From Constrained to Unconstrained Maximum Agreement Subtree in Linear Time*

V. Berry† Z.S. Peng‡ H.F. Ting‡

Abstract

We propose and study the Maximum Constrained Agreement Subtree (MCAST) problem, which is a variant of the classical Maximum Agreement Subtree (MAST) problem. Our problem allows users to apply their domain knowledge to control the construction of the agreement subtrees in order to get better results. We show that the MCAST problem can be reduced to the MAST problem in linear time and thus we have algorithms for MCAST with running times matching the fastest known algorithms for MAST.

Key Words. Maximum Agreement Subtrees, Constrained Maximum Agreement Subtrees, Consensus, Reduction, Bioinformatics, Evolutionary trees.

1 Introduction

Evolutionary trees, which are rooted trees with their leaves labeled by some unique species, are commonly used to capture the evolutionary relationship of the species in nature. Different biological theories capture different kinds of evolutionary relationships and induce different evolutionary trees. To find out how much these theories have in common, we compare the corresponding evolutionary trees and find some consensus of these trees.

*A preliminary version of this paper appears in the *Proceedings of the Fifth Workshop on Algorithms in Bioinformatics* (WABI 2005).

†Departement Informatique, L.I.R.M.M., Université de Montpellier II - C.N.R.S., vberry@lirmm.fr.

‡Department of Computer Science, The University of Hong Kong, Pokfulam Road, Hong Kong, {zspeng,hfting}@cs.hku.hk.

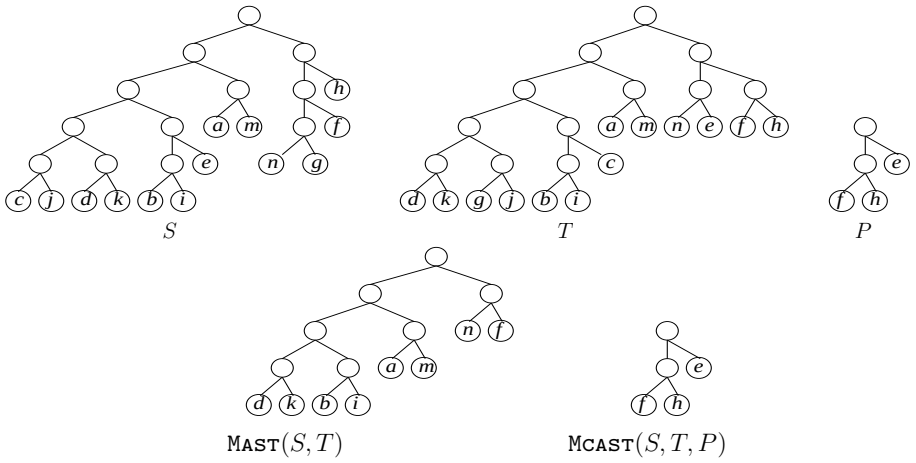


Figure 1: Maximum agreement and maximum constrained agreement subtrees

One successful approach for finding a consensus of different evolutionary trees is to construct their *maximum agreement subtree* (MAST), which is the largest evolutionary tree that is a topological subtree of all the given trees. There are many algorithms proposed for constructing MAST; for example, [5, 6, 9, 10, 12, 13, 17], or more recently, [1, 2, 4, 14].

A problem of these algorithms is that they do not allow users to apply their biological knowledge to control the construction of the consensus. For example, the evolutionary relationships of many species are well understood. Any evolutionary tree including these species should be consistent with these commonly accepted relationships. With this additional constraint, MAST is not a good measure for comparing evolutionary trees. Let us consider the trees S and T in Figure 1. Note that the maximum agreement subtree of S and T is large, and one would consider that the two trees are similar. However, the two trees agree on almost nothing if we insist that the agreement subtree must be consistent with the evolutionary relationships of e, f, h, which are given by the tree P. In fact, if P is a correct relationship, then S and T infer different evolutionary relationships for many other species. For example, for the species a, S suggests that the least common ancestor of a and e is different from the least common ancestor of a and f, while T suggests they are the same.

To allow users to enforce such predefined relationship in the agreement subtree, we propose and study the *maximum constrained agreement subtree* (MCAST) problem, which is defined as follows:

Let S and T be two evolutionary trees, and P be an agreement

subtree of S and T. Find a largest agreement subtree of S and T that contains P as a subtree. We say that this agreement subtree is a *maximum constrained agreement subtree* of S and T with respect to P.

In [15], we gave an $O(n \log n)$ time recursive algorithm for this problem when the input trees are binary. However, it is difficult to generalize the algorithm for general trees. In this paper, we give a deeper analysis of the structure of the constrained agreement subtrees and show that the MCAST problem can be reduced to the Maximum Agreement Subtree (MAST) problem in linear time. Note that this reduction is not surprising when P is empty or has only one leaf. If P is the empty tree, our MCAST problem is just the MAST problem. If P has only one leaf κ, the problem is equivalent to finding a largest agreement subtree A of S and T that contains κ. By a simple trick, we can reduce the problem to the MAST problem as follows. Let $|S|$ and $|T|$ be the number of leaves in S and T, respectively. To find A, we simply replace the leaf κ in S and T by some large tree X of size at least $|S| + |T|$. Then, any maximum agreement subtree A' of the enlarged trees must contain X. In other words, the role of X is the same as the role of κ in S and T. By replacing X in A' by κ, we get A.

The major contribution of this paper is to show that we have this reduction even for general P. We prove in Theorem 11 that given S, T and P, we can find in linear time (i.e., $O(|S|+|T|)$ time) subtrees $S_1, S_2, \ldots, S_m$ of S, $T_1, T_2, \ldots, T_m$ of T, and $P_1, \ldots, P_m$ of P such that

1. to find a maximum constrained agreement subtree of S and T with respect to P, it suffices to find a maximum constrained agreement subtrees of S_i and T_i with respect to P_i for $1 \leq i \leq m$,
2. $\sum_{1 \leq i \leq m} (|S_i| + |T_i|) \leq 2(|S| + |T|)$ and
3. each P_i has only one leaf.

As mentioned above, finding a MCAST of S_i and T_i with respect to the single leaf tree P_i can be reduced to finding a MAST of two trees with size doubled. Therefore, if $\phi(n)$ is the worst case running time of an algorithm for finding a maximum agreement subtree of two trees with totally n leaves, then $\tau(S, T, P)$, the time complexity of finding a maximum constrained agreement

	MAST	MCAST
Binary trees	$O(n \log n)$ [3]	$O(n \log n)$
Trees with constant degree d	$O(\sqrt{d} n \log n)$ [16]	$O(\sqrt{d} n \log n)$
General trees	$O(n^{1.5})$ [11]	$O(n^{1.5})$

Table 1: Time complexity of MAST and MCAST

subtree of S and T with respect to P, can be bounded as follows:

$$\begin{aligned} \tau(S,T,P) &= \sum_{1 \le i \le m} \tau(S_i, T_i, P_i) + O(|S| + |T|) \\ &\le \sum_{1 \le i \le m} \phi(2(|S_i| + |T_i|)) + O(|S| + |T|). \end{aligned} \tag{1}$$

We note that for all existing algorithms for MAST, their running times are upper bounded by some convex functions $\phi(n)$, and by Jensen's inequality [8], we have

$$\sum_{1 \le i \le m} \phi(2(|S_i| + |T_i|)) \le \phi\Big(\sum_{1 \le i \le m} (2(|S_i| + |T_i|)\Big) \le \phi(4(|S| + |T|)). \tag{2}$$

From (1) and (2), we conclude that the time complexity of solving an instance of MCAST is no more than that of solving an instance of MAST with input size four times of the original one. For a summary, Table 1 lists the running time of the MCAST problem by our reduction using the fastest known MAST algorithms for different kinds of trees. Note that our method can indeed handle the more general case where the constraint is a forest instead of a tree; we will give the details in the last section.

Our paper is organized as follows. In Section 2, we give the necessary definitions and notations for our discussion. We also prove some properties on agreement subtrees that help simplify our analysis. In Sections 3 and 4, we analyze the structure of the agreement subtrees, and in Section 5, we detail our reduction. We handle the case with forest constraints in Section 6.

