Improved Parametrized Complexity of Maximum Agreement Subtree and Maximum Compatible Tree problems

Vincent Berry, François Nicolas

Improved parameterized complexity of the Maximum Agreement Subtree and Maximum Compatible Tree problems
LIRMM, Tech.Rep. num 04026

Vincent Berry*, François Nicolas
Équipe *Méthodes et Algorithmes pour la Bioinformatique – L.I.R.M.M.*
161, rue Ada 34392 Montpellier cedex 5, France
E-mails:vberry@lirmm.fr, nicolas@lirmm.fr *

Abstract

Given a set of evolutionary trees on a same set of taxa, the maximum agreement subtree problem (MAST), respectively maximum compatible tree problem (MCT), consists of finding a largest subset of taxa such that all input trees restricted to these taxa are isomorphic, respectively compatible. These problems have several applications in phylogenetics such as the computation of a consensus of phylogenies obtained from different datasets, the identification of species subjected to horizontal gene transfers and, more recently, the inference of supertrees, e.g. Trees Of Life.

We provide two linear time algorithms to check the isomorphism, respectively compatibility, of a set of trees or otherwise identify a conflict between the trees with respect to the relative location of a small subset of taxa.

Then, we use these algorithms as subroutines to solve MAST and MCT on rooted or unrooted trees of unbounded degree. More precisely, we give exact fixed-parameter tractable algorithms, whose running time is uniformly polynomial when the number of taxa on which the trees disagree is bounded. The improves on a known result for MAST and proves fixed-parameter tractability for MCT.

Keywords: Phylogenetics, algorithms, consensus, pattern matching, trees, compatibility, fixed-parameter tractability.

1 Introduction

This paper investigates two tree consensus problems with applications in phylogenetics. This field aims at reconstructing the evolutionary history of species or

*Supported by the *Action Incitative Informatique-Mathématique-Physique en Biologie Moléculaire* [ACI IMP-Bio].

more generally taxa. This evolutionary history is represented by an *evolutionary* tree, or *phylogeny*, in which leaves are labelled by present-day taxa and internal nodes correspond to hypothetical ancestors of taxa. The branching pattern of such a tree shows how speciation events have resulted in different taxonomic groups, *i.e.* shows how taxa relate to one another in terms of common ancestors.

1.1 Overview of MAST and MCT

The two problems considered in this paper take as input a set of evolutionary trees on a same set of taxa. We begin by stating the problems and indicating motivation for their study.

1.1.1 Maximum agreement subtree

Given a set of evolutionary trees on the same set of taxa, the Maximum Agreement SubTree problem (MAST) consists of finding a subtree homeomorphically included in all input trees and with the largest number of taxa [1, 2, 3, 4, 5, 6]. In other words, this involves selecting a largest set of taxa such that the input trees are isomorphic, *i.e. agree*, when restricted to these taxa.

This problem arises in various areas, including phylogenetics, where it can be used to reach different practical goals:

- to obtain the largest *intersection* of a set of phylogenies inferred from different molecular or morphological datasets. These datasets can be, *e.g.* different regions of the same molecular sequences, or sequences of different genes, suspected to result from different evolutionary histories. This largest intersection is used to measure the similarity of the different estimated histories or to identify species that could be implied in horizontal transfers of genes.

- Systematic biologists also use MAST (*e.g.*, implemented in the well-known PAUP package [7]) as a method to obtain a consensus of a set of phylogenies that are optimal for some tree-building criterion. *E.g.*, for the parsimony criterion, some datasets can induce several dozens (sometimes hundreds) of optimal trees. Similarly, methods that build trees according to a maximum likelihood criterion can give numerous trees with non-significantly different likelihood values. In such cases, when ranking the output phylogenies by decreasing likelihood values, the first tree alone may not be a good representative of the evolutionary hypotheses for the studied species, and a consensus of the first dozen trees is often preferred.

  Depending on the differences in the trees considered, the MAST method can be the consensus method giving the most informative output [7, 8].

- Recently, MAST also appeared as a subproblem of a supertree inference method [9, 10]. This method builds an agreement *supertree* of a collection of input trees having *non-identical* leaf sets. The main application of supertree methods is the construction of *Trees of Life* spanning several

thousands of species. Here, fast polynomial time algorithms are of crucial importance.

1.1.2 Maximum compatible tree

A variant of MAST, most often called Maximum Compatible Tree (MCT) [11, 12, 13, 8] is also of particular interest in phylogenetics when the input trees are not binary. In an evolutionary tree, a node with more than two descendants usually represents uncertainty with respect to the relative groupings of its descendants rather than a multi-speciation event. The MCT problem takes this into account by seeking a tree that is *compatible* with the input trees and that contains a largest number of taxa The compatibility of two trees means that the least common ancestor of a subset of taxa can be of high degree in one tree and of low degree in the other, as long as the groups defined by both trees on this subset of taxa can be represented together in a same output tree. In practice, phylogenetic softwares usually output binary trees from primary data. However, one can typically resort to the MCT problem when the input trees are provided with confidence values assigned to their edges (*e.g.* thanks to a bootstrap process of the primary data). The edges with the lowest confidence are usually discarded from the analysis, which results in the creation of some higher degree nodes in the input trees.

Note that a maximum compatible tree of a collection of trees always contains at least as many taxa as a maximum agreement subtree of the collection, since compatibility is a weaker constraint than isomorphism. Also, the MCT and MAST problems are identical when the input trees are binary.

1.2 Previous results

1.2.1 Polynomial cases of MAST and MCT

The MAST problem is NP-hard on only three rooted trees of unbounded degree [3], and MCT on two rooted trees as soon as one of them is of unbounded degree [13]. Efficient polynomial time algorithms have been recently proposed for MAST on two rooted n-leaf trees: $O(n \log n)$ for binary trees [6], and $O(\sqrt{d}n \log \frac{2n}{d})$ for trees of degree bounded by d [5]. When the two input trees are unrooted and of unbounded degree, the $O(n^{1.5})$ algorithm of [14] can be used.

Suppose k rooted trees are given as input:

- if at least one of these input trees has maximum degree d then MAST can be solved in $O(n^d + kn^3)$ time [2, 3, 15] and,
- if all of the input trees have maximum degree d then MCT can be solved in $O(2^{2kd}n^k)$ time [12].

1.2.2 Fixed parameter tractability

MAST is known to be *fixed-parameter tractable* (FPT) in p, the smallest number of labels to remove from the input set of labels such that the input trees agree: [16] describe an algorithm in $O(3^p kn \log n)$ time and [17] give an algorithm in $O(2.27^p + kn^3)$ time. This parameterized version of the problem is of particular interest in phylogenetics where many instances of MAST and MCT now consist of phylogenies inferred by different tree-building methods on the basis of molecular sequences of reasonable length. Hence, the trees given as input to MAST and MCT usually differ w.r.t. the location of a small number p of species.

1.2.3 Approximability

See [18] and references therein.

1.3 Our contribution

1.3.1 Linear-time algorithms

We propose two linear time algorithms that check the isomorphism, respectively compatibility of a collection of k input trees or that otherwise identify a small set of taxa on which two input trees conflict. By identifying such a set of taxa, our algorithms extend the work of [19], respectively [20] that only decide isomorphism, respectively compatibility, without increasing the running time. We provide algorithms for both collections of rooted trees and collections of unrooted trees.

1.3.2 Fixed parameter tractability

Building on the work of [16], we obtain an $O(\min\{3^p kn, 2.27^p + kn^3\})$ parameterized algorithm for both MAST and MCT on rooted trees. This improves the time bound for the MAST problem with respect to [16] and is the first result of fixed-parameter tractability for MCT. Moreover, from this standpoint, MCT has the same complexity as MAST which was not expected.

We show how these algorithms can be used at most $p+1$ times to handle the case of collections of unrooted trees. The exponential term in the complexity of the presented FPT algorithms for MCT does not depend on the degree or number of input trees, which might be an advantage in practice over the algorithm of [12], although the latter may be faster for trees with a high level of disagreement.

1.4 Organization of our paper

Sect. 2 presents definitions and preliminary results, then Sect. 3 presents linear time algorithms that conclude on the isomorphism and compatibility of trees, or otherwise identify a conflict on a small subset of labels. Then Sect. 4 shows

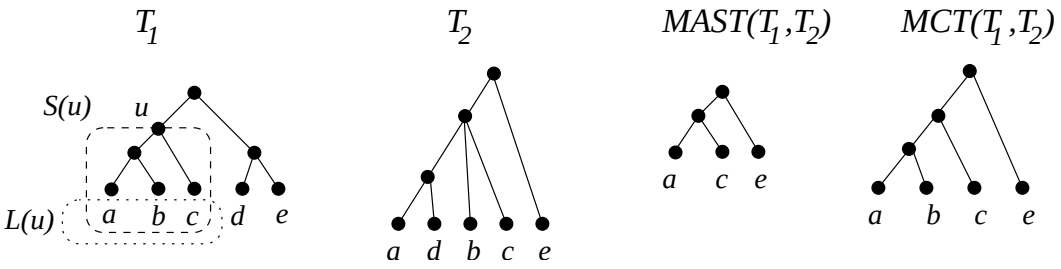

Figure 1: Four rooted trees. A collection $\mathcal{T} := \{T_1, T_2\}$, one of the $MAST(\mathcal{T})$ trees, and the $MCT(\mathcal{T})$.

how these algorithms can be used as subroutines of fixed-parameter algorithms to solve MAST and MCT for rooted and unrooted trees.

2 Definitions and preliminaries

Formally, any tree T considered in this paper has its leaf set $L(T)$ in bijection with a *label* set representing taxa, and is either *rooted*, in which case all internal nodes have at least two children, or *unrooted*, in which case internal nodes have degree at least three. When there is no ambiguity, we identify a leaf with its label. The *size* of a tree T is the number of its leaves and is denoted $\#T$. Let u be a node in a rooted tree, we denote $S(u)$ the subtree rooted at u (*i.e.* u and its offspring) and denote $L(u)$ the set of leaves of this subtree. As an example, for the node u in the tree T_1 of Fig. 1, the subtree $S(u)$ is the one enclosed in the dashed area and $L(u)$ is the set of leaves enclosed in the dotted area, *i.e.* $\{a, b, c\}$. If C is a set of nodes in a tree, then define $L(C) := \bigcup_{u \in C} L(u)$. Given a rooted tree T and a set of leaves $L \subseteq L(T)$, we denote $lca_T(L)$ the node that is the lowest common ancestor of L in T.

2.1 Definition of MAST and MCT

The definitions and results of this section apply both to rooted and unrooted trees.

Definition 1 *Given a set L of labels and a tree T, the* restriction *of T to L, denoted $T|L$, is the tree obtained in the following way: take the smallest induced subgraph of T connecting leaves with labels in $L \cap L(T)$; then remove any degree two (non-root) node to make the tree homeomorphically irreducible. If $\mathcal{T}$ is a collection of trees, then define $\mathcal{T}|L := \{T|L : T \in \mathcal{T}\}$.*

See trees U, U' in Fig. 2 for an example. Note that for any tree T and any two label sets L and L', it holds that $(T|L)|L' = T|(L \cap L') = (T|L')|L$.

