Linear time 3-approximation for MAST problem

Vincent Berry, Sylvain Guillemot, François Nicolas, Christophe Paul

Linear time 3-approximation for the MAST problem

Vincent BERRY
LIRMM, Université Montpellier II, France
and
Sylvain GUILLEMOT
LIRMM, Université Montpellier II, France
and
François NICOLAS
LIRMM, Université Montpellier II, France
and
Christophe PAUL
CNRS - LIRMM, Université Montpellier II, France

Given a set of leaf-labeled trees with identical leaf sets, the well-known Maximum Agreement Subtree (MAST) problem consists in finding a subtree homeomorphically included in all input trees and with the largest number of leaves. MAST and its variant called Maximum Compatible Tree (MCT) are of particular interest in computational biology. This paper presents a linear-time approximation algorithm to solve the complement version of MAST, namely identifying the smallest set of leaves to remove from input trees to obtain isomorphic trees. We also present an $O(n^2 + kn)$ algorithm to solve the complement version of MCT. For both problems, we thus achieve significantly lower running times than previously known algorithms. Fast running times are especially important in phylogenetics where large collections of trees are routinely produced by resampling procedures, such as the non parametric bootstrap or Bayesian MCMC methods.

Categories and Subject Descriptors: ... [**...**]: ...

General Terms: ...
Additional Key Words and Phrases: Approximation algorithm, Phylogenetic tree, Maximum Agreement Subtree, Maximum Compatible Subtree

---

1. INTRODUCTION

The evolutionary history of species is usually displayed by a tree whose leaves are labeled by current species and whose internal nodes represent hypothetical ancestors. Such trees, also called *phylogenies*, are inferred from primary data consisting of molecular sequences or pairwise distances between species, e.g. obtained from gene order data. Several inference methods are available to infer a phylogeny from a data set. Most often, these methods lead to different phylogenies. Moreover, several data sets are usually analyzed, still increasing the number of candidate phylogenies to represent the evolutionary history of the species under study. As these phylogenies usually conflict on the position of some leaves (or species) or group of leaves it is common practice to obtain a congruent view of the topological signal they encode by resorting to a tree consensus method. When only reliable parts of the input phylogenies are conserved (e.g. identified by a high support value obtained through a sampling process such as bootstrap or MCMC), consensus methods that produce a tree not contradicting the input phylogenies are preferred. The strict consensus [McMorris et al. 1983] is the most widespread consensus method satisfying this property. It has the desirable feature to be computable in linear time [Berger-Wolf 2004], however, it tends to provide trees with a poor level of information as soon as all topological conflicts are not restricted to local parts of the phylogenies. In such cases, more involved consensus methods are used, among which is the well-known Maximum Agreement Subtree (MAST) consensus. Given a set of leaf-labeled trees with identical leaf sets, the MAXIMUM AGREEMENT SUBTREE problem (MAST) consists in finding a subtree homeomorphically included in all input trees and with the largest number of leaves [Steel and Warnow 1993; Farach et al. 1995; Amir and Keselman 1997; Gupta and Nishimura 1998; Kao et al. 2001; Cole et al. 2001]. In other words, this involves selecting a largest set of input leaves such that the input trees are isomorphic, i.e. *agree* with each other, when restricted to these leaves. Note that MAST is used to reach other practical goals in phylogenetics, such as identifying horizontal gene transfers, and has numerous other applications, such as comparing XML trees, or images.

The MAXIMUM COMPATIBLE TREE problem (MCT) is a variant of MAST that is of particular interest in phylogenetics when the input trees are not binary [Hamel and Steel 1996; Hein et al. 1996; Ganapathysaravanabavan and Warnow 2001; Ganapathy and Warnow 2002]. By requiring that induced subtrees of the input trees are *compatible*, and not strictly isomorphic, MCT usually leads to selecting a larger set of leaves than allowed by MAST. Note that another variant of MAST has been recently proposed to build phylogenetic supertrees, where input trees have different leaf sets [Berry and Nicolas 2004].

Unfortunately, the MAST problem is NP-hard on three rooted trees of unbounded degree [Amir and Keselman 1997], and MCT on two rooted trees if one of them is of unbounded degree [Hein et al. 1996]. Subquadratic algorithms have been proposed for MAST on two rooted n-leaf trees [Kao et al. 1999; Cole et al. 2001; Kao et al. 2001]. When dealing with k rooted trees, MAST can be solved in $O(n^d + kn^3)$ time provided that the degree of one of the input trees is bounded by d [Farach et al. 1995; Amir and Keselman 1997; Bryant 1997]. In comparison, MCT can be solved in $O(2^{2kd}n^k)$ time provided that all input trees have degree bounded

by d [Ganapathysaravanabavan and Warnow 2001]. However, when the unreliable parts of the input phylogenies have been filtered prior to the consensus inference, input phylogenies usually contain some high-degree nodes. A possibility is then to turn to algorithms that are FPT for some parameter with a small value in practice. Following [Downey et al. 1999], Berry and Nicolas [Berry and Nicolas pear] proposed an $O(\min\{3^p kn, 2.27^p + kn^3\})$ time algorithm, where the parameter p is the smallest number of leaves to be removed from the input set of leaves so that the input trees agree.

An alternative to exact algorithms, in such cases, lies in fast approximation algorithms. Several works (starting from [Amir and Keselman 1997]) proposed 3-approximation algorithms for CMAST and CMCT, where CMAST, resp. CMCT, is the complement version of MAST, resp. MCT, i.e. aims at selecting the smallest number of leaves to be removed from the input trees in order to obtain their agreement. In practice, input trees usually agree on the position of the majority of leaves, thus approximating CMAST and CMCT is more relevant than approximating MAST and MCT.

For CMAST, we achieve in this paper $O(kn)$, i.e. linear time, significantly improving on the former $O(kn^4)$ time algorithm of [Amir and Keselman 1997], refined into an $O(kn^3)$ algorithm in [Berry and Nicolas 2004]. The improvement in the complexity results from ordering the subtrees of two compared trees such that conflicting triples of leaves are readily identified from the minimum and maximum leaves contained in a subtree. For the CMCT problem, [Ganapathy and Warnow 2002] propose an $O(k^2n^2)$ time 3-approximation algorithm. We propose here an $O(n^2 + kn)$ time algorithm.

2. DEFINITIONS AND PRELIMINARIES

A rooted *evolutionary tree* is a tree whose leaf set $L(T)$ is in bijection with a label set, and whose internal nodes have at least two children. Hereafter, we only consider such trees and, without loss of generality, labels are assumed to be numbers from 1 to n. We also identify leaf nodes with their labels. The *size* of a tree T (denoted $|T|$) is the number of its leaves: $|T| = |L(T)|$.

If u is a node of a tree T, $S(u)$ stands for the subtree rooted at u, $L(u)$ for the leaves of this subtree, and $d^+(u)$ for the number of children of u. For a set of leaves $L \subseteq L(T)$, $lca_T(L)$ denotes the lowest common ancestor of leaves L in T. Given a set L of labels and a tree T, the *restriction* of T to L, denoted $T|L$, is the tree homeomorphic to the smallest subtree of T connecting leaves of L. Given a collection $\mathcal{T}$ of trees and a leaf-set L, the collection $\mathcal{T}|L$ is $\{T_i|L \text{ s.t. } T_i \in \mathcal{T}\}$.

Given a collection $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ of k trees on a common leaf set L of cardinality n, an *agreement subtree* of $\mathcal{T}$ is any tree T with leaves in L s.t. $\forall T_i \in \mathcal{T}$, $T = T_i|L(T)$ (see Figure 1). The MAST problem consists in finding an agreement subtree of $\mathcal{T}$ with the largest number of leaves.

The following result states that isomorphism is invariant under leaf removal. It will be used by the approximation algorithm of Section 4 for MAST which proceeds by removing progressively leaves from input trees.

