Maximum Agreement and Compatible Supertrees

Vincent Berry, François Nicolas

Maximum agreement and compatible supertrees ⋆,⋆⋆

Vincent Berry *

Équipe Méthodes et Algorithmes pour la Bioinformatique, *LIRMM, CNRS–Université Montpellier 2*

François Nicolas

Department of Computer Science, P. O. Box 68 (Gustaf Hällströmin katu 2 b), 00014 University of Helsinki, Finland.

keywords: Algorithms; Trees; Isomorphism and refinement relations; Supertrees; Computational Biology;

Abstract

Given a set of leaf-labelled trees with identical leaf sets, the MAST problem, respectively MCT problem, consists of finding a largest subset of leaves such that all input trees restricted to these leaves are isomorphic, respectively compatible. In this paper, we propose extensions of these problems to the context of supertree inference, where input trees have non-identical leaf sets. This situation is of particular interest in phylogenetics. The resulting problems are called SMAST and SMCT.

A sufficient condition is given that identifies cases where these problems can be solved by resorting to MAST and MCT as subproblems. This condition is met, for instance, when only two input trees are considered. Then we give algorithms for SMAST and SMCT that benefit from the link with the subtree problems. These algorithms run in time linear to the time needed to solve MAST, respectively MCT, on an instance of the same or smaller size.

It is shown that arbitrary instances of SMAST and SMCT can be turned in polynomial time into instances composed of trees with a bounded number of leaves.

SMAST is shown to be W[2]-hard when the considered parameter is the number of input leaves that have to be removed to obtain the agreement of the input trees. A simlar result holds for SMCT. Moreover, the corresponding optimization problems, that is the complements of SMAST and SMCT, can not be approximated in polynomial time within a constant factor, unless P = NP. These results also hold when the input trees have a bounded number of leaves.

The presented results apply to both collections of rooted and unrooted trees.

1 Introduction

Supertree problems and methods.

This paper proposes two new methods for building *supertrees*, *i.e.* trees inferred from other trees. Building supertrees is a problem whose importance increased markedly in the last decade in phylogenetics. Trees considered in this field are called *phylogenies* or *evolutionary trees* because any such tree is an estimation of the evolutionary history of a set of species or sequences (*e.g.* genes) called *taxa*: the leaves of the tree are each labelled by a current taxon and the branching pattern of the tree describes a speciation scenario leading from ancestral taxa to current ones. In phylogenetics, the major work in progress is the building of the so-called *Tree of Life*, a huge tree interrelating all species of the living realm (see e.g. [39]). Currently, some trees of life are still assembled by hand. According to systematic biologists, the key problem remains to obtain reliable computational methods to assemble several source phylogenies into a single supertree [10].

The input of any supertree building method is a collection of trees with different but overlapping sets of leaves. The output is a tree whose leaf set includes all (or most) species of the input trees and that displays as much as possible

⋆ Part of these results were briefly covered in a conference paper [8].
⋆⋆Supported by the *Action Incitative Informatique-Mathématique-Physique en Biologie Moléculaire* [ACI IMP-Bio].
* Corresponding author
Email address: vberry@lirmm.fr (Vincent Berry).

of the branching pattern of the input trees on these leaves. The input trees, usually inferred from different datasets, often differ upon the position of some leaves or groups of leaves. Current supertree methods can be divided into two categories depending on the way they handle these conflicts: *(i)* optimization methods tend to resolve conflicts, i.e. choose one of the proposed scenarii, according to a specified optimization criterion (*e.g.* [3,35,33]); *(ii)* consensus methods produce supertrees displaying only the parts of the species' history for which the input trees agree. The drawback of approach *(i)* is that output supertrees sometimes contain undesirable or unjustified resolutions of conflicts [33]. Approach *(ii)* has been poorly investigated in the supertree context, in contrast with the many consensus methods available to deal with collection of trees having identical leaf sets. The two known supertree methods of this kind are the pioneering *strict consensus* [22] and *reduced consensus* [38]. Unfortunately, strict consensus usually produces a supertree with a scant amount of information [32,11] and only applies to the rare case of compatible input trees: the trees can differ from one another but not actually conflict [10]. Moreover, the use of reduced consensus is not widespread because a few conflicts only will likely result in a whole set of complementary partial trees as output [38, Section 4], instead of the single synthetic supertree that is sought. Below, we propose alternative methods affiliated to approach *(ii)* that do not suffer from the drawbacks just mentioned.

Extending MAST *and* MCT *to the supertree context.*

Almost all supertree methods proposed so far focus on *clusters* (sets of leaves under internal nodes). This is a problem whenever the input trees contain some "rogue" leaves, *i.e.* leaves whose position differs greatly from one input tree to the other. Indeed, changing the position of just one leaf in a tree can lead to a completely different set of clusters. Unfortunately, this phenomenon happens quite often in real supertree instances. Thus, several authors have suggested that an alternative in designing supertree methods would be to focus on leaves individually rather than to consider clusters of leaves [22,32,11]. The rational for this is that, in a number of cases, removing a few leaves upon whose position the input trees disagree is sufficient to produce a single informative supertree.

Here we respond to this suggestion by extending to the case of trees on overlapping sets of leaves a well-known classical consensus problem and one of its variants. Given a set of leaf-labelled trees with identical leaf sets, the Maximum Agreement SubTree (MAST) problem consists of finding a subtree homeomorphically included in all input trees and with the largest number of leaves [18,2,24,31,15]. In other words, this involves selecting a largest set of input leaves such that the source trees are isomorphic (*i.e. agree*) when restricted to these leaves. Note that this problem is also considered in various domains other than computational biology.

In phylogenetics, when input trees are non-binary, a node with more than

two descendants usually represents uncertainty with respect to the branching pattern of its descendants rather than a multi-speciation event. The Maximum Compatible Tree problem (MCT) is a variant of MAST that takes this into account by seeking a largest set of leaves such that the source trees restricted to these leaves are *compatible* [25,27,20,7] (note that this problem is also called MRST in [27]). Compatibility allows a high-degree node of a source tree to be resolved (split into several nodes) according to the information present in other source trees. Note that this is a weaker constraint than the isomorphism required by MAST, and thus allows inclusion of more input leaves in the output tree.

We call SMAST and SMCT the respective variants of MAST and MCT concerned with supertree inference, *i.e.* which allow input trees with differing leaf sets. The use of SMCT rather than SMAST can be advocated when the edges of the input trees are associated with confidence values (*e.g.* bootstrap values in phylogenetic analysis). To obtain a more reliable supertree, edges with insufficient support can be collapsed before the supertree inference is performed, which gives rise to nodes of higher degree in some input trees. In this case, SMCT is more propitious than SMAST for inferring a supertree, as it allows a high degree node of an input tree to be resolved according to the highly supported branching patterns present in other input trees. In other words, highly supported clusters that remain in some input trees will not be contradicted by weakly supported alternatives collapsed in other input trees.

Apart from inferring a partial estimate of the species' history, SMAST and SMCT can also be used to achieve the following goals in phylogenetics:

- to measure the topological similarity between the input trees considered for a supertree reconstruction. The proportion of leaves not conserved in a produced supertree when solving SMAST and SMCT measures the intrinsic difficulty of the particular instance considered for supertree building. This difficulty is currently assessed only through indirect measures, such as the average number of triples or quartets common to two input trees.
- by explicitly indicating leaves upon whose position the input trees conflict, SMAST and SMCT help to identify leaves that may be involved in horizontal transfers of genes and to identify paralogous sequences in the original datasets.
- to increase the accuracy of other supertree building methods. For instance, the popular *matrix representation with parsimony* (MRP) method [3,35] has relatively low accuracy when the input trees overlap moderately and [12] recommends adding to the set of input trees a tree with leaves spanning most input trees, that they call a *seed* tree. Any supertree that is a solution of SMAST or SMCT most likely contains leaves from most, if not all, input trees (see Theorem 3) and, moreover, fully agrees with all of these trees by definition. Thus, it is a good candidate for being a seed tree.

Related work

We first review known results on the complexity of MAST and MCT. The MAST problem is NP-hard for three rooted trees of unbounded degree [2], while MCT is already NP-hard for two rooted trees of unbounded degree [27]. When k rooted trees with n leaves are given as input, 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 [2,13], and MCT can be solved in $O(2^{2kd}n^k)$ time provided that all input trees have degree bounded by d [20]. MCT is solvable in polynomial time provided that the maximum degree of all input trees is bounded [23]. Polynomial-time algorithms with sub-quadratic running times have been obtained for MAST in the special case of two input trees [30,15,31].

MAST and MCT are known to be *fixed-parameter tractable* (FPT) in p, the smallest number of leaves to remove from the input set of leaves such that the input trees agree. The latest result being an $O(\min\{3^p kn, 2.27^p + kn^3\})$ time algorithm for the case of rooted trees (considering unrooted trees adds a p factor) [9]. The MAST problem (maximizing the number of leaves in an agreement subtree) is hard to approximate on a bounded number of trees [27] or on trees with a bounded height [21]. The same results hold for MCT [7]. However, the complement problem of MAST (*i.e.* minimizing the number of leaves to remove so that input trees are isomorphic) can be approximated within a constant ratio in polynomial time in both rooted and unrooted cases

[2,7,28]. The same result holds for the complement problem of MCT [19,7].

The extension of MAST to supertree inference has also been considered in [24], and very recently in [29]. However, the "supertrees" considered in [24] (and subsequent papers) have a different meaning from that considered in phylogenetics and here. The work of [29] is independent of the results presented here, but studies an extension of MAST similar to the one we present. [29] give an algorithm for the case of two input rooted trees and present an approximation result that is complementary to the results shown here. However, they neither consider the case of unrooted trees, nor the extension of the MCT problem to the supertree context.

Results.

We show how to extend MAST and MCT in a natural way to obtain the SMAST and SMCT problems on supertrees. We prove that the maximal degree d of input trees does not play any role in solving SMAST and SMCT as any instance of these problems can be reduced to an instance with small bounded degree. This contrasts with MAST and MCT problems, for which polynomial-time algorithms are available for input trees of bounded degree only (in the case of more than two input trees).

We show that any leaf appearing in a single input tree will definitely be included in all supertrees that are solutions of SMAST and SMCT. We give a sufficient condition for SMAST and SMCT to be solved by using MAST and

MCT as subproblems. This condition is always fulfilled for collections of only two input trees but also applies to instances including more trees.

We give algorithms that take advantage of the link between these problems and detail when this leads to polynomial cases for SMAST and SMCT. The presented algorithms run in time linear to the algorithm solving MAST and MCT, one of them generalizing the algorithm of [29]. The MergeTrees algorithm we propose also enables computation of the strict consensus supertree of two trees in $O(n)$ time (where n is the total number of input leaves), which improves the $O(n^3)$ bound stated in [22].

In general, SMAST and SMCT are NP-hard as they are equivalent to MAST, respectively MCT, in the case of input trees with identical leaf sets. However, by reduction from Hitting Set, we show that SMAST and SMCT are more difficult than MAST and MCT, as they are W[2]-hard for p (the minimum number of input leaves to remove from input trees to obtain their agreement, respectively compatibility). This holds even when the instance only consists of rooted triples (binary trees with three leaves) or unrooted quartets (trees with four leaves).