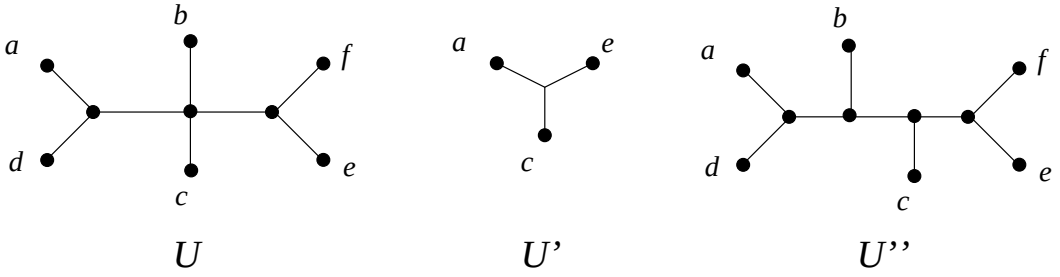

Figure 2: Three unrooted trees U, U', U'' such that $U' = U|\{a, c, e\} \sqsubseteq U$ and $U'' \trianglerighteq U$.

Definition 2 *Two rooted (respectively unrooted) trees T, T' are* isomorphic*, which is denoted $T = T'$, iff there exists a one-to-one mapping from the nodes of T onto the nodes of T' preserving leaf labels and descendancy (respectively leaf labels and adjacency). Let T, T' be two trees, T is* homeomorphically included *in T', which is denoted $T \sqsubseteq T'$, iff $T = T'|L(T)$.*

The well-known MAST problem is defined as follows:

Definition 3 *Given a collection $\mathcal{T}$ of rooted, respectively unrooted, trees with identical leaf set L, an* agreement subtree *of $\mathcal{T}$ is any rooted, respectively unrooted, tree T with leaves in L s.t. $\forall T_i \in \mathcal{T}$, $T \sqsubseteq T_i$. An agreement subtree of $\mathcal{T}$ that is of maximum size is called a* maximum agreement subtree *of $\mathcal{T}$ and is denoted $MAST(\mathcal{T})$. The corresponding optimization problem is stated as follows:*

Name: Maximum Agreement SubTree (MAST)
Input: *A collection $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ of k trees (all rooted or all unrooted) with identical leaf set L of cardinality n.*
Task: *Find a maximum agreement subtree of $\mathcal{T}$.*

The MCT variant of MAST is based on the relation of refinement instead of that of isomorphism.

Definition 4 *A tree T* refines *a tree T', which is denoted $T \trianglerighteq T'$, whenever T can be transformed into T' by contracting some of its internal edges (contracting an edge (u, v) means removing nodes u and v, replacing them by a single new node that is made adjacent to every node previously adjacent to u or v). More generally, a tree T refines a collection $\mathcal{T}$, which is denoted $T \trianglerighteq \mathcal{T}$, whenever T refines all trees in $\mathcal{T}$.*

Note that an evolutionary tree T refining another tree T' agrees with the entire evolutionary history of T', while containing additional history absent from T'. See Fig. 2 for an illustration of the $\trianglerighteq$ notation on unrooted trees. The MCT problem is defined as:

Definition 5 *Given a collection $\mathcal{T}$ of rooted, respectively unrooted, trees, with identical leaf set L, a rooted, respectively unrooted, tree T with leaves in L is said to be* compatible with $\mathcal{T}$ *iff* $\forall T_i \in \mathcal{T}$, $T \trianglerighteq T_i|L(T)$. *If there is a tree T compatible with $\mathcal{T}$ s.t. $L(T) = L$,* i.e. *that is a common refinement to all trees in $\mathcal{T}$, then the collection $\mathcal{T}$ is simply said to be* compatible. *Note that this is generally not the case and it is thus interesting to find a* maximum compatible tree *of $\mathcal{T}$, defined as a tree compatible with $\mathcal{T}$ that contains a maximum number of leaves. Such a tree is denoted $MCT(\mathcal{T})$. The corresponding optimization problem is stated as follows:*

Name: Maximum Compatible Tree (MCT)
Input: *A collection $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ of k trees (all rooted or all unrooted) with identical leaf set L of cardinality n.*
Task: *Find a maximum compatible tree of $\mathcal{T}$.*

Fig. 1 shows an example of a tree $MAST(\mathcal{T})$ and a tree $MCT(\mathcal{T})$ for a collection of two rooted trees. Note that for all collections $\mathcal{T}$, a maximum compatible tree of $\mathcal{T}$ includes at least as many leaves as a maximum agreement subtree of $\mathcal{T}$. Also note that the tree $MAST(\mathcal{T})$, and similarly $MCT(\mathcal{T})$, may not be unique, so then the notations $MAST(\mathcal{T})$ and $MCT(\mathcal{T})$ denote any single tree among the possible trees. However, the number of leaves in a maximum agreement subtree, respectively maximum compatible tree, of a collection $\mathcal{T}$ is unique and denoted $\#MAST(\mathcal{T})$, respectively $\#MCT(\mathcal{T})$. Note also that the MCT problem is equivalent to MAST when input trees are binary. However, as stated before, MCT is of particular interest when considering evolutionary trees that are not binary as input.

The particular case where $\#MAST(\mathcal{T}) = \#L$ arises whenever all trees in $\mathcal{T}$ are isomorphic. Similarly, $\#MCT(\mathcal{T}) = \#L$ occurs whenever the collection $\mathcal{T}$ is compatible.

From now on, by default, leaves are denoted by ℓ, rooted trees are denoted by T and unrooted trees are denoted by U. In a similar way, collections of rooted trees, respectively unrooted trees, are denoted $\mathcal{T}$, respectively $\mathcal{U}$.

2.2 Other formalisms to describe trees

Rooted trees can be described in terms of rooted triples and fans:

Definition 6 (Triples and fans) *Let T be a rooted tree.*

For any three distinct leaves ℓ, ℓ', $\ell'' \in L(T)$, there are only three possible binary shapes for $T|\{\ell, \ell', \ell''\}$, denoted, respectively, $\ell|\ell'\ell''$, $\ell'|\ell\ell''$ or $\ell''|\ell\ell'$, depending on their innermost grouping, respectively, $\{\ell', \ell''\}$, $\{\ell, \ell''\}$ or $\{\ell, \ell'\}$. These trees are called rooted triples *(or* resolved triples*). Alternatively $T|\{\ell, \ell', \ell''\}$ can be a* fan *(also called* unresolved *triple), which is the tree in where the root is directly connected to the three leaves. The fan on the leaves $\{\ell, \ell', \ell''\}$ is denoted (ℓ, ℓ', ℓ'').*

We define $rt(T)$, respectively $f(T)$, to be the set of rooted triples, respectively fans, induced by the leaves of a tree T. We extend these definitions to define

rooted triples and fans of a collection $\mathcal{T}$ of rooted trees: $rt(\mathcal{T}) := \bigcap_{T_i \in \mathcal{T}} rt(T_i)$ *and* $f(\mathcal{T}) := \bigcap_{T_i \in \mathcal{T}} f(T_i)$.

For example, in Fig. 1:

- $rt(T_2) = \{b|ad, c|ad, e|ad, e|ab, e|ac, e|bd, e|cd, e|bc\}$,
- $f(T_2) = \{(a, b, c), (b, c, d)\}$,
- $rt(\mathcal{T}) = \{e|ab, e|ac, e|bc\}$,
- $f(T_1)$ is empty, hence also is $f(\mathcal{T})$.

Given a rooted tree T, note that the *lca* relationships enable us to know which rooted triple or fan is induced by T. Hence, given $\ell, \ell', \ell'' \in L(T)$,

- $\ell|\ell'\ell'' \in rt(T)$ iff $lca_T(\ell', \ell'')$ is a node strictly below $lca_T(\ell, \ell', \ell'')$ and
- $(\ell, \ell', \ell'') \in f(T)$ iff $lca_T(\ell, \ell') = lca_T(\ell, \ell'') = lca_T(\ell', \ell'')$.

Note also that a fan is compatible with any rooted triple having the same set of leaves. However, two different rooted triples on the same set of leaves are incompatible.

We now recall the translation in terms of triples and fans of the relations between trees defined in Sect. 2.1:

Lemma 1 *Let $\mathcal{T}$ be a collection of rooted trees with identical leaf set L and let T, T' be two rooted trees.*

(i) T is an agreement subtree of $\mathcal{T}$ iff $rt(T) \subseteq rt(\mathcal{T})$ and $f(T) \subseteq f(\mathcal{T})$.

(ii) T is isomorphic to T' iff $rt(T) = rt(T')$ and $f(T) = f(T')$.

(iii) T refines T' iff $rt(T') \subseteq rt(T)$ and $L(T) = L(T')$.

(iv) T is compatible with $\mathcal{T}$ iff $L(T) \subseteq L$ and $\forall T_i \in \mathcal{T}, rt(T_i|L(T)) \subseteq rt(T)$.

Proof. (i) is [15, Lem. 6.6]. (ii) derives from (i) and is the rooted equivalent of [3, Thm. 2]. By [21, Thm. 1], $T|L(T')$ refines T' iff $rt(T') \subseteq rt(T)$ and $L(T') \subseteq L(T)$. From that we deduce (iii). Let us now prove (iv).

T is compatible with $\mathcal{T}$ means that

$$\forall T_i \in \mathcal{T} \quad T \trianglerighteq T_i|L(T)$$

by (iii), this is equivalent to

$$\forall T_i \in \mathcal{T} \quad \big(L(T_i|L(T)) = L(T) \quad \text{and} \quad rt(T_i|L(T)) \subseteq rt(T)\big)$$

which is equivalent to

$$L(T) \subseteq L \quad \text{and} \quad \big(\forall T_i \in \mathcal{T} \quad rt(T_i|L(T)) \subseteq rt(T)\big)\,.$$

□

Lemma 1-(i) means that T is an agreement subtree of $\mathcal{T}$ iff any restriction of T to a set of 3 leaves is an agreement subtree of $\mathcal{T}$. Similarly, Lem. 1-(iv), means that T is a tree compatible with $\mathcal{T}$ iff any restriction of T to a 3-leaf set is a tree compatible with $\mathcal{T}$.

Definition 7 (Hard and soft conflicts) *Let T, T' be two rooted trees.*

- *A* hard conflict *between T and T' is a 3-leaf set $\{\ell, \ell', \ell''\} \subseteq L(T) \cap L(T')$ such that $\ell|\ell'\ell'' \in rt(T)$ and $\ell'|\ell\ell'' \in rt(T')$.*
- *A* soft conflict *between T and T' is a 3-leaf set $\{\ell, \ell', \ell''\} \subseteq L(T) \cap L(T')$ such that $\ell|\ell'\ell'' \in rt(T)$ and $(\ell, \ell', \ell'') \in f(T')$.*

Let $\mathcal{T}$ be a collection of rooted trees. A hard, respectively soft, conflict in $\mathcal{T}$ is a hard, respectively soft, conflict between two trees of $\mathcal{T}$.

For example, in Fig. 1:

- T_1 and T_2 have a hard conflict on $\{a, c, d\}$ (since $d|ac \in rt(T_1)$, while $c|ad \in rt(T_2)$) and,
- T_1 and T_2 have a soft conflict on $\{a, b, c\}$ (since $c|ab \in rt(T_1)$, while $(a, b, c) \in f(T_2)$).

The previous definitions together with Lem. 1 have the following straightforward consequences:

Proposition 1 ([22, 15, 16, 8]) *Let $\mathcal{T}$ be a collection of rooted trees on the same leaf set.*

(i) $\{\ell, \ell', \ell''\}$ is a hard or soft conflict in $\mathcal{T}$ if and only if there is no agreement subtree T of $\mathcal{T}$ s.t. $\{\ell, \ell', \ell''\} \subseteq L(T)$.

(ii) $\{\ell, \ell', \ell''\}$ is a hard conflict in $\mathcal{T}$ if and only if there is no compatible tree T with $\mathcal{T}$ s.t. $\{\ell, \ell', \ell''\} \subseteq L(T)$.