LEMMA 2.1. *Let T_1 and T_2 be two isomorphic trees with leaf set L. If $L' \subseteq L$, then $T_1|L'$ is isomorphic to $T_2|L'$.*

A tree T *refines* a tree T', if T' can be obtained by collapsing some internal edges of T, (i.e. merging their extremities). More generally, a tree T refines a collection $\mathcal{T}$, whenever T refines all T_i's in $\mathcal{T}$. Given a collection $\mathcal{T}$ of k trees with identical leaf set L of cardinality n, a tree T with leaves in L is *compatible with* $\mathcal{T}$ if and only if T refines $\mathcal{T}|L(T)$. If there is a tree T compatible with $\mathcal{T}$ s.t. $L(T) = L$, i.e. that is a common refinement of all trees in $\mathcal{T}$, then the collection $\mathcal{T}$ is *compatible*. In this case, a *minimum refinement* T of $\mathcal{T}$ is a tree such that any tree T' refining $\mathcal{T}$ also refines T (see Figure 1). Note that input collections considered in phylogenetics are usually not compatible. The MCT problem aims at finding a tree T compatible with $\mathcal{T}|L(T)$ and having the largest number of leaves. Note that MCT is equivalent to MAST when input trees are binary. However, when input trees have some nodes with outdegree larger than 2, solving the MCT problem usually enables one to keep more leaves in the output tree than MAST does, as the refinement relation is weaker than the isomorphism relation.
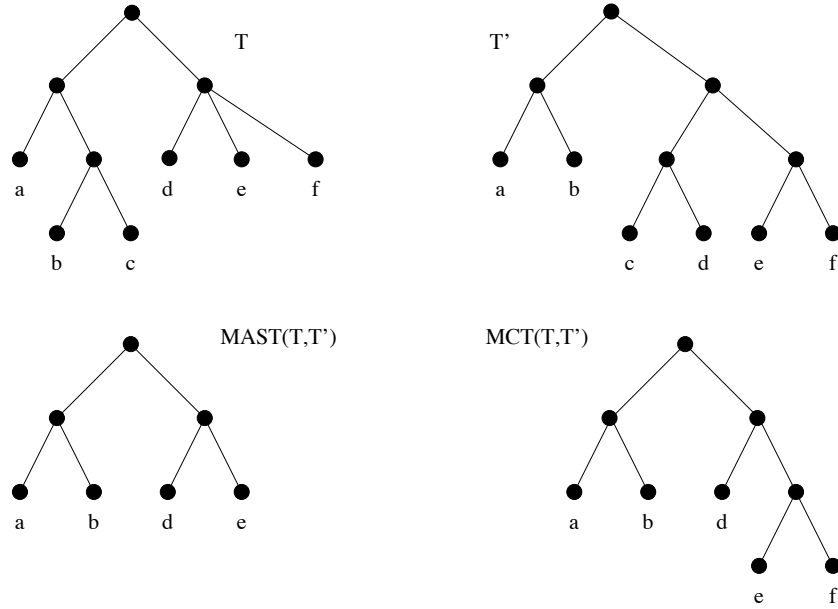
Fig. 1. In the tree T, $a|bc$ is a rooted triple whereas $\{d, e, f\}$ is a fan. As $ab|c$ is a rooted triple of the tree T', the leaves a, b, c form a hard conflict. On the other hand the leaves d, e, f form a soft conflict between T and T'. Removing leaf c kills all the hard conflicts between T and T' and yields a maximum compatible tree (Contracting in the resulting tree the edge between the parent of d and the parent of e gives a subtree of T). To eliminate the soft conflict on $\{d, e, f\}$ one of its three leaves has to be removed, say f. So the subtree on a, b, d, e defines a maximum agreement subtree.

Rooted trees of arbitrary size can be defined in terms of rooted subtrees on three leaves, see [Semple and Steel 2003] for numerous results on this aspect. For any tree on three leaves a, b, c, there are only three possible binary shapes, denoted $a|bc$, resp. $b|ac$, resp. $c|ab$, depending on their innermost grouping of two leaves (bc, resp. ac, resp ab). These trees are called *rooted triples*. Alternatively the tree can

be a *fan*, also called a *star-tree* on 3 leaves, i.e. composed of a unique internal node connected to the three leaves. A fan is denoted $\{a, b, c\}$. Let T be a tree on more than three leaves, for any subset $\{a, b, c\}$ of leaves in T, $T|\{a, b, c\}$ is either a rooted triple or a fan. We define $rt(T)$, resp. $f(T)$, as the set of rooted triples, resp. fans, induced by the 3-leaf subsets of leaves in T. Given a collection $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ of trees with leaf set L, a set $\{a, b, c\} \subseteq L$ is a *hard conflict* between $\mathcal{T}$ whenever $\exists T_i, T_j \in \mathcal{T}$ s.t. $a|bc \in rt(T_i)$ and $b|ac \in rt(T_j)$. The set $\{a, b, c\}$ is a *soft conflict* between (trees of) $\mathcal{T}$ whenever $a|bc \in rt(T_i)$ and $\{a, b, c\} \in f(T_j)$ (see Figure 1).

The topological disagreement between trees of arbitrary size translates into disagreement on three-leaf sets:

> LEMMA 2.2 [AMIR AND KESELMAN 1997; GANAPATHYSARAVANABAVAN AND WARNOW 2001; BERRY AND
> *Two trees on a same leaf set are isomorphic iff there is no hard nor soft conflict between them, i.e. the two trees have identical sets of rooted triples and identical sets of fans. More generally, all trees of a collection $\mathcal{T}$ are isomorphic if and only if there is no hard nor soft conflict between $\mathcal{T}$.*

> (ii) *A collection $\mathcal{T}$ of trees with the same leaf set is compatible if and only if no two trees in $\mathcal{C}$ have a hard conflict between them.*

Given a set of conflicts $\mathcal{C}$, let $L(\mathcal{C})$ denote the leaves appearing in $\mathcal{C}$. Given a collection $\mathcal{T}$ with conflicts, define an *hs-peacemaker* to be any set $\mathcal{C}$ of disjoint hard and soft conflicts between $\mathcal{T}$ s.t. $\mathcal{T}|(L - L(\mathcal{C}))$ is a collection of isomorphic trees. Similarly, define an *h-peacemaker* of $\mathcal{T}$ to be any set $\mathcal{C}$ of disjoint hard conflicts between $\mathcal{T}$ s.t. $\mathcal{T}|(L - L(\mathcal{C}))$ is a collection of compatible trees. In other words, removing $L(\mathcal{C})$ from the input trees removes all conflicts between them, according to compatibility, respectively isomorphism.

A 3-approximation T for CMAST is an agreement subtree of $\mathcal{T}$, such that $|L(\mathcal{T}) - L(T)| \leq 3|L(\mathcal{T}) - L_{opt}|$, where L_{opt} is the set of leaves of a maximum agreement subtree of $\mathcal{T}$. Likewise, a 3-approximation for CMCT is a tree T compatible with $\mathcal{T}$ such that $|L(\mathcal{T}) - L(T)| \leq 3|L(\mathcal{T}) - L_{opt}|$, where L_{opt} is a largest set of leaves s.t. $\mathcal{T}|(L(\mathcal{T}) - L_{opt})$ is compatible. The hs-peacemakers, resp. h-peacemakers, play an important role in finding a 3-approximation for CMAST, resp. CMCT:

> LEMMA 2.3 [AMIR AND KESELMAN 1997; GANAPATHY AND WARNOW 2002; BERRY AND NICOLAS 2004].
> *Let $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ be a collection of trees on a leaf set L*

> (i) *If $\mathcal{C}$ is an hs-peacemaker of $\mathcal{T}$, then any tree in $\mathcal{T}|(L - L(\mathcal{C}))$ is a 3-approximation for CMAST.*
> (ii) *If $\mathcal{C}$ is an h-peacemaker of $\mathcal{T}$, then any tree refining $\mathcal{T}|(L - L(\mathcal{C}))$ is a 3-approximation for CMCT.*

3. AN $O(N^2 + KN)$ TIME 3-APPROXIMATION ALGORITHM FOR CMCT

Let $\mathcal{T}$ be a collection of k trees whose leaves are mapped to a common set L of n labels. Given a node u in a tree, the set $L(u)$ of leaves in the subtree rooted at u is called a *cluster*. The set of clusters of a tree T is denoted $\mathcal{C}l(T)$. The following observation states the relationship between clusters and rooted triples.

Remark 3.1. Let ℓ, ℓ', ℓ'' be three distinct leaves of a rooted tree T. Then, $\ell|\ell'\ell'' \in rt(T)$ iff there is a cluster $L(v) \in \mathcal{C}l(T)$ s.t. $\ell \notin L(v)$ and $\{\ell', \ell''\} \subseteq L(v)$.