2 Preliminaries

A *labeled* tree S is a rooted tree with every leaf being labeled with a unique species. In this paper, we use the label of the leaf as its name. Let $\mathcal{L}(S)$

denote the set of leaves of S. For any two leaves a, b, let $\texttt{lca}_S(a, b)$ denote the *least common ancestor* of a, b in S. Given any subset $H \subseteq \mathcal{L}(S)$ of leaves, the *restricted subtree* of S on H, denoted as $S\|_H$, is the subtree of S whose nodes include the set of leaves in H as well as the least common ancestors of any two leaves in H, and whose edges preserve the ancestor-descendant relationship of S. Intuitively, $S\|_H$ can be constructed as follows: Discard those leaves of S not in H, as well as those internal nodes whose degrees eventually become one; then contract every path whose intermediate nodes are each of degree two into an edge. The following fact comes directly from the definition.

Fact 1. *Suppose that $H \subseteq L \subseteq \mathcal{L}(S)$. Then, we have (i) for any two leaves $a, b \in H$, $\texttt{lca}_{S\|_H}(a, b) = \texttt{lca}_{S\|_L}(a, b)$, and (ii) $(S\|_L)\|_H = S\|_H$.*

Let T be another labeled tree. We say that S and T are *leaf-label preserving isomorphic* if (i) they have the same set of leaves (i.e., $\mathcal{L}(S) = \mathcal{L}(T)$) and (ii) there exists a bijection f from the nodes of S to the nodes of T such that for any pair of leaves a, b of S, $f(\texttt{lca}_S(a, b)) = \texttt{lca}_T(a, b)$. Note that for any leaf a, $f(a) = f(\texttt{lca}_S(a, a)) = \texttt{lca}_T(a, a) = a$; f maps every leaf in S to the leaf in T with the same label. We write $S = T$ if the two trees are leaf-label preserving isomorphic.

Observe that given any two trees S and T with the same set of leaves, we can always define a mapping f such that for any pair of leaves a, b, $f(\texttt{lca}_S(a, b)) = \texttt{lca}_T(a, b)$. However, the necessary and sufficient condition for f being bijective, and hence $S = T$, is that for any two pairs of leaves a, b and c, d (not necessarily distinct), we have

$$\texttt{lca}_S(a, b) = \texttt{lca}_S(c, d) \text{ if and only if } \texttt{lca}_T(a, b) = \texttt{lca}_T(c, d). \tag{3}$$

The following lemma gives a somewhat simpler condition; it helps to simplify our analysis given in the rest of the paper.

Lemma 1. *Following is a necessary and sufficient condition for $S = T$: for any three leaves a, b, c, we have*

$$\texttt{lca}_S(a, b) = \texttt{lca}_S(a, c) \iff \texttt{lca}_T(a, b) = \texttt{lca}_T(a, c). \tag{4}$$

Proof. It suffices to prove that (3) is equivalent to (4). Obviously, (3) implies (4). To prove the other direction, suppose that (3) does not hold. In other words, there are four leaves a, b, c, d such that in one tree, say S, we have

$\texttt{lca}_S(a,b) = \texttt{lca}_S(c,d)$, but $\texttt{lca}_T(a,b) \neq \texttt{lca}_T(c,d)$. Below, we identify three leaves from a, b, c, d that violate (4).

In T, since $\texttt{lca}_T(a,b) \neq \texttt{lca}_T(c,d)$, they cannot be descendant of each other at the same time. Thus, one of them, say $\texttt{lca}_T(a,b)$, is not a descendant of $\texttt{lca}_T(c,d)$, and this further implies either a or b, say a, is not a descendant of $\texttt{lca}_T(c,d)$. In other words, all of the ancestors of a are not $\texttt{lca}_T(c,d)$, and it follows that

$$\texttt{lca}_T(a,c) \neq \texttt{lca}_T(c,d) \text{ and } \texttt{lca}_T(a,d) \neq \texttt{lca}_T(c,d). \tag{5}$$

In S, since $\texttt{lca}_S(a,b) = \texttt{lca}_S(c,d)$, a is a descendant of $\texttt{lca}_S(c,d)$. Let w be the least common ancestor of $\texttt{lca}_S(a,c)$ and $\texttt{lca}_S(a,d)$. Since $\texttt{lca}_S(a,c)$ and $\texttt{lca}_S(a,d)$ are on the same path from a to the root, their least common ancestor w must be equal to one of these two nodes, i.e., $w = \texttt{lca}_S(a,c)$ or $w = \texttt{lca}_S(a,d)$. We observe that $w = \texttt{lca}_S(c,d)$ because

- $\texttt{lca}_S(c,d)$ is an ancestor of a, c, d in S and hence it is an ancestor of w, and
- w is an ancestor of c and d in S, and hence is an ancestor of $\texttt{lca}_S(c,d)$.

Therefore

$$\texttt{lca}_S(a,c) = \texttt{lca}_S(c,d) \text{ or } \texttt{lca}_S(a,d) = \texttt{lca}_S(c,d). \tag{6}$$

Taking (5) and(6) together, we conclude that (4) does not hold; the lemma follows. □

We say that a subset $K \subseteq \mathcal{L}(S) \cap \mathcal{L}(T)$ of leaves is an *agreement leaf subset* of S and T if $S\|_K = T\|_K$; the two restricted subtrees are called *agreement subtrees* of S and T. Suppose that K is an agreement leaf subset of S and T. A leaf subset $L \subseteq \mathcal{L}(S) \cap \mathcal{L}(T)$ is called a *constrained agreement leaf subset of S and T with respect to K* if

(i) $K \subseteq L$ and

(ii) L is an agreement leaf subset of S and T.

Note that given a constrained agreement leaf subset, we can find the corresponding agreement subtree in linear time, and *vice versa*. The classical *maximum agreement subtree problem* asks to find the largest agreement leaf subset of S and T. In this paper, we study the *maximum constrained agreement subtree*, which asks for finding the maximum constrained agreement leaf subset

of S and T with respect to K. As shown in Figure 1, the output of the two problems can be very different.

In the rest of the paper, we assume that $K \neq \emptyset$ and $S\|_K = T\|_K$. We define $\texttt{CAST}(S, T, K)$ to be the set of all constrained agreement leaf subsets of S and T with respect to K, and define $\texttt{MCAST}(S, T, K) \subseteq \texttt{CAST}(S, T, K)$ to be the subset of those with maximum size. In the next two sections, we describe some structural properties on S, T and K, which help us to design efficient algorithms for solving the maximum constrained agreement subtree problem, or equivalently, finding an element in $\texttt{MCAST}(S, T, K)$. Our analysis depends on an arbitrary, but fixed leaf κ in K. We consider two cases. In the following section, we focus on the case when κ is a child of the root of both S and T; we call such leaf a *shallow* leaf. The existence of a shallow leaf in K greatly simplifies our analysis. We handle the other case, that is when κ is not a shallow leaf, in Section 4.

3 The case when κ is a shallow leaf

In this section, we show that the existence of a shallow leaf imposes some restrictions on the structure of a constrained agreement leaf subset. The following lemma describes one such restriction. Recall that a *rooted subtree* of some tree X is the whole subtree rooted at some child of X's root.

Lemma 2. *Suppose that $L \in \texttt{CAST}(S, T, K)$ and κ is a shallow leaf in K. For any rooted subtrees S' of S and T' of T, if S' and T' have a common leaf in L (i.e., $L \cap \mathcal{L}(S') \cap \mathcal{L}(T') \neq \emptyset$) then $L \cap \mathcal{L}(S') = L \cap \mathcal{L}(T')$.*

Proof. It suffices to prove that for any leaves $a, b \in L$, a, b are in different rooted subtrees of S if and only if a, b are in different rooted subtrees of T, or equivalently, $\texttt{lca}_S(a, b)$ is the root of S if and only if $\texttt{lca}_T(a, b)$ is the root of T.