Another well-known alternative description of rooted trees, is as a set of clusters (see *e.g.* [23]):

Definition 8 (Clusters) *Let T be a rooted tree and v be an internal node of T, then the set of leaves $L(v)$ is called the* cluster *of T induced by v. A cluster is also commonly called a* clade *in phylogenetics. Let $\mathcal{C}l(T)$ denote the set of clusters of a tree T, defined as the clusters induced by all internal nodes of T.*

For example, in Fig. 1:

- $\mathcal{C}l(T_1) = \{\{a, b\}, \{d, e\}, \{a, b, c\}, \{a, b, c, d, e\}\}$ and
- $\mathcal{C}l(T_2) = \{\{a, d\}, \{a, b, c, d\}, \{a, b, c, d, e\}\}$.

Here, we do not need to consider single leaves as clusters, unlike the practise in [24, 23].

When dealing with unrooted trees, *splits* or *bipartitions* play the role that clusters play for rooted trees. In the unrooted context, there is a well-known characterization of a minimum refinement of a compatible collection of trees [25, 20, 23]. Here, we require a version of this result applying to rooted trees.

Lemma 2 (Minimum refinement) *Let $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ be a collection of rooted trees on a leaf set L and let T be a rooted tree on L. The three following assertions are equivalent:*

(i) T is a minimum refinement of $\mathcal{T}$ (that is any tree T' refining $\mathcal{T}$ also refines T),

(ii) $\mathcal{C}l(T) = \mathcal{C}l(T_1) \cup \mathcal{C}l(T_2) \cup \ldots \cup \mathcal{C}l(T_k)$,

(iii) $rt(T) = rt(T_1) \cup rt(T_2) \cup \ldots \cup rt(T_k)$.

Moreover, if $\mathcal{T}$ is compatible then there exists a minimum refinement of $\mathcal{T}$.

Proof. See Appendix 6. □

2.3 Rooting and unrooting collections of trees

Definition 9 *Given an unrooted tree U and $\ell \in L(U)$, $U^{-\ell}$ is the rooted tree on $L(U) - \{\ell\}$ obtained by rooting U at leaf ℓ and then removing ℓ and its incident edge. Let $\mathcal{U}$ be a collection of unrooted trees and ℓ a leaf common to all trees of $\mathcal{U}$, the collection of rooted trees $\{U^{-\ell} : U \in \mathcal{U}\}$ is denoted $\mathcal{U}^{-\ell}$.*

Conversely, given a rooted tree T and a leaf $\ell \notin L(T)$, define $T^{+\ell}$ as the unrooted tree on $L(T) \cup \{\ell\}$ obtained by grafting ℓ at the root of T by a new edge and unrooting the tree. Let $\mathcal{T}$ be a collection of rooted trees and a leaf ℓ not appearing in any tree of $\mathcal{T}$, the collection of unrooted trees $\{T^{+\ell} : T \in \mathcal{T}\}$ is denoted $\mathcal{T}^{+\ell}$.

For example, considering the tree U in Fig. 2 and leaf $f \in L(U)$, then tree U^{-f} is the tree T_2 in Fig. 1. Reciprocally, considering this tree T_2, then tree T_2^{+f} is tree U in Fig. 2.

Clearly, the ways of rooting and unrooting trees defined above are symmetric. More formally:

Lemma 3

(i) Let U be an unrooted tree and $\ell \in L(U)$. Then $U = \left(U^{-\ell}\right)^{+\ell}$.

(ii) Let T be a rooted tree and $\ell \notin L(T)$. Then $T = \left(T^{+\ell}\right)^{-\ell}$.

Isomorphism and refinement relations between trees are also conserved by rooting or unrooting the trees in the same way:

Lemma 4
Let U_1, U_2 be two unrooted trees s.t. $L(U_1) \subseteq L(U_2)$ and let ℓ be a leaf appearing in U_1 (and U_2):

$$U_1^{-\ell} \sqsubseteq U_2^{-\ell} \iff U_1 \sqsubseteq U_2\,, \tag{1}$$
$$U_1^{-\ell} \trianglerighteq U_2^{-\ell} \iff U_1 \trianglerighteq U_2\,. \tag{2}$$

Let T_1, T_2 be two rooted trees s.t. $L(T_1) \subseteq L(T_2)$ and let ℓ be a leaf not appearing in T_2 (and T_1):

$$T_1 \sqsubseteq T_2 \iff T_1^{+\ell} \sqsubseteq T_2^{+\ell}\,, \tag{3}$$
$$T_1 \trianglerighteq T_2 \iff T_1^{+\ell} \trianglerighteq T_2^{+\ell}\,. \tag{4}$$

Proof. (1) results from the fact that the tree modifications to go from U_1, U_2 to $U_1^{-\ell}, U_2^{-\ell}$, or reciprocally, preserve the isomorphism between U_1 and $U_2|L(U_1)$.

Concerning (2), $U_1^{-\ell} \trianglerighteq U_2^{-\ell}$ means that $U_2^{-\ell}$ can be obtained from $U_1^{-\ell}$ by contracting some edges. Contracting these same edges in the unrooted tree U_1 leads to U_2, and thus $U_1 \trianglerighteq U_2$. The converse holds for the same reason.

(3) and (4) follow immediately from (1) and (2) by Lem. 3-(i). □

For collections of trees, the two previous lemmas have the following consequences that will play a role for solving MAST and MCT on unrooted trees:

Lemma 5

(i) Let $\mathcal{T}$ be a collection of rooted trees with identical leaf set L, let ℓ be a leaf not in L and let T be an agreement subtree of $\mathcal{T}$, respectively a tree compatible with $\mathcal{T}$. Then, $T^{+\ell}$ is an agreement subtree of $\mathcal{T}^{+\ell}$, respectively a tree compatible with $\mathcal{T}^{+\ell}$.

(ii) Let $\mathcal{U}$ be a collection of unrooted trees with identical leaf set L, let U be an agreement subtree of $\mathcal{U}$, respectively a tree compatible with $\mathcal{U}$, and let $\ell \in L(U)$. Then, $U^{-\ell}$ is an agreement subtree of $\mathcal{U}^{-\ell}$, respectively a tree compatible with $\mathcal{U}^{-\ell}$.

Proof. A direct consequence of Lem. 3 and 4. □

This induces a relation between the sizes of *maximum* agreement subtrees, respectively *maximum* compatible trees, of collections of unrooted trees and corresponding collections of rooted trees:

Lemma 6 *Let $\mathcal{U}$ be a collection of unrooted trees with identical leaf set L, and let $\ell \in L$:*

(i)
$$\#MAST(\mathcal{U}) \geq \#MAST(\mathcal{U}^{-\ell}) + 1 \tag{5}$$
and equality holds iff ℓ appears in some maximum agreement subtree of $\mathcal{U}$.

(ii)

$$\#MCT(\mathcal{U}) \geq \#MCT(\mathcal{U}^{-\ell}) + 1 \tag{6}$$

and equality holds iff ℓ appears in some maximum compatible tree of $\mathcal{U}$.

Proof. (i) Let $M := \#MAST(\mathcal{U})$ and $M_\ell := \#MAST(\mathcal{U}^{-\ell})$. Let $T := MAST(\mathcal{U}^{-\ell})$. By Lem. 5-(i), the unrooted tree $T^{+\ell}$ is an agreement subtree of the collection $\left(\mathcal{U}^{-\ell}\right)^{+\ell}$ which is equal to $\mathcal{U}$ by Lem. 3. Hence, we have $M \geq \#T^{+\ell} = M_\ell + 1$, *i.e.* (5).

Suppose ℓ appears in no maximum agreement supertree of $\mathcal{U}$. In this case $T^{+\ell}$ cannot be a maximum agreement subtree of $\mathcal{U}$, and thus we have $M > \#T^{+\ell} = M_\ell + 1$: inequality (5) is strict.

Conversely, suppose ℓ appears in some maximum agreement subtree U of $\mathcal{U}$. By Lem. 5-(ii), the rooted tree $U^{-\ell}$ is an agreement subtree of $\mathcal{U}^{-\ell}$. Hence, we have $M_\ell \geq \#U^{-\ell} = M - 1$. Together with (5), this yields $M = M_\ell + 1$: inequality (5) is an equality.

The proof of (ii) is similar. □

In the next section, we describe linear time algorithms that will be used as subroutines in efficient FPT algorithms (see Sect. 4) for MAST and MCT problems.

3 Linear time algorithms for finding a conflict or checking isomorphism, respectively compatibility of trees

Prop. 1 shows that the identification of conflicts between two input trees is essential to solve MAST and MCT. This suggests extending the algorithms of [19], respectively [20] to identify a conflict, in the case of non-isomorphism, respectively incompatibility. Identifying such conflicts is the basis of an approximation algorithm [3] and of an FPT algorithm [16] for MAST. Also, [8] improve ideas of [3] to propose a conflict-based approximation algorithm for MCT.

Concerning the running time, [16] use a subroutine to check the isomorphism of two rooted trees or otherwise identify a conflict, that runs in $O(n \log n)$ time. [8] describe data structures that enable in time $O(n^2)$ to check the compatibility of two rooted trees or otherwise identify a conflict. In this section, we provide algorithms with better running time than the ones cited above:

- an $O(n)$ time algorithm to check that two rooted trees are isomorphic or otherwise identify three leaves on which the trees conflict;
- an $O(n)$ time algorithm to check that two rooted trees are compatible or otherwise identify three leaves on which the trees conflict. Moreover, in case of compatibility, the algorithm actually returns a certificate, *i.e.* a tree refining the input trees. This certificate is minimum (see Thm. 2).

In each case, it is outlined how linear time algorithms follow also for unrooted trees (Sect. 3.3) and for collections of more than two trees.

3.1 The Check-Isomorphism-or-Find-Conflict algorithm

Let $\mathcal{T} = \{T_1, T_2\}$ be a collection of two rooted trees with identical leaf set L of cardinality n. We detail here an algorithm called CHECK-ISOMORPHISM-OR-FIND-CONFLICT($\mathcal{T}$) that checks whether T_1 and T_2 are isomorphic or alternatively identifies a hard or soft conflict. This algorithm is obtained by modification of the linear time tree isomorphism algorithm proposed by [19] for leaf-labelled trees and extended by [20] to more general trees. Note that the algorithm of [19, 20] does not find a conflict when the input trees are not isomorphic, but we show here that it can be modified to achieve this goal while preserving linear time complexity.

The algorithm of [19] implicitly relies on nodes of a tree that have only leaves as children. Such a node is called a *cherry*.

Lemma 7 ([19]) *Let T_1, T_2 be two isomorphic trees and let v_1 be a cherry in T_1. Then, there is a cherry $v_2 \in T_2$ s.t. $L(v_1) = L(v_2)$.*

In case of non-isomorphism, the following result states how a conflict can be identified:

Lemma 8 *Let v_1 be a cherry in a tree T_1, let $\ell \in L(v_1)$ and v_2 be the parent node of ℓ in a tree T_2. There is a conflict between T_1 and T_2 involving ℓ whenever $L(v_1) \neq L(v_2)$ or v_2 is not a cherry. Moreover, knowing v_1 and ℓ, such a conflict can be identified in $O(n)$ time.*

Proof. If $L(v_1) \neq L(v_2)$ then there are two cases:

(i) $\#\big(L(v_1) \cap L(v_2)\big) = 1$, *i.e.* ℓ is the only common leaf of v_1 and v_2. Then, $L(v_1) - L(v_2) \neq \emptyset$ and $L(v_2) - L(v_1) \neq \emptyset$. Picking any $\ell' \in L(v_1) - L(v_2)$ and any $\ell'' \in L(v_2) - L(v_1)$, the set $\{\ell, \ell', \ell''\}$ is a hard conflict between T_1 and T_2 since $\ell''|\ell'\ell \in rt(T_1)$ while $\ell'|\ell''\ell \in rt(T_2)$.