LEMMA 3.2. $\mathcal{T}$ *is a compatible collection if and only if there is a minimum refinement T of $\mathcal{T}$. Moreover $rt(T) = \bigcup_{T_i \in \mathcal{T}} rt(T_i)$.*

PROOF. $\Leftarrow$ Directly follows from definition of a compatible collection.

$\Rightarrow$ Assume $\mathcal{T}$ is compatible and let T be a *minimal* refinement of $\mathcal{T}$ (i.e. for any refinement T' of $\mathcal{T}$, T is not a refinement of T'). By the definition of a refinement, $\mathcal{Cl}(T_i) \subseteq \mathcal{Cl}(T)$ for any $T_i \in \mathcal{T}$. Also, by Remark 3.1, we have $rt(T_i) \subseteq rt(T)$ for all $T_i \in \mathcal{T}$ and thus

$$\bigcup_{T_i \in \mathcal{T}} rt(T_i) \subseteq rt(T) \tag{1}$$

Assume the inclusion is proper. Then there is a triple $\ell|\ell'\ell'' \in rt(T)$ such that for any $T_i \in \mathcal{T}$, $\ell|\ell'\ell'' \notin rt(T_i)$. Equivalently, by Remark 3.1, there is an internal node v of T s.t. for all $T_i \in \mathcal{T}$, $L(v) \notin \mathcal{Cl}(T_i)$. Since $L(T)$ is a cluster of all T_i's (they all have the same set of leaves as T), v is not the root of T. Let T' be the tree obtained from T by collapsing the edge between v and its parent. Thereby, T refines T' and for all $T_i \in \mathcal{T}$, $\mathcal{Cl}(T_i) \subseteq \mathcal{Cl}(T) \setminus \{L(v)\} = \mathcal{Cl}(T')$, i.e. T' is a refinement of every tree in $\mathcal{T}$. This is in contradiction with the minimality of T. So we prove that, unless (1) is an equality, the tree T is not a minimal refinement of $\mathcal{T}$. Moreover the equality shows the unicity of a minimal refinement: i.e. T is a minimum refinement.

$\square$

Note that if $\mathcal{T}$ is compatible, a minimum refinement of $\mathcal{T}$ is a solution to MCT as $L(T) = L$. Such a minimum refinement can be obtained by first computing a minimum refinement of two trees $T_1, T_2 \in \mathcal{T}$, and then repeating this operation between the current refinement and each other input tree taken successively ($k - 2$ additional runs). If $\mathcal{T}$ is not compatible, then Lemma 2.3 can be applied. Given two trees, [Berry and Nicolas pear] give an $O(n)$ time algorithm called FIND-REFINEMENT-OR-CONFLICT that either returns a minimum refinement of the trees (when they are compatible), or otherwise identifies a hard conflict C between the trees. Thus, from Lemma 2.3-*(ii)*, the procedure sketched above can be adapted to obtain a 3-approximation of CMCT for a collection $\mathcal{T}$, whether $\mathcal{T}$ is compatible or not. Apply FIND-REFINEMENT-OR-CONFLICT to a pair $\{T_1, T_2\} \subseteq \mathcal{T}$ to obtain either a minimum refinement T or a hard conflict C. In the latter case, remove C from all input trees and iterate with the same trees. In the former case, iterate with T and another input tree from $\mathcal{T}$. When $\mathcal{T}$ has been entirely processed, $O(k + n)$ calls to FIND-REFINEMENT-OR-CONFLICT have been issued, and the set $\mathcal{C}$ of all removed conflicts is an h-peacemaker of $\mathcal{T}$. Hence:

THEOREM 3.3. *The CMCT problem for a collection of k rooted trees on an n-leaf set can be 3-approximated in $O(n^2 + kn)$ time.*

PROOF. The time complexity is obvious from the above discussion. Let us prove the correctness of the described algorithm. Assume a set of disjoint conflicts has been removed so that $\{T_1, T_2\} \subseteq \mathcal{T}$ is a pair of compatible trees and let T be their minimum refinement and $\mathcal{T}'$ be the collection of trees $(\mathcal{T} \setminus \{T_1, T_2\}) \cup \{T\}$. From

Lemma 3.2, $rt(T) = rt(T_1) \cup rt(T_2)$, which implies that the trees of the collection $\mathcal{T}$ induce as a whole the same set of rooted triples as those of $\mathcal{T}'$. In other words any conflict of $\mathcal{T}'$ is a conflict of $\mathcal{T}$ and vice versa. Thereby a solution to the CMCT problem on $\mathcal{T}'$ is a solution to CMCT on $\mathcal{T}$. As the conflicts removed all along the process are disjoint, this procedure actually computes an h-peacemaker. By Lemma 2.3-*(ii)*, it provides a 3-approximation for CMCT. $\square$

4.   A LINEAR TIME 3-APPROXIMATION ALGORITHM FOR CMAST

First consider collections $\mathcal{T} = \{T_1, T_2\}$ of two trees on a leaf set L. As for MCT, Lemma 2.3 guides the computation of a 3-approximation of CMAST.

   Though the definition of a maximum agreement subtree is independent of any order on the children of nodes in the input trees, considering the input trees as ordered enables us to efficiently compute an approximation of CMAST. Fixing an arbitrary order on the children of every node u of T_2 uniquely defines a left-right order π_2 on the leaves L.

   Definition 4.1. A subset S of L is an *interval* of π_2 whenever the elements of S occur consecutively in π_2.

   Note that the cluster $L(v)$ corresponding to a node v of T_2 is an interval of π_2. Remark that T_1 is isomorphic to T_2 if and only if each node u of T_1 defines an interval $L(u)$ in π_2 and the intervals defined by the nodes of T_1 are the same as the intervals defined by that of T_2. Thus, considering the order π_2 on the leaves L induced by T_2, we can compute an agreement subtree of T_1 and T_2 in two steps:

(1) identify a set $\mathcal{C}$ of disjoint conflicts s.t. removing $L(\mathcal{C})$ from the trees ensures that nodes u of $T_1|L - L(\mathcal{C})$ define intervals $L(u)$ in $\pi_2|L - L(\mathcal{C})$;
(2) identify an additional set $\mathcal{C}$ of disjoint conflicts s.t. the nodes of $T_1|L - L(\mathcal{C})$ define the same set of intervals in $\pi_2|L - L(\mathcal{C})$ as the nodes of $T_2|L - L(\mathcal{C})$.

Together, these two steps identify an *hs*-peacemaker of $\{T_1, T_2\}$, i.e. a set of leaves whose removal leads to obtain an agreement subtree of $\{T_1, T_2\}$. Given a collection $\mathcal{T}$ of $k > 2$ trees, applying these two steps to the first two input trees then to the obtained agreement subtree and the other input trees successively, enables us to obtain an agreement subtree for $\mathcal{T}$ that 3-approximates CMAST (as the conflicts removed during the process form an *hs*-peacemaker of $\mathcal{T}$). The approximation result is stated in Section 4.3.

4.1   Ensuring that nodes of T_1 define intervals in π_2

The aim of the first step is to remove a set $\mathcal{C}$ of disjoint conflicts so that for any node u of T_1, $L(u) - L(\mathcal{C})$ is an interval of $\pi_2|(L - L(\mathcal{C}))$. Before describing the algorithm, let us introduce some definitions and properties.

   LEMMA 4.2. *Let T_1, T_2 be trees on a leaf set $L \subseteq \{1, \ldots, n\}$ and let $\{a, b, c\} \subseteq L$. If both $a <_{\pi_2} b <_{\pi_2} c$ and $ac|b \in rt(T_1)$, then $\{a, b, c\}$ is a (hard or soft) conflict between T_1 and T_2.*

   PROOF. Note that $a <_{\pi_2} b <_{\pi_2} c$ only allows $ab|c \in rt(T_2)$, $cb|a \in rt(T_2)$ or $\{a, b, c\} \in f(T_2)$. In all cases, this conflicts with $ac|b \in rt(T_1)$. $\square$

For any node u of T_1 or T_2, let $m_{\pi_2}(u)$ and $M_{\pi_2}(u)$ be respectively the smallest and largest leaf of $L(u)$ according to π_2. (In the example of Figure 2, $m_\pi(u) = 1$ and $M_\pi(u) = 27$). If an element $x \in L - L(u)$ is s.t. $m(u) <_{\pi_2} x <_{\pi_2} M(u)$, then $prev_{\pi_2}(x, u)$ stands for the maximum element of $L(u)$ w.r.t. π_2 that is smaller than x. Similarly, $next_{\pi_2}(x, u)$, stands for the minimum element of $L(u)$ w.r.t. π_2 larger than x. (In the example of Figure 2, $prev_\pi(4, u) = 3$ and $next_\pi(4, u) = 20$).