From Fact 1, $\texttt{lca}_S(a, b) = \texttt{lca}_{S\|_{\mathcal{L}(S)}}(a, b) = \texttt{lca}_{S\|_L}(a, b)$, and $\texttt{lca}_T(a, b) = \texttt{lca}_{T\|_{\mathcal{L}(T)}}(a, b) = \texttt{lca}_{T\|_L}(a, b)$. Since $\kappa \in K \subseteq L$, $a, b \in L$, and $S\|_L = T\|_L$, by Lemma 1, $\texttt{lca}_{S\|_L}(a, b) = \texttt{lca}_{S\|_L}(a, \kappa) \iff \texttt{lca}_{T\|_L}(a, b) = \texttt{lca}_{T\|_L}(a, \kappa)$. The lemma follows immediately because κ is a shallow leaf, and $\texttt{lca}_{S\|_L}(a, \kappa)$ and $\texttt{lca}_{T\|_L}(a, \kappa)$ are the root of S and T, respectively. □

Note that $K \in \texttt{CAST}(S, T, K)$ and we can apply Lemma 2 to conclude that for any rooted subtree S' of S, if S' has a leaf in K, then there is a rooted

subtree T' of T such that $K \cap \mathcal{L}(S') = K \cap \mathcal{L}(T')$. Let $S_1, S_2, \ldots, S_m$ be all the rooted subtrees of S that contain some leaf in K, and $T_1, T_2, \ldots, T_m$ be the rooted subtrees of T where $K \cap \mathcal{L}(S_i) = K \cap \mathcal{L}(T_i)$. Suppose that S_m and T_m are the subtrees composed of the single shallow leaf κ. Define S_0 to be the tree obtained by removing $S_1, S_2, \ldots, S_{m-1}$ from S. Note that only S_m remains in S_0 and thus S_0 has a single leaf in K, namely κ. The other rooted subtrees of S_0 are those of S which do not contain any leaf of K. Define T_0 similarly. It should be clear that $K \cap \mathcal{L}(S_0) = K \cap \mathcal{L}(T_0) = \{\kappa\}$. We call $\langle (S_0, S_1, \ldots, S_{m-1}), (T_0, T_1, \ldots, T_{m-1}) \rangle$ the κ-*decomposition* of S and T with respect to K. The following lemma shows that κ-decomposition imposes another restriction on the structure on any constrained agreement leaf subset.

Lemma 3. *Suppose that* $L \in \textsc{Cast}(S, T, K)$. *Then, for* $0 \leq i \leq m-1$,

$$L_i = L \cap \mathcal{L}(S_i) \in \textsc{Cast}(S_i, T_i, K \cap \mathcal{L}(S_i))$$

Proof. Since for each $1 \leq i \leq m-1$, $K \cap \mathcal{L}(S_i) = K \cap \mathcal{L}(T_i) \neq \emptyset$ and $K \subseteq L$, S_i and T_i have a common leaf in L. By Lemma 2, we conclude $L \cap \mathcal{L}(S_i) = L \cap \mathcal{L}(T_i)$. It follows that the remaining leaves of L in S and T are the same; in other words, $L \cap \mathcal{L}(S_0) = L \cap \mathcal{L}(T_0)$. Therefore, for each $0 \leq i \leq m-1$, $L_i = L \cap \mathcal{L}(S_i) = L \cap \mathcal{L}(T_i)$. We use this fact to prove that $S_i\|_{L_i} = T_i\|_{L_i}$, and hence $L_i = L \cap \mathcal{L}(S_i) \in \textsc{Cast}(S_i, T_i, K \cap \mathcal{L}(S_i))$ as follows: Consider any leaves $a, b, c \in L_i$. We have

$$\begin{aligned}
\mathtt{lca}_{S_i\|_{L_i}}(a,b) = \mathtt{lca}_{S_i\|_{L_i}}(a,c) &\iff \mathtt{lca}_{S\|_{L_i}}(a,b) = \mathtt{lca}_{S\|_{L_i}}(a,c) \\
&\qquad\qquad (\text{as } L_i \subseteq \mathcal{L}(S_i)) \\
&\iff \mathtt{lca}_{S\|_L}(a,b) = \mathtt{lca}_{S\|_L}(a,c) \quad (\text{as } L_i \subseteq L) \\
&\iff \mathtt{lca}_{T\|_L}(a,b) = \mathtt{lca}_{T\|_L}(a,c) \\
&\qquad\qquad (\text{as } S\|_L = T\|_L) \\
&\iff \mathtt{lca}_{T\|_{L_i}}(a,b) = \mathtt{lca}_{T\|_{L_i}}(a,c) \\
&\iff \mathtt{lca}_{T_i\|_{L_i}}(a,b) = \mathtt{lca}_{T_i\|_{L_i}}(a,c).
\end{aligned}$$

By Lemma 1, we conclude that $S_i\|_{L_i} = T_i\|_{L_i}$. □

The following theorem shows that based on the κ-decomposition, we can solve the MCAST problem by solving some smaller subproblems.

Theorem 4. *For* $0 \leq i \leq m-1$, *let* $H_i \in \textsc{McAst}(S_i, T_i,\ K \cap \mathcal{L}(S_i))$. *Then,* $H = \bigcup_{0 \leq i \leq m-1} H_i$ *is in* $\textsc{McAst}(S, T, K)$.

Proof. Note that $K = \bigcup_{0 \le i \le m-1}(K \cap \mathcal{L}(S_i)) \subseteq \bigcup_{0 \le i \le m-1} H_i = H$. Below, we prove that $S\|_H = T\|_H$ and hence $H \in \texttt{Cast}(S, T, K)$. By Lemma 1, it suffices to prove that for any three leaves $a, b, c \in H$, we have

$$\texttt{lca}_{S\|_H}(a, b) = \texttt{lca}_{S\|_H}(a, c) \iff \texttt{lca}_{T\|_H}(a, b) = \texttt{lca}_{T\|_H}(a, c). \tag{7}$$

Note that if a, b, c are all in the same leaf set H_i, then

$$\begin{aligned}\texttt{lca}_{S\|_H}(a, b) = \texttt{lca}_{S\|_H}(a, c) &\iff \texttt{lca}_{S\|_{H_i}}(a, b) = \texttt{lca}_{S\|_{H_i}}(a, c) \\ &\iff \texttt{lca}_{T\|_{H_i}}(a, b) = \texttt{lca}_{T\|_{H_i}}(a, c) \iff \texttt{lca}_{T\|_H}(a, b) = \texttt{lca}_{T\|_H}(a, c),\end{aligned}$$

and we have (7). Suppose that a, b, c are not in the same leaf set. Either a, b or a, c are in different sets. Assume that a and c are in different sets H_i and H_j. Then, $\texttt{lca}_{S\|_H}(a, c)$ and $\texttt{lca}_{T\|_H}(a, c)$ are the root of S and T, respectively. Therefore, to prove (7), it suffices to prove that

$$\texttt{lca}_{S\|_H}(a, b) \text{ is the root of } S \iff \texttt{lca}_{T\|_H}(a, b) \text{ is the root of } T. \tag{8}$$

Note that if a, b are in different leaf sets, $\texttt{lca}_{S\|_H}(a, b)$ and $\texttt{lca}_{T\|_H}(a, b)$ are the roots of S and T, respectively. If a, b are in the same set H_i where $i \neq 0$, a, b are within the rooted subtrees S_i in S and subtree T_i in T; hence, $\texttt{lca}_{S\|_H}(a, b)$ and $\texttt{lca}_{T\|_H}(a, b)$ are not the root of S and T. For the case when $a, b \in H_0$, recall that $\kappa \in K \cap \mathcal{L}(S_0) \subseteq H_0$ and $S_0\|_{H_0} = T_0\|_{H_0}$. Thus, for the three leaves $a, b, \kappa \in H_0$, we have

$$\texttt{lca}_{S_0\|_{H_0}}(a, b) = \texttt{lca}_{S_0\|_{H_0}}(a, \kappa) \iff \texttt{lca}_{T_0\|_{H_0}}(a, b) = \texttt{lca}_{T_0\|_{H_0}}(a, \kappa). \tag{9}$$

Note that (9) is equivalent to (8) because (i) $\texttt{lca}_{S_0\|_{H_0}}(a, b)$=$\texttt{lca}_{S\|_{H_0}}(a, b)$= $\texttt{lca}_{S\|_H}(a, b)$, $\texttt{lca}_{T_0\|_{H_0}}(a, b)$=$\texttt{lca}_{T\|_{H_0}}(a, b)$=$\texttt{lca}_{T\|_H}(a, b)$, and (ii) $\texttt{lca}_{S_0\|_{H_0}}(a, \kappa)$ and $\texttt{lca}_{T_0\|_{H_0}}(a, \kappa)$ are the root of S and T, respectively. Hence, in all possible cases, we have (8), and hence (7). Therefore $S\|_H = T\|_H$ and $H \in \texttt{Cast}(S, T, K)$.

To see that $H \in \texttt{McAst}(S, T, K)$, i.e., H is a largest element in $\texttt{Cast}(S, T, K)$, let us consider any $L \in \texttt{Cast}(S, T, K)$. Lemma 3 asserts that for $0 \le i \le m-1$, $L_i = L \cap \mathcal{L}(S_i) \in \texttt{Cast}(S_i, T_i, K \cap \mathcal{L}(S_i))$. Since $H_i \in \texttt{McAst}(S_i, T_i, K \cap \mathcal{L}(S_i))$, we have $|L_i| \le |H_i|$. Then, $|L| = \sum_{0 \le i \le m-1} |L_i| \le \sum_{0 \le i \le m-1} |H_i| = |H|$. □

4 The case when κ is not a shallow leaf

In this section, we analyze the structure of the maximum agreement leaf subsets of S and T with respect to K under the assumption that κ is not a shallow leaf.