This suggests that heuristic algorithms may be required to solve these supertree problems in general. However, no heuristics with a tight approximation ratio can exist for these problems: SMAST and SMCT are hard to approximate (from the results of [27,21] for MAST), and the reduction from Hitting Set

is approximation preserving, which proves that no polynomial-time algorithm can approximate within a *constant factor* the complement of the SMAST and SMCT problems (unless P = NP).

Note in passing that, compared to the reduction from Independent Set / Vertex Cover given in [29], the reduction given here from Hitting Set leads to tighter results on the parameterized complexity and approximability of the complement of SMAST. Moreover, our result also applies to the complement of SMCT and to the unrooted case. Note that the strong limitations shown here on the approximability of SMAST and SMCT do not impede the existence of approximation algorithms with non-constant ratio. *E.g.* [29] provides a $(n/\log n)$-approximation algorithm for SMAST on rooted trees.

Organization of the paper.

In the following, Section 2 reviews definitions of MAST and MCT with associated results, and introduces the SMAST and SMCT supertree problems. Section 3 presents algorithms to solve SMAST and SMCT in the particular cases where MAST and MCT can be used as subproblems. A sufficient condition for applying these algorithms is also stated there. Then, Section 4 details how general instances of SMAST and SMCT can be polynomially transformed into instances of trees having a bounded number of leaves (hence also a bounded degree). On the basis of such instances, Section 5 shows the intractability and inapproximability results.

2 Definitions and preliminaries

The trees we consider are *evolutionary trees* (also called *phylogenies*). Such a tree T has its leaf set $L(T)$ in bijection with a label set and is either *rooted* (at a node denoted root(T)), in which case all internal nodes have at least two children each, or *unrooted*, in which case internal nodes have a degree of at least three. In the following, trees are denoted T, respectively R, respectively U, in statements applying to both rooted and unrooted trees, respectively applying only to rooted trees, respectively applying only to unrooted trees. When there is no ambiguity, we identify leaves with their labels. Given a set S, Card (S) denotes the cardinality of S. In particular, if L is a leaf set, Card (L) denotes the number of leaves in L. The size $|T|$ of a tree T is the number of its leaves: $|T| = \text{Card}\,(L(T))$. For a node u in a rooted tree, we denote $S(u)$ the subtree rooted at u (*i.e.* u and its descendant nodes) and $L(u)$ the leaves of this subtree. See Figure 2 for an example.

The following definitions apply to rooted and unrooted trees.

Definition 1 (Restriction of a tree) *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 $\mathcal{T} \,|\, L := \{T \,|\, L : T \in \mathcal{T}\}$.*

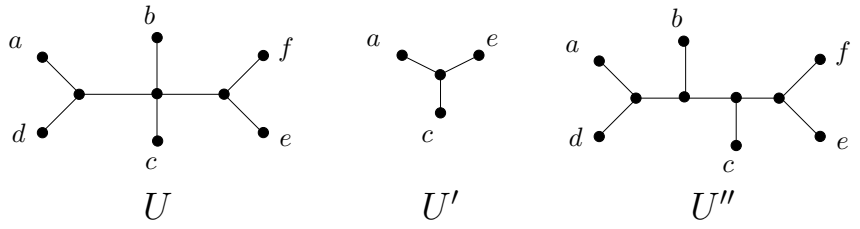

Fig. 1. Three unrooted trees. A tree U, a tree U' such that $U' = U \mid \{a, c, e\}$ and a tree U'' such that $U'' \trianglerighteq U$.

See trees U, U' in Figure 1 for an example. Note that for any tree T and any two label sets L,L', $(T \mid L) \mid L' = T \mid (L \cap L') = (T \mid L') \mid L$.

Definition 2 (Tree isomorphism and inclusion) *Two trees T, T' are* isomorphic*, denoted $T = T'$, if and only if there is a graph isomorphism $T \mapsto T'$ preserving leaf labels (and the root if both trees are rooted). Given two trees T, T', T is* homeomorphically included *in T' if and only if $T = T' \mid L(T)$.*

Definition 3 (Tree refinement) *A tree T* refines *a tree T', and we write $T \trianglerighteq T'$, whenever T can be transformed into T' by collapsing some of its internal edges (*collapsing *an edge means removing it and merging its extremities). See Figure 1 for an example. More generally, a tree T refines a collection $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$, denoted $T \trianglerighteq \mathcal{T}$, whenever T refines all T_i's in $\mathcal{T}$.*

When considering a set of trees with different leaf sets, the preceding definition can be extended [37]:

Definition 4 (Tree compatibility) *Let T be a tree with leaf set L, let L' be a subset of L and T' be a tree with leaf set L'. We say T* displays *T' whenever $T|L' \trianglerighteq T'$. Furthermore, a collection $\mathcal{T}$ of trees with different leaf sets is* compatible *if there is a tree T that displays every tree in $\mathcal{T}$. In that*

case, T is said to display $\mathcal{T}$.

Isomorphism and compatibility issues between rooted trees can also be expressed in terms of ancestor relationships. Given two nodes u and v in a rooted tree R, $u < v$ means that u is a proper ancestor of v and $u \leq v$ means that u is either a proper ancestor of v, or v itself. The *least common ancestor* (or *lca*) of a set of leaves $L \subseteq L(R)$ is the unique node u such that $u \leq \ell$ for all $\ell \in L$, and $v < u$ for any other node v that is also an ancestor of every leaf in L. The lca of L in R is denoted $\mathrm{lca}_R(L)$. More particularly, the lca of any pair $\{\ell, \ell'\} \subseteq L(R)$ is denoted $lca_R(\ell, \ell')$.

The following statements are directly derived from the definitions given previously and are implicitly or explicitly used in a number of works (for instance [19,34]).

Observation 1 *Let R and R' be two rooted trees on the same leaf set L.*

The following two statements are equivalent:

(i) *R and R' are isomorphic*

(ii) *$\forall \ell, \ell', \ell'' \in L$, the two following equations hold*

$$lca_R(\ell, \ell') < lca_R(\ell, \ell'') \iff lca_{R'}(\ell, \ell') < lca_{R'}(\ell, \ell'') \quad (1)$$

$$\textit{and} \;\; lca_R(\ell, \ell') = lca_R(\ell, \ell'') \iff lca_{R'}(\ell, \ell') = lca_{R'}(\ell, \ell'') \quad (2)$$

Moreover, the following two statements are equivalent:

(iii) *R refines R'*

(iv) *$\forall \ell, \ell', \ell'' \in L$, the following holds:*

$$lca_{R'}(\ell, \ell') < lca_{R'}(\ell, \ell'') \Rightarrow lca_R(\ell, \ell') < lca_R(\ell, \ell'') \quad (3)$$

2.1 Agreement problems for trees with identical leaf sets

The well-known MAST problem is defined as follows:

Definition 5 (MAST **problem)** *Given a collection $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ of trees with identical leaf sets L, an* agreement subtree *of $\mathcal{T}$ is any tree T with leaves in L such that $\forall T_i \in \mathcal{T}$, $T = T_i \,|\, L(T)$. The* Maximum Agreement SubTree *problem (*MAST*) consists in finding an agreement subtree of $\mathcal{T}$ with the largest number of leaves. Such a tree is denoted $MAST(\mathcal{T})$.*

The MCT problem is a variant of MAST introduced in phylogenetics to deal with cases where high-degree nodes represent uncertainty with respect to the relative branching of their child subtrees.

Definition 6 (MCT **problem)** *Given a collection $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ of input trees with identical leaf sets L, a tree T with leaves in L is said to be* compatible with *$\mathcal{T}$ if and only if $\forall T_i \in \mathcal{T}$, $T \trianglerighteq T_i \,|\, L(T)$. The* Maximum Compatible Tree *problem (*MCT*) consists in finding a tree compatible with $\mathcal{T}$ having the largest number of leaves. Such a tree is denoted $MCT(\mathcal{T})$. If there is a tree T compatible with $\mathcal{T}$ such that $L(T) = L$, then the collection $\mathcal{T}$ is said to be* compatible.

Note that an evolutionary tree T properly refining another tree T', agrees with the entire evolutionary history of T', while containing additional history absent

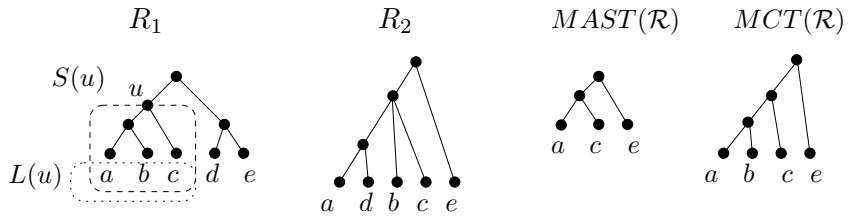

Fig. 2. A collection $\mathcal{R} = \{R_1, R_2\}$, one of the $MAST(\mathcal{R})$ trees and the $MCT(\mathcal{R})$ tree. $S(u)$ denotes the subtree induced by a node u and $L(u)$ the corresponding set of leaves.

from T': at least one high degree node of T' is replaced in T by several nodes, hence T specifies more speciation events than T'. Figure 2 shows examples of trees $MAST(\mathcal{T})$ and $MCT(\mathcal{T})$ for a collection $\mathcal{T}$ of two rooted trees. Note that $\forall \mathcal{T}$, $|MCT(\mathcal{T})| \geq |MAST(\mathcal{T})|$ and that MCT is equivalent to MAST when input trees are binary. Note also that some instances of the MAST and MCT problems have several optimum solutions.

2.2 Extending agreement problems to the supertree context

We now consider the case of supertree inference, where input trees are allowed to have different sets of leaves. We first show how to extend MAST and MCT to this context. Then we distinguish different kinds of leaves that appear in the input trees, depending on the overlap of these trees. Without loss of generality, the rest of the paper assumes that any input tree shares at least two leaves with other input trees.

Definition 7 (Leaf set of a collection) *Given a collection* $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ *of trees, we denote* $L(\mathcal{T}) := \bigcup_{T_i \in \mathcal{T}} L(T_i)$ *the set of all leaves appearing in at least one tree of* $\mathcal{T}$*.*

Definition 8 (SMAST **problem)** *Given a collection* $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ *of trees, an* agreement supertree *of* $\mathcal{T}$ *is a tree* T *with* $L(T) \subseteq L(\mathcal{T})$ *such that* $\forall T_i \in \mathcal{T}$, $T \,|\, L(T_i) = T_i \,|\, L(T)$. *An agreement supertree of* $\mathcal{T}$ *that is of maximum size is called a* maximum agreement supertree *of* $\mathcal{T}$ *and is denoted* $SMAST(\mathcal{T})$. *The corresponding optimization problem is stated as follows:*

Name: Maximum Agreement Supertree *(*SMAST*)*

Instance: *A finite collection* $\mathcal{T}$ *of trees (all rooted or all unrooted).*

Solution: *An agreement supertree* T *of* $\mathcal{T}$.