COROLLARY 4.3. *For a node u of T_1 and a leaf $x \in L - L(u)$ s.t. $m_{\pi_2}(u) <_{\pi_2} x <_{\pi_2} M_{\pi_2}(u)$, the set $\{prev_{\pi_2}(x, u), x, next_{\pi_2}(x, u)\}$ is a conflict between T_1 and T_2.*

PROOF. Since, $x \notin L(u)$ unlike $prev_{\pi_2}(x, u)$ and $next_{\pi_2}(x, u)$ we have

$$x \big| prev_{\pi_2}(x, u) next_{\pi_2}(x, u) \in rt(T_1)$$

Thus Lemma 4.2, applies on the set $\{prev_{\pi_2}(x, u), x, next_{\pi_2}(x, u)\}$. □

Algorithm 1 details in pseudo-code how a set of disjoint conflicts $\mathcal{C}$ is identified using this result. For the sake of simplicity the parameters of the algorithm are stated to be a tree T (instead of T_1) and an order π (instead of π_2). Algorithm 1 searches the tree T in post-order, so that the children of a node u are already known to be intervals of $\pi|L - L(\mathcal{C})$ when u is processed. A list $\mathcal{I}$ of disjoint intervals of L is maintained sorted according to π. Each interval I in $\mathcal{I}$ corresponds to a processed node whose parent has not yet been processed. Given an interval $I \in \mathcal{I}$, the corresponding node u in T is identified by a pointer $node(I)$. Conversely, a pointer $I(u)$ indicates the interval in $\mathcal{I}$ associated with some processed node u. $\mathcal{I}$ is initially composed of unit intervals ($\{1\}, \ldots, \{n\}$) corresponding to leaves of T. The function $precInt_{\mathcal{I}}(I)$ (resp. $nextInt_{\mathcal{I}}(I)$) used in the pseudo-code returns the interval in $\mathcal{I}$ preceding (resp. following) I. Processing a node u consists in ensuring that the intervals defined by its children are consecutive in $\pi_2|L - L(\mathcal{C})$, so that u itself corresponds to an interval. While processing u, if Algorithm 1 finds in $\mathcal{I}$ three intervals $J <_\pi I <_\pi J'$ such that $node(J)$ and $node(J')$ are children of u but not $node(I)$, then obviously $L(u)$ is not currently an interval. Then conflicts can be identified (according to Lemma 4.2 and Corollary 4.3). To handle these conflicts, the algorithm calls a specific routine named `RemoveConflict`. This routine iteratively deletes triples of leaves $a \in J$, $b \in I$ and $c \in J'$ (which are conflicts according to Corollary 4.3) until one of the three intervals becomes empty. Finally, when no more such intervals J, I, J' can be found, then u is an interval in $\pi|L - L(\mathcal{C})$ and the set of intervals of the children of u are merged into a single one, namely $I(u)$, which indicates the so far preserved leaves of u.

We now describe in more details the `RemoveConflict` procedure which returns a set of conflicts and is also allowed to update the intervals denoted by I_l and I_r. The two possible calls to this procedure, namely `RemoveConflict`$(I(c), I, I_l)$ and `RemoveConflict`$(I_r, I, I(c))$, are issued in the `while` loop of Algorithm 1 (respectively in line 4 and in line 5). Below we describe the behavior of the `RemoveConflict` procedure in the context of the first call (the second one being processed symmetrically). The procedure receives as input three intervals of $\mathcal{I}$ such that $I(c) <_\pi I <_\pi I_l$ and I is the interval preceding I_l in $\mathcal{I}$. As already mentioned, the procedure removes triples $a \in I(c), b \in I, c \in I_l$ until one of the three intervals becomes empty

---

Algorithm 1: $\text{MakeIntervals}(T, \pi)$

Input: A tree T with leaf set L and an arbitrary order π on its leaf labels

Result: A set $\mathcal{C}$ of disjoint conflicts s.t. each node of $T|(L-L(\mathcal{C}))$ is an interval in $\pi|(L - L(\mathcal{C}))$.

$\mathcal{C} \leftarrow \emptyset \; ; \mathcal{I} \leftarrow \emptyset$

1 for *each leaf-node u of T with label i* **do**
    $I \leftarrow \{i\}; node(I) \leftarrow u; I(u) \leftarrow I; \mathcal{I} \leftarrow \mathcal{I} \cup \{I\};$

2 for *each internal node u in a post-order traversal of T* **do**
    Let L_{ch} be a copy of the list of children of u
3 $c \leftarrow ExtractFirst(L_{ch})$
    $I_l \leftarrow I(c); I_r \leftarrow I(c)$
    while L_{ch} *is not empty* **do**
        $c \leftarrow \text{ExtractFirst}(L_{ch})$
        if $I(c) <_\pi I_l$ **then**
            $I \leftarrow precInt_{\mathcal{I}}(I_l)$
            while $I(c) \neq \emptyset$ *and* $I \neq I(c)$ **do**
                if $node(I)$ *is a child of u* **then**
                    Extract $node(I)$ from L_{ch}
                    $I_l \leftarrow I$
4   **else** $\mathcal{C} \leftarrow \mathcal{C} \cup \text{RemoveConflict}(I(c), I, I_l)$
                $I \leftarrow precInt_{\mathcal{I}}(I)$
            if $I(c) \neq \emptyset$ **then** $I_l \leftarrow I(c)$
5   **else**   Perform symetrically, examining intervals $I \in \mathcal{I}$ from $nextInt_{\mathcal{I}}(I_r)$ to $precInt_{\mathcal{I}}(I(c))$

6   **if** $I_l \neq \emptyset$ **then**
        Replace in $\mathcal{I}$ all elements from I_l to I_r by a new element $I_u = [m(I_l), M(I_r)]$
        $node(I_u) \leftarrow u \; ; I(u) \leftarrow I_u$

return $\mathcal{C}$

(note that several of them may be simultaneously empty). Whenever $I(c)$ or I becomes empty, it is readily removed from $\mathcal{I}$ in $O(1)$ time. The crux is when I_l becomes empty and the procedure has to update it, as also potentially I_r (recall it is allowed to do such updates). When I_l becomes empty, it is removed from $\mathcal{I}$, and I_l is set to the following interval in $\mathcal{I}$ in the case I_l was different from I_r. The most involved case is when $I_l = I_r$ and I_l is removed. In such a case, if u still has at least another child (not yet removed) then the procedure extracts a new child c of u from $L(c)$ and sets $I_l = I_r = I(c)$ (as in line 3). In the other subcase where u has no more child (ie, all leaves in $L(u)$ are now in conflicts of $\mathcal{C}$), then the procedure sets $I_l = I_r = \emptyset$ and returns, which ends the current iteration of the `for` loop of line 2.

The following remark states that removing leaves in T when processing a node u does not put into question the previous processing of nodes in the subtree $S(u)$.

Remark 4.4. Let T be a tree with leaf set L and u a node of T s.t. $L(u)$ is an interval in some order π on L, then for any set $S \subseteq L(u)$, $L(u) - S$ defines an interval in $\pi|L - S$.

The correctness of Algorithm 1 follows from the following invariants.

INVARIANT 4.5. *The set $\mathcal{C}$ always contains disjoint 3-leaf subsets C of L such that C is a conflict between T and any tree T_2 s.t. $\pi_2 = \pi$.*

PROOF. $\mathcal{C}$ is initially empty, so the invariant is initially correct. $\mathcal{C}$ grows by addition of 3-leaf sets returned by the procedure `RemoveConflict` in line 4 and line 5. From the above description of this procedure, it must be clear that any such set is composed of leaves in intervals of $\mathcal{I}$, ie of leaves in $L - L(\mathcal{C})$. Thus, any 3-leaf set added to $\mathcal{C}$ is disjoint from sets already in $\mathcal{C}$. Moreover, the sets $\{a, b, c\}$ returned by the procedure are such that $a \in I(c), b \in I, c \in I_l$ (with $I(c) <_\pi I <_\pi I_l$) or $a \in I_r, b \in I, c \in I(c)$ (with $I_r <_\pi I <_\pi I(c)$) where $I(c), I_l, I_r$ contain only leaves in $L(u)$, $b \notin L(u)$.