Consider the unique path from the root of S to κ. We call the nodes on this path κ-nodes of S. Given any two different κ-nodes u, u', we say that u *is higher than* u', denoted as $u \succ u'$, if u is nearer the root. We say $u \succeq u'$ if either $u = u'$ or $u \succ u'$. Note that κ itself is the lowest κ-node in S. For any leaf a of S, define the κ-parent of a, denoted as $\kappa_S(a)$, to be the least ancestor of a that is κ-node. For any κ-node u, let $\mathcal{L}_\kappa(u) = \{a \mid \kappa_S(a) = u\}$ be the set of leaves whose κ-parents are u. Note that $\mathcal{L}_\kappa(\kappa) = \{\kappa\}$, and for any other κ-node u, $\mathcal{L}_\kappa(u)$ includes all the leaves descendant of u except those that are in the subtree rooted at the unique κ-node child of u. For any set I of κ-nodes, define $\mathcal{L}_\kappa(I) = \bigcup_{u \in I} \mathcal{L}_\kappa(u)$. We say that a κ-node u is *precious* if $\mathcal{L}_\kappa(u)$ has at least one leaf in K, i.e., $K \cap \mathcal{L}_\kappa(u) \neq \emptyset$. We have similar definitions for T.

Lemma 5. *Suppose that* $L \in \textsc{Cast}(S, T, K)$. *For any two leaves* $a, b \in L$, *we have (i)* $\kappa_S(a) \succ \kappa_S(b) \iff \kappa_T(a) \succ \kappa_T(b)$, *and (ii)* $\kappa_S(a) \neq \kappa_S(b) \iff \kappa_T(a) \neq \kappa_T(b)$.

Proof. To prove (i), suppose that $\kappa_S(a) \succ \kappa_S(b)$. Since $\kappa \in L$ and $S\|_L = T\|_L$, the three leaves in L are related as follows:

$$\texttt{lca}_{S\|_L}(a, b) = \texttt{lca}_{S\|_L}(a, \kappa) \iff \texttt{lca}_{T\|_L}(a, b) = \texttt{lca}_{T\|_L}(a, \kappa). \tag{10}$$

Note that among the ancestors of b that are on the path from b to $\kappa_S(a)$, there is only one node, namely $\kappa_S(a)$ that is an ancestor of a; hence $\texttt{lca}_{S\|_L}(a, b) = \kappa_S(a) = \texttt{lca}_{S\|_L}(a, \kappa)$ (because κ is the lowest κ-node and all κ-nodes are its ancestors). Together with (10), $\texttt{lca}_{T\|_L}(a, b) = \texttt{lca}_{T\|_L}(a, \kappa) = \kappa_T(a)$, or equivalently, we have $\kappa_T(a) \succ \kappa_T(b)$. The other direction of (i) can be proved symmetrically.

Note that (ii) follows from (i) directly. □

Let $u_1 \succ u_2 \succ \cdots \succ u_m$ be the sequence of precious κ-nodes in S. We define the κ-*decomposition* of S to be the sequence of sets $(I_1, I_2, \ldots, I_{2m})$ where

- $I_{2\ell}$ is a singleton containing the ℓth precious κ-node u_ℓ,
- I_1 contains all the κ-nodes higher than u_1, and
- for $2 \leq \ell \leq m$, $I_{2\ell-1}$ contains those κ-nodes between $u_{\ell-1}$ and u_ℓ.

See Figure 2 for an example. Note that $I_{2m} = \{\kappa\}$ and the κ-decomposition covers all the κ-nodes. We define the κ-decomposition $(J_1, J_2, \ldots, J_{2n})$ for T similarly.

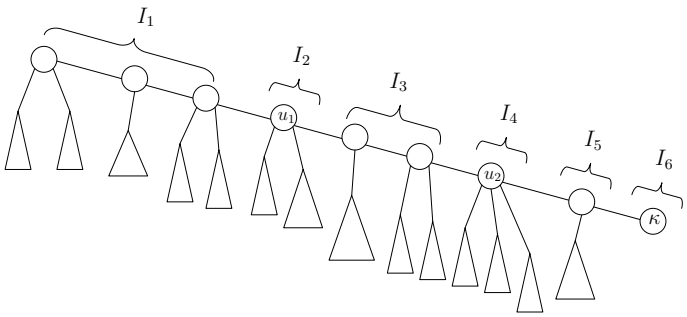


Figure 2: The κ-decomposition of S

Recall that we assume $S\|_K = T\|_K$ and hence $\textsc{Cast}(S, T, K)$ is not empty. In the rest of the section, we study the structure of any $L \in \textsc{Cast}(S, T, K)$ according the κ-decompositions $(I_1, I_2, \ldots, I_{2m})$ and $(J_1, J_2, \ldots, J_{2n})$ of S and T, respectively. The following lemma shows that the two lists have the same length, i.e., $m = n$, and there is a one-one correspondence between the sets in the lists.

Lemma 6. *Given any $L \in \textsc{Cast}(S, T, K)$, $L \cap \mathcal{L}_\kappa(I_\ell) = L \cap \mathcal{L}_\kappa(J_\ell)$ for $1 \le \ell \le 2m$. Furthermore, $m = n$.*

Proof. We claim that for every $1 \le \ell \le \min\{m, n\}$, a leaf $a \in L$ is in $\mathcal{L}_\kappa(I_\ell)$ if and only if a is in $\mathcal{L}_\kappa(J_\ell)$. This implies $L \cap \mathcal{L}_\kappa(I_\ell) = L \cap \mathcal{L}_\kappa(J_\ell)$ for $1 \le \ell \le \min\{m, n\}$. Together with the fact that $\mathcal{L}_\kappa(I_{2m}) = \mathcal{L}_\kappa(J_{2n}) = \{\kappa\}$, we conclude that $m = n$ and the lemma follows.

We prove our claim by induction. Note that by symmetry, we only need to prove that if a leaf $a \in L$ is in $\mathcal{L}_\kappa(I_\ell)$, then a is in $\mathcal{L}_\kappa(J_\ell)$. For the base case, suppose to the contrary that L has a leaf a in $\mathcal{L}_\kappa(I_1)$ but not in $\mathcal{L}_\kappa(J_1)$. Recall that J_2 has only one element, which is a precious κ-node v. It follows that $\mathcal{L}_\kappa(J_2)$ has a leaf $b \in K \subseteq L$, and $\kappa_T(b) = v \succeq \kappa_T(a)$ (because $a \notin \mathcal{L}_\kappa(J_1)$). On the other hand, by definition, $\mathcal{L}_\kappa(I_1)$ contains no leaf in K and thus $b \notin \mathcal{L}_\kappa(I_1)$ and $\kappa_S(a) \succ \kappa_S(b)$. Note that the κ-parent of the two leaves $a, b \in L$ have different relationships in S and T. By Lemma 5, $S\|_L \neq T\|_L$, a contradiction. Thus, the claim is true for $\ell = 1$.

Suppose that the claim is true for $1, 2, \ldots, \ell - 1$ and we consider ℓ. Assume that L has a leaf a in $\mathcal{L}_\kappa(I_\ell)$ but not in $\mathcal{L}_\kappa(J_\ell)$. Note that if ℓ is odd, the assumption will lead us to the contradictory conclusion that $S\|_L \neq T\|_L$ as in the base case $\ell = 1$. Suppose that ℓ is even. Then, J_ℓ has a single precious κ-node v, and there is a leaf $b \in K \subseteq L$ that is in $\mathcal{L}_\kappa(J_\ell)$. Together with

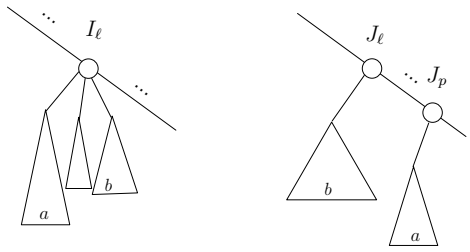


Figure 3: Relative positions of a and b.

induction hypothesis that $L \cap \mathcal{L}_\kappa(I_h) = L \cap \mathcal{L}_\kappa(J_h)$ for $1 \leq h \leq \ell - 1$, we conclude that (i) $a \in \mathcal{L}_\kappa(I_\ell)$ and $a \in \mathcal{L}_\kappa(J_p)$ for some $p > \ell$, and (ii) $b \in \mathcal{L}_\kappa(J_\ell)$ and $b \in \mathcal{L}_\kappa(I_q)$ for some $q \geq \ell$. (See Figure 3.) Therefore, $\kappa_S(a) \succeq \kappa_S(b)$ and $\kappa_T(b) \succ \kappa_T(a)$, and by Lemma 5, $S\|_L \neq T\|_L$, a contradiction. Thus the claim is also true for ℓ. □

Corollary 7. *Suppose that $L \in \textsc{Cast}(S, T, K)$. For any two leaves $a, b \in L$, if $a \in \mathcal{L}_\kappa(I_p)$ and $b \in \mathcal{L}_\kappa(I_q)$ where $p < q$ then*

(i) $\kappa_S(a) \succ \kappa_S(b)$ and $\mathtt{lca}_S(a, b) = \kappa_S(a)$, and

(ii) $\kappa_T(a) \succ \kappa_T(b)$ and $\mathtt{lca}_T(a, b) = \kappa_T(a)$.