(ii) $\#\big(L(v_1) \cap L(v_2)\big) > 1$. Let $\ell' \neq \ell, \ell' \in L(v_1) \cap L(v_2)$. Since $L(v_1) \neq L(v_2)$, we can pick a leaf ℓ'' in the symmetrical difference $L(v_1) \ominus L(v_2)$. Then $\{\ell, \ell', \ell''\}$ is a conflict. More precisely, if $\ell'' \in L(v_1) - L(v_2)$ then $\{\ell, \ell', \ell''\}$ is a soft conflict because $(\ell, \ell', \ell'') \in f(T_1)$ while $\ell''|\ell\ell' \in rt(T_2)$. Otherwise, $\ell'' \in L(v_2) - L(v_1)$ and the conflict is soft if $lca_{T_2}(\ell', \ell'') = v_2$ (because $(\ell, \ell', \ell'') \in f(T_2)$ while $\ell''|\ell\ell' \in rt(T_1)$), and hard otherwise (because $\ell''|\ell\ell' \in rt(T_1)$ while $\ell|\ell'\ell'' \in rt(T_2)$).

Now consider the case where v_2 is not a cherry and assume also that $L(v_1) = L(v_2)$ (otherwise the first part of the proof applies). Since v_2 is not a cherry, it has a non-leaf child c. Let ℓ', ℓ'' be any two leaves in $L(c)$, then $\{\ell, \ell', \ell''\}$ is a soft conflict because $\ell|\ell'\ell'' \in rt(T_2)$, while $(\ell, \ell', \ell'') \in f(T_1)$ (since $L(c) \subseteq L(v_2) = L(v_1)$ and v_1 is a cherry). Moreover, if T_1, T_2 conflict, a simple linear

time search of the subtrees rooted at v_1 and v_2 is sufficient to identify three leaves ℓ, ℓ', ℓ'' involved in a conflict, according to the guidelines given above. □

Sketch of the algorithm.
Lem. 7 suggests examining cherries of a tree in a bottom-up process. Given a cherry $v_1 \in T_1$, choose $\ell \in L(v_1)$ and let v_2 be the parent node of leaf ℓ in T_2. If v_2 is not a cherry or if $L(v_1) \neq L(v_2)$ then Lem. 8 states that three leaves ℓ, ℓ', ℓ'' on which T_1 and T_2 conflict can be identified in $O(n)$ time. If, however, v_2 is a cherry and $L(v_1) = L(v_2)$, then the cherries can be *eaten* in both trees. This means that leaves hanging from v_1 and v_2 are deleted, turning v_1 and v_2 into leaves to which a same label is assigned (the label is arbitrarily chosen in $L(v_1) = L(v_2)$). Note that this modification of the tree can in turn transform the parent node of v_1, respectively v_2, in a cherry node.

The processing of cherries in T_1 is iterated until the trees are both reduced to a single leaf with the same label (then we know that the input trees are isomorphic) or until a conflict is identified (that is also present in the original trees). For this algorithm to be used as a subroutine of other algorithms in the paper, we assume that the tree T_1 is returned in the case where isomorphism is detected, and otherwise that the three leaves of the identified conflict are returned.

Theorem 1 *Let T_1, T_2 be two rooted trees with identical leaf set L of cardinality n. In time $O(n)$ algorithm* Check-Isomorphism-or-Find-Conflict($\{T_1, T_2\}$) *either concludes that the trees are isomorphic whenever this is the case, or otherwise identifies a hard or soft conflict between T_1, T_2.*

Proof. Correctness stems from Lemmas 7 and 8, and from the fact that only identical parts of the trees are eaten. In that case, assigning to v_1 and v_2 the same label chosen in $L(v_1) = L(v_2)$ guarantees that the modified trees will be isomorphic iff the original trees are isomorphic. Moreover, if $\{\ell, \ell', \ell''\}$ is a conflict between the modified trees then $\{\ell, \ell', \ell''\}$ is also a conflict in the original trees.

Concerning the running time, computing and maintaining the list of cherries in T_1 costs $O(n)$ time globally. Given a cherry $v_1 \in T_1$, finding the corresponding node $v_2 \in T_2$ is $O(1)$. Eating v_1 and v_2 costs a time proportional to the number of their children, hence $O(n)$ amortized time over the whole process. When non-isomorphism is detected, identifying a conflict requires $O(n)$ time (cf Lem. 8). □

Consider now the case of a collection $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ of k trees on n leaves. The problem is still solvable in linear time $O(kn)$: run the above-stated algorithm successively on all pairs (T_1, T_i), where $1 < i \leq k$, until a conflict is found (and then returned) or all trees are processed (and then the tree T_1 is returned).

3.2 The Find-Refinement-or-Conflict algorithm

Let $\mathcal{T} = \{T_1, T_2\}$ be a collection of two input trees with identical leaf set L of cardinality n. We detail here an $O(n)$ algorithm, called FIND-REFINEMENT-OR-CONFLICT($\{T_1, T_2\}$), that either identifies a hard conflict between trees of $\mathcal{T}$ or returns a minimum refinement of $\mathcal{T}$.

Similarly to the previous section, this algorithm could be obtained by direct modification of an existing algorithm that decides whether two trees are compatible [19, 8]. However, the algorithm of [8] maintains $O(n^2)$-sized data structures (to identify conflicts) when we aim for linear time. Moreover, the linear time algorithm of [19] performs several passes over the trees (first refining T_1 according to T_2, then T_2 according to T_1, and finally doing an isomorphism check of the resulting trees). Instead, we present a somewhat different linear time algorithm performing a single pass over the trees and identifying a conflict in case of non-compatibility.

Since soft conflicts are allowed by compatibility, a cherry $v_1 \in T_1$ does not always correspond to a cherry $v_2 \in T_2$ with the same leaf set. Given v_2 the parent in T_2 of a leaf $\ell \in L(v_1)$, cases where $L(v_1) \subseteq L(v_2)$ or $L(v_2) \subseteq L(v_1)$ are now allowed. Moreover, v_2 is no longer required to be a cherry[1]. We use the following result, which plays for compatibility the same role as Lem. 7 plays for isomorphism:

Lemma 9 *Let $\{T_1, T_2\}$ be a compatible collection, v_1 a cherry in T_1 and $v_2 := lca_{T_2}(L(v_1))$. Then there is a subset C of (at least two) children of v_2 such that $L(v_1) = L(C)$.*

Proof. Let C be the set of children c of v_2 s.t. $L(c) \cap L(v_1) \neq \emptyset$.

First, let ℓ be any leaf in $L(v_1)$. Because v_2 is a proper ancestror of ℓ, there is a child c of v_2 on the path from ℓ to v_2. Hence, $\ell \in L(c) \cap L(v_1)$, thus $c \in C$ and $\ell \in L(c) \subseteq L(C)$. It follows that $L(v_1) \subseteq L(C)$. Now, if C was to contain a single child of v_2, then this child would be a common ancestor of $L(v_1)$ and thus v_2 would not be the *least* common ancestor of $L(v_1)$. Thus, we have $\#C \geq 2$.

Finally, suppose that $L(v_1)$ is a proper subset of $L(C)$. Then, there is $c \in C$ s.t. $\exists\, \ell \in L(c) - L(v_1)$. Consider $\ell' \in L(c) \cap L(v_1)$, $c' \in C$ s.t. $c' \neq c$ and $\ell'' \in L(c') \cap L(v_1)$: $\ell''|\ell\ell' \in rt(T_2)$ while $\ell|\ell'\ell'' \in rt(T_1)$. Hence, by Prop. 1-(ii), $\{T_1, T_2\}$ is not compatible which is a contradiction. Therefore, $L(v_1) = L(C)$. □

Sketch of algorithm.
Let $\mathcal{T} = \{T_1, T_2\}$ be a collection of two input trees with identical leaf set L. The algorithm gradually prunes parts of T_1 and T_2, repeatedly eating cherries in T_1 and corresponding parts in T_2. This process ends when the trees are reduced to a single leaf or a (hard) conflict is found. At each step, a cherry v_1 in T_1 is chosen, and the corresponding node $v_2 := lca_{T_2}(L(v_1))$ identified, as well as the subset C of v_2's children c such that $L(c) \cap L(v_1) \neq \emptyset$. Then either:

[1] *e.g.* using parenthetical notation, $T_1 = (\ell_1, \ell_2, (\ell_3, \ell_4))$ and $T_2 = ((\ell_1, \ell_2), \ell_3, \ell_4)$, admit $((\ell_1, \ell_2), (\ell_3, \ell_4))$ as common refinement.

(i) $L(v_1) = L(C) = L(v_2)$ (*i.e.* C is the set of all children of v_2), then subtrees $S(v_1)$ and $S(v_2)$ are pruned;

(ii) $L(v_1) = L(C) \subset L(v_2)$, then subtree $S(v_1)$ and the set $\mathcal{S}$ of subtrees $S(c)$, with $c \in C$, are pruned;

(iii) $L(v_1) \neq L(C)$, then a conflict involving a leaf $\ell \in L(C) - L(v_1)$ and two leaves $\ell', \ell'' \in L(v_1)$ is identified (see proof of Lem. 9) and returned.

Pruning a subtree $S(v_1), S(v_2)$ or a set $\mathcal{S}$ of subtrees means deleting all its nodes and replacing it (the subtree or the whole set $\mathcal{S}$ under v_2) by a single leaf. Both in T_1 and T_2 this new leaf is given a new label, say ℓ^* (which changes at every step).

To build the refinement of T_1 and T_2, a forest of trees is maintained, initially containing a leaf-tree for each leaf in L. Unlike T_1 and T_2, trees of the forest have labels on their internal nodes. At each step of the algorithm, some trees in the forest are assembled to mimic subtrees of T_1 and T_2 that are pruned (cases (i) and (ii)). Trees to assemble are identified thanks to the label of their root, which is found at a leaf in the part of T_1 and T_2 to reproduce. Every such assembly adds a new cluster of T_1 or T_2 in the forest. The clusters formed by trees in the forest are thus all clusters identified by the algorithm in the two input trees. More precisely, each tree of the forest is a minimum refinement of a subtree in T_1 and the corresponding subtree in T_2. As the assembling process goes on, the number of trees in the forest decreases and each tree contains more and more leaves, *i.e.* is a refinement of a larger part of T_1 and T_2. When only one tree T remains in the forest, it contains all leaves of T_1 and T_2 and is a minimum refinement of the entire T_1 and T_2 trees.

The pseudo-code FIND-REFINEMENT-OR-CONFLICT($\{T_1, T_2\}$) (see Algorithm 1) details this process that either identifies a hard conflict between T_1 and T_2, or returns a tree minimally refining them.

Algorithm 1: FIND-REFINEMENT-OR-CONFLICT($\{T_1, T_2\}$)

Input: Two rooted trees T_1, T_2 on the same leaf set L.
Result: A hard conflict between T_1 and T_2, or a tree T on L minimally refining T_1 and T_2.