Measure: $|T|$, *to be maximized.*

In a similar way, we define:

Definition 9 (SMCT **problem)** *Given a collection* $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ *of trees, a* supertree compatible with $\mathcal{T}$ *is a tree* T *with* $L(T) \subseteq L(\mathcal{T})$ *such that* $\forall T_i \in \mathcal{T}$, $T \,|\, L(T_i) \trianglerighteq T_i \,|\, L(T)$. *A supertree compatible with* $\mathcal{T}$ *that is of maximum size is called a* maximum compatible supertree *of* $\mathcal{T}$ *and is denoted* $SMCT(\mathcal{T})$. *The corresponding optimization problem is stated as follows:*

Name: Maximum Compatible Supertree *(*SMCT*)*

Instance: *A finite collection* $\mathcal{T}$ *of trees (all rooted or all unrooted).*

Solution: *A supertree* T *compatible with* $\mathcal{T}$.

Measure: $|T|$, *to be maximized.*

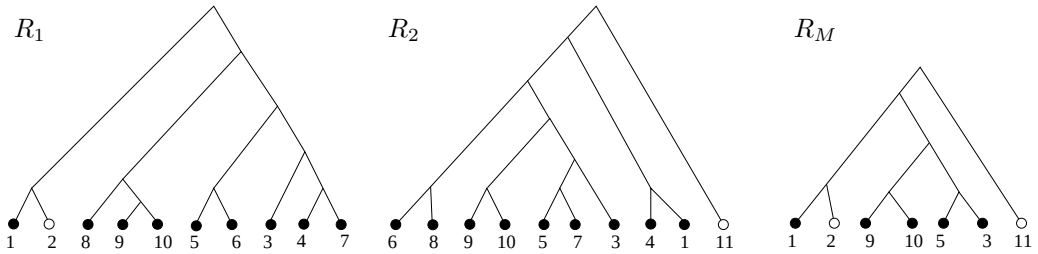

Fig. 3. A collection $\mathcal{R} = \{R_1, R_2\}$ of two source trees on tomatoes taken from [4] and a supertree R_M. In this example, the supertree both represents a $SMAST(\mathcal{R})$ and a $SMCT(\mathcal{R})$. Leaves appearing in only one source tree are displayed in white. Correspondance between numbers and species: 1 – *L. lycopersicoides*, 2 – *L. juglandifolium*, 3 – *L. peruvianum*, 4 – *L. chilense*, 5 – *L. pennellii*, 6 – *L. hirsutum*, 7 – *L. chmielewskii*, 8 – *L. esculentum*, 9 – *L. pimpinellifolium*, 10 – *L. cheesmanii*, 11 – *L. rickii*.

Figure 3 shows a collection $\mathcal{R}$ of two source trees of species of tomato (*Lycopersicon*) and a supertree, that both is a $SMAST(\mathcal{R})$ and a $SMCT(\mathcal{R})$. Figure 7 shows an example where these two supertrees differ. The two problems stated above are natural extensions of the problems defined in Section 2.1. More precisely, SMAST, respectively SMCT, is equivalent to MAST, respectively MCT, when all input trees have the same set of leaves.

Remark 1 *Let $\mathcal{T}$ be a collection of trees.*

Any restriction of an agreement supertree of $\mathcal{T}$ is also an agreement supertree of $\mathcal{T}$ and any restriction of a supertree compatible with $\mathcal{T}$ is also a supertree compatible with $\mathcal{T}$.

Hence, $SMAST(\mathcal{T})$ and $SMCT(\mathcal{T})$ can potentially contain less leaves than some trees in $\mathcal{T}$.

Let $\mathcal{T}$ be a collection of trees with identical leaf set L. Given any subset $L' \subseteq L$, there can be only one agreement *subtree* of $\mathcal{T}$ with leaf set L'. This

contrasts with what can happen for agreement *supertrees*, due to the lack of cross information between source trees:

Remark 2 *Given a collection $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ of trees and a subset $L \subseteq L(\mathcal{T})$, there may be more than one agreement supertree, respectively compatible supertree, of $\mathcal{T}$ with leaf set L.*

For instance, consider the collection $\mathcal{R} = \{R_1, R_2\}$ where $R_1 := ((a, c), b)$ and $R_2 = ((a, d), b)$, in parenthetical notation (*e.g.* Newick format). Any tree in the set $\{(((a, c), d), b),\ (((a, d), c), b),\ ((a, (c, d)), b),\ ((a, c, d), b)\}$ is a $SMAST(\mathcal{R})$ or a $SMCT(\mathcal{R})$.

Definition 10 (Types of leaves) *Let $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ be a collection of trees. Leaves in $L(\mathcal{T})$ can be partitioned in three subsets:*

(1) leaves appearing in every *tree of $\mathcal{T}$. We note $L_\cap(\mathcal{T}) := \bigcap_{T_i \in \mathcal{T}} L(T_i)$ this subset of leaves;*

(2) leaves appearing in several but not all *trees of $\mathcal{T}$. This subset of leaves is denoted $L_\Delta(\mathcal{T})$;*

(3) leaves specific *to a tree of $\mathcal{T}$,* i.e. *leaves appearing in a* single *tree of $\mathcal{T}$. This subset of leaves is denoted $L_{\mathrm{S}}(\mathcal{T})$.*

3 Computing $SMAST$ and $SMCT$

We first study, in Section 3.1, the particular case of a collection of only two trees with different leaf sets but such that one refines (respectively, is equal to) the other when restricted to common leaves. In that case, it is always possible to efficiently produce a tree that displays the collection, *i.e.* a maximum compatible supertree (respectively, a maximum agreement supertree). We provide a linear-time algorithm that fulfills this purpose.

Then, Section 3.2 shows that for a collection containing an arbitrary number of trees, any maximum agreement supertree or maximum compatible supertree includes all specific leaves, *i.e.* leaves that appear in a single tree of the collection. Based on this property, Section 3.3 shows cases where the MAST and MCT problems can be used to solve SMAST and SMCT problems, respectively, describing appropriate algorithms. The link between subtree problems and supertree problems induces polynomial cases for the latter, as listed in Section 3.4.

3.1 Merging two rooted trees in linear time

Let $\mathcal{R} = \{R_I, R_A\}$ be a compatible collection of two rooted trees with $L(R_I) \neq L(R_\text{A})$, such that $R_\text{A}|L(R_I) \trianglerighteq R_I|L(R_\text{A})$. In other words, R_A (loosely or strictly) refines R_I when they are both restricted to their common leaves. This section describes an algorithm called MergeTrees that returns a tree

R displaying $\mathcal{R}$. By definition, R contains all leaves of the two trees and R is a maximum compatible supertree of $\mathcal{R}$. Moreover, when $R_{\mathrm{A}}|L(R_I)$ loosely refines $R_I|L(R_{\mathrm{A}})$, *i.e.* when both restricted trees are isomorphic, then the tree R output by MergeTrees is a maximum agreement supertree of $\mathcal{R}$. To compute a $SMCT(\mathcal{R})$ or $SMAST(\mathcal{R})$ of a collection $\mathcal{R}$ of more than two trees, repeated calls to MergeTrees operating on two trees will be used, as described in Section 3.3. For the rest of Section 3.1, trees are considered to be rooted.

Definition 11 (Specific subtree) *Let $\mathcal{R}$ be a collection of trees and $R_I \in \mathcal{R}$. A* specific subtree *of R_I is any maximal tree of the form $S(v_i)$, where v_i is a node of R_I such that $L(v_i) \cap \Big(L(\mathcal{R}) - L(R_I)\Big) = \emptyset$. Here,* maximal *means that if p_i is the parent node of v_i, then $L(p_i) \cap \Big(L(\mathcal{R}) - L(R_I)\Big) \neq \emptyset$. A leaf in a specific subtree is called a* specific leaf.

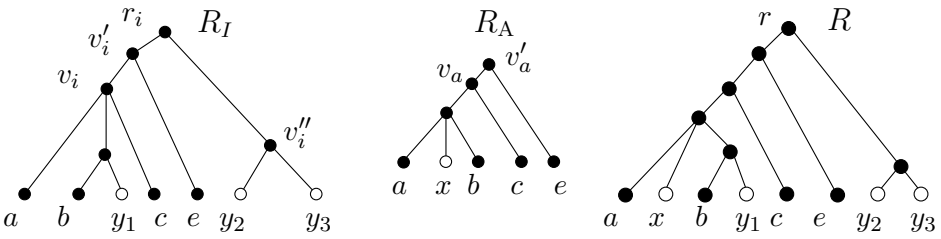

Fig. 4. Two rooted trees R_I and R_{A} with overlapping sets of leaves such that $R_{\mathrm{A}}|L(R_I) \trianglerighteq R_I|L(R_{\mathrm{A}})$, and the tree R returned by the call MergeTrees (R_I, R_{A}). Specific leaves are indicated by white circles.

For instance, consider the collection $\{R_I, R_{\mathrm{A}}\}$ displayed in Figure 4. R_I hosts two specific subtrees: the leaf $\{y_1\}$ and the subtree rooted at node v''_i. The leaf $\{x\}$ is the only specific subtree of R_{A}. In a collection $\mathcal{R} = \{R_{\mathrm{A}}, R_I\}$ of two trees, leaves are either specific to R_I, specific to R_{A}, or common to

both trees. To obtain the desired tree R, MergeTrees proceeds by grafting specific subtrees of R_I into R_A. In a way, its goal is similar to the grafting step of the well-known algorithm of Gordon for computing a strict consensus supertree [22]. However, Gordon's algorithm attaches, one by one, specific *leaves* of the input trees to a "backbone" tree, while the algorithm detailed here proceeds by grafting each time a whole specific *subtree*. Using this idea and two simple data structures, we can achieve linear running time when the grafting step of [22] runs in cubic time. As the tree R output by MergeTrees has to display R_I, the specific subtrees of R_I have to be grafted in R_A so as to respect the ancestor-descendant constraints of R_I between lcas of leaves (see Observation 1). Sometimes there is a unique place where a subtree can be grafted in order to respect these relationships, and sometimes there can be several. However, a correspondence between some nodes of R_I and some nodes of R_A can be maintained such that a correct place is always easily identified. This correspondence is explained by the fact that, when restricted to common leaves $L_\cap(\{R_\mathrm{A}, R_I\})$, R_I is refined by R_A. This ensures that for any node $v_i \in R_I|L(R_\mathrm{A})$ there is a unique node $v_a \in R_\mathrm{A}|L(R_I)$, such that $L(v_i) = L(v_a)$. This correspondence between nodes of the restricted trees $R_\mathrm{A}|L(R_I)$ and $R_I|L(R_\mathrm{A})$ can be translated as a correspondence between nodes of the complete input trees R_I and R_A:

Definition 12 *Let R and R' be two trees. To any node v in R such that $v := \mathrm{lca}_R(S)$ with $S \subseteq L_\cap(\{R, R'\}), \mathrm{Card}\,(S) \geq 1$, we associate an* anchor

node v' *in* R'*, defined as* $v' := lca_{R'}(S)$*. Nodes of* R *with an anchor in* R' *are called* anchored nodes.

Note that the previous definition allows both internal nodes and leaf nodes of R to have an anchor in R'. However, note that some nodes of R can have no anchor in R' and that some nodes of R' are not anchors of any node in R.