Proof. (i) follows directly from definition. From Lemma 6, we have $L \cap \mathcal{L}_\kappa(I_p) = L \cap \mathcal{L}_\kappa(J_p)$ and $L \cap \mathcal{L}_\kappa(I_q) = L \cap \mathcal{L}_\kappa(J_q)$. Hence, $a \in \mathcal{L}_\kappa(J_p)$ and $b \in \mathcal{L}_\kappa(J_q)$ and we have (ii). □

Below, we prove some properties that are similar to those given in Lemma 3 and Theorem 4. First, we need to extend the leaf sets of the κ-decomposition as follows: for $1 \leq \ell \leq 2m$, let $\bar{\mathcal{L}}_\kappa(I_\ell) = \mathcal{L}_\kappa(I_\ell) \cup \{\kappa\}$ and $\bar{\mathcal{L}}_\kappa(J_\ell) = \mathcal{L}_\kappa(J_\ell) \cup \{\kappa\}$. Note that $K \in \textsc{Cast}(S, T, K)$ and by Lemma 6, we have $K \cap \mathcal{L}_\kappa(I_\ell) = K \cap \mathcal{L}_\kappa(J_\ell)$, and hence $K \cap \bar{\mathcal{L}}_\kappa(I_\ell) = K \cap \bar{\mathcal{L}}_\kappa(J_\ell)$ for $1 \leq \ell \leq 2m$. It follows that $\textsc{Cast}(S\|_{\bar{\mathcal{L}}_\kappa(I_\ell)}, T\|_{\bar{\mathcal{L}}_\kappa(J_\ell)}, K \cap \bar{\mathcal{L}}_\kappa(I_\ell))$ is not empty.

Lemma 8. *Suppose that $L \in \textsc{Cast}(S, T, K)$. For $1 \leq \ell \leq 2m$, the set $L_\ell = L \cap \bar{\mathcal{L}}_\kappa(I_\ell)$ is in $\textsc{Cast}(S\|_{\bar{\mathcal{L}}_\kappa(I_\ell)}, T\|_{\bar{\mathcal{L}}_\kappa(J_\ell)}, K \cap \bar{\mathcal{L}}_\kappa(I_\ell))$.*

Proof. Obviously $K \cap \bar{\mathcal{L}}_\kappa(I_\ell) \subseteq L_\ell$. Below, we show that $(S\|_{\bar{\mathcal{L}}_\kappa(I_\ell)})\|_{L_\ell} = (T\|_{\bar{\mathcal{L}}_\kappa(J_\ell)})\|_{L_\ell}$ and the lemma follows.

By Lemma 6, we have $L \cap \mathcal{L}_\kappa(I_\ell) = L \cap \mathcal{L}_\kappa(J_\ell)$ and hence $L \cap \bar{\mathcal{L}}_\kappa(I_\ell) = L \cap \bar{\mathcal{L}}_\kappa(J_\ell)$. Therefore, $L_\ell = L \cap \bar{\mathcal{L}}_\kappa(I_\ell) = L \cap \bar{\mathcal{L}}_\kappa(J_\ell)$, and $(S\|_{\bar{\mathcal{L}}_\kappa(I_\ell)})\|_{L_\ell} = S\|_{L_\ell}$

and $(T\|_{\bar{\mathcal{L}}_\kappa(J_\ell)})\|_{L_\ell} = T\|_{L_\ell}$. As in the proof of Lemma 3, we have, for any three leaves $a, b, c \in L_\ell$,

$$\begin{aligned}
&\mathtt{lca}_{(S\|_{\bar{\mathcal{L}}_\kappa(I_\ell)})\|_{L_\ell}}(a,b) = \mathtt{lca}_{(S\|_{\bar{\mathcal{L}}_\kappa(I_\ell)})\|_{L_\ell}}(a,c) \iff \mathtt{lca}_{S\|_{L_\ell}}(a,b) = \mathtt{lca}_{S\|_{L_\ell}}(a,c) \\
&\iff \mathtt{lca}_{S\|_L}(a,b) = \mathtt{lca}_{S\|_L}(a,c) \iff \mathtt{lca}_{T\|_L}(a,b) = \mathtt{lca}_{T\|_L}(a,c) \iff \\
&\mathtt{lca}_{T\|_{L_\ell}}(a,b) = \mathtt{lca}_{T\|_{L_\ell}}(a,c) \iff \mathtt{lca}_{(T\|_{\bar{\mathcal{L}}_\kappa(J_\ell)})\|_{L_\ell}}(a,b) = \mathtt{lca}_{(T\|_{\bar{\mathcal{L}}_\kappa(J_\ell)})\|_{L_\ell}}(a,c),
\end{aligned}$$

and by Lemma 1, $(S\|_{\bar{\mathcal{L}}_\kappa(I_\ell)})\|_{L_\ell} = (T\|_{\bar{\mathcal{L}}_\kappa(J_\ell)})\|_{L_\ell}$. □

The next theorem is similar to Theorem 4; it suggests a divide-and-conquer approach to find a maximum constrained agreement leaf subset.

Theorem 9. *For $1 \le \ell \le 2m$, let $H_\ell \in \mathtt{McAst}(S\|_{\bar{\mathcal{L}}_\kappa(I_\ell)}, T\|_{\bar{\mathcal{L}}_\kappa(J_\ell)}, K \cap \bar{\mathcal{L}}_\kappa(I_\ell))$. Then, $H = \bigcup_{1\le\ell\le 2m} H_\ell$ is in $\mathtt{McAst}(S, T, K)$.*

Proof. Note that $K = \bigcup_{1\le\ell\le 2m} K \cap \bar{\mathcal{L}}_\kappa(I_\ell) \subseteq \bigcup_{1\le\ell\le 2m} H_\ell = H$. Below, we show that $S\|_H = T\|_H$, and hence $H \in \mathtt{Cast}(S, T, K)$. By Lemma 1, it suffices to prove that for any three leaves $a, b, c \in H$, we have

$$\mathtt{lca}_{S\|_H}(a,b) = \mathtt{lca}_{S\|_H}(a,c) \iff \mathtt{lca}_{T\|_H}(a,b) = \mathtt{lca}_{T\|_H}(a,c) \tag{11}$$

Note that if a, b, c are all in the same leaf set H_ℓ, then,

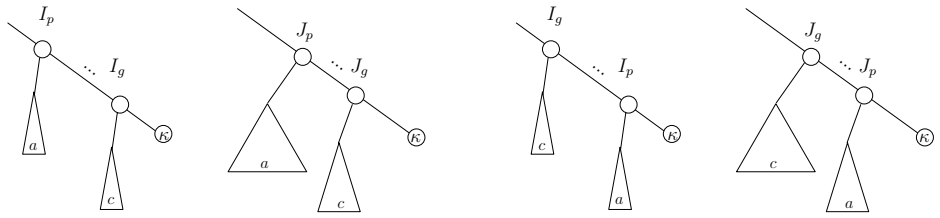
$$\begin{aligned}
&\mathtt{lca}_{S\|_H}(a,b) = \mathtt{lca}_{S\|_H}(a,c) \iff \mathtt{lca}_{S\|_{H_\ell}}(a,b) = \mathtt{lca}_{S\|_{H_\ell}}(a,c) \\
&\quad\iff \mathtt{lca}_{T\|_{H_\ell}}(a,b) = \mathtt{lca}_{T\|_{H_\ell}}(a,c) \iff \mathtt{lca}_{T\|_H}(a,b) = \mathtt{lca}_{T\|_H}(a,c),
\end{aligned}$$

and we have (11). Suppose that a, b, c are not in the same leaf set. Then, either a, b or a, c, say a, b are in different leaf sets. Suppose $a \in H_p$ and $b \in H_q$. Note that κ is in all the leaf sets because $\kappa \in K \cap \bar{\mathcal{L}}_\kappa(I_\ell) \subseteq H_\ell$ for $1 \le \ell \le 2m$; hence a and b cannot be κ. We consider two cases.