Thus, $ac|b \in rt(T)$ while $a <_\pi b <_\pi c$ or $c <_\pi b <_\pi a$ and lemma 4.2 applies to show that $\{a, b, c\}$ is a conflict between $T = T_1$ and any tree T_2 s.t. $\pi_2 = \pi$. □

INVARIANT 4.6. *When a node u of T has just been processed, any interval of the list $\mathcal{I}$ contains the leaf set $L(u)$ of some node u of T that is an interval in $\pi|\big(L - L(\mathcal{C})\big)$.*

PROOF. Initially, the invariant holds because $\mathcal{I}$ is initialized with unitary intervals corresponding to the leaf-nodes of T (line 1), and these nodes are trivial intervals. Assume by induction that the invariant holds before an internal node u is processed. Because nodes are processed in post-order, u's children have already been processed and therefore are intervals in $\pi|\big(L - L(\mathcal{C})\big)$.

Let c be a child of u currently examined and $I(c)$ the corresponding interval. Assume there is an interval I such that $I(c) <_\pi I <_\pi I_l$ (or equiv. $I_r <_\pi I <_\pi I(c)$) such that $node(I)$ is not a child of u. In that case, repeated calls to `RemoveConflict` are performed until the intervals corresponding to remaining children of u occur consecutively in $\mathcal{I}$. Indeed, each call to `RemoveConflict` removes a complete interval of $\mathcal{I}$ and, if needed, updates I_l and I_r s.t. the leaves of the already examined children of u that remain form an interval in $\pi|L - L(\mathcal{C})$. Since consecutive intervals in $\mathcal{I}$ cover consecutive elements in $L - L(\mathcal{C})$ (see Figure 2), it follows that u is now an interval in $\pi|L - L(\mathcal{C})$. □

The data structures used by Algorithm 1 are illustrated in Figure 2. The current set $L - L(\mathcal{C})$ of leaves of T is stored in a doubly linked list. As already mentioned, a second doubly linked list $\mathcal{I}$ (sorted according to π) contains intervals of $L - L(\mathcal{C})$. Each interval of $\mathcal{I}$ stores its two extremities as pointers to the corresponding elements of $L - L(\mathcal{C})$. These intervals are associated with the already processed nodes via two symmetric pointers: $node(I)$ indicates the corresponding node v in T, while $I(v)$ refers to v's interval in $\mathcal{I}$. Finally, the list of children of a node u in T is stored as a doubly linked list s.t. removing an element can be done in $O(1)$. The pointers between $\mathcal{I}$ and $L - L(\mathcal{C})$ ensure that the extremities of a new interval, resulting from the union of existing consecutive intervals (see line 6), are automatically set.
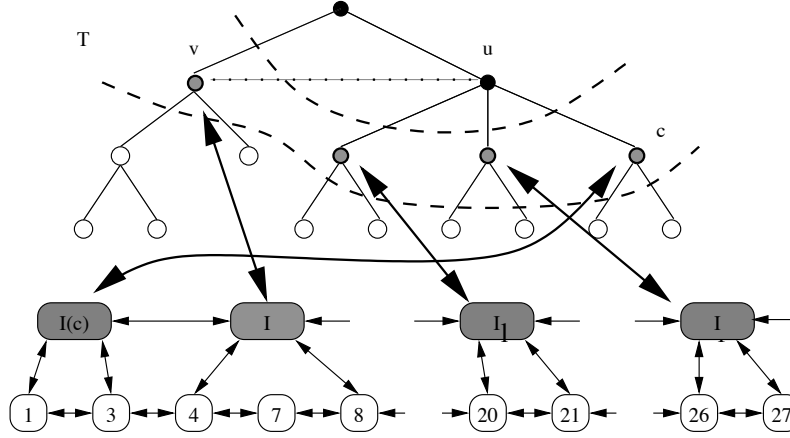
Fig. 2. Illustration of data structures for Algorithm 1. Lists $\mathcal{I}$ (in grey) and $L-L(\mathcal{C})$ are maintained sorted according to $\pi = 1, 2, 3, \ldots n$. Some leaves belonging to conflicts have already been removed from (e.g., leaves 2,5,6). White circles represent processed nodes whose corresponding interval is properly contained in an interval $I \in \mathcal{I}$. Grey circles are recently processed nodes, ie nodes whose corresponding interval is currently an element of $\mathcal{I}$. Black circles are nodes not yet processed. While processing a node u, a subset of consecutive intervals of $\mathcal{I}$, delimited by I_l and I_r, are associated to u. This subset is gradually enlarged by considering new children c of u and their intervals $I(c)$ in $\mathcal{I}$. At that point in the figure, an interval $I \in \mathcal{I}$ such that $I(c) <_\pi I <_\pi I_l$ is met, and since $node(I) = v$ is not a child of u, unlike c and $node(I_l)$, conflicts are identified between T and π (e.g., $\{1, 4, 20\}$).

LEMMA 4.7. *Given an internal node u of T to process such that the children $c_1, \ldots, c_{d^+(u)}$ of u define intervals in $\pi|L - L(\mathcal{C})$, Algorithm 1 identifies in time $O(d^+(u) + |C(u)|)$ a set $C(u)$ of disjoint 3-leaf sets of L s.t. $L(u) - L(C(u))$ is an interval in $\pi|(L - (L(\mathcal{C}) \cup L(C(u))))$.*

PROOF. Let $\mathcal{C}$ be the set of conflicts identified before a node u of T is processed. When processing u Algorithm 1 removes a set $C(u)$ of conflicting triples such that $L(u) - L(C(u))$ is an interval in $\pi|(L - (L(\mathcal{C}) \cup L(C(u))))$. Any interval I of $\mathcal{I}$ considered while processing u either *(i)* corresponds to a child of u or *(ii)* generates calls to Procedure RemoveConflict:

(i) if $node(I)$ is a child of u, I is extracted in constant time from L_{ch}, thanks to pointer $node(I)$ and to pointers between nodes in the tree. Extracting the $d^+(u)$ children of u hence costs $O(d^+(u))$;

(ii) otherwise Procedure RemoveConflict is called with intervals $I(c)$, I and I_b, where I_b is one of I_l, I_r. It identifies n_I disjoint conflicts, where n_I is the cardinal of the smallest set among I, I_b and $I(c)$. Removing from $L - L(\mathcal{C})$ any leaf of such a conflict is done in $O(1)$, as well as updating with pointers between $\mathcal{I}$ and $L - L(\mathcal{C})$. Removing the last leaf of the smallest considered interval leads the corresponding interval to be removed from the doubly linked list $\mathcal{I}$, which is done in $O(1)$. Hence the procedure removes n_I conflicts in time $O(n_I)$. Remark that, as stated in Invariant 4.5, leaves belonging to conflicts are readily removed from $L - L(\mathcal{C})$, thus conflicts removed over the different calls to the procedure are all

disjoint. As $|C(u)|$ conflicts are removed when processing node u, the whole cost of all calls it generates to Procedure `RemoveConflict` is $O(|C(u)|)$.

Summing the costs of the two cases leads to the $O(d^+(u) + |C(u)|)$, the running time claimed for processing node u.   □

PROPOSITION 4.8. *Let T be a tree with an n-leaf set L and π be an order on L. In time $O(n)$, Algorithm 1 identifies a set $\mathcal{C}$ of disjoint conflicts between T and any tree T_2 s.t. $\pi_2 = \pi$ and s.t. any node of $T|(L - L(\mathcal{C}))$ is an interval in $\pi|(L - L(\mathcal{C}))$.*

PROOF. Correctness follows from Invariant 4.5 and Invariant 4.6. Let us analyze the time complexity. Let $C(u)$ be the set of conflicting triples removed while processing any node u of T. From Lemma 4.7, identifying and removing $C(u)$ costs $O(d^+(u) + |C(u)|)$ time. As $\mathcal{C} = \bigcup_{u \in T} C(u)$ contains disjoint 3-leaf sets of L (Invariant 4.5), we have $\sum_{u \in T} |C(u)| \leq n$ and since $\sum_{u \in T} d^+(u) \in O(n)$, the whole complexity of Algorithm 1 is $O(n)$.   □

4.2   Ensuring that nodes of T_1 and T_2 define identical intervals

From the previous subsection, we can reduce in linear time the agreement subtree problem for $\{T_1, T_2\}$ (with leaf set L) to the particular case where $L(u)$ is an interval of π_2 for any node u of T_1. But T_1 and T_2 may still host conflicting triples due to the fact that their nodes can define different intervals. E.g. the first interval in T_1 can be $\{1, 2\}$ while the first interval in T_2 is $\{1, 2, 3\}$, in which case $\{1, 2, 3\}$ is a conflict between the two trees.

