximum agreement forest problem. In J. D. P. Rolim M. Goemans, K. Jansen and L. Trevisan, editors, *Approximation, Randomization and Combinatorial Optimization: Algorithms and Techniques (APPROX and RANDOM 2001), Berkeley, California, Lecture Notes in Computer Science*, volume 2129, pages 159–169, 2001.

[23] D.L. Swofford. *PAUP*. Phylogenetic Analysis Using Parsimony (*and Other Methods). Version 4*. Sinauer Associates, Sunderland, Massachusetts, 2002.

[24] T. Warnow, D. Ringe, and A. Taylor. Reconstructing the evolutionary history of natural languages. In *Association for Computing Machinery and the Society of Industrial and Applied Mathematics, Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 314–322, 1996.

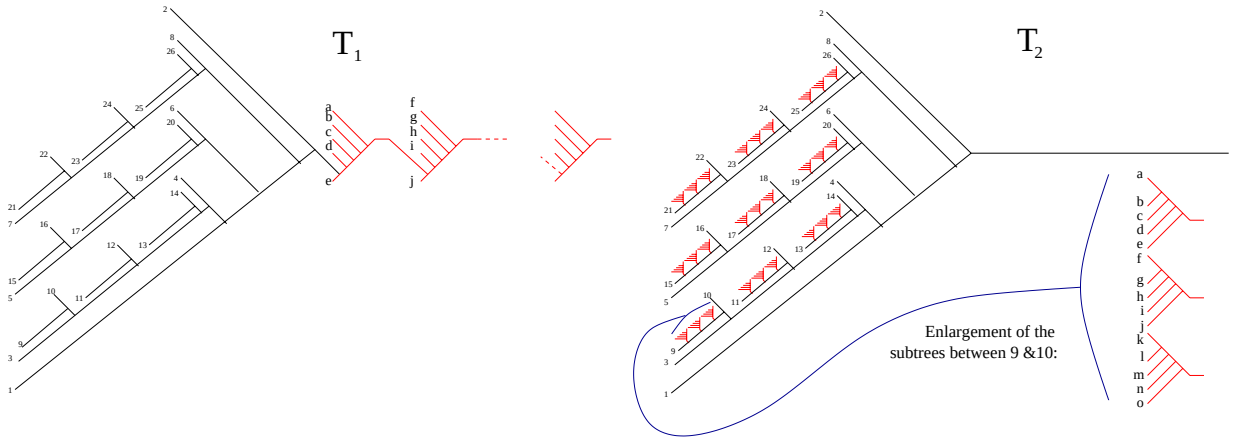


Figure 4: A counterexample to the approximation to TBR claims of Rodriguez *et al.*. Note that the trees are constructed so that the removal of the red subtrees yields a maximum agreement forest. The algorithm starts by choosing a sibling pair in T_1, say $(9, 10)$. By Case 2, 9 and 10 are cut in both trees, leaving behind the red subtrees in T_1, listed to the left side of the picture. This is repeated for $(11, 12), (13, 14), \ldots, (25, 26)$. Note 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 red subtrees. The red subtrees are then cut apart starting at their sibling pair, using 3 cuts, when a single cut at the root of the red subtrees would suffice. This example takes $\frac{3\times 27+26}{27} = 3 + \frac{26}{27}$ times the optimal.

A Appendix - Counterexamples

Figure 4 shows a counterexample to the claim that the algorithm of Rodriguez *et al.* [22] gives a 3-approximation to the TBR distance.

We have isolated where the proofs break down in [11, 22]. For Hein *et al.*, they claim that

> Let T_1 and T_2 be rooted phylogenetic trees on the same leaf set. Then there is F a rSPR MAF for T_1 and T_2 such that the edges cut by the algorithm in Case 1 are links. (Lemma 10 of [11]).

However, there are pairs of trees for which no choice of the MAF will give that the edges cut in Case 1 are links. For example, see Figure 5.

The analysis of Rodriguez *et al.* [22] also fails in a subtle way. In Lemma 4 of [22], 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 and then charge it 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 makes their analysis not hold.

For example, let us run the algorithm on the trees in Figure 4. Assume you start with the sibling pair $(9, 10)$ in the first tree. 9 and 10 in the second tree 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 is missing 9 and 10. Above each of the subtrees, S_1, S_2, and S_3 is a link in the MAF. So, the edges directly above the subtrees S_1, S_2, and S_3, are now 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.

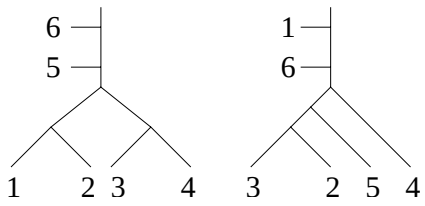


Figure 5: The pair of trees give a counterexample to Lemma 10 of Hein *et al.*. Any MAF for the pair of trees has $\{2, 3, 5, 6\}$ as a component, but when the algorithm is run, the leaf 5 will be cut by a Case 1.