$F \leftarrow L$ /* *F is a forest of rooted trees (initially leaf-trees)* */
Let Ch be the list of cherries in T_1
1 **while** $Ch \neq \emptyset$ **do**
  Choose v_1 in Ch
2 Let ℓ^* be a new label, v a new node in F labelled ℓ^* and let $v_2 = lca_{T_2}(L(v_1))$
  $C \leftarrow \emptyset$ /* *C are the subtrees of v_2 to prune because of v_1* */
3 $P \leftarrow L(v_1)$ /* *P are leaves leading to identify new subtrees to prune* */
4 **foreach** *leaf* $\ell \in L(v_1)$ **do**
5 **if** $\ell \in P$ **then**
6 Let c be the child of v_2 s.t. $\ell \in L(c)$
7 **foreach** *leaf* $\ell'' \in L(c)$ **do**
8 **if** $\ell'' \in P$ **then** $P \leftarrow P - \{\ell''\}$
  else
  /* *case* (iii) *in the text* */
9 Let ℓ' be a leaf in $L(v_1) - L(c)$
  Let T, respectively T', respectively T'', be the tree of F whose root is labelled by ℓ, respectively ℓ', respectively ℓ''
10 $\ell \leftarrow$ a leaf of T, $\ell' \leftarrow$ a leaf of T', $\ell'' \leftarrow$ a leaf of T''
  return $\{\ell, \ell', \ell''\}$ /* *conflict on* $\{\ell, \ell', \ell''\}$ */
11 Add to F a tree T_c that is a copy of $S(c)$ then connect T_c to other trees in F by merging respectively each of its leaves with the root of the tree having the same label
12 Add an edge in F making T_c a new child subtree of v.
  Add c to C
13 Replace $S(v_1)$ in T_1 by a new leaf labelled ℓ^* and add its parent to Ch if it becomes a cherry
14 **foreach** *node* $c \in C$ **do** Remove the subtree $S(c)$ from T_2
15 **if** *v_2 has become a leaf in T_2* **then** /* *case* (i) *in the text* */ Label v_2 by ℓ^*
16 **else** /* *case* (ii) *in the text* */ Graft a new leaf labelled ℓ^* by a new edge under v_2
return *a tree T in F /* in fact, there is only one tree left in F at that stage* */*

Theorem 2 *Let T_1, T_2 be two rooted trees with identical leaf sets, in time $O(n)$ algorithm* FIND-REFINEMENT-OR-CONFLICT($\{T_1, T_2\}$) *either returns a tree T*

minimally refining T_1 and T_2 if such a tree exists, or otherwise returns a hard conflict between T_1 and T_2.

Proof.
Correctness. (i) First consider the case where the algorithm returns a tree T. Note that the progressive eating of T_1 (line 13) guarantees that each internal node of the original T_1 becomes at some step a cherry v_1 in the modified T_1. Moreover, during the processing of v_1 at some iteration of loop 1, the updates of F (lines 2,11,12) guarantee that F contains a tree T_v rooted at v s.t. $L(T_v)$ is the cluster of the original T_1 induced by v_1 when the inner loop (initiated at line 4) ends. Hence, in particular, after processing the root of T_1, there is a tree in F with leaf set L. Note in passing that this is the only tree remaining in F at that point since the set of leaves of trees in F is always exactly L (this is true when F is created and after each update of F). Hence, returning the only tree left in F at the end of the algorithm gives a tree T s.t. $L(T) = L$.

F is initialized with trees of size one, each one being a leaf labelled by an element of L, *i.e.* containing no cluster. Then changes in the forest only consist in connecting some of its trees, which adds new clusters and never removes already formed clusters. The assembling of trees continues until they all have been connected into one tree T that hence contains all clusters formed in F during execution of the algorithm. We now show that $\mathcal{C}l(T) = \mathcal{C}l(T_1) \cup \mathcal{C}l(T_2)$, *i.e.* that all clusters of T_1 and T_2 are formed in F, and only those.

- *Clusters of T_1:* let C_1 be a cluster of T_1 induced by an internal node v_1 of T_1. The gradual eating of cherries of T_1 guarantees that v_1 is the considered node at an iteration of loop 1. During this iteration, after the end of loop 4, F contains a new tree, say T_v, with root v s.t. $L(T_v) = L(v_1)$, *i.e.* C_1 is induced by a tree in F.

- *Clusters of T_2:* each cluster C_2 of $\mathcal{C}l(T_2)$ is either induced by the node c in T_2 considered on line 6 or induced by a node inside $S(c)$. In both cases, after the execution of line 11, a copy of C_2 has been added in a tree of F.

This shows that every cluster of T_1 and T_2 is formed in F at some step, *i.e.* is present in the tree T output by the algorithm. Moreover, new clusters are only formed in F due to changes done at line 11 and line 12. These changes in F respectively involve:

- creating in F a copy of the subtree $S(c)$ of T_2, whose leaves are merged with roots of trees previously in F having respectively the same label. Each such label either belongs to L or is a label ℓ^* corresponding to an internal node v in the original T_2. In the latter case, the tree of F with root labelled by ℓ^* has $L(v)$ as leaf-set. This guarantees that line 11 adds in F only clusters present in the original tree T_2;

- adding existing trees or newly formed trees as child subtrees of the node v of F, until it becomes the root of a tree having $L(v_1)$ as leaves. Thus, these executions of line 12 form in F a cluster of T_1.

Therefore, only clusters present in T_1 and T_2 are formed in F.

As a result, if the algorithm returns a tree T, this tree is s.t. $\mathcal{C}l(T) = \mathcal{C}l(T_1) \cup \mathcal{C}l(T_2)$. By Lem. 2, this implies that T is a minimum refinement of T_1 and T_2.

(ii) Finally, consider the case where the algorithm returns a conflict. The algorithm returns a conflict ℓ, ℓ', ℓ'' whenever a leaf $\ell'' \notin P$, *i.e.* $\ell'' \notin L(v_1)$ is found in the subtree rooted at a child c of $v_2 := lca_{T_2}(L(v_1))$. But since $L(c)$ contains both a leaf in $\ell \in L(v_1)$ and leaf $\ell'' \notin L(v_1)$ then there is no subset C of v_2 children s.t. $L(C) = L(v_1)$. Hence, by Lem. 9 there is a conflict between T_1 and T_2 in their current state (recall that these trees are gradually reduced during the algorithm). Indeed, let $\ell' \in L(v_1) - L(c)$ (such a leaf exists by definition of v_2), then $\ell''|\ell\ell' \in rt(T_1)$ while $\ell'|\ell\ell'' \in rt(T_2)$. Now if $\{\ell, \ell', \ell''\}$ is a conflict between T_1 and T_2, the way trees are gradually reduced by the algorithm on lines 13-16 implies that there is a conflict in the original trees T_1 and T_2. Such a conflict is returned by the algorithm by replacing ℓ, respectively ℓ', ℓ'' by a leaf of the original subtree it represents (the tree of the forest to which ℓ, respectively ℓ', ℓ'', belongs is a refinement of a subtree of the original tree T_1 and of a subtree of the original tree T_2). Thus, if the algorithm returns a conflict, this is a conflict between the input trees T_1 and T_2.

Running time. The algorithm is traversing T_1, T_2 a constant number of times, spending a constant amount of time at each of the $O(n)$ nodes and edges. Nodes v_2 are identified in $O(n)$ amortized time by exploring a different subtree of T_2 each time (or using dynamic data structures proposed by [26]). The list of cherries in T_1 is maintained in $O(n)$ globally, sets of subtrees $S(c)$ corresponding to processed cherries of T_1 are identified and removed in $O(n)$ globally. See Appendix 5 for more details. □

We now generalize Thm. 2: given a collection $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ of k rooted trees with leaf set L of cardinality n, we want to compute a minimum refinement of $\mathcal{T}$ if $\mathcal{T}$ is compatible. Otherwise, a hard conflict between two trees of $\mathcal{T}$ has to be identified.

Note that we can not proceed exactly as done in the previous section for isomorphism, because the compatibility relation is *not transitive*. However, taking minimum refinement of (compatible) trees is an *associative* operation. Thus, we can iterate the process described above for two trees in the following way: choose two trees of $\mathcal{T}$ and replace them in $\mathcal{T}$ by their minimum refinement output by the process. Repeat that operation until either a conflict is found or until $\mathcal{T}$ has only one tree left, which is the minimum refinement of the initial collection. In the first hand, the running time is clearly $O(kn)$ since at most $k-1$ pairs of trees with n leaves are considered. On the other hand, Lem. 2-(iii) ensures that the set $\bigcup_{T_i \in \mathcal{T}} rt(T_i)$ is left unchanged after each iteration of the algorithm. Hence, if a hard conflict is returned then this hard conflict is present between two trees of the original collection.

3.3 Dealing with unrooted trees

Let $\mathcal{U}$ be a collection of *unrooted* trees with identical leaf set L and let $\rho \in L$. As suggested by [3, 8]:

- all (rooted) trees of the collection $\mathcal{U}^{-\rho}$ are isomorphic iff all (unrooted) tree of $\mathcal{U}$ are isomorphic and,
- the collection $\mathcal{U}^{-\rho}$ is compatible iff the collection $\mathcal{U}$ is compatible.

Moreover, if $\{\ell, \ell', \ell''\}$ is a hard or soft conflict, respectively a hard conflict, between two trees T_1, $T_2 \in \mathcal{U}^{-\rho}$, then the trees $T_1^{+\rho}$, $T_2^{+\rho}$ which both belong to $\mathcal{U}$ are such that $T_1^{+\rho}|\{\rho, \ell, \ell', \ell''\}$ and $T_2^{+\rho}|\{\rho, \ell, \ell', \ell''\}$ are not isomorphic, respectively $\left\{T_1^{+\rho}|\{\rho, \ell, \ell', \ell''\}, T_2^{+\rho}|\{\rho, \ell, \ell', \ell''\}\right\}$ is not compatible. Thus, using the algorithms presented in this section on $\mathcal{U}^{-\rho}$, it is possible to check in linear time whether all trees in $\mathcal{U}$ are isomorphic or compatible, and otherwise to identify a quartet of conflicting leaves.

4 Fixed-Parameter Tractability of MAST and MCT

The previous section considered the problem of deciding whether trees of an input collection conflict on the relative location of leaves, *i.e.* taxa. In most practical cases, the answer is positive and one can then aim at producing a consensus of the input trees by removing a minimum set of conflicting leaves, that is solving the MAST and MCT problems. The present section proposes exact algorithms to solve these problems. They use as subroutines the algorithms presented in the previous section.

The MAST and MCT problems are both NP-hard in general. However, different algorithms have been proposed for MAST with a running time that is exponential only on a given parameter, for instance the degree. [16] showed that a parameterized version of MAST is fixed-parameter tractable (FPT). More formally, a problem is FPT whenever it can be solved by an algorithm with $O(f(p)N^{\alpha})$ running time, where p is the parameter, N is the size of the input, α is a constant (independant of both p and N) and f is an arbitrary function, though usually exponential [16]. The interest in designing fixed-parameter algorithms is that for some practical instances, the value of the parameter is known to be small. Hence, the exponential term hidden in the function f is not penalizing that much the running time, which means that the problem is tractable for that kind of instances.

We first consider the fixed-parameter tractability of MAST and MCT on *rooted* trees. The parameterized version of MAST considered in [16, 17] is the following search problem:

Name: Parameterized Rooted Maximum Agreement SubTree (PRMAST)
Input: A collection $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ of k rooted trees with identical leaf set L of cardinality n.
Parameter: an integer $p \geq 0$.

Task: Find an agreement subtree T of $\mathcal{T}$ s.t. $\#T \geq n - p$, if such a tree exists.

Similarly, we define

Name: Parameterized Rooted Maximum Compatible Tree problem (PRMCT)
Input: A collection $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ of k rooted trees with identical leaf set L of cardinality n.
Parameter: an integer $p \geq 0$.
Task: Find a tree T compatible with $\mathcal{T}$ s.t. $\#T \geq n - p$, if such a tree exists.