For any subtree $S(u)$ induced by a node u of a tree T, let its set of rooted triples be

$$rt(u) = \{x|yz \in rt(T) \ s.t. \ |\{x, y, z\} \cap L(u)| \geq 2\}$$

and its set of fans be

$$f(u) = \{\{x, y, z\} \in f(T) \ s.t. \ \{x, y, z\} \subseteq L(u)\}$$

Note that if r is the root node of tree T, then $rt(r) = rt(T)$ and $f(r) = f(T)$.

Definition 4.9. Define a node u in tree T_1 to be *valid* w.r.t. tree T_2 if both $rt(u) \subseteq rt(T_2)$ and $f(u) \subseteq f(T_2)$ hold.

It should be noticed that the validity of a node is an invariant property under leaf removal (because of Remark 4.4).

LEMMA 4.10. *If a node u in tree T_1 is valid w.r.t. T_2, then there is a node v of T_2 such that $L(u) = L(v)$.*

PROOF. Let v be the lowest node of T_2 such that $L(u) \subseteq L(v)$. If follows that $L(u)$ intersects the leaf set of at least two different children of v, say v_1 and v_2. Assume v has more than two children and let v_i be a child of v different from v_1 and v_2. Consider three leaves $l_1 \in L(u) \cap L(v_1)$, $l_2 \in L(u) \cap L(v_2)$ and $l \in L(v_i) \setminus L(u)$. By definition of l_1, l_2 and l, $\{l, l_1, l_2\} \in f(T_2)$ while $l|l_1, l_2 \in rt(u)$: contradiction with $rt(u) \subseteq rt(T_2)$. It follows that if v has more than two children then for any child v_i, $L(v_i) \subset L(u)$, thereby $L(v) = L(u)$. Assume now v_1 and v_2 are the only children of v. Assume also without loss of generality that there is $l \in L(v_1) \setminus L(u)$. Then the rooted triple $l_1 l|l_2$ belongs to $rt(T_2)$, while $l_1 l_2|l \in rt(u)$, which is in contradiction with $rt(u) \subseteq rt(T_2)$. It follows that $L(v) = L(v_1) \cup L(v_2) = L(u)$.   □

Thus, if all nodes of T_1 are valid w.r.t. T_2 then these trees are isomorphic (both trees have the same leaf set). The second filtering step aims to remove disjoint conflicting triples $\mathcal{C}$ so that any node of $T_1|\big(L - L(\mathcal{C})\big)$ becomes valid w.r.t. $T_2|\big(L - L(\mathcal{C})\big)$. As a consequence of next Lemma, we will show that the set $\mathcal{C}$ can be identified by a post-order search of T_1. Given a node u, let $p(m(u))$ be the leaf preceding $m(u)$ in in π_2 if it exists. Similarly let $s(M(u))$ be the leaf following $M(u)$ in π_2 if it exists. On the example of Figure 2, $p(m(u))$ is not defined (there is no leaf smaller than 1) while, assuming that the leaf 28 still exists (even if not drawn on the figure), $s(M(u)) = 28$.

LEMMA 4.11. *Assuming any node u of T_1 defines an interval $L(u)$ in π_2, a node u of T_1 whose children are all valid w.r.t. T_2, is also valid w.r.t. T_2 if both:*

(i) $\{p(m(u))|m(u)M(u),\ s(M(u))|m(u)M(u)\} \subseteq rt(T_2)$;

(ii) *and if u has children $c_1, c_2, \ldots, c_{d^+(u)}$ with $d^+(u) > 2$, then for any $i \in \{1, 2, \ldots, d^+(u) - 2\}$, $\{m(c_i), m(c_{i+1}), m(c_{i+2})\} \in f(T_2)$.*

PROOF. Let us prove that (i) implies $rt(u) \subseteq rt(T_2)$ and that (ii) implies $f(u) \subseteq f(T_2)$. Thus, if both conditions hold then u is valid.

(i) Assume $\{p(m(u))|m(u)M(u),\ s(M(u))|m(u)M(u)\} \subseteq rt(T_2)$. Rooted triples of $rt(u)$ are of two kinds: those having their three leaves in $L(u)$ and those having two leaves in $L(u)$ and one leaf in $L - L(u)$. Notice that any rooted triple $l_1|l_2l_3$ of the first kind belongs to $rt(c_i)$ for some child c_i of u. As by assumption c_i is valid, $l_1|l_2l_3 \in rt(T_2)$. The other rooted triples $l_1|l_2l_3$ of $rt(u)$ are s.t. $\{l_2, l_3\} \in L(u)$ and $l_1 \notin L(u)$.

CLAIM 4.12. $p(m(u))|m(u)M(u) \in rt(T_2) \Rightarrow l_1|m(u)M(u) \in rt(T_2), \forall l_1$ s.t. $l_1 \leq_{\pi_2} p(m(u))$.

Proof: Let v be the lca of $m(u)$ and $M(u)$ in T_2. As $L(v)$ is an interval of π_2, if $p(m(u))|m(u)M(u) \in rt(T_2)$ then $p(m(u)) \notin L(v)$. This implies that $m(v) = m(u)$ hence that $l_1|m(u)M(u) \in rt(T_2)$ for any leaf l_1 s.t. $l_1 \leq_{\pi_2} p(m(u))$. Similarly, the following claim holds.

CLAIM 4.13. $s(M(u))|m(u)M(u) \in rt(T_2) \Rightarrow l_1|m(u)M(u) \in rt(T_2), \forall l_1$ s.t. $l_1 \geq_{\pi_2} s(M(u))$.

Now considering triples $l_1|l_2l_3$ of $rt(u)$ s.t. $\{l_2, l_3\} \in L(u)$ and $l_1 \notin L(u)$, note that since $L(u)$ is an interval of π_2 then either $l_1 \leq_{\pi_2} p(m(u))$ or $l_1 \geq_{\pi_2} s(M(u))$. Thus, assuming (i), for any $l_1 \notin L(u)$ one of the two above claims applies to show $l_1|l_2l_3 \in rt(T_2)$.
Overall, this shows that (i) implies $rt(u) \subseteq rt(T_2)$.

(ii) A fan $\{l_1, l_2, l_3\} \in L(u)$ is either a fan of some child c_i of u (in which case it belongs by assumption to $f(T_2)$) or l_1, l_2 and l_3 are respectively leaves of three distinct children of u. Thereby let us assume that u has at least three children, namely c_1, c_2, $\ldots$, $c_{d^+(u)}$. Let us denote $f_i := \{m(c_i), m(c_{i+1}), m(c_{i+2})\}$ and $F_i := \bigcup_{1..i} f_i$.

CLAIM 4.14. *If all children of u are leaves and $F_{d^+(u)-2} \subseteq f(T_2)$, then $f(u) \in f(T_2)$.*

Proof: By induction on the number $d^+(u)$ of children. It is obviously true for $d^+(u) = 3$. Assume it holds for any value less than k. Let u be a node with k children $l_1 \ldots l_k$. By induction, any fan $\{l, l', l''\} \in f(u)$ such that $l_k \notin \{l, l', l''\}$ belongs to $f(T_2)$. Thus we only have to consider fans involving l_k. Note that by assumption, $\{l_{k-2}, l_{k-1}, l_k\} \in f(u)$. Thus it remains to show that $\{l, l', l_k\} \in f(u)$ with l, l' distinct from l_{k-2}, l_{k-1} belongs to $f(T_2)$. As by induction, $\{l, l', l_{k-2}\} \in f(T_2)$ and $\{l, l', l_{k-1}\} \in f(T_2)$, the lca in T_2 of any pair of leaves among $\{l, l', l_{k-1}, l_{k-2}\}$ is the same node v. As by assumption, $\{l_{k-2}, l_{k-1}, l_k\} \in f(T_2)$, then $v = lca_{T_2}(l_{k-2}, l_k) = lca_{T_2}(l_{k-1}, l_k)$. which implies that $\{l, l', l_k\} \in f(T_2)$.