Remark 3 *Let* R_I *and* R_{A} *be two rooted trees such that* $R_{\mathrm{A}}|L(R_I) \trianglerighteq R_I|L(R_{\mathrm{A}})$ *and* $v_i \in R_I$ *be a node with an anchor node* $v_a \in R_{\mathrm{A}}$*. We have* $L(v_i) \cap L_{\cap}(\{R_{\mathrm{A}}, R_I\}) = L(v_a) \cap L_{\cap}(\{R_{\mathrm{A}}, R_I\})$.

If the root r_i of R_I has no anchor in R_{A} according to the previous definition, then algorithm MergeTrees artificially anchors it to a node in R_{A} to enable grafting of specific subtrees hanging from r_i or between r_i and the highest anchored node in R_I. This artificial anchor is set in the following way: if the root r_a of R_{A} is not anchored to any node in R_I then it is used as the anchor for r_i. Otherwise, the anchor of r_i is set at a new node that is added as a parent of r_a (the algorithm will graft some specific subtrees to this new node that will hence not remain of degree 1). In Figure 4, leaves $\{a, b, c, e\}$ of R_I are respectively anchored at similarly labelled leaves in R_I; the internal nodes v_i, respectively v'_i, is anchored at v_a, respectively v'_a. The root r_i of R_I is artificially anchored to a node added as parent of the original root v'_a of R_{A} (not shown in the figure).

The position of any specific subtree of R_I can be considered w.r.t. anchored nodes: either *(i)* the subtree is hanging from an anchored node, or *(ii)* it is hanging from a node that is in between two anchored nodes. Thanks to the corresponding anchors between R_I and R_A, any specific subtree of R_I can be grafted in R_A so as to respect ancestor/descendant relationships needed for the produced tree to display R_I (see Observation 1). Algorithm 1 details in pseudo-code how this is done given two preprocessed data structures:

- Anchor that associates nodes of R_I to their respective anchors in R_A.
- SpecificChildren that associates to each node of R_I the list of its children that are roots of specific subtrees.

Algorithm 1: MERGETREES (R_I, R_A)

Input: Two rooted trees R_I and R_A with overlapping sets of leaves and such that $R_\mathrm{A}|L(R_I) \trianglerighteq R_I|L(R_\mathrm{A})$.

Result: A tree R on $L(R_\mathrm{A}) \cup L(R_I)$ such that $R|L(R_\mathrm{A}) = R_\mathrm{A}$ and $R|L(R_I) \trianglerighteq R_I|L(R)$

$R \leftarrow R_\mathrm{A}$

1 Compute the Anchors of R_I in R and the SpecificChildren for R and R_I

***(i)* Graft specific subtrees to anchored nodes of R**

2 **for** *each node* $v \in R_I$ *such that* Anchor$(v) \neq \emptyset$ **do**

  for *each node* $c \in$ SpecificChildren(v) **do**

    Add a copy of $S(c)$ as a new child of Anchor(v) in R

***(ii)* Graft specific subtrees on the path between two anchored nodes of R**

3 **for** *each non-root node* $v \in R_I$ *such that* Anchor$(v) \neq \emptyset$ **do**

  $v_i \leftarrow$ parent node of v ; $v_a \leftarrow$ Anchor (v)

4 **while** Anchor(v_i) *is* $\emptyset$ **do**

    Insert a new node v_new between v_a and its parent node in R

5   **for** *each* $c \in$ SpecificChildren(v_i) **do**

      Add a copy of $S(c)$ as a new child subtree of v_new

    $v_a \leftarrow v_\mathrm{new}$; $v_i \leftarrow$ parent node of v_i

return R

For an example of an execution of MergeTrees, consider the trees R_I and R_{A} displayed in Figure 4. The algorithm progressively builds tree R, starting from a copy of the tree R_{A}. Then anchors between R_I and R are computed. During this process, the initial root of R (corresponding to the node labelled v'_a in R_{A}) is given a parent node (called r) to serve as an artificial anchor for $r_i \in R_I$. During step *(i)* of the algorithm (*i.e.* loop in line 2), a copy of the specific subtree $S(v''_i)$ of R_I is grafted to the anchor r of r_i in R. Then during step *(ii)* (*i.e.* loop in line 3), the specific leaf-subtree y_1 hanging in R_I from the parent of the anchored node b is grafted in R to a new node inserted between the leaf labelled b and its parent. Figure 4 shows the resulting tree R.

Theorem 1 *Given a collection $\mathcal{R} = \{R_I, R_{\mathrm{A}}\}$ of two rooted trees such that $R_{\mathrm{A}}|L(R_I) \trianglerighteq R_I|L(R_{\mathrm{A}})$, the algorithm* MergeTrees (R_I, R_{A}) *returns a tree R such that $L(R) = L(\mathcal{R})$ and such that R is a $SMCT(\mathcal{R})$. In the particular case where $R_{\mathrm{A}}|L(R_I) = R_I|L(R_{\mathrm{A}})$, then R is a $SMAST(\mathcal{R})$.*

Proof. First, it is easy to see that $L(R) = L(R_I) \cup L(R_{\mathrm{A}})$: R is initially set at R_{A} and copies of all specific subtrees of R_I, *i.e.* containing all leaves in $L(\mathcal{R}) - L(R_{\mathrm{A}})$, are then grafted into R. Hence, if R displays these two trees, then it is a $SMCT(\mathcal{R})$, because there is no tree larger than R with leaves in $L(\mathcal{R})$.

The output tree R displays R_{A}, because R is initially set at R_{A}, and the only modifications made to this tree are additions of subtrees containing leaves not

belonging to R_{A}, *i.e.* not changing $R|L(R_{\mathrm{A}})$.

To show that R is a $SMCT(\mathcal{R})$, it remains to be proven that R displays R_I. To that aim, we prove that $R|L(R_I) \trianglerighteq R_I|L(R)$ because, together with $L(R_I) \subseteq L(R)$, this proves that R displays R_I. $R|L(R_I) \trianglerighteq R_I|L(R)$ is proven by induction on the number of grafts performed by the algorithm. The initial step of the induction holds as, before the first graft, $R = R_{\mathrm{A}}$, and we know by assumption that $R_{\mathrm{A}}|L(R_I) \trianglerighteq R_I|L(R_{\mathrm{A}})$. Now suppose the result holds for the first $g \geq 0$ grafts and that a $g+1$th specific subtree S_p of R_I is grafted into R, and for the needs of the proof, let R' be the resulting tree. We have to prove that $R'|L(R_I) \trianglerighteq R_I|L(R')$ which, by Observation 1, is equivalent to showing that for all ℓ, ℓ', ℓ'' in $L(R') \cap L(R_I)$ the following holds:

$$\mathrm{lca}_{R_I}(\ell, \ell') < \mathrm{lca}_{R_I}(\ell, \ell'') \Rightarrow \mathrm{lca}_{R'}(\ell, \ell') < \mathrm{lca}_{R'}(\ell, \ell'') \tag{4}$$

There are several cases depending on the number of these leaves already present in R before S_p is grafted:

(1) $\mathrm{Card}\left(\{\ell, \ell', \ell''\} \cap L(R)\right) = 0$: then the three leaves belong to $S_p \in R_I$ and, since an exact copy of S_p is grafted into R, the relationships between lcas of leaves in $L(S_p)$ are reproduced in R' as they are in R_I, hence (4) holds.

(2) $\mathrm{Card}\left(\{\ell, \ell', \ell''\} \cap L(R)\right) = 3$: this means that the three leaves belong to R before the grafting of S_p, hence (4) holds by induction hypothesis since grafting a subtree does not alter lca relationships between already present leaves in R.

(3) $\text{Card}\,(\{\ell, \ell', \ell''\} \cap L(R)) = 1$: w.l.o.g. suppose $\ell \in L(R)$, that is $\ell \notin L(S_p)$, hence $\text{lca}_{R_I}(\ell, \ell') = \text{lca}_{R_I}(\ell, \ell'') < \text{lca}_{R_I}(\ell', \ell'')$. As R' is obtained by adding a copy of S_p (containing leaves ℓ' and ℓ'') by a new edge (v, v') as a new child subtree of a node $v \in R$, this means that $\text{lca}_{R'}(\ell, \ell') = \text{lca}_{R'}(\ell, \ell'') \leq v < v' \leq \text{lca}_{R'}(\ell', \ell'')$, hence (4) holds.

(4) $\text{Card}\,(\{\ell, \ell', \ell''\} \cap L(R)) = 2$: numerous but simple sub-cases arise here. For the sake of readability, this part of the proof is presented in Appendix B.

In the particular case where $R_\text{A}|L(R_I) = R_I|L(R_\text{A})$, we also have to show that the equivalent of equation (1) of Observation 1 holds for R' and R_I. This proof strictly follows the proof given above for (4), and is thus omitted. □

Theorem 2 *Algorithm* MergeTrees(R_I, R_A) *runs in* $O(n)$ *time, where* $n = \text{Card}\,(L(R_I) \cup L(R_\text{A}))$.

Proof. We first detail the cost of preprocessing the data structures used by the algorithm (line 1). To initialize the SpecificChildren data structure, a simple $O(n)$ search of each tree R_I and R_A enables us to know which leaves of each tree are specific. Then an $O(n)$ postorder search of each tree enables us to identify the children of each node that are specific. To initialize the Anchor data structure, leaf-nodes of R_I with a label in $L_\cap(\{R_\text{A}, R_I\})$ are directly anchored at nodes of R_I sharing the same label. Then lca relationships are preprocessed in R_I and R_A in $O(n)$ time [26]. Let $\mathcal{O}$ be the left-right order

in which the leaves of $L_\cap(\{R_\mathrm{A}, R_I\})$ appear in R_I. The $O(n)$ pairs (ℓ_i, ℓ_{i+1}) of successive leaves in $\mathcal{O}$ are then considered. For each of these pairs, a single query for $v_i := lca_{R_I}(\ell_i, \ell_{i+1})$ and for $v_a := lca_{R_\mathrm{A}}(\ell_i, \ell_{i+1})$ is performed, each time costing only $O(1)$ thanks to the preprocessing step. Only these pairs of leaves have to be considered to span all internal nodes v_i of R_I for which an anchor has to be determined, and to obtain the anchor v_a for each of them (see Appendix A for more details). Hence, initializing the Anchor data structure costs $O(n)$ time.

Step *(i)* of the MergeTrees algorithm is performed by a recursive search of the tree R_I, during which $O(n)$ nodes v_i are considered. Knowing whether a given node v_i has an anchor in R_A is $O(1)$ time, thanks to the Anchor data structure. If so, knowing each specific child of v_i is also $O(1)$ time thanks to the SpecificChildren data structure. Note that each tree contains $O(n)$ specific children. For each specific child c of v_i, a copy of $S(c)$ is grafted under $\mathsf{Anchor}(v_i)$, which costs a time proportional to the size of this subtree, $O(|L(c)|)$. Since non-intersecting subtrees $S(c)$ are considered over all examined nodes v_i, the total size of grafted subtrees is bounded by the number of nodes in the tree, *i.e.* by $O(n)$, which is then the cost of step *(i)*.