Continuing on the trees in Figure 4, the sibling pairs examined are $(9, 10), \ldots, (25, 26)$. 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 [22], it's 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 red subtrees. But instead, the algorithm cuts off $1, 2, \ldots, 26$, plus three edges for each of the red subtrees (the three edges are needed due to the way the red subtrees are arranged in T_1 and T_2), using a $26 + 3 * 27$ when 27 edges would be enough.

B Appendix - Proofs of Technical Lemmata

Proof of Lemma 12: 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. So 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. □

Proof of Lemma 15. We prove it by induction on d the depth of a. Base Case: if the depth of a is 0, then a is not contracted and is also a leaf of T_1^i. In this case $nop(a) = 0$ and $nol(a) = 0$, and the statement of the lemma holds.

Induction Case: suppose the lemma holds for nodes of depth $\leq d - 1$. Let a be a node of depth d, and suppose $a = (a', b')$. We will use the following facts:

- if a' and b' share a component, $nop(a') + nop(b') = nop(a)$
- if a' and b' don't share a component, $nop(a') + nop(b') + 1 = nop(a)$
- if $u_{a'}$ and $u_{b'}$ are links, $nol(a') + nol(b') + 2 = nol(a)$.
- if only one among $u_{a'}$ and $u_{b'}$ is a link, then $nol(a') + nol(b') + 1 = nol(a)$.
- if none of $u_{a'}$ and $u_{b'}$ are links, then $nol(a') + nol(b') = nol(a)$.

Case I: u_a is a virtual link that is not a link. Then a' and b' don't share any components, and since there has to be a link from u_a to every leaf contracted into a, $u_{a'}$ and $u_{b'}$ have to be virtual links. If $u_{a'}$ and $u_{b'}$ are links, then by the induction hypothesis, $nop(a') \leq nol(a')$ and $nop(b') \leq nol(b')$. So,

$$\begin{aligned} nop(a) &= nop(a') + nop(b') + 1 \\ &< nol(a') + nol(b') + 2 = nol(a) \end{aligned}$$

If $u_{a'}$ is a link and $u_{b'}$ is a virtual link not link then by the induction hypothesis, $nop(a') \leq nol(a')$ and $nop(b') < nol(b')$. So,

$$\begin{aligned} nop(a) &= nop(a') + nop(b') + 1 \\ &< nol(a') + nol(b') + 1 = nol(a) \end{aligned}$$

If $u_{a'}$ and $u_{b'}$ are virtual links but not links, then by induction hypothesis, $nop(a') < nol(a')$ and $nop(b') < nol(b')$. So,

$$\begin{aligned} nop(a) &= nop(a') + nop(b') + 1 \\ &< nol(a') + nol(b') = nol(a) \end{aligned}$$

Case II: u_a is either a link or not a virtual link.
Case II.1: a' and b' share a MAF component. Then a' and b' are not virtual links. So by the induction hypothesis, $nop(a') \leq nol(a')$ and $nop(b') \leq nol(b')$. So

$$\begin{aligned} nop(a) &= nop(a') + nop(b') \\ &\leq nol(a') + nol(b') = nol(a) \end{aligned}$$

Case 2.2: a' and b' don't share any components. Then $u_{a'}$ or $u_{b'}$ are virtual links. Wlog $u_{a'}$ is the virtual link. Now, if $u_{a'}$ is a not a link, then

$$\begin{aligned} nop(a) &= nop(a') + nop(b') + 1 \\ &\leq nol(a') + nol(b') = nol(a) \end{aligned}$$

Now, if $u_{a'}$ is a link, then

$$\begin{aligned} nop(a) &= nop(a') + nop(b') + 1 \\ &\leq nol(a') + nol(b') + 1 = nol(a) \end{aligned}$$

□

C Appendix– Proof of Theorem 19

Proof of Theorem 19: Lemmas 11, 12, 15 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 wlog 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 MAF component. We identify the nodes of T_2 corresponding to a and b, and a path from a to b. Say $A_1, \ldots, A_k$ are the trees that hang directly

from this path, and let $e_1, \ldots, e_k$ 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 don't 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. Wlog say they are $A_1, \ldots, A_s$, where $3 \leq s \leq k$. Then we charge $2/s$ to each of the s edges $e_1, \ldots, e_s$. 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 asigned 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 has to say that the maximum charge on a link for cases 1A+2A is 1. This is because charges on links for "1A" cases 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 $2/3 + 2/9 + 2/27 \ldots$.

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. □