Case 1: $p < q$. Since $a \in H_p \subseteq \bar{\mathcal{L}}_\kappa(I_p)$, $b \in H_q \subseteq \bar{\mathcal{L}}_\kappa(I_q)$ and a, b are not κ, we conclude that $a \in \mathcal{L}_\kappa(I_p)$ and $b \in \mathcal{L}_\kappa(I_q)$. Together with $p < q$, we have $\mathtt{lca}_{S\|_H}(a,b) = \kappa_S(a)$ and $\mathtt{lca}_{T\|_H}(a,b) = \kappa_T(a)$ (Corollary 7). To prove (11), it suffices to show that

$$\mathtt{lca}_{S\|_H}(a,c) = \kappa_S(a) \iff \mathtt{lca}_{T\|_H}(a,c) = \kappa_T(a). \tag{12}$$

Suppose that a, c are in the same leaf set, i.e., $a, c \in H_p$. Since $\kappa \in H_p$ and $S\|_{H_p} = T\|_{H_p}$, the three leaves a, c, κ are related by $\mathtt{lca}_{S\|_{H_p}}(a,c)$=$\mathtt{lca}_{S\|_{H_p}}(a,\kappa)$



(a) When $g > p$.

(b) When $p > g$.

Figure 4: Relative positions of a and c

$=\kappa_S(a) \iff \mathtt{lca}_{T\|_{H_p}}(a,c) = \mathtt{lca}_{T\|_{H_p}}(a,\kappa) = \kappa_T(a)$. Then, we have (12) because $\mathtt{lca}_{S\|_H}(a,c) = \mathtt{lca}_{S\|_{H_p}}(a,c)$ and $\mathtt{lca}_{T\|_H}(a,c) = \mathtt{lca}_{T\|_{H_p}}(a,c)$.

Suppose that a, c are in different leaf sets and let $c \in H_g \subseteq \bar{\mathcal{L}}_\kappa(I_g)$. Again, c cannot be κ and thus $c \in \mathcal{L}_\kappa(I_g)$. From Corollary 7, if $g > p$, then $\mathtt{lca}_{S\|_H}(a,c) = \kappa_S(a)$ and $\mathtt{lca}_{T\|_H}(a,c) = \kappa_T(a)$ (see Figure 4(a)), and if $g < p$, then $\mathtt{lca}_{S\|_H}(a,c) = \kappa_S(c) \neq \kappa_S(a)$ and $\mathtt{lca}_{T\|_H}(a,c) = \kappa_T(c) \neq \kappa_T(a)$ (see Figure 4(b)). Therefore, regardless of where c is, we have (12), and hence (11).

Case 2: $p > q$. Similar to Case 1, we have $\mathtt{lca}_{S\|_H}(a,b) = \kappa_S(b)$ and $\mathtt{lca}_{T\|_H}(a,b) = \kappa_T(b)$. To prove (11), it suffices to prove that

$$\mathtt{lca}_{S\|_H}(a,c) = \kappa_S(b) \iff \mathtt{lca}_{T\|_H}(a,c) = \kappa_T(b). \tag{13}$$

Suppose $c \notin H_q$. Then neither a nor c are in $\mathcal{L}_\kappa(I_q)$ and thus their least common ancestor in S and T are not in I_q and J_q, respectively. Since $b \in \mathcal{L}_\kappa(I_q)$, $\kappa_S(b)$ and $\kappa_T(b)$ are in I_q and J_q respectively. Hence, $\mathtt{lca}_{S\|_H}(a,c) \neq \kappa_S(b)$ and $\mathtt{lca}_{T\|_H}(a,c) \neq \kappa_T(b)$ and we have (13).

Suppose $c \in H_q$. Since $b, c, \kappa \in H_q$ and $S\|_{H_q} = T\|_{H_q}$, the three leaves are related by $\mathtt{lca}_{S\|_H}(c,\kappa) = \mathtt{lca}_{S\|_H}(b,\kappa) \iff \mathtt{lca}_{T\|_H}(c,\kappa) = \mathtt{lca}_{T\|_H}(b,\kappa)$, or equivalently,

$$\kappa_S(c) = \kappa_S(b) \iff \kappa_T(c) = \kappa_T(b). \tag{14}$$

Since $p > q$, we have $\mathtt{lca}_{S\|_H}(a,c) = \kappa_S(c)$ and $\mathtt{lca}_{T\|_H}(a,c) = \kappa_T(c)$. Together with (14), we have (13) and hence (11).

In both cases, we have (11) and hence $S\|_H = T\|_H$ and $H \in \textsc{Cast}(S,T,K)$. Together with Lemma 8, we can prove easily that $H \in \textsc{McCast}(S,T,K)$ as in the proof of Theorem 4. □

5 The Reduction

In this section, we show that by applying the decomposition theorems stated in the previous sections, we can reduce in $O(|S|+|T|)$-time the problem of finding

a maximum constrained agreement subtree of S and T with respect to K to some instances of the MAST problem. To make our reduction efficient, we need to order the internal nodes according to some common measure. Without loss of generality, we assume that the set of leaves are totally ordered. Furthermore, we assume that every leaf in K is smaller than all the leaves not in K.

For every node u, define $cl(u)$, the *classifying leaf* of u, to be the smallest leaf in the subtree rooted at u. Observe that given $cl(u)$, we can decide in constant time whether the subtree rooted at u has some leaf in K, and if there is any, what is the smallest one. For every internal node u, define $\Gamma(u)$ to be the list $\langle(\ell_1, c_1), (\ell_2, c_2), \ldots, (\ell_k, c_k)\rangle$ where (i) $c_1, c_2, \ldots, c_k$ are the children of u, (ii) ℓ_i is the classifying leaf of c_i $(1 \leq i \leq k)$, and (iii) the list is in ascending order of the classifying leaves, i.e., $\ell_1 < \ell_2 < \cdots < \ell_k$. Let $cl(\Gamma(u))$ be the set of classifying leaves $\ell_1, \ell_2, \ldots, \ell_k$ that appear in $\Gamma(u)$. The following lemma shows how to construct these lists efficiently.

Lemma 10. *We can compute $\Gamma(u)$ for every internal node u of S and T in total $O(|S| + |T|)$ time.*

Proof. We consider this computation in the tree S; the case of T is done similarly. Note that by performing a depth first search on S, we can decide in $O(|S|)$ time the classifying leaf $cl(u)$ of every node u. To compute the lists $\Gamma(u)$, we initialize them as empty lists and then we fill up these lists correctly and efficiently by picking the leaves $\ell \in \mathcal{L}(S)$ one by one, from the smallest to the largest, every time appending ℓ and the corresponding child to the correct $\Gamma(u)$'s. Followings are the details.

Observe that a leaf ℓ can only appear in those lists $\Gamma(u)$ where u is along the path from ℓ to the root, and for any node u along this path, $\ell \in cl(\Gamma(u))$ if and only if ℓ is the classifying leaf of some child v of u. Hence, we "push" ℓ upward along this path as follows: Starting from the node v equal to the leaf ℓ,

- if $\ell = cl(v)$, then push ℓ to its parent u and append (ℓ, v) at the end of $\Gamma(u)$.
- if $\ell \neq cl(v)$, then stop pushing ℓ. (Note that in this case, $\ell \notin cl(\Gamma(u))$ for all the ancestor nodes u of v.)

Note that for every node $u \in S$, $\Gamma(u)$ will eventually include $(cl(c_i), c_i)$ for all the children c_i of u, and they will be in order because a pair containing a

smaller leaf will always be inserted before a pair containing a larger leaf. To see that the whole process takes linear time, observe that for every edge (u, v) of S where u is a parent of v, only the leaf $cl(v)$ can be pushed along (u, v). $\square$

We are now ready to explain how to use the lists $\Gamma(u)$ and apply the results of the previous sections to find an MCAST of S and T with respect to K.