Step *(ii)* is performed by a recursive postorder search of R_I. Every time an anchored node v is met, the edges on the path from v_i to its closest ascendant that is also anchored are explored (note that $O(n)$ such edges exist in R_I). During this upward walk, each time a non-anchored node v_i is met, copies

of specific subtrees hanging from v_i are grafted into R_A to a place identified in $O(1)$ (nodes v_a and v_{new}). Specific subtrees $S(c)$ to be grafted are each identified in $O(1)$ and grafted in $O(|L(c)|)$. This costs $\sum_c |L(c)| \in O(n)$ total time. Hence, step *(ii)* also costs $O(n)$ time. □

Corollary 1 *The strict consensus supertree [22] of two trees containing n leaves in total can be computed in $O(n)$ time.*

Proof. Computing the strict consensus *supertree* T of a collection $\mathcal{T} = \{T_1, T_2\}$ of two trees involves four steps:

(1) computing the restrictions T_1' and T_2' of the input trees to $L_\cap(\mathcal{T})$;

(2) computing the strict consensus *tree* T' of the trees T_1', T_2' (having the same set of leaves);

(3) collapsing suitable edges (by joining their two extremities) in T_1 and T_2 such that $T_1|L_\cap(\mathcal{T}) = T'$ and $T_2|L_\cap(\mathcal{T}) = T'$;

(4) T is obtained by grafting specific subtrees of the modified T_1 and T_2 in tree T', then collapsing edges in the parts of T' where both specific subtrees of T_1 and T_2 have been inserted.

Step 1. is clearly done in $O(n)$ by traversals of the trees. Several $O(n)$ algorithms are known for Step 2. (*e.g.* [6]). Step 3 is done by first anchoring nodes of T_1' and T_2' in T', then jointly traversing T_1 and T' and similarly for T_2 and T', hence requiring $O(n)$ time. Step 4. first performs two calls to MergeTrees, obtaining a tree T in $O(n)$ time. When keeping track of which input tree the

specific subtrees originate from, a single traversal of T (requiring $O(n)$ time) is then enough to decide which specific subtrees have to be collapsed. Collapsing a subtree is linear in the number of its edges, *i.e.* all collapsing operations in T require $O(n)$ total time. □

When considering collections of more than two trees, the MergeTrees algorithm will be used several times to attach specific subtrees from the different input trees to an initial backbone tree. The order in which input trees are processed does not change the set of leaves of the produced supertree. However, the shape of the supertree can vary depending on this order. This is not relevant to solve SMAST and SMCT, but can lead the supertree to possess some edges that can be considered as arbitrary from a phylogenetic standpoint. However, such edges can easily be detected and collapsed through known algorithms [36].

3.2 The inclusion of specific leaves

The following result states that all specific leaves of a collection are systematically included in any maximum agreement supertree or maximum compatible supertree of the collection. This result is not surprising since the information for positioning a specific leaf comes only from one input tree. Thus, no disagreement or incompatibility arises by positioning the leaf according to this input tree. Nonetheless, the proof requires handling a certain number of restrictions of trees and intersection of leaf sets.

Theorem 3 *Let $\mathcal{R}$ be a collection of rooted trees with overlapping sets of leaves. All specific leaves of $\mathcal{R}$ appear in any $SMAST(\mathcal{R})$ and in any $SMCT(\mathcal{R})$.*

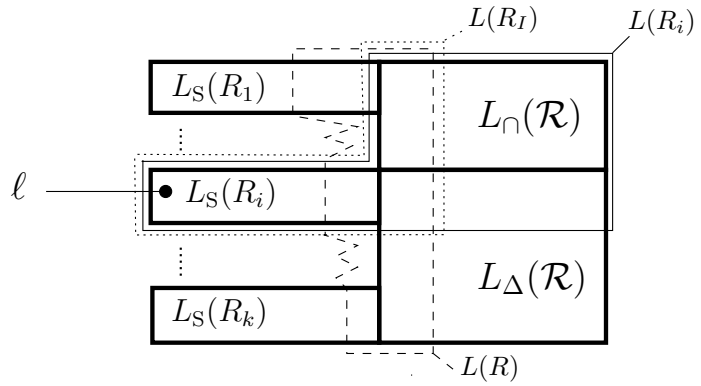

Fig. 5. The set of leaves $L(\mathcal{R})$ of a tree collection $\mathcal{R} = \{R_1, R_2, \ldots, R_k\}$, decomposed into the three sets $L_\cap(\mathcal{R}), L_\Delta(\mathcal{R})$ and $L_{\mathrm{S}}(\mathcal{R}) = \cup_{R_i \in \mathcal{R}} L_{\mathrm{S}}(R_i)$, all displayed in bold lines. The figure also displays the leaf set $L(R_i)$ of a tree $R_i \in \mathcal{R}$, in plain thin lines, and leaf sets $L(R_I)$, respectively $L(R)$, in dotted, respectively dashed lines, mentioned in the proof of Theorem 3, where R is a $SMCT(\mathcal{R})$ assumed to not cover all leaves of $L_{\mathrm{S}}(\mathcal{R})$ for the sake of contradiction.

Proof. The proof is given for the SMCT problem. The proof for SMAST is quite similar. Let R be a $SMCT(\mathcal{R})$. The proof proceeds by supposing that there is a specific leaf $\ell \in L_{\mathrm{S}}(\mathcal{R})$ such that $\ell \notin L(R)$ and shows that a single run of algorithm MergeTrees gives a tree containing ℓ that is both a $SMCT(\mathcal{R})$ and larger than R, which is in contradiction with the maximality of R. Several leaf sets involved in the proof are examplified in Figure 5.

Let R_i be the tree of $\mathcal{R}$ from which ℓ originates and let $R_I := R_i|\Big(L(R) \cup L_{\mathrm{S}}(R_i)\Big)$. Basic set operations show that

$$L(R_I) \cap L(R) = L(R_i) \cap L(R). \tag{5}$$

By definition of R, $R|L(R_i) \trianglerighteq R_i|L(R)$, *i.e.*

$$R|\Big(L(R)\cap L(R_i)\Big)\trianglerighteq R_i|\Big(L(R_i)\cap L(R)\Big) \tag{6}$$
$$R|L(R_I)\trianglerighteq R_i|\Big(L(R_I)\cap L(R)\Big) \tag{7}$$
$$R|L(R_I)\trianglerighteq R_I|L(R) \tag{8}$$

where (7) results from the use of (5) on both sides of (6), and (8) derives from $R_i|L(R_I) = R_I$.

Let R' be the tree returned by the call to MergeTrees (R_I, R). From (8), Theorem 1 applies and gives

$$L(R') = L(R) \cup L(R_I). \tag{9}$$

As, $\ell \in L(R_I) - L(R)$, we deduce from (9) that

$$\text{Card}\left(L(R')\right) > \text{Card}\left(L(R)\right). \tag{10}$$

Also, by definition of R_I, we have $L(R_I) = L(R_i)\cap\Big(L(R)\cup L_{\text{S}}(R_i)\Big) = \Big(L(R_i)\cap L(R)\Big) \cup L_{\text{S}}(R_i)$. Combined with (9), we then have

$$L(R') = L(R) \cup L_{\text{S}}(R_i). \tag{11}$$

From (8) and Theorem 1, we also know that R' is a $SMCT(\{R_I, R\})$, thus

$$R'|L(R_I)\trianglerighteq R_I|L(R'). \tag{12}$$

From (9) and the definition of R_I, basic set operations show that $L(R')\cap L(R_I) = L(R')\cap L(R_i)$, thus the left term of (12) can be rewritten as $R'|L(R_i)$; its right term can be rewritten as $R_i|L(R')$ (replacing R_I by its definition and

then using (9)), leading to

$$R'|L(R_i) \trianglerighteq R_i|L(R'). \tag{13}$$

Moreover, from Theorem 1, $R'|L(R) \trianglerighteq R|L(R')$. Since, $L(R) \subseteq L(R')$, this means that

$$R'|L(R) \trianglerighteq R. \tag{14}$$

Now consider any $R_j \in \mathcal{R}$ such that $R_j \neq R_i$. From $L_{\mathrm{S}}(R_i) \cap L(R_j) = \emptyset$ and (11) we obtain

$$L(R_j) \cap L(R) = L(R_j) \cap L(R'), \tag{15}$$

from which we deduce $(R'|L(R))\Big|L(R_j) = R'|\Big(L(R) \cap L(R_j)\Big) = R'|L(R_j)$. Thus, restricting both terms of (14) to $L(R_j)$, we obtain $R'|L(R_j) \trianglerighteq R|L(R_j)$. Combining this with $R|L(R_j) \trianglerighteq R_j|L(R)$ (which holds by definition of R) shows by transitivity that $R'|L(R_j) \trianglerighteq R_j|L(R)$. This can be rewritten as

$$R'|L(R_j) \trianglerighteq R_j|L(R') \tag{16}$$

since (15) also implies $R_j|L(R) = R_j|L(R')$.

Equation (13) for R_i and equation (16) for all $R_j \in \mathcal{R}, R_j \neq R_i$, show that R' is a supertree compatible with $\mathcal{R}$. Moreover, from (10), R' contains more leaves than $R := SMCT(\mathcal{R})$, a contradiction with the definition of R. $\square$

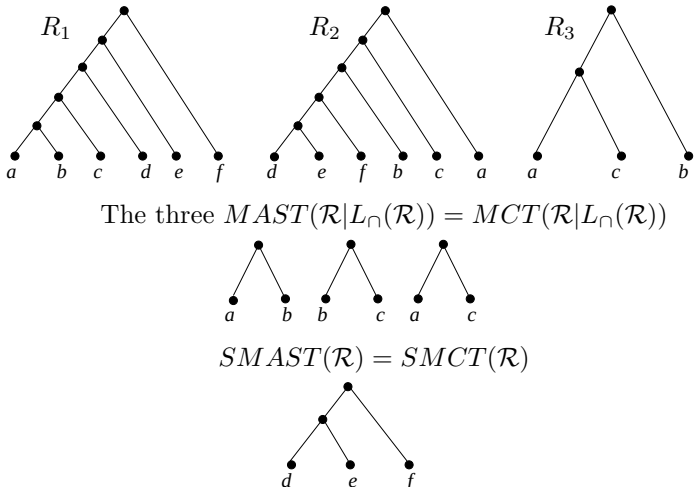

Fig. 6. A collection $\mathcal{R} = \{R_1, R_2, R_3\}$ of rooted input trees for which the trees $MAST(\mathcal{R})$ and $MCT(\mathcal{R})$ can not be used as backbones of $SMAST(\mathcal{R})$ and $SMCT(\mathcal{R})$.

3.3 Using MAST *and* MCT *as subproblems*

In the general case, it is not possible to solve SMAST, respectively SMCT, by considering MAST, respectively MCT, as a subproblem. For instance, Figure 6 shows a collection $\mathcal{R}$ with only three rooted trees, where the trees $SMAST(\mathcal{R})$ and $SMCT(\mathcal{R})$ do not include $MAST(\mathcal{R})$ and $MCT(\mathcal{R})$ as restrictions. However, in the particular case where every leaf of the collection belongs either to a single tree or to all trees of the collection, the connection between subtree and supertree problems can be exploited. See algorithm BuildSMCT to solve SMCT. The algorithm proceeds from a maximum compatible tree of the input trees restricted to common leaves. Then, specific subtrees of each original input tree are added by successive calls to the MergeTrees algorithm.