On practical data, the value of p is likely to be reasonnably small. Indeed, source trees are now usually inferred from lenghty molecular sequences and through more and more accurate inference methods. Thus, trees inferred on a same set of taxa and given as input to PRMAST and PRMCT are unlikely to differ on the location of a large number of taxa. Moreover, confidence values enable to detect and collapse edges with unsufficient statistical support, which incidentally reduces the number of conflicts between the source trees.

Links between MAST, respectively MCT, and the Hitting Set problem [3, 15, 16, 17], respectively [8], have suggested two ways to solve the former: Section 4.1 describes a recursive method sketched in [16], whose complexity is slightly improved here. Then, Sect. 4.2 describes a method explicitly solving 3-Hitting Set as a subproblem [16, 17]. These two methods lead to FPT algorithms having complementary running times. Indeed, which approach is the fastest depends on the particular values taken by p and n. Both methods were originally introduced for solving PRMAST. They also apply to solve PRMCT as shown below. Moreover, Sect. 4.3 shows that the two methods can be extended to deal with unrooted trees.

4.1 Recursive FPT algorithms

Starting from the remark that if any two trees of a collection have a conflict, then the leaves involved in the conflict do not appear in any agreement subtree of the whole collection (Prop. 1-(i)), a recursive algorithm for finding an agreement subtree of an initial collection $\mathcal{T}$ of k rooted trees is the following [16]: identify a conflict $\{\ell, \ell', \ell''\}$ between two input trees, then try alternatively to remove one of ℓ, ℓ', ℓ'' from all trees of $\mathcal{T}$ and iterate on the three possible restricted collections until a collection of isomorphic trees is obtained or until p leaves have been removed. Hence, to solve PRMAST, we need a subroutine that checks that k trees are isomorphic or otherwise returns a hard or soft conflict between two of these trees. Algorithm Check-Isomorphism-or-Find-Conflict of Sect. 3.1 can be used for this purpose. We call Recursive-Mast the resulting recursive algorithm solving PRMAST.

To solve PRMCT, a similar algorithm can be used. It needs a subroutine that returns a minimum refinement of a collection of k trees when such a tree exists, or otherwise returns a hard conflict between two trees of the collection. The

linear-time algorithm FIND-REFINEMENT-OR-CONFLICT of Sect. 3.2 can be used for this purpose. We call RECURSIVE-MCT this algorithm solving PRMCT. Note that the only difference between RECURSIVE-MAST and RECURSIVE-MCT is that the former issues calls to CHECK-ISOMORPHISM-OR-FIND-CONFLICT, while the latter issues calls to FIND-REFINEMENT-OR-CONFLICT. The pseudo-code for RECURSIVE-MCT is given in Algorithm 2.

Algorithm 2: RECURSIVE-MCT$(\mathcal{T}, p)$

Input: A collection $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ of k rooted trees with identical leaf set L and an integer $p \geq 0$.

Result: A tree T compatible with $\mathcal{T}$ s.t. $\#T \geq \#L - p$ if such a tree exists or, otherwise, the empty tree $\emptyset$.

res $\leftarrow$ FIND-REFINEMENT-OR-CONFLICT$(\mathcal{T})$
if res *is a tree* T **then return** T */* this tree is compatible with* $\mathcal{T}$ **/*
/ Otherwise* res *is a set of three leaves that is a conflict in* $\mathcal{T}$ **/*
if $p > 0$ **then**
  foreach *leaf* $\ell \in$ res **do**
17 $T \leftarrow$RECURSIVE-MCT$\big(\mathcal{T}|(L - \{\ell\}), p - 1\big)$
18 **if** $T \neq \emptyset$ **then return** T
19 **return** $\emptyset$

Theorem 3

(i) Algorithm RECURSIVE-MAST *solves the* PRMAST *problem in* $O(3^p kn)$ *time.*

(ii) Algorithm RECURSIVE-MCT *solves the* PRMCT *problem in* $O(3^p kn)$ *time.*

Proof.
Correctness. We give the proof of (ii), the proof of (i) is similar.
We proceed by induction on p. If $p = 0$, then the result of RECURSIVE-MCT is the result of the algorithm FIND-REFINEMENT-OR-CONFLICT, which is correct.

If $p > 0$, and $\mathcal{T}$ is compatible, then FIND-REFINEMENT-OR-CONFLICT returns a minimum refinement of $\mathcal{T}$, *i.e.* a tree of size $\#L \geq \#L - p$ and compatible with $\mathcal{T}$, which is correct.

If $p > 0$ and $\mathcal{T}$ is not compatible, then the result res of the algorithm FIND-REFINEMENT-OR-CONFLICT is a hard conflict $\{\ell, \ell', \ell''\}$ between two trees of $\mathcal{T}$.

By Prop. 1-(ii), this implies that there is no tree compatible with $\mathcal{T}$ including all leaves of res. This means that there is a tree of size at least $\#L - p$ and compatible with $\mathcal{T}$ iff there is a tree of size at least $\#L - p$ and compatible with $\mathcal{T}|(L - \{\ell\})$, $\mathcal{T}|(L - \{\ell'\})$ or $\mathcal{T}|(L - \{\ell''\})$. On line 17 of the algorithm RECURSIVE-MCT, are issued recursive calls on the three collections, whose respective leaf sets are all of cardinality $\#L - 1$. By induction, each of these calls, taking as input a collection with leaf set $\tilde{L} \in \{L - \{\ell\}, L - \{\ell'\}, L - \{\ell''\}\}$,

returns a tree of size at least $\#\tilde{L} - (p-1) = \#L - p$ and compatible with the considered collection iff such a tree exists. There are two cases:

- the three recursive calls return an empty tree, which then means that there is no tree of size at least $\#L - p$ and compatible with $\mathcal{T}$ and justifies returning an empty tree on line 19;
- one of the three recursive calls returns a tree T of size at least $\#L - p$ and compatible with the considered collection. Returning this tree on line 18 as a solution to $(\mathcal{T}, p)$ is correct.

Running time. The recursive calls in the algorithms RECURSIVE-MCT form a search tree of depth at most p (p is decreased by one at each recursive call until it reaches 0) whose nodes have degree bounded by 3 (0 to 3 recursive calls are issued at each execution of the pseudo-code). Hence, the search tree explored contains at most $O(3^p)$ nodes. Moreover, by results of Sect. 3 each node is processed in $O(nk)$ because it requires a single call to FIND-REFINEMENT-OR-CONFLICT (restricting $\mathcal{T}$ to $L - \{\ell\}$ only costs $O(k)$). □

For PRMAST, this improves on the complexity of [16] by a $\log n$ factor. Concerning PRMCT, this is the first time that the problem is shown to be FPT. The burden of the complexity depends only on the level of disagreement between the input trees. When considering a collection of trees disagreeing on few species, we obtain an efficient algorithm, whatever the size, number and degree of the input trees.

4.2 Algorithms resorting explicitly to 3-HITTING SET

4.2.1 The HITTING SET problem

Let $\mathcal{C}$ be a collection of subsets of a ground set L. A *hitting set* of $\mathcal{C}$ is a set H s.t. for all $X \in \mathcal{C}$, $H \cap X$ is non-empty. The corresponding search problem is:

Name: HITTING SET

Input: A collection $\mathcal{C}$ of subsets of a finite ground set L and an integer $p \geq 0$.

Task: Find a hitting set H of $\mathcal{C}$ s.t. $\#H \leq p$, if such a set exists.

HITTING SET is an alternate formulation of SET COVER. It is NP-complete [27] and W[2]-complete for parameter p [28, Prop. 10].

The d-HITTING SET problem (where d is a fixed positive integer) is the restriction of HITTING SET to instances where sets in $\mathcal{C}$ have cardinality d. The d-HITTING SET problem is known to be fixed-parameter tractable, the best current algorithm running in $O(c^p + \#\mathcal{C})$ time where $c = d - 1 + O(d^{-1})$ [29].

The particular cases where $d = 2$ and $d = 3$ have been extensively considered. The 2-HITTING SET problem can be seen as an alternate formulation of the VERTEX COVER problem, for which there is very efficient FPT algorithms (see [30] and references therein). For 3-HITTING SET, [29] give an algorithm running in $O(2.27^p + \#\mathcal{C})$ time, which is more efficient than the algorithm for general d.

4.2.2 Reducing PRMCT and PRMAST to 3-Hitting Set

PRMCT and PRMAST can be solved by reduction to 3-Hitting Set:

Proposition 2 *Let $\mathcal{T}$ be a collection of rooted trees with identical leaf set L and let $H \subseteq L$.*

(i) Let $\mathcal{C}$ be the set of hard and soft *conflicts in $\mathcal{T}$: H is a hitting set of $\mathcal{C}$ iff there is an agreement subtree of $\mathcal{T}$ with leaf set $L - H$.*

(ii) Let $\mathcal{C}$ be the set of hard *conflicts in $\mathcal{T}$: H is a hitting set of $\mathcal{C}$ iff there is a tree compatible with $\mathcal{T}$ with leaf set $L - H$.*

Proof.

(i) If H is a hitting set of $\mathcal{C}$ then for every hard or soft conflict on three leaves in $\mathcal{T}$, at least one of these leaves is removed in $L - H$. Thus, all trees in $\mathcal{T}|(L-H)$ induce the same triple and fan sets, *i.e.* by Lem. 1-(ii) are isomorphic. These isomorphic trees on $L - H$ are agreement subtrees of $\mathcal{T}$. Conversely, let T be an agreement subtree of $\mathcal{T}$ with leaf set $L-H$. Let $X \in \mathcal{C}$, we have $X \subseteq L$ and $X \not\subseteq L - H$ by Prop. 1-(i). This implies $X \cap H \neq \emptyset$, hence H is a hitting set of $\mathcal{C}$.

(ii) If H hits all hard conflicts between trees of $\mathcal{T}$, then trees in $\mathcal{T}|(L - H)$ have no hard conflict. Thus, by Prop. 1-(ii), $\mathcal{T}|(L-H)$ is compatible, *i.e.* there is a tree with leaf set $L - H$ that is compatible with $\mathcal{T}$. Conversely, let T be a tree compatible with $\mathcal{T}$ having leaf set $L-H$, the same reasoning as the second part of the proof of (i) applies thanks to Prop. 1-(ii) to show that H is a hitting set of $\mathcal{C}$. □

Proposition 2-(i) is implicitly used in [17] and Prop. 2-(ii) in [8].

Theorem 4 PRMAST *and* PRMCT *problems can be solved in $O(2.27^p + kn^3)$ time.*

Proof. Knowing the rooted triples and fans induced by a tree can be done in $O(n^3)$ [24]. Hence, knowing the set $\mathcal{C}$ of hard and soft conflicts (respectively only hard conflicts) between the k input trees requires $O(kn^3)$ time. Using $\mathcal{C}$ as input, the FPT algorithm of [29] either gives a hitting set H of size at most p or concludes that no such set exists, in $O(2.27^p + \#\mathcal{C})$ time, where $\#\mathcal{C} = O(n^3)$. In the latter case, Prop. 2-(i), respectively Prop. 2-(ii), implies that there is no feasible solution to PRMAST, respectively PRMCT. This conclusion is reached in $O(2.27^p + kn^3)$ time.