Given any instance (S, T, K) of the MCAST problem, we say that (S, T, K) is *simple* if $|K| \leq 1$. We say that a collection $\mathcal{G} = \{(S_1, T_1, K_1), \ldots, (S_m, T_m, K_m)\}$ of MCAST problem instances is a *reduction* for (S, T, K) if for any $H_1 \in \texttt{MCAST}(S_1, T_1, K_1)$, $H_2 \in \texttt{MCAST}(S_2, T_2, K_2), \ldots, H_m \in \texttt{MCAST}(S_m, T_m, K_m)$, we have $\bigcup_{1 \leq i \leq m} H_i \in \texttt{MCAST}(S, T, K)$. We say that $\mathcal{G}$ is a *simple reduction* for (S, T, K) if every instance in $\mathcal{G}$ is simple. Define $\texttt{size}(\mathcal{G}) = \sum_{(S_i, T_i, K_i) \in \mathcal{G}} (|S_i| + |T_i|)$. The following fact follows directly from definitions.

Fact 2. *Let $\mathcal{G} = \{(S_1, T_1, K_1), (S_2, T_2, K_2), \ldots, (S_m, T_m, K_m)\}$ be a reduction for (S, T, K). Suppose that $\mathcal{G}_i$ is a reduction for some instance $(S_i, T_i, K_i) \in \mathcal{G}$. Then, $(\mathcal{G} - \{(S_i, T_i, K_i)\}) \cup \mathcal{G}_i$ is a reduction for (S, T, K).*

Note that by using the trick described in Section 1, we can solve any simple instance (S, T, K) of the MCAST problem by solving some instance of the MAST problem. We can extend this trick to handle the case when (S, T, K) is not simple by first finding a simple reduction for (S, T, K). The following theorem shows how to find such a reduction efficiently.

Theorem 11. *Given any instance (S, T, K) for the MCAST problem, we can construct in $O(\sum_{u \in S} |\Gamma(u)| + \sum_{v \in T} |\Gamma(v)|) = O(|S| + |T|)$ time a reduction $\mathcal{G}$ for (S, T, K) such that $\mathcal{G}$ is simple (i.e., $|K_i| \leq 1$ for every $(S_i, T_i, K_i) \in \mathcal{G}$) and $\texttt{size}(\mathcal{G}) = \sum_{(S_i, T_i, K_i) \in \mathcal{G}} |S_i| + |T_i| \leq 2(|S| + |T|)$.*

Proof. We pick an arbitrary leaf $\kappa \in K$ and consider the following two cases.
Case 1: κ is a shallow leaf. Observe that by a simple left-right examination of the lists $\Gamma(r_S)$ and $\Gamma(r_T)$ where r_S and r_T are the root of S and T respectively, we find the κ-decomposition $\langle (S_0, S_1, \ldots, S_{m-1}), (T_0, T_1, \ldots, T_{m-1}) \rangle$ of (S, T, K). Recall that the decomposition satisfies the following properties:

(i) $K \cap \mathcal{L}(S_0) = K \cap \mathcal{L}(T_0) = \{\kappa\}$,

(ii) $K \cap \mathcal{L}(S_i) = K \cap \mathcal{L}(T_i)$ for $1 \leq i \leq m - 1$, and

(iii) $S_1, S_2, \ldots S_{m-1}$ are disjoint rooted subtrees of S, and $T_1, T_2, \ldots T_{m-1}$ are disjoint rooted subtrees of T.

By Theorem 4, the set $\{(S_0, T_0, \{\kappa\}), (S_1, T_1, K \cap \mathcal{L}(S_1)), \ldots, (S_{m-1}, T_{m-1}, K \cap \mathcal{L}(S_{m-1}))\}$ is a reduction for (S, T, K). Note that the first instance $(S_0, T_0, \{\kappa\})$ is simple. For each of the remaining instances $(S_i, T_i, K \cap \mathcal{L}(S_i))$, we construct recursively in $O(\sum_{u \in S_i} |\Gamma(u)| + \sum_{v \in T_i} |\Gamma(v)|)$ time a simple reduction $\mathcal{G}_i$ for $(S_i, T_i, K \cap \mathcal{L}(S_i))$ where $size(\mathcal{G}_i) \leq 2(|S_i| + |T_i|)$. By Fact 2, we conclude that $\mathcal{G} = \{(S_0, T_0, \{\kappa\})\} \cup \bigcup_{1 \leq i \leq m-1} \mathcal{G}_i$ is a simple reduction for (S, T, K). The total time taken is $O\left(|\Gamma(r_S)| + |\Gamma(r_T)| + \sum_{1 \leq i \leq m-1}(\sum_{u \in S_i} |\Gamma(u)| + \sum_{v \in T_i} |\Gamma(v)|)\right)$, which is $O(\sum_{u \in S} |\Gamma(u)| + \sum_{v \in T} |\Gamma(v)|)$. Furthermore, note that

$$\begin{aligned}\texttt{size}(\mathcal{G}) &= (|S_0| + |T_0|) + \texttt{size}(\mathcal{G}_1) + \cdots + \texttt{size}(\mathcal{G}_{m-1}) \\ &\leq (|S_0| + |T_0|) + 2(|S_1| + |T_1|) + \cdots + 2(|S_{m-1} + T_{m-1}|) \leq 2(|S| + |T|).\end{aligned}$$

Case 2: κ is not a shallow leaf. By examining the list $\Gamma(u)$ for each node u on the path σ_S from the root of S to κ, we find the κ-decomposition $(I_1, I_2, \ldots, I_{2m})$ of S in $O(\sum_{u \in \sigma_S} |\Gamma(u)|)$ time. Similarly, we can find $(J_1, J_2, \ldots, J_{2m})$, the κ-decomposition of T in $O(\sum_{v \in \sigma_T} |\Gamma(v)|)$ time where σ_T is the path from the root of T to κ. By Theorem 9, the set $\{(S_1, T_1, K_1), (S_2, T_2, K_2), \ldots, (S_{2m}, T_{2m}, K_{2m})\}$ where $S_i = S\|_{\mathcal{L}_\kappa(I_i) \cup \{\kappa\}}$, $T_i = \|_{\mathcal{L}_\kappa(I_i) \cup \{\kappa\}}$ and $K_i = K \cap (\mathcal{L}_\kappa(I_i) \cup \{\kappa\})$ is a reduction for (S, T, K). Recall that when i is odd, $\mathcal{L}_\kappa(I_i)$ does not contain any leaf in K; hence, $|K_i| = |K \cap (\mathcal{L}_\kappa(I_i) \cup \{\kappa\})| = 1$ and (S_i, T_i, K_i) is simple. When i is even, recall that $I_i = \{u_i\}$ and $J_i = \{v_i\}$ are singletons and u_i and v_i are respectively the root of S_i and T_i. Furthermore, κ is a shallow leaf of S_i and T_i. Let $A(u_i)$ be the set of nodes in the subtrees rooted at some children of u_i that is not on σ_S. Define $A(v_i)$ similarly. Below, we show that using $O(|\Gamma(u_i)| + |\Gamma(v_i)| + \sum_{w \in A(u_i)} |\Gamma(w)| + \sum_{w \in A(v_i)} |\Gamma(w)|)$ time, we can find a simple reduction $\mathcal{G}_i$ for (S_i, T_i, K_i) with $\texttt{size}(\mathcal{G}_i) \leq 2(|\{u_i, v_i\}| + |A(u_i)| + |A(v_i)|\})$. Then, by Fact 2, we have a simple reduction $\mathcal{G} = \bigcup_{i \text{ odd}} \{(S_i, T_i, K_i)\} \cup \bigcup_{i \text{ even}} \mathcal{G}_i$ where

$$\begin{aligned}\texttt{size}(\mathcal{G}) &= \sum_{i \text{ odd}} (|S_i| + |T_i|) + \sum_{i \text{ even}} 2(|\{u_i, v_i\}| + |A(u_i)| + |A(v_i)|) \\ &\leq 2(|S| + |T|).\end{aligned}$$

Note that the total time taken is $O(\sum_{u \in \sigma_S} |\Gamma(u)| + \sum_{i \text{ even}} (\sum_{w \in A(u_i)} |\Gamma(w)|)) + O(\sum_{v \in \sigma_T} |\Gamma(v)| + \sum_{i \text{ even}} \sum_{w \in A(v_i)} |\Gamma(w)|)) = O(\sum_{u \in S} |\Gamma(u)| + \sum_{v \in T} |\Gamma(v)|)$. The theorem follows.