To prove the correctness of BuildSMCT, we first need to establish the three following invariants:

Algorithm 2: BUILDSMCT ($\mathcal{R}$)

Input: A collection $\mathcal{R} = \{R_1, R_2, \ldots, R_k\}$ of rooted trees such that $L_\Delta(\mathcal{R}) = \emptyset$.

Result: A tree $SMCT(\mathcal{R})$.

1 $R_M^0 \leftarrow MCT(\mathcal{R}|L_\cap(\mathcal{R}))$

for i *in* $1 \rightarrow k$ **do**

2 $\quad R_M^i \leftarrow$ MERGETREES $\left(R_i \middle| (L(R_M^0) \cup L_{\mathrm{S}}(R_i)), R_M^{i-1}\right)$

return R_M^k.

Lemma 1 *Given a collection* $\mathcal{R} = \{R_1, R_2, \ldots, R_k\}$ *such that* $L_\Delta(\mathcal{R}) = \emptyset$, *the following statements hold at each iteration* i $(1 \leq i \leq k)$ *of the loop of algorithm* BUILDSMCT*:*

(A) let $R_I := R_i \middle| (L(R_M^{i-1}) \cup L_{\mathrm{S}}(R_i))$ *and* $R_A := R_M^{i-1}$ *be the trees given as input to* MERGETREES *in line 2,* $R_A|L(R_I) \trianglerighteq R_I|L(R_A)$ *holds;*

(B) $L(R_M^i) = L(R_M^0) \cup \bigcup_{j \leq i, R_j \in \mathcal{R}} L_{\mathrm{S}}(R_j)$*;*

(C) R_M^i *is a supertree compatible with* $\mathcal{R}$*.*

The proof of the Lemma is done by induction on the iterations of the loop in algorithm BUILDSMCT, and is included in Appendix C. The correctness of the algorithm BUILDSMCT directly derives from Lemma 1. Moreover, its running time mainly depends on that of the algorithm for solving MCT on an instance of the same or smaller size.

Theorem 4 *Let* $\mathcal{R} = \{R_1, R_2, \ldots, R_k\}$ *be a collection of rooted trees such that* $L_\Delta(\mathcal{R}) = \emptyset$*. Algorithm* BUILDSMCT($\mathcal{R}$) *computes a maximum compatible supertree of* $\mathcal{R}$ *in* $O(N+kn)$ *time where* n *is the maximum number of leaves in a tree of* $\mathcal{R}$*, and* N *is the time needed to compute a rooted maximum compatible tree of* $\mathcal{R} \,|\, L_\cap(\mathcal{R})$*.*

Proof. From Lemma 1-(C), the tree R^k_M returned by the algorithm is a supertree compatible with $\mathcal{R}$. Moreover, it is of maximum size among such supertrees. Indeed, suppose there is a tree $R = SMCT(\mathcal{R})$ such that $|R| > |R^k_M|$. Since $L(R^k_M) = L_{\mathrm{S}}(\mathcal{R}) \cup L(R^0_M)$ (from Lemma 1-(B)) and $L(\mathcal{R}) = L_{\mathrm{S}}(\mathcal{R}) \cup L_{\cap}(\mathcal{R})$ (from $L_{\Delta}(\mathcal{R}) = \emptyset$) then R contains more leaves of $L_{\cap}(\mathcal{R})$ than R^k_M, *i.e.*

$$\left|R|L_{\cap}(\mathcal{R})\right| > \left|R^k_M|L_{\cap}(\mathcal{R})\right| = \left|R^0_M|L_{\cap}(\mathcal{R})\right|. \tag{17}$$

However, as $\mathcal{R}|L_{\cap}(\mathcal{R})$ is a collection of trees on the same leaf set, $L_{\cap}(\mathcal{R})$, and $L(R|L_{\cap}(\mathcal{R})) \subseteq L_{\cap}(\mathcal{R})$, the fact that R is a supertree compatible with $\mathcal{R}$ implies that $R|L_{\cap}(\mathcal{R})$ is a tree compatible with the collection $\mathcal{R}|L_{\cap}(\mathcal{R})$. But then (17) is in contradiction with the maximality of R^0_M among the trees compatible with this collection. Thus, R^k_M is a maximum compatible supertree of $\mathcal{R}$.

Concerning the running time, the kn term results from both the restrictions of input trees and from calls to MergeTrees: lines 1 and 2 restrict each of the k input trees to a subset of its leaves, necessitating a single $O(n)$ traversal of the tree each time; moreover, line 2 performs k calls to MergeTrees, each requiring a time proportional to the size $O(n)$ of the trees given as input to the call, by Theorem 2. The N term results from the computation of a maximum compatible tree of $\mathcal{R}|L_{\cap}(\mathcal{R})$ in line 1. □

Note that the kn term in the complexity of BuildSMCT can be reduced to

an n term by integrating this algorithm with MergeTrees and computing all anchors between the trees of $\mathcal{R}$ and R_M^0 before performing the grafts of specific subtrees. However, the N majoring term would remain.

A simple modification of the algorithm BuildSMCT yields an algorithm, BuildSMAST, that solves SMAST: in line 1, use a tree $MAST(\mathcal{R}|L_\cap(\mathcal{R}))$ instead of a tree $MCT(\mathcal{R}|L_\cap(\mathcal{R}))$ to initialize R_M^0.

Theorem 5 *Let $\mathcal{R} = \{R_1, R_2, \ldots, R_k\}$ be a collection of rooted trees such that $L_\Delta(\mathcal{R}) = \emptyset$. Algorithm* BuildSMAST($\mathcal{R}$) *computes a maximum agreement supertree of $\mathcal{R}$ in $O(N'+kn)$ time where n is the maximum number of leaves in a tree of $\mathcal{R}$ and N' is the time needed to compute a rooted maximum agreement subtree of $\mathcal{R} \,|\, L_\cap(\mathcal{R})$.*

Proof. The correctness of BuildSMAST is shown in a very similar way as that of BuildSMCT, replacing in the above results (*e.g.* in Lemma 1-(A)) each refinement relation between trees by an equality (*i.e.* isomorphism) of these trees. The complexity proof is quite similar to that of Theorem 4, with N being replaced with N'. □

Now consider the case where the input trees are unrooted. By rooting unrooted trees on the edge leading to a common leaf, isomorphism and refinement relations between unrooted trees translate into the same relations between corresponding rooted trees [9, Lemma 4]. Hence, the SMAST and SMCT problems on unrooted trees can be easily reduced to the same problems on rooted trees.

Theorem 6 *Let $\mathcal{U} = \{U_1, U_2, \ldots, U_k\}$ be a collection of unrooted trees sets such that $L_\Delta(\mathcal{U}) = \emptyset$.*

One can compute a tree $SMAST(\mathcal{U})$, respectively $SMCT(\mathcal{U})$, in $O(M + kn)$ time where M is the time needed to compute an unrooted *tree $MAST(\mathcal{U} \,|\, L_\cap(\mathcal{U}))$, respectively $MCT(\mathcal{U} \,|\, L_\cap(\mathcal{U}))$.*

Proof. Consider the case of the SMCT problem (the proof for SMAST is similar). Make the following modifications to BuildSMCT: first compute $U_M^0 = MCT(\mathcal{U}|L_\cap(\mathcal{U}))$ by applying an algorithm to solve the problem on unrooted trees [20]. Then choose an arbitrary leaf $\ell \in L(U_M^0)$, compute the collection $\mathcal{R} = \{R_1, R_2, \ldots, R_k\}$ of rooted trees such that $R_i \in \mathcal{R}$ is obtained by rooting $U_i \in \mathcal{U}$ (inserting a new node) on the external edge leading to leaf ℓ. Similarly, R_M^0 is initialized as the tree obtained by rooting U_M^0 on the edge leading to ℓ. Then the `for` loop remains the same. The last modification is to unroot the obtained tree R_M^k before returning it.

Concerning the correctness of the modified algorithm, first note that $\mathcal{R}_M^0 = MCT(\mathcal{R}|L_\cap(\mathcal{R}))$ [9, Lemma 5]. Now, the k calls to algorithm MergeTrees give a tree R_M^k such that $R_M^k = SMCT(\mathcal{R})$ (Theorem 4) and such that $L(R_M^k) = L(R_M^0) \cup L_{\mathrm{S}}(\mathcal{R})$ (Lemma 1-(B)). Let U_M^k be the tree obtained by unrooting R_M^k. Since refinement relations are preserved by unrooting trees [9, Lemma 4], U_M^k is a supertree compatible with $\mathcal{U}$. Moreover, it is of maximum size. Indeed, a maximum compatible supertree U' of $\mathcal{U}$ including more leaves

than U_M^k would necessarily contain more leaves of $L_\cap(\mathcal{U})$ than U_M^k (because $L(\mathcal{U}) = L_\cap(\mathcal{U}) \cup L_\mathrm{S}(\mathcal{U})$ and $L_\mathrm{S}(\mathcal{U}) \subseteq L(U_M^k) = L(R_M^k) = L(R_M^0) \cup L_\mathrm{S}(\mathcal{R}) = L(U_M^0) \cup L_\mathrm{S}(\mathcal{U})$). Thus, U' would contain more leaves of $L_\cap(\mathcal{U})$ than U_M^0 does, implying that $U'|L_\cap(\mathcal{U})$ would be a tree compatible with $\mathcal{U}|L_\cap(\mathcal{U})$ of larger size than U_M^0, which is a contradiction with the definition of the latter.

The running time differs from the original BUILDSMCT by the fact that the MCT is computed on unrooted trees, requiring $O(M)$ time instead of $O(N)$. Choosing ℓ is $O(1)$ time, computing $\mathcal{R}$ is $O(kn)$ time, and unrooting R_M^k is $O(1)$ time. Taking restrictions of trees in line 2 is $O(kn)$. Thus, the modified algorithm requires $O(M + kn)$ time. □

The previous theorems enable to state the relationships between subtree and supertree problems for a collection $\mathcal{T}$ when $L_\Delta(\mathcal{T}) = \emptyset$.

Corollary 2 *Let $\mathcal{T}$ be a collection of trees such that $L_\Delta(\mathcal{T}) = \emptyset$. Any tree $MAST(\mathcal{T} \,|\, L_\cap(\mathcal{T}))$, respectively $MCT(\mathcal{T} \,|\, L_\cap(\mathcal{T}))$, is the restriction to $L_\cap(\mathcal{T})$ of some tree $SMAST(\mathcal{T})$, respectively $SMCT(\mathcal{T})$.*

Note that the condition required for Corollary 2 to apply is always fulfilled for collections $\mathcal{T}$ of only two trees, because $L(\mathcal{T}) = L_\cap(\mathcal{T}) \cup L_\mathrm{S}(\mathcal{T})$. Figure 7 shows an illustration of the corollary in such a case. Lastly, note that it is also true that in this case the restriction to $L_\cap(\mathcal{T})$ of any tree $SMAST(\mathcal{T})$, respectively $SMCT(\mathcal{T})$, is a tree $MAST(\mathcal{T}|L_\cap(\mathcal{T}))$, respectively $MCT(\mathcal{T}|L_\cap(\mathcal{T}))$.