To solve PRMAST, when the algorithm of [29] returns a hitting set H of $\mathcal{C}$, then choose any tree $T_i \in \mathcal{T}$ and return $T_i|(L - H)$. This tree, of at least $n - p$ size, is computed in time $O(n)$ and is a solution for PRMAST, as induced by Prop. 2-(i). To solve PRMCT from a hitting set H returned by the algorithm of [29], compute the collection $\mathcal{T}|(L - H)$. Prop. 2-(ii) guarantees that there is a tree compatible with $\mathcal{T}$ that has $L - H$ as leaf set, *i.e.* that has at least $n - p$

leaves. Algorithm Find-Refinement-or-Conflict (described at the end of Sect. 3.2) produces such a tree in $O(kn)$ time. Thus, the most computational intensive steps to obtain a solution to PRMCT are computing $\mathcal{C}$ and obtaining H, *i.e.* PRMCT can be solved in $O(2.27^p + kn^3)$ time. □

The fact that PRMAST can be solved in $O(2.27^p + kn^3)$ time is already stated in [17].

4.3 Unrooted trees

We now consider variant of the problem PRMAST, respectively PRMCT, that takes a collection of *unrooted* trees as input. We call PUMAST, respectively PUMCT, the resulting problem (U is for Unrooted). Suppose given an algorithm Find-Rooted-Tree that solves PRMAST, respectively PRMCT(*e.g.*, see Sect. 4.1 and Sect 4.2). For each collection $\mathcal{T}$ of rooted trees with identical leaf set L and for each integer $p \geq 0$, Find-Rooted-Tree$(\mathcal{T}, p)$ returns

- the empty tree if $\#MAST(\mathcal{T}) < \#L - p$, respectively if $\#MCT(\mathcal{T}) < \#L - p$,
- an agreement subtree of, respectively a tree compatible with, $\mathcal{T}$ of size at least $\#L - p$ otherwise.

Results of Sect. 2.3 suggest that PUMAST, respectively PUMCT, on a collection $\mathcal{U}$ of unrooted trees can be solved by n runs of Find-Rooted-Tree, one call for each $\mathcal{U}^{-\ell}$ ($\ell \in L$). This procedure would add an n factor to the complexity for the rooted case. However, Algorithm 3 below solves PUMAST, respectively PUMCT, with at most $p + 1$ calls to Find-Rooted-Tree.

Algorithm 3: Find-Unrooted-Tree$(\mathcal{U}, p)$

Input: A collection $\mathcal{U}$ of unrooted trees with identical leaf set L and an integer $p \geq 0$.
Result: a solution to PUMAST, respectively PUMCT, if one exists, the empty tree $\emptyset$ otherwise.
Choose arbitrarily $L' \subseteq L$ s.t. $\#L' = p + 1$
foreach $\ell \in L'$ **do**
  $T_\ell \leftarrow$ Find-Rooted-Tree$(\mathcal{U}^{-\ell}, p)$
  if $T_\ell \neq \emptyset$ **then return** $(T_\ell)^{+\ell}$
return $\emptyset$

We now prove the correctness of this algorithm:

Proposition 3 *Given a collection $\mathcal{U}$ of unrooted trees with identical leaf set L and an integer $p \geq 0$, algorithm* Find-Unrooted-Tree *returns an unrooted agreement subtree of $\mathcal{U}$, respectively an unrooted tree compatible with $\mathcal{U}$, of size at least $\#L - p$ iff such a tree exists.*

Proof. Assume that FIND-ROOTED-TREE solves the PRMAST. We show that FIND-UNROOTED-TREE solves PUMAST (the proof for PRMCT / PUMCT is similar).

Below, in *a)* we show that if $\#MAST(\mathcal{U}) < \#L - p$ then FIND-UNROOTED-TREE$(\mathcal{U}, p)$ returns the empty tree. In *b)* we show that if $\#MAST(\mathcal{U}) \geq \#L - p$ then FIND-UNROOTED-TREE$(\mathcal{U}, p)$ returns a tree of size at least $\#L - p$ that is an agreement subtree of $\mathcal{U}$.

Suppose $\#MAST(\mathcal{U}) < \#L - p$ Then, for any $\ell \in L$, by Lem. 6-(i), we have

$$\#MAST(\mathcal{U}^{-\ell}) + 1 \leq \#MAST(\mathcal{U}) < \#L - p\,,$$

i.e. $\#MAST(\mathcal{U}^{-\ell}) < (\#L-1)-p$. Since $\mathcal{U}^{-\ell}$ is a collection of rooted trees with leaf set $L - \{\ell\}$ of cardinality $\#L - 1$, the tree T_ℓ returned by FIND-ROOTED-TREE $(\mathcal{U}^{-\ell}, p)$ is the empty tree for all $\ell \in L'$. Hence FIND-UNROOTED-TREE returns the empty tree.

Suppose $\#MAST(\mathcal{U}) \geq \#L - p$ The size of L' guarantees that at least one leaf ℓ_{M} in L' is in a maximum agreement subtree of $\mathcal{U}$. By Lem. 6-(i) we have

$$\#MAST(\mathcal{U}^{-\ell_{\mathrm{M}}}) + 1 = \#MAST(\mathcal{U}) \geq \#L - p\,,$$

i.e. $\#MAST(\mathcal{U}^{-\ell_{\mathrm{M}}}) \geq (\#L - 1) - p$. Hence, $T_{\ell_{\mathrm{M}}}$ is an agreement subtree of $\mathcal{U}^{-\ell_{\mathrm{M}}}$ s.t. $\#T_{\ell_{\mathrm{M}}} \geq (\#L - 1) - p$. This guarantees that at least a call to FIND-ROOTED-TREE returns a non-empty tree, hence that FIND-UNROOTED-TREE$(\mathcal{U}, p)$ returns an non-empty tree. Let ℓ be the first leaf of L' s.t. $T_\ell \neq \emptyset$. Then, by Lems. 3 and 5-(i), $\big(T_\ell\big)^{+\ell}$ is an agreement subtree of $\big(\mathcal{U}^{-\ell}\big)^{+\ell} = \mathcal{U}$ and is of size $\#T_\ell + 1$. Thus $\#\big(T_\ell\big)^{+\ell} \geq \#L - p$. □

Using the algorithms of the previous section (for the rooted case) as subroutines in the algorithm FIND-UNROOTED-TREE, enables us to state a running time in which PUMAST and PUMCT can be solved.

Theorem 5 *Given a collection* $\mathcal{U} = \{U_1, U_2, \ldots, U_k\}$ *of* k *unrooted trees on an identical set of* n *leaves,* PUMAST *and* PUMCT *can be solved in time* $O\big((p+1) \times \min\{3^p kn, 2.27^p + kn^3\}\big)$.

Proof. Use the algorithm FIND-UNROOTED-TREE, where choosing L' requires $O(n)$ time and obtaining $\big(T_\ell\big)^{+\ell}$ from a tree T_ℓ requires $O(1)$. Then, the only other thing to do is to perform at most $p+1$ calls to FIND-ROOTED-TREE. Using the algorithms of Sect. 4.1 and 4.2 to instantiate the calls to FIND-ROOTED-TREE gives the claimed result by Thms. 3 and 4. □

4.4 Remarks for solving related problems

The computational problems considered above can be seen as generalizations of the well-known Tree Isomorphism and Tree Compatibility problems. The latter is of particular interest in phylogenetics and is deciding whether a collection of rooted input trees with identical leaf sets is compatible [25]. Tree Compatibility for rooted trees is identical to the restriction of PRMCT to instances for which $p = 0$. Algorithm Recursive-Mct solves this particular problem in linear time (Thm. 3 with $p = 0$). The Tree Compatibility problem for *unrooted* trees is identical to the PUMCT problem with $p = 0$ and is then solved in linear time also (Thm. 5). Linear algorithms are obtained in a similar way for the Tree Isomorphism problem on rooted or unrooted trees, which are particular cases of PRMAST and PUMAST respectively.

Hence, the general algorithms proposed in this paper allow to solve Tree Isomorphism and Tree Compatibility in the same running time as dedicated algorithms [19, 20].

References

[1] M. A. Steel and T. J. Warnow, "Kaikoura tree theorems: Computing the maximum agreement subtree," *Information Processing Letters*, vol. 48, no. 2, pp. 77–82, 1993.

[2] M. Farach, T. M. Przytycka, and M. Thorup, "On the agreement of many trees," *Information Processing Letters*, vol. 55, no. 6, pp. 297–301, 1995.

[3] A. Amir and D. Keselman, "Maximum agreement subtree in a set of evolutionary trees: metrics and efficient algorithm," *SIAM Journal on Computing*, vol. 26, no. 6, pp. 1656–1669, 1997.

[4] A. Gupta and N. Nishimura, "Finding largest subtrees and smallest supertrees," *Algorithmica*, vol. 21, no. 2, pp. 183–210, 1998.

[5] M.-Y. Kao, T. W. Lam, W.-K. Sung, and H.-F. Ting, "An even faster and more unifying algorithm for comparing trees via unbalanced bipartite matchings," *Journal of Algorithms*, vol. 40, no. 2, pp. 212–233, 2001.

[6] R. Cole, M. Farach-Colton, R. Hariharan, T. M. Przytycka, and M. Thorup, "An $O(n \log n)$ algorithm for the Maximum Agreement SubTree problem for binary trees," *SIAM Journal on Computing*, vol. 30, no. 5, pp. 1385–1404, 2001.

[7] D. Swofford, G. Olsen, P. Wadell, and D. Hillis, "Phylogenetic inference," in *Molecular systematics (2nd edition)*, D. Hillis, D. Moritz, and B. Mable, Eds. USA: Sunderland, 1996, pp. 407–514.

[8] G. Ganapathy and T. J. Warnow, "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), 2002, pp. 122–134.

[9] V. Berry and F. Nicolas, "Maximum agreement and compatible supertrees," in *Proceedings of CPM*, ser. LNCS, S. C. Sahinalp, S. Muthukrishnan, and U. Dogrusoz, Eds., vol. 3109, 2004, pp. 205–219.

[10] J. Jansson, J. H.-K. Ng, K. Sadakane, and W.-K. Sung, "Rooted maximum agreement supertrees," in *Proceedings of the 6th Latin American Symposium on Theoretical Informatics (LATIN)*, 2004, (in press).

[11] A. M. Hamel and M. A. Steel, "Finding a maximum compatible tree is NP-hard for sequences and trees," *Applied Mathematics Letters*, vol. 9, no. 2, pp. 55–59, 1996.

[12] G. Ganapathysaravanabavan and T. J. Warnow, "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., 2001, pp. 156–163.

[13] J. Hein, T. Jiang, L. Wang, and K. Zhang, "On the complexity of comparing evolutionary trees," *Discrete Applied Mathematics*, vol. 71, no. 1–3, pp. 153–169, 1996.

[14] M.-Y. Kao, T. W. Lam, W.-K. Sung, and H.-F. Ting, "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)*, 1999, pp. 438–449.

[15] D. Bryant, "Building trees, hunting for trees and comparing trees: theory and method in phylogenetic analysis," Ph.D. dissertation, University of Canterbury, Department of Mathemathics, 1997.

[16] R. G. Downey, M. R. Fellows, and U. Stege, "Computational tractability: The view from mars," *Bulletin of the European Association for Theoretical Computer Science*, vol. 69, pp. 73–97, 1999.

[17] J. Alber, J. Gramm, and R. Niedermeier, "Faster exact algorithms for hard problems: a parameterized point of view," *Discrete Mathematics*, vol. 229, no. 1–3, pp. 3–27, 2001.

[18] V. Berry, S. Guillemot, F. Nicolas, and C. Paul, "On the approximation of computing evolutionary trees," in *Proceedings of the 11th International Computing and Combinatorics Conference (COCOON'05)*, ser. LNCS, L. Wang, Ed., 2005.