Now consider the more general case where children of u are not restricted to be leaves.

CLAIM 4.15. *If $f_i \in f(T_2)$ then $\{l_i, l_{i+1}, l_{i+2}\} \in f(T_2)$ for all $l_i \in L(c_i)$, $l_{i+1} \in L(c_{i+1})$, $l_{i+2} \in L(c_{i+2})$.*

Proof: By assumption, nodes c_i, c_{i+1} and c_{i+2} are valid. Thus by Lemma 4.10, there are three nodes v_i, v_{i+1} and v_{i+2} of T_2 such that $L(c_i) = L(v_i)$, $L(c_{i+1}) = L(v_{i+1})$ and $L(c_{i+2}) = L(v_{i+2})$. Thus, $f_i \in f(T_2)$ implies that v_i, v_{i+1}, v_{i+2} have a common lca, i.e. that $\{l_i, l_{i+1}, l_{i+2}\} \in f(T_2)$ for all $l_i \in L(c_i)$, $l_{i+1} \in L(c_{i+1})$, $l_{i+2} \in L(c_{i+2})$.

The two previous claims have for direct consequence that $F_{d^+(u)-2} \subseteq f(T_2)$ implies $f(u) \subseteq f(T_2)$.

□

This lemma directly gives rise to a simple algorithm (see pseudo-code Algorithm 2) that identifies an agreement subtree of T_1 and T_2 by identifying a set $\mathcal{C}$ of disjoint conflicts between the trees. Nodes u of T_1 are processed in post-order so that u is processed knowing the validity of its children. In this case, the previous lemma states that only a small number of rooted triples and fans have to be tested to ensure the validity of u. Namely, $O(|C(u)| + d^+(u))$ 3-leaf sets are examined when processing u, where $C(u)$ denotes disjoint conflicts to remove from L, additionally to those previously removed, to ensure the validity of u.

THEOREM 4.16. *Given two trees T_1 and T_2 on a same n-leaf set L s.t. any node of T_1 is an interval in π_2, then Algorithm 2 outputs in $O(n)$ time an agreement subtree of T_1, T_2 by removing a set $\mathcal{C}$ of disjoint conflicts between the trees.*

PROOF. We first prove that $\mathcal{C}$ contains disjoint conflicts between T_1 and T_2. Because for any node u of T_1 $L(u)$ is an interval of π_2, we have

$$\{p(m(u))|m(u)M(u), s(M(u))|m(u)M(u)\} \subseteq rt(T_1)$$

Thus, if any of these rooted triples is not in $rt(T_2)$ then its three leaves are a conflict between T_1 and T_2. Similarly, $\{m(c_i), m(c_{i+1}), m(c_{i+2})\} \in f(T_1)$, thus if this 3-leaf set is not a fan of T_2, then it is a conflict between the two trees. Thus, only 3-leaf sets representing conflicts between the trees are put in $\mathcal{C}$. Moreover, each time a conflict is discovered, it involves 3 leaves currently in T_1, and these leaves are then readily removed from the tree. Hence, they cannot belong to any conflict

---

Algorithm 2: AGREEMENTSUBTREE (T_1, T_2)

$\mathcal{C} \leftarrow \emptyset$

for *each node u in a post order traversal of T_1* **do**

    /* *Ensures that $rt(u) \subseteq rt(T_2)$* */

    repeat

        if $p(m(u))|m(u)M(u) \notin rt(T_2)$ **then**

            $\lfloor \mathcal{C} \leftarrow \mathcal{C} \cup \{p(m(u)), m(u), M(u)\}$; remove these leaves from T_1

        else if $s(M(u))|m(u)M(u) \notin rt(T_2)$ **then**

            $\lfloor \mathcal{C} \leftarrow \mathcal{C} \cup \{s(M(u)), m(u), M(u)\}$; remove these leaves from T_1

    until $\{p(m(u))|m(u)M(u), s(M(u))|m(u)M(u)\} \subseteq rt(T_2)$ *or* $d^+(u) < 2$

    /* *Ensures that $f(u) \subseteq f(T_2)$* */

    $i \leftarrow 1$

    while $d^+(u) > 2$ *and* $i \leq d^+(u) - 2$ **do**

        let $c_1, c_2, \ldots, c_{d^+(u)}$ be the children of u

        if $\{m(c_i), m(c_{i+1}), m(c_{i+2})\} \in f(T_2)$ **then** $i \leftarrow i + 1$

        else $\mathcal{C} \leftarrow \mathcal{C} \cup \{m(c_i), m(c_{i+1}), m(c_{i+2})\}$; remove these leaves from T_1

    /* *now u is valid* */

return T_1

---

identified afterwards, which proves that the conflicts identified during Algorithm 2 are disjoint.

Now, the fact that T_1 and T_2 are isomorphic when restricted to $L - L(\mathcal{C})$ results from Lemma 2.1-(ii) and Lemma 4.11. Indeed, when processing a node u and a conflict is encountered, its leaves can be removed from the trees without changing the pre-established validity of inner nodes of $S(u)$. Moreover, when the algorithm stops after processing the root r_1 of T_1, Lemma 4.11 guarantees that $rt(r_1) = rt(T_1) \subseteq rt(T_2)$ and $f(r_1) = f(T_1) \subseteq f(T_2)$. As T_1 and T_2 have the same leaf set, this implies in fact that $f(T_1) = f(T_2)$ and $rt(T_1) = rt(T_2)$, i.e. that the trees are isomorphic (from Lemma 2.2).

Concerning the complexity issue, when processing a node u, only $O(d^+(u) + |C(u)|)$ 3-leaf sets are examined, where $C(u)$ denotes the conflicts encountered when specifically processing node u. Thus, processing the whole tree T_1 globally involves examining $O(n)$ 3-leaf sets as $\Sigma_{u \in T_1} d^+(u) = O(n)$ and $|\mathcal{C}| = \Sigma_{u \in T_1} |C(u)| = O(n)$.

So the main issue is to maintain, for the current node u of T_1, constant time access to $m(u)$ and $M(u)$ under leaf removals. The leaves are stored in a doubly linked list L sorted according to π_2. Initially each node u of T_1 is given two symmetric pointers toward $m(u)$ and $M(u)$ in L. These pointers are initialized only for leaf nodes and receive the *null* value for the other nodes. When a node u has been processed, it forwards its pointers $m(u)$ and $M(u)$ to its parent v in T_1. Then, if needed, node v updates its own pointers $m(v)$ and $M(v)$ (e.g. $m(v) \leftarrow m(u)$ if $m(u) <_{\pi_2} m(v)$ or $m(v)$ is *null*). Notice that now leaf $m(u)$ does not point toward u anymore. Hence at any step, each leaf is associated to at most one node of T_1. Assume a leaf l is removed and that for some node u, $l = m(u)$ or $l = M(u)$. Then if $m(u) = M(u)$, u contains no more leaf in its subtree and the algorithm starts

processing another node u of T_1. Otherwise, $m(u)$ (resp. $M(u)$) now points to the leaf following $m(u)$ (resp. preceding $M(u)$ in L. Any of these operations costs constant time.

Finally, it is necessary to preprocess T_2 in $O(n)$ time so that the least common ancestor of any two of its nodes is identified in $O(1)$ [Harel and Tarjan 1984]. This allows to know in constant time whether a 3-leaf set is a fan or a rooted triple in T_2. Note that leaves involved in conflicts need not really to be removed from T_2 as this does not change the lca of other leaves. Hence, dynamic lcas are not needed here.

Thus, running Algorithm 2 on two n-leaf trees costs $O(n)$ time.   □

4.3   Approximating a collection of k trees

THEOREM 4.17. *The* CMAST *problem on a collection of k rooted trees on the same leaf set can be 3-approximated in $O(kn)$ time.*

PROOF. Computing an agreement subtree for a collection $\mathcal{T}$ of k trees is done by first computing an agreement subtree T of two trees $T_1, T_2 \in \mathcal{T}$, which requires one call to Algorithm 1 followed by one call to Algorithm 2. Then leaves of the conflicts identified between T_1, T_2 are removed from all trees of the collection and another tree is chosen in $\mathcal{T}$ to be considered with T to obtain an agreement subtree. The process is iterated until all trees of $\mathcal{T}$ have been processed. The agreement subtree obtained at the last iteration is an agreement subtree of any tree considered in the process, i.e. of any tree in the initial collection (as isomorphism is invariant under leaf removal, see Lemma 2.1).