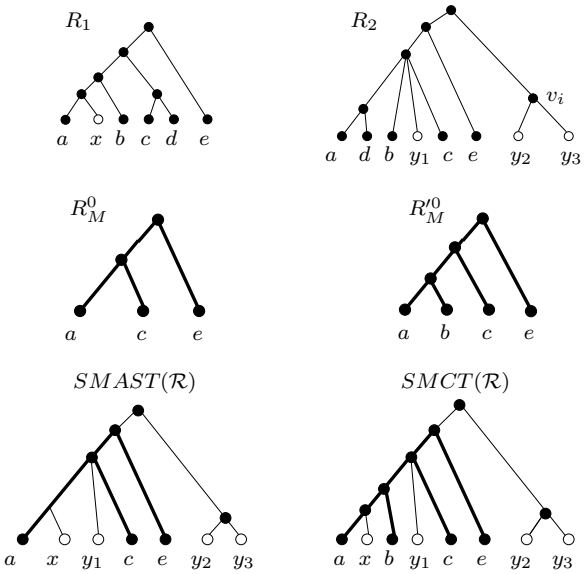

Fig. 7. A collection $\mathcal{R} = \{R_1, R_2\}$ of two input trees ($L_\cap(\mathcal{R}) = \{a, b, c, d, e\}$), two trees $R^0_{\rm M} := MAST(\mathcal{R}|L_\cap(\mathcal{R}))$ and $R'^0_{\rm M} := MCT(\mathcal{R}\,|\,L_\cap(\mathcal{R}))$ and two trees $SMAST(\mathcal{R})$ and $SMCT(\mathcal{R})$, in which the structure of $R^0_{\rm M}$, respectively $R'^0_{\rm M}$, is displayed in bold lines.

3.4 Polynomial cases

Particular cases where SMAST and SMCT problems can be solved in polynomial time are deduced from the above results and from works on MAST and MCT.

Corollary 3 *Let $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ be a collection of trees (all rooted or all unrooted).*

(i) *The* SMAST *problem on two trees can be solved in polynomial time.*

Moreover, when $L_\Delta(\mathcal{T}) = \emptyset$,

(ii) *the* SMAST *problem can be solved in polynomial time whenever the maximum degree of an input tree is bounded,*

(iii) *the* SMCT *problem on* $\mathcal{T}$ *can be solved in polynomial time whenever the degree of all input trees is bounded.*

Proof. Consider the case where trees in $\mathcal{T}$ are rooted. In this case: (i) the result derives from Theorem 5 and from efficient algorithms that solve MAST when $\mathcal{T}$ contains only two trees [30,31,15]; (ii) follows from Theorem 5 and the algorithms of [2,13]; (iii) follows from Theorem 4 and [23]. If $\mathcal{T}$ is a collection of unrooted trees, then the result follows from Theorem 6 and the works cited in the rooted case. □

4 Reduction to instances involving smaller trees

We now describe how trees of arbitrary size can be described by subtrees of small bounded size. This decomposition will be used to prove intractability results in the next section. Rooted trees of arbitrary size can be described by rooted trees on three leaves.

Definition 13 (Rooted triples and fans) *A* rooted triple *(or* resolved triple*) is a binary rooted tree on three leaves. A* fan *(also called* unresolved triple*) is a rooted tree on three leaves with only one internal node. On three given distinct leaves* a, b *and* c*, there are three possible rooted triples, denoted* $bc|a$*, respectively* $ac|b$*, respectively* $ab|c$*, depending on their innermost grouping of two leaves (*bc*, respectively* ac*, respectively* ab*).* E.g. *tree* R_M^0 *of Figure 7 is the rooted triple* $ac|e$*. The only one possible fan on this set of leaves is denoted*

(a, b, c).

Let R be a rooted tree. For any set $\{a, b, c\}$ of three leaves in $L(R)$, $R \mid \{a, b, c\}$ is either a rooted triple or a fan. We define $rt(R)$, respectively $f(R)$, as the set of rooted triples, respectively fans, of R induced by the 3-leaf subsets of $L(R)$.

For instance, in Figure 2,

$$rt(R_2) = \{ad|b, ad|c, ad|e, ab|e, ac|e, bd|e, cd|e, bc|e\},$$
$$f(R_2) = \{(a, b, c), (b, c, d)\}.$$

The basic building stones of unrooted trees are *quartets* and *stars*:

Definition 14 (Unrooted quartets and stars) *A* quartet *is a binary unrooted tree on four leaves. A* star *is an unrooted tree with only one internal node to which four leaves are connected. Given four distinct leaves a, b, c and d, there are three possible quartets, respectively denoted $ab|cd$ (corresponding to the binary tree where the path from a to b does not intersect the path from c to d), $ac|bd$ and $ad|bc$, and only one possible star denoted (a, b, c, d).*

Let U be an unrooted tree. For any set Q of four leaves appearing in U, $U \mid Q$ is either a quartet or a star. We define $q(U)$, respectively $s(U)$, as the set of quartets, respectively stars, of U induced by 4-leaf subsets of $L(U)$.

For instance, in Figure 1,

$$q(U) = \{ad|bc, ad|be, ad|bf, ad|ce, ad|cf, ad|ef, bc|ef, bd|ef, cd|ef, ab|ef, ac|ef\},$$
$$s(U) = \{(a, b, c, e), (a, b, c, f), (b, c, d, e), (b, c, d, f)\}.$$

The following well-known results show that isomorphism and compatibility of rooted, respectively unrooted, trees can be expressed by relations on sets of induced rooted triples and fans, respectively quartets and stars:

Lemma 2 *Let R, R' be two rooted trees.*

(i) *R is isomorphic to R' if and only if $rt(R) = rt(R')$ and $f(R) = f(R')$.*

(ii) *R refines R' if and only if $rt(R') \subseteq rt(R)$ and $L(R) = L(R')$.*

Let U, U' be two unrooted trees.

(iii) *U is isomorphic to U' if and only if $q(U) = q(U')$ and $s(U) = s(U')$.*

(iv) *U refines U' if and only if $q(U') \subseteq q(U)$ and $L(U) = L(U')$.*

Proof. (i) derives from [13, Lemma 6.6], (iii) is [2, Theorem 2] and [14, Theorem 1] yields (ii) and (iv). □

We can now show that solving SMAST and SMCT on instances with input trees of arbitrary degree is equivalent to solving the same problems on trees with both degree and size bounded by a small constant. This contrasts with MAST and MCT which are trivial when the input trees contain a bounded number of leaves. Moreover, MAST is polynomial in the case where an input tree has a bounded degree [2,13]. Note also that having trees with bounded degree is a sufficient condition for the algorithm of [23] to solve MCT in polynomial time.

We first define collections of rooted triples and fans, respectively unrooted quartets and stars that can be obtained from a collection of general rooted trees, respectively general unrooted trees:

Definition 15 *Given a collection $\mathcal{R}$ of rooted trees, define:*

$$rt_{\cup}(\mathcal{R}) = \bigcup_{R_i \in \mathcal{R}} rt(R_i) \qquad \text{and} \qquad f_{\cup}(\mathcal{R}) = \bigcup_{R_i \in \mathcal{R}} f(R_i)\,.$$

Similarily, given a collection $\mathcal{U}$ of unrooted trees, define:

$$q_{\cup}(U) = \bigcup_{U_i \in \mathcal{U}} q(U_i) \qquad \text{and} \qquad s_{\cup}(U) = \bigcup_{U_i \in \mathcal{U}} s(U_i)\,.$$

From Lemma 2, we deduce:

Corollary 4 *Let $\mathcal{R}$ be a collection of rooted trees and let R be a rooted tree with $L(R) \subseteq L(\mathcal{R})$.*

(i) *R is an agreement supertree of $\mathcal{R}$ if and only if R is an agreement supertree of $rt_{\cup}(\mathcal{R}) \cup f_{\cup}(\mathcal{R})$,*

(ii) *R is a supertree compatible with $\mathcal{R}$ if and only if R is a supertree compatible with $rt_{\cup}(\mathcal{R})$.*

Let $\mathcal{U}$ be a collection of unrooted trees and let U be an unrooted tree with $L(U) \subseteq L(\mathcal{U})$.

(iii) *U is an agreement supertree of $\mathcal{U}$ if and only if U is an agreement supertree of $q_{\cup}(\mathcal{U}) \cup s_{\cup}(\mathcal{U})$,*

(iv) *U is a supertree compatible with $\mathcal{U}$ if and only if U is a supertree compatible*

with $q_\cup(\mathcal{U})$.

Proof. Assertions (i), (ii), (iii) and (iv) of Corollary 4 are easily deduced from statements (i), (ii), (iii) and (iv) of Lemma 2, respectively. □

If k and n respectively denote the number of trees and the number of leaves in the collection, note that

- $rt_\cup(\mathcal{R})$ and $f_\cup(\mathcal{R})$ are computable in $O(kn^3)$ time from $\mathcal{R}$;
- $q_\cup(\mathcal{U})$ and $s_\cup(\mathcal{U})$ are computable in $O(kn^4)$ time from $\mathcal{U}$.

Since approximating MAST on rooted trees is at least as hard as approximating Maximum Clique [21], approximating SMAST on rooted triples and fans is at least as hard as approximating Maximum Clique. In the same way, building on a result of [7], approximating SMCT on rooted triples is at least as hard as approximating Maximum Independent Set.

5 Intractability of SMAST and SMCT

In this section, we show that there is a substantial gap in complexity between MAST and SMAST, respectively MCT and SMCT.

The complement of the SMAST problem, denoted CSMAST, is defined as the minimization problem obtained from SMAST by changing, in Definition 8, "**Measure**: $|T|$, *to be maximized*", into "**Measure**: $\mathrm{Card}\,(L(\mathcal{T})) - |T|$, *to be minimized*". The complement of SMCT, denoted CSMCT, is obtained in the

same way from Definition 9. Note that trees involved in practical phylogenetic instances are expected to conflict on a small proportion of leaves. Thus, $\mathrm{Card}\,(L(\mathcal{T})) - |SMAST(\mathcal{T})|$ and $\mathrm{Card}\,(L(\mathcal{T})) - |SMCT(\mathcal{T})|$ are expected to be small. Hence, approximating CSMAST and CSMCT is more interesting than approximating SMAST and SMCT.

The complement of MAST, respectively MCT, is defined to be the restriction of CSMAST, respectively CSMCT, to instances consisting in collections of trees sharing the same leaf set. The complement of MCT is approximable within ratio 3 [19], as is also well-known for the complement of MAST ([2,7]). The latter result was also recently improved to a ratio $3 - \frac{6 \log\log n}{\log n}$ [28]. In contrast to these positive results, CSMAST, respectively CSMCT, in its general form is NP-hard to approximate within any constant ratio, as shown below in Theorem 9.

Moreover, consider the decision problem corresponding to CSMAST:

Instance: A finite collection $\mathcal{T}$ of trees and an integer $p \geq 0$.

Question: Is there an agreement supertree of $\mathcal{T}$ of size at least $\mathrm{Card}\,(L(\mathcal{T})) - p$?

The decision problem corresponding to CSMCT is defined in the same way (replace "agreement supertree of $\mathcal{T}$" by "supertree compatible with $\mathcal{T}$" in the above statement of the "**Question**:"). Theorem 8 below shows that CSMAST and CSMCT are hard for parameter p unlike the complements of MAST and

MCT which are FPT in p (see [9] for the latest algorithms).