[19] D. Gusfield, "Efficient algorithms for inferring evolutionary trees," *Networks*, vol. 21, pp. 19–28, 1991.

[20] T. J. Warnow, "Tree compatibility and inferring evolutionary history," *Journal of Algorithms*, vol. 16, no. 3, pp. 388–407, 1994.

[21] D. Bryant and M. A. Steel, "Extension operations on sets of leaf-labelled trees," *Advances in Applied Mathematics*, vol. 16, no. 4, pp. 425–453, 1995.

[22] G. F. Eastabrook, C. S. Johnson, and F. R. McMorris, "An algebraic analysis of cladistic characters," *Discrete Mathematics*, vol. 16, pp. 141–147, 1976.

[23] C. Semple and M. Steel, *Phylogenetics*, ser. Oxford Lecture Series in Mathematics and its Applications. Oxford University Press, 2003, vol. 24.

[24] D. Bryant and V. Berry, "A structured family of clustering and tree construction methods," *Advances in Applied Mathematics*, vol. 27, no. 4, pp. 705–732, 2001.

[25] G. F. Eastabrook and F. R. McMorris, "When is one estimate of evolutionary relationships a refinement of another?" *Journal of Mathematical Biology*, vol. 10, pp. 367–373, 1980.

[26] R. Cole and R. Hariharan, "Dynamic LCA queries on trees," in *Proceedings of 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'99)*, 1999, pp. 235 – 244.

[27] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. Cambridge, Massachusetts: M.I.T. Press, 2001.

[28] U. Feige, M. M. Halldórsson, and G. Kortsarz, "Approximating the domatic number," in *Proceedings of the 32nd Annual A.C.M. Symposium on Theory of Computing (STOC'00)*, 2000, pp. 134–143.

[29] R. Niedermeier and P. Rossmanith, "An efficient fixed parameter algorithm for 3-Hitting Set," *Journal of Discrete Algorithms*, vol. 1, no. 1, pp. 89–102, 2003.

[30] R. G. Downey, "Parameterized complexity for the skeptic," in *Proceedings of the 18th IEEE Conference on Computational Complexity (CCC'03)*, 2003, pp. 147–168, invited paper.

Acknowledgement

The authors thank C. Paul and J. Cassaigne for careful readings of the manuscript and help in simplifying some proofs. The authors are also grateful to anonymous reviewers for their valuable comments.

Appendices

5 Details on the implementation of Find-Refinement-or-Conflict on two trees

The $O(n)$ running time of the algorithm is shown here in detail by a successive examination of data structures and operations they support:

- Trees T_1 and T_2 are stored by usual pointers.

  Each edge of T_1 is processed once, when its higher node is the cherry v_1 processed by the main loop. Its higher node is either a cherry at start, or becomes a cherry because of the repeated process of replacing cherries of T_1 by a new leaf each.

  Edges of T_2 are each examined a constant number of times: when considering a node v_2, the edges of each subtree $S(c)$ containing leaves in $L(v_1)$ will be considered once when traversing $S(c)$ (line 7), plus once for some of them, when c has to be identified (line 6: edges of the path from leaf ℓ to the first ascendant c that is a child of v_2). In case of conflict, edges of a subtree can be traversed another time to identify a leaf ℓ' (line 9) before stopping the algorithm. Finding such a leaf is the reason why subtrees $S(c)$ are not readily removed from T_2 when processed (hence the reason for list C).

  When all leaves of $L(v_1)$ have been processed successfully, each edge of a subtree $S(c)$ has been traversed twice and is traversed a final time to make a copy of the subtree in F (line 11), before being removed (line 14).

- Forest F consists of a set of nodes arranged in a set of non-overlapping trees that are subtrees of the final output tree on n leaves. Thus, at any step, the forest contains $O(n)$ nodes. Assembling some trees in F (line 14) involves identifying nodes with labels corresponding to leaves in a subtree of $S(v_2)$ and connecting them according to the topology of this subtree of T_2. For this purpose, an additional array can be easily maintained to find each required node of F in $O(1)$. Creating a new node v in F with a given label (line 2) is done $O(n)$ times and costs $O(1)$ each time. In case of conflict, at most three different trees of F, *i.e.* $O(n)$ nodes, are traversed (line 10) to find leaves of L (*i.e.* leaves of trees in F).

- List C is a simple linked list of root nodes of child subtrees of v_2 to be removed from T_2 after all leaves of P have been processed. Each element is added in $O(1)$ and removed in $O(1)$ when the list is emptied (line 14). Removing each subtree from the list of child subtrees of v_2 is performed in $O(1)$ when coding its children as a bidirectional linked list.

- The list of leaves P is managed as an array of $2n-1$ bits: one for each of the n original labels of leaves, plus one for each of the $n-1$ new labels (assigned to a cherry node that becomes a leaf when pruning its child nodes. Initially, all entries are zeroed, indicating the absence of any leaf in P. When considering leaves $L(v_1)$ of a cherry $v_1 \in T_1$, only bits corresponding to these labels are set (line 3). Then leaves put in P (line 3) are successively taken until none remains (loop line 4). This is done by listing the children of v_1 (they are all leaves).

  Testing whether a leaf ℓ'' is in P (line 8) is just checking whether the corresponding bit is set at 1. On the same line, removing the leaf from P is just setting this bit at 0. Note that after the last iteration of the loop line 4, P has returned to its initial state, *i.e.* all the bits that were set at 1 have been turned back to 0. Thus, using P to handle leaves of a cherry v_1 during an iteration of the main while loop (line 1) costs a time proportional to $\#L(v_1)$. After this iteration, the leaves $L(v_1)$ are removed from the tree, hence the amortized cost for maintaining P during the whole algorithm is $O(n)$.

- For *lca* queries, we can use the dynamic structure of [26], initialized in $O(n)$ which enables us to obtain the lca of any two nodes in $O(1)$ worst case time and supports insertion/deletions of leaves in $O(1)$. Globally, $O(n)$ lca queries issue from line 2: queries issue from set of leaf labels $L(v_1)$ taken from a cherry in $v_1 \in T_1$ and concern nodes in $v_2 \in T_2$. To identify $v_2 := lca_{T_2}(L(v_1))$, we need $\#L(v_1)-1$ queries. But then these leaves are removed from the trees and v_1 becomes a leaf, that will be implied in a cherry at a latter step (if no conflict arises), thus giving rise to one lca query in turn. Thus, each node of T_1 will be used in at most one lca query, so the algorithm performs $O(n)$ lca queries, each in $O(1)$. The data structure maintaining lca relationships also has to be updated during the algorithm, but this requires $O(n)$ insertions and deletions of leaves, hence $O(n)$ globally: the number of leaves inserted (line 15) is bounded by the number of processed cherries $v_1 \in T_1$, so is $O(n)$. Removing a subtree from T_2 (line 14) costs a number of leaf deletions proportional to the number of its nodes (performing a postorder traversal). There are $O(n)$ nodes initially in T_2 and $O(n)$ will be added (line 15), thus $O(n)$ deletions are performed, each costing $O(1)$. Hence, deletions will cost $O(n)$ time to update the lca structure. Note that an alternative to using the dynamic data structure to identify lcas is to perform careful traversal of parts of T_2.

6 Proof of Lemma 2

To prove the lemma we first need two remarks and a preliminary claim. The first remark precises the link between clusters and contractions of edges in a tree.

Remark 1 *Let T be a rooted tree and let v be an internal non-root node of T. Contracting the edge of T between v and its parent gives a tree with $\mathcal{C}l(T) - \{L(v)\}$ as set of clusters.*

The next remark precises the link between clusters and rooted triples of a tree.

Remark 2 *Let T be a rooted tree and let ℓ, ℓ', ℓ'' be three distinct leaves of T: $\ell|\ell'\ell'' \in rt(T)$ iff there is an internal node v in T s.t. $\ell \notin L(v)$ and $\{\ell', \ell''\} \subseteq L(v)$.*

Lemma 10 ([25]) *Let T and T' be two trees on the same set of leaves. T refines T' iff $\mathcal{C}l(T') \subseteq \mathcal{C}l(T)$.*

Proof. If T refines T' then, Rem. 1 implies $\mathcal{C}l(T') \subseteq \mathcal{C}l(T)$. Conversely, assume $\mathcal{C}l(T') \subseteq \mathcal{C}l(T)$. Then, we have $rt(T') \subseteq rt(T)$ by Rem. 2. Thus, by Lem. 1-(iii), T refines T'. □

Proof of Lemma 2:
(i) ⇒ (ii). Assume that T is the minimum refinement of $\mathcal{T}$. For all $T_i \in \mathcal{T}$, T refines T_i and, thus, by Lem. 10 $\mathcal{C}l(T_i)$ is a subset of $\mathcal{C}l(T)$. Hence, we have

$$\mathcal{C}l(T_1) \cup \mathcal{C}l(T_2) \cup \ldots \cup \mathcal{C}l(T_k) \subseteq \mathcal{C}l(T)\,. \tag{7}$$

By contradiction, assume that this inclusion is proper.

Then there is an internal node v of T s.t. for all $T_i \in \mathcal{T}$, $L(v) \notin \mathcal{C}l(T_i)$. Since L is a cluster of all T_i's, v is not the root of T.

Let T' be the tree obtained from T by contracting the edge between v and its parent. For all $T_i \in \mathcal{T}$, Rem. 1 yields

$$\mathcal{C}l(T') = \mathcal{C}l(T) \setminus \{L(v)\} \supseteq \mathcal{C}l(T_i)$$

and thus, T' refines $\mathcal{T}$. Since T' has less edges than T, T' can not refine T. Therefore, T is not a minimum refinement of $\mathcal{T}$. Hence, we have shown that inclusion (7) is an equality.
(ii) ⇒ (iii) is easily deduced from Rem. 2.
(iii) ⇒ (i). Assume that $rt(T) = rt(T_1) \cup rt(T_2) \cup \ldots \cup rt(T_k)$.

For all $T_i \in \mathcal{T}$, $rt(T_i)$ is a subset of $rt(T)$ and thus, by Lem. 1-(iii), T refines T_i. Hence, T refines $\mathcal{T}$.

Moreover, let T' be a tree on L refining $\mathcal{T}$. For all $T_i \in \mathcal{T}$, T' refines T_i and, thus, we have $rt(T_i) \subseteq rt(T')$. From that we deduce $rt(T) = rt(T_1) \cup rt(T_2) \cup \ldots \cup rt(T_k) \subseteq rt(T')$: T' refines T from Lem. 1-(iii). Hence, we have shown that T is a minimum refinement of $\mathcal{T}$.
Finally, we have to prove the existence of a minimum refinement whenever $\mathcal{T}$ is compatible. Suppose that $\mathcal{T}$ is compatible and let T' be a tree refining $\mathcal{T}$. By Lem. 10, we have $\mathcal{C}l(T_i) \subseteq \mathcal{C}l(T')$ for all $T_i \in \mathcal{T}$ and thus

$$\mathcal{C}l(T_1) \cup \mathcal{C}l(T_2) \cup \ldots \cup \mathcal{C}l(T_k) \subseteq \mathcal{C}l(T')\,.$$

Moreover, we can modify T' to remove clusters in

$$\mathcal{C}l(T') - \big(\mathcal{C}l(T_1) \cup \mathcal{C}l(T_2) \cup \ldots \cup \mathcal{C}l(T_k)\big)$$

by contracting corresponding edges according to Rem. 1. Thus, we obtain a tree T satisfying Lem. 2-(ii), *i.e.* T is a minimum refinement of $\mathcal{T}$. □