Suppose that i is even. Let u_i and v_i be the root of S_i and T_i, respectively. Since κ is a shallow leaf of S_i and T_i, as in Case 1, we construct a reduction $\{(S_{i0}, T_{i0}, \{\kappa\}), (S_{i1}, T_{i1}, K_{i1}), \ldots, (S_{i\ell}, T_{i\ell}, K_{i\ell})\}$ for (S_i, T_i, K_i) in time

$O(|\Gamma(u_i)|+|\Gamma(v_i)|)$. For each $1 \leq j \leq \ell$, we find recursively in $O(\sum_{u\in S_{ij}} |\Gamma(u)|+ \sum_{v\in T_{ij}} |\Gamma(u)|)$ time a simple reduction $\mathcal{G}_{ij}$ for (S_{ij}, T_{ij}, K_{ij}) where $\texttt{size}(\mathcal{G}_{ij}) \leq 2(|S_{ij}| + |T_{ij}|)$. Hence, we construct a simple reduction $\mathcal{G}_i = \{(S_{i0}, T_{i0}, \{\kappa\})\} \cup \bigcup_{1\leq j\leq \ell} \mathcal{G}_{ij}$ for (S_i, T_i, K_i) using total time

$$O(\sum_{1\leq j\leq \ell}\sum_{u\in S_{ij}} |\Gamma(u)| + \sum_{1\leq j\leq \ell}\sum_{v\in T_{ij}} |\Gamma(u)|)) = O(\sum_{w\in A(u_i)} |\Gamma(w)| + \sum_{w\in A(v_i)} |\Gamma(w)|)$$

total time. To estimate $\texttt{size}(\mathcal{G}_i)$, let U_{i0} and V_{i0} be the sets of nodes in S_{i0} and T_{i0}, respectively. Observe that $U_{i0} - \{u_i, \kappa\} \subseteq A(u_i)$ and it does not contain any node in $S_{i1}, S_{i2}, \ldots, S_{i\ell}$. Furthermore, $|U_{i0} - \{u_i\}| \leq 2|U_{i0} - \{u_i, \kappa\}|$. We have similar properties for V_{i0}. Then, $\texttt{size}(\mathcal{G}_i) = |S_{i0}| + |T_{i0}| + 2\sum_{1\leq j\leq \ell}(|S_{ij}| + |T_{ij}|)$ $= |\{u_i, v_i\}| + |U_{i0} - \{u_i\}| + |V_{i0} - \{v_i\}| + 2\sum_{1\leq j\leq \ell}(|S_{ij}| + |T_{ij}|)$, which is no more than $|\{u_i, v_i\}| + 2(|U_{i0} - \{u_i, \kappa\}| + |V_{i0} - \{v_i, \kappa\}|) + 2\sum_{1\leq j\leq \ell}(|S_{ij}| + |T_{ij}|)$ $\leq |\{u_i, v_i\}| + 2(|A(u_i)| + |A(v_i)|)$.

□

6 The MCAST problem with Forest Constraint

In this section, we explain how our method can be used to handle the MCAST problem with forest constraint, which is defined formally as follows:

> Given two evolutionary trees S and T, and a forest $\mathcal{F}$ of evolutionary trees, find a largest agreement subtree M of S and T that contains every tree in $\mathcal{F}$.

We say that M is a *maximum constrained agreement subtree of S and T with respect to the forest $\mathcal{F}$*. Note that $\mathcal{F}$ may not be uniquely labeled; a same label may appear in different trees in $\mathcal{F}$. Furthermore, it is easy to verify that the necessary conditions for the existence of M are

(1) for every tree $P \in \mathcal{F}$, P is a subtree of both S and T, and

(2) $S\|_{K_\mathcal{F}} = T\|_{K_\mathcal{F}}$ where $K_\mathcal{F}$ is the set of leaf labels of the trees in $\mathcal{F}$.

(We note that the case where these conditions are not met also gives rise to several interesting optimization problems that deserve further study.) We say that the forest $\mathcal{F}$ is *consistent* with S and T if the above two conditions are satisfied. Note that by using some efficient tree isomorphism algorithms (e.g., [7, 18]), we can check the consistency of $\mathcal{F}$ in linear time. The following

theorem suggests that our reduction method can be used to solve the MCAST problem with forest constraints directly.

Theorem 12. *Suppose that the forest $\mathcal{F}$ is consistent with the trees S and T. Let $K_{\mathcal{F}}$ be the set of leaf labels of the trees in $\mathcal{F}$, and $L \in \texttt{MCAST}(S, T, K_{\mathcal{F}})$ be a maximum constrained agreement leaf subset of S and T with respect to $K_{\mathcal{F}}$. Then, $M = S\|_L$ is a maximum constrained agreement subtree of S and T with respect to $\mathcal{F}$.*

Proof. By definition, $M = S\|_L = T\|_L$ is a largest agreement subtree that contains $K_{\mathcal{F}}$. To show that it is a maximum constrained agreement subtree of S and T with respect to $\mathcal{F}$, it suffices to prove that for any tree $P \in \mathcal{F}$, P is a subtree of M. Let L_P be the set of leaf labels in P. Since $\mathcal{F}$ is consistent with S and T, P is a subtree of S and thus $S\|_{L_P} = P$. Together with Fact 1, we conclude that $M\|_{L_P} = (S\|_L)\|_{L_P} = S\|_{L_P} = P$, and hence P is a subtree of M. The theorem follows. □

Acknowledgement. We thank the anonymous referees for their comments on the paper and their suggestion to study the case where the constraint is a forest.

References

[1] K. Amenta and F. Clarke. A linear-time majority tree algorithm. In *Proceedings of the 3rd International Workshop on Algorithms in Bioinformatics*, pages 216–227, 2003.

[2] T.Y. Berger-Wolf. Online consensus and agreement of phylogenetic trees. In *Proceedings of the 4th International Workshop on Algorithms in Bioinformatics*, pages 350–361, 2004.

[3] R. Cole, M. Farach, R. Hariharan, T. Przytycka, and M. Thorup. An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees. *SIAM Journal on Computing*, 30(5):1385–1404, 2000.

[4] S. Dong and E. Kraemer. Calculation, visualization and manipulation of MASTs (maximum agreement subtrees). In *Proceedings of the IEEE Computational Systems Bioinformatics Conference*, pages 1–10, 2004.

[5] M. Farach and M. Thorup. Optimal evolutionary tree comparison by sparse dynamic programming. In *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science*, pages 770–779, 1994.

[6] M. Farach and M. Thorup. Fast comparison of evolutionary trees. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 481–488, 1995.

[7] D. Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, 21:19–28, 1991.

[8] G.H. Hardy, J.E. Littlewood, and G. Pólya. *Inequalities*. Cambridge, 1952.

[9] M.Y. Kao. Tree contractions and evolutionary trees. *SIAM Journal on Computing*, 27:1592–1616, 1998.

[10] M.Y. Kao, T.W. Lam, W.K. Sung, and H.F. Ting. A decomposition theorem for maximum weight bipartite matchings with applications in evolution trees. In *Proceedings of the 7th Annual European Symposium on Algorithms*, pages 438–449, 1999.

[11] M.Y. Kao, T.W. Lam, W.K. Sung, and H.F. Ting. An even faster and more unifying algorithm comparing trees via unbalanced bipartite matchings. *Journal of Algorithms*, 20(2):212–233, 2001.

[12] D. Keselman and A. Amir. Maximum agreement subtree in a set of evolutionary trees– metrics and efficient algorithms. In *Proceedings of 35th Annual Symposium on the Foundations of Computer Sciences*, pages 758–769, 1994.

[13] E. Kubicka, G. Kubicki, and F. McMorris. An algorithm to find agreement subtrees. *Journal of Classification*, 12:91–99, 1995.

[14] A. Messmark, J. Jansson, A. Lingas, and E. Lundell. Polynomial-time algorithms for the ordered maximum agreement subtree problem. In *Proceedings of the 15th Annual Symposium on Combinatorial Pattern Matching*, pages 220–229, 2004.

[15] Z.S. Peng and H.F. Ting. An $O(n \log n)$-time algorithm for the maximum constrained agreement subtree problem for binary trees. In *Proceedings of the 15th Symposium on Algorithms and Computations*, pages 754–765, 2004.

[16] T. Przytycka. Sparse dynamic programming for maximum agreement subtree problem. In *Mathematical Hierarchies and Biology*, pages 249–264. DIMACS series in Discrete Mathematics and Theoretical Computer Science, 1997.

[17] M. Steel and T. Warnow. Kaikoura tree theorems: computing the maximum agreement subtree. *Information Processing Letters*, 48(2):77–82, 1994.

[18] T.J. Warnow. Tree compatibility and inferring evolutionary history. *Journal of Algorithms*, 16(3):388–407, 1994.