5.1 The Hitting Set *problem*

As often done in previous works (*e.g.* [2,13]), we exploit links between the MAST problem and the Hitting Set problem. A *hitting set* of a collection of sets $\mathcal{C}$ is a set H such that for all $C \in \mathcal{C}$, $H \cap C$ is non-empty. Consider the decision problem:

Name: Hitting Set

Instance: A finite collection $\mathcal{C}$ of finite sets and an integer $p \geq 0$.

Question: Is there a hitting set of $\mathcal{C}$ of cardinality at most p?

Hitting Set is an alternative formulation of Set Cover. It is thus NP-complete [16] and W[2]-complete for parameter p [17, Proposition 10]. Moreover, its optimization version can not be approximated within any constant ratio unless P = NP [5].

5.2 A graph representing rooted triples

Definition 16 ([1,14,37]) *Let $\mathcal{R}$ be a finite collection of rooted triples and let $L \subseteq L(\mathcal{R})$.*

Let $[\mathcal{R}, L]$ be the undirected graph such that:

- *there is a vertex for every element of L,*

- *there is an edge between two vertices u and v if and only if there exists $\ell \in L$ such that $uv|\ell \in \mathcal{R}$.*

Theorem 2 in [14] can be restated as follows:

Theorem 7 ([14]) *Let $\mathcal{R}$ be a collection of rooted triples and $L \subseteq L(\mathcal{R})$.*

There is an agreement supertree of $\mathcal{R}$ with leaf set L if and only if for each subset $L' \subseteq L$ of cardinality at least 3, the graph $[\mathcal{R}, L']$ is disconnected.

5.3 The gadget

Definition 17 *We recursively define the function* rake *associating a rooted tree to a given non-empty ordered sequence of rooted trees with non-intersecting leaf sets:*

- $\text{rake}(R_1) = R_1$ *for any rooted tree R_1 (sequence of length 1).*
- $\text{rake}(R_1, R_2, \ldots, R_k)$ *is the rooted tree whose root has R_1 and* $\text{rake}(R_2, R_3, \ldots, R_k)$ *as two child subtrees for any sequence of rooted trees $R_1, R_2, \ldots, R_k$ of length $k \geq 2$ such that*

$$\forall i, j \in [1, k] \qquad i \neq j \iff L(R_i) \cap L(R_j) = \emptyset\,.$$

Figure 8 illustrates the previous definition. We now describe the gadget that is used to reduce Hitting Set to SMAST:

Definition 18 (Gadget) *Let m be an integer such that $m \geq 1$ and let $x^1, x^2, \ldots, x^m, y^1, y^2, \ldots, y^m$ be $2m$ distinct labels. We define $\mathcal{G} =$*

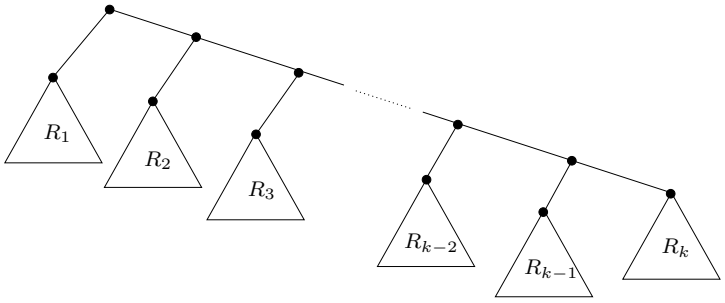

Fig. 8. The tree rake$(R_1, R_2, \ldots, R_k)$.

$\mathcal{G}(x^1, x^2, \ldots, x^m, y^1, y^2, \ldots, y^m)$ *to be the following collection of rooted triples:*

$$\left\{y^{h+1}x^{h+1}|y^h, x^{h+1}x^{h+2}|y^h\right\}_{h\in[1,m]}$$

setting $x^{m+1} := x^1$, $x^{m+2} := x^2$ *and* $y^{m+1} := y^1$.

Lemma 3 $\mathcal{G}$ *has the following properties:*

(i) *There is no agreement supertree of* $\mathcal{G}$ *having leaf set* $L(\mathcal{G})$.

(ii) *Let* $j \in [1, m]$. *The following trees with leaf set* $L(\mathcal{G}) - \{x^j\}$ *are agreement supertrees of* $\mathcal{G}$*:*

$$\text{rake}(y^j, y^{j+1}, \ldots, y^m, y^1, y^2, \ldots, y^{j-1}, R^*)$$

where R^* *is any rooted tree on* $\{x^1, x^2, \ldots, x^m\} - \{x^j\}$.

Proof. (i) The graph $[\mathcal{G}, L(\mathcal{G})]$ associated with $\mathcal{G}$ is connected (see Figure 9). Therefore, by Theorem 7, there is no agreement supertree of $\mathcal{G}$ having leaf set $L(\mathcal{G})$.

(ii) Assume w.l.o.g. that $j = 1$ ($\mathcal{G}$ is not altered by a common circular permutation of the two sequences x^1, x^2,..., x^m and y^1, y^2, ..., y^m). Fixing an

arbitrary rooted tree R^* on $\{x^2, x^3, \ldots, x^m\}$, we have to show that the tree

$$R_{\mathrm{A}} := \mathrm{rake}(y^1, y^2, \ldots, y^m, R^*)$$

on $L(\mathcal{G}) - \{x^1\}$ is an agreement supertree of $\mathcal{G}$. For this purpose, we distinguish trees in $\mathcal{G}$ that do not contain x^1 from those that do.

In the one hand, it is easily seen that

$$\begin{aligned} &\forall h \in [1, m-1] \qquad y^{h+1}x^{h+1}|y^h = R_{\mathrm{A}} \,\Big|\, \{y^h, y^{h+1}, x^{h+1}\} \ , \\ &\forall h \in [1, m-2] \qquad x^{h+1}x^{h+2}|y^h = R_{\mathrm{A}} \,\Big|\, \{y^h, x^{h+1}, x^{h+2}\} \ . \end{aligned}$$

On the other hand, $x^1 = x^{m+1}$ is a leaf of $y^{m+1}x^{m+1}|y^m$ and of $x^{h+1}x^{h+2}|y^h$ for $h \in \{m-1, m\}$. Hence, restricting these three trees to $L(\mathcal{G}) - \{x^1\}$ reduces them to only two leaves, belonging to R_{A}.

We have shown that $\forall G_i \in \mathcal{G},\ R_{\mathrm{A}} \,|\, L(G_i) = G_i \,|\, L(R_{\mathrm{A}})$. As also $L(R_{\mathrm{A}}) \subseteq L(\mathcal{G})$, this proves that R_{A} is an agreement supertree of $\mathcal{G}$. □

In other words, $\mathcal{G}$ is a collection of conflicting trees in the sense that there is no tree R with the entire $L(\mathcal{G})$ as leaf set and displaying all trees of $\mathcal{G}$. However, choosing only one leaf x^j (any one) and removing from $\mathcal{G}$ all triples containing x^j guarantees that such a tree exists. It is formed by making leaves y^h (with $h \in [1, m]$) pending in a specific order from the successive internal nodes of the tree (starting from the root and going downward), last appending a subtree containing the leaves x^h (with $h \in [1, m], h \neq j$) but which can have any shape.

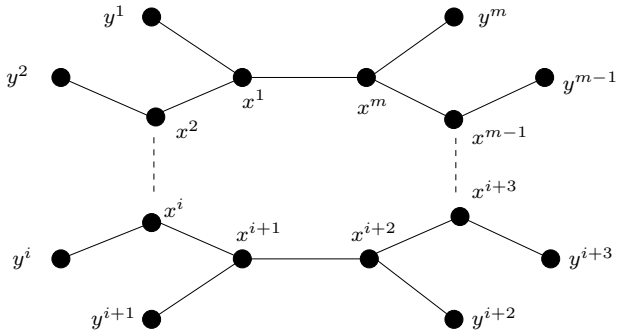

Fig. 9. The graph $[\mathcal{G}, L(\mathcal{G})]$ induced by the gadget $\mathcal{G}$.

5.4 The reductions

Theorem 8 *The* CSMAST *problem is* NP*-hard and* W[2]*-hard for parameter* p*, even for instances* $\mathcal{T}$ *only composed of rooted triples, respectively unrooted quartets.*

Proof.

Rooted case. We reduce Hitting Set to CSMAST, polynomially and preserving the parameter p. Let $(\mathcal{C}, p)$ be an instance of Hitting Set,

$$\begin{aligned}\mathcal{C} &= \{X_1, X_2, \ldots, X_c\} \\ &= \left\{\left\{x_1^1, x_1^2, \ldots, x_1^{m_1}\right\}, \left\{x_2^1, x_2^2, \ldots, x_2^{m_2}\right\}, \ldots, \left\{x_c^1, x_c^2, \ldots, x_c^{m_c}\right\}\right\} .\end{aligned}$$

where $c := \mathrm{Card}\,(\mathcal{C})$ and $m_i := \mathrm{Card}\,(X_i)$. Then let (y_i^j) be an injective family of labels not appearing in $X_1 \cup X_2 \cup \ldots \cup X_c$, indexed on the set of ordered pairs (i, j) with $i \in [1, c], j \in [1, m_i]$. Based on the model of $\mathcal{C}$, we build a collection of non-intersecting sets

$$\left\{\left\{y_1^1, y_1^2, \ldots, y_1^{m_1}\right\}, \left\{y_2^1, y_2^2, \ldots, y_2^{m_2}\right\}, \ldots, \left\{y_c^1, y_c^2, \ldots, y_c^{m_c}\right\}\right\}$$

whose elements are distinct from those of $\mathcal{C}$. Let

$$\mathcal{G}_i := \mathcal{G}(x_i^1, x_i^2, \ldots, x_i^{m_i}, y_i^1, y_i^2, \ldots, y_i^{m_i})$$

for all $i \in [1, c]$ and consider the collection of trees

$$\mathcal{R} := \mathcal{G}_1 \cup \mathcal{G}_2 \cup \ldots \cup \mathcal{G}_c \,.$$

From the definition of gadgets (Definition 18), the transformation of the instance $(\mathcal{C}, p)$ of HITTING SET to the instance $(\mathcal{R}, p)$ of the decision problem CSMAST obviously takes a polynomial time ($\mathcal{R}$ is of cardinality $2(m_1 + m_2 + \cdots + m_c)$) and preserves parameter p. By construction, all trees in $\mathcal{R}$ are rooted triples. It remains to be proven that the following two statements are equivalent:

(1) $\mathcal{C}$ admits a hitting set of size at most p and

(2) there is an agreement supertree of $\mathcal{R}$ whose size is at least $\text{Card}\,(L(\mathcal{R})) - p$.

(2) $\Rightarrow$ (1). Let R_{A} be an agreement supertree of $\mathcal{R}$ of size of at least $\text{Card}\,(L(\mathcal{R})) - p$. Thus, $H := L(\mathcal{R}) - L(R_{\text{A}})$ is a set of cardinality at most p. Moreover, for any $i \in [1, c]$, we know that $L(\mathcal{G}