Overall, $k - 1$ calls to Algorithms 1 and 2 are performed, requiring $O(kn)$ total time, and disjoint conflicts are globally identified. Together, these conflicts form an hs-peacemaker of $\mathcal{T}$, hence the last agreement subtree is a 3-approximation for CMAST on $\mathcal{T}$ (from Lemma 2.3-(i)).   □

5. DISCUSSION

In this paper we have presented algorithms for MAST and MCT which are 3-approximations and whose complexity is linear. Let us discuss these two aspects on the MAST problem.

Approximation ratio. It is often the case that the theoretical approximation ratio of an algorithm differs significantly from the practical approximation ratio observed on real instances. Indeed the theoretical ratio is a worst case upper bound. Let us briefly report on the results obtained from real instances for MAST. Let $s_{opt} = \frac{|T_{opt}|}{n}$ be the relative size of an optimal tree T_{opt}. We mainly compare the practical ratio of our algorithm with respect to the s_{opt} value, averaged over a large (several hundreds) number of instances. Two kinds of simulations were conducted, first on two random trees, then on $k > 2$ trees inferred by phylogenetic software from simulated nucleotide data sets (see [Guindon and Gascuel 2003] for a precise protocol). The former trees contained 15 leaves, while the latter contained 24 leaves.

On both kinds of simulations, the results clearly indicate that the practical ratio improves when s_{opt} decreases. In other words, for a fixed n, the ratio decreases when k grows and when the conditions considered for obtaining the input trees become more difficult (eg, shorter sequences, more heterogeneous evolutionary rates between edges of the model tree). Moreover we also observed that for two trees (and most likely for a fixed $k > 2$), increasing n leads to decrease both *sopt* and the practical ratio. This behaviour is in accordance with the result of [Bryant et al. 1983] showing that the expected size of an optimal tree for MAST on two random input trees is in $O(\sqrt{n})$. Thus, $s_{opt} = O(\frac{\sqrt{n}}{n})$ tends to 0 as n increases.

Over all experiments, average absolute values observed for the practical ratio are the following: 1.5 on two random trees; 2.74 on phylogenetic trees inferred from sequences of 5000 nucleotides; and 1.54 when resorting to sequences of 100 nucleotides.

Time complexity issues. Besides its own interest, improving the time complexity of the 3-approximation algorithm for MAST has interesting consequences. Mainly it enables the use of this algorithm as a fast subroutine in more sophisticated algorithms, for instance to compute an exact solution (within exponential or parameterizedcomplexity). Indeed, in the context of parameterized complexity, an approximation algorithm is required as part of polynomial time reduction rules to obtain a polynomial kernel for the 3-HITTING SET problem [Nishimura et al. 2004]. Thanks to the well-known relationship between 3-HITTING SET and MAST, similar polynomial reduction rules can be derived for MAST. Optimizing the time complexity of these reduction rules is an important issue.

REFERENCES

Amir, A. and Keselman, D. 1997. Maximum agreement subtree in a set of evolutionary trees: metrics and efficient algorithm. *SIAM Journal on Computing 26,* 6, 1656–1669.

Berger-Wolf, T. 2004. Consensus and agreement of phylogenetic trees. In *4th Workshop on Algorithms in Bioinformatics (WABI)*. LNCS. Springer-Verlag, 350–361.

Berry, V. and Nicolas, F. 2004. Maximum agreement and compatible supertrees. In *Proceedings of The 15th Annual Symposium on Combinatorial Pattern Matching (CPM'04)*, S. C.

Sahinalp, S. Muthukrishnan, and U. Dogrusoz, Eds. Lecture Notes in Computer Science, vol. 3109. Springer-Verlag, 205–219.

BERRY, V. AND NICOLAS, F. to appear. Improved parametrized complexity and approximation of the maximum agreement subtree and maximum compatible tree problems. *IEEE/ACM Transactions in Computational Biology and Bioinformatics*.

BRYANT, D. 1997. Building trees, hunting for trees and comparing trees: theory and method in phylogenetic analysis. Ph.D. thesis, University of Canterbury, Department of Mathemathics.

BRYANT, D., STEEL, M., AND MACKENZIE, A. 1983. The size of a maximum agreement subtree for random binary trees. In *BioConsensus*, M. Janowitz, F.-J. Lapointe, F. McMorris, B. Mirkin, and F. Roberts, Eds. DIMACS AMS, 55–66.

COLE, R., FARACH-COLTON, M., HARIHARAN, R., PRZYTYCKA, T. M., AND THORUP, M. 2001. An $O(n \log n)$ algorithm for the Maximum Agreement SubTree problem for binary trees. *SIAM Journal on Computing 30,* 5, 1385–1404.

DOWNEY, R. G., FELLOWS, M. R., AND STEGE, U. 1999. Computational tractability: The view from mars. *Bulletin of the European Association for Theoretical Computer Science 69*, 73–97.

FARACH, M., PRZYTYCKA, T. M., AND THORUP, M. 1995. On the agreement of many trees. *Information Processing Letters 55,* 6, 297–301.

GANAPATHY, G. AND WARNOW, T. J. 2002. Approximating the complement of the maximum compatible subset of leaves of k trees. In *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX'02)*. Lecture Notes in Computer Science, vol. 2462. Springer-Verlag, 122–134.

GANAPATHYSARAVANABAVAN, G. AND WARNOW, T. J. 2001. Finding a maximum compatible tree for a bounded number of trees with bounded degree is solvable in polynomial time. In *Proceedings of the 1st International Workshop on Algorithms in Bioinformatics (WABI'01)*, O. Gascuel and B. M. E. Moret, Eds. Lecture Notes in Computer Science, vol. 2149. Springer-Verlag, 156–163.

GUINDON, S. AND GASCUEL, O. 2003. A simple, fast and accurate method to estimate large phylogenies by maximum-likelihood. *Systematic Biology 52,* 5, 696–704.

GUPTA, A. AND NISHIMURA, N. 1998. Finding largest subtrees and smallest supertrees. *Algorithmica 21,* 2, 183–210.

HAMEL, A. M. AND STEEL, M. A. 1996. Finding a maximum compatible tree is NP-hard for sequences and trees. *Applied Mathematics Letters 9,* 2, 55–59.

HAREL, D. AND TARJAN, R. E. 1984. Fast algorithms for finding nearest common ancestor. *SIAM Journal on Computing 13,* 2, 338–355.

HEIN, J., JIANG, T., WANG, L., AND ZHANG, K. 1996. On the complexity of comparing evolutionary trees. *Discrete Applied Mathematics 71,* 1–3, 153–169.

KAO, M.-Y., LAM, T. W., SUNG, W.-K., AND TING, H.-F. 1999. A decomposition theorem for maximum weight bipartite matchings with applications to evolutionary trees. In *Proceedings of the 7th Annual European Symposium on Algorithms (ESA'99)*. Lecture Notes in Computer Science, vol. 1643. Springer-Verlag, 438–449.

KAO, M.-Y., LAM, T. W., SUNG, W.-K., AND TING, H.-F. 2001. An even faster and more unifying algorithm for comparing trees via unbalanced bipartite matchings. *Journal of Algorithms 40,* 2, 212–233.

MCMORRIS, F., MERONIK, D., AND NEUMANN, D. 1983. A view of some consensus methods for trees. In *Numerical Taxonomy*, J. Felsenstein, Ed. Springer-Verlag, 122–125.

NISHIMURA, N., RAGDE, P., AND THILIKOS, D. 2004. Smaller kernels for hitting set problems of constant arity. In *International Workshop on Parameterized and Exact Computation (IWPEC)*. Number 3162 in Lecture Notes in Computer Science. 121–126.

SEMPLE, C. AND STEEL, M. 2003. *Phylogenetics*. Oxford Lecture Series in Mathematics and its Applications, vol. 24. Oxford University Press.

STEEL, M. A. AND WARNOW, T. J. 1993. Kaikoura tree theorems: Computing the maximum agreement subtree. *Information Processing Letters 48,* 2, 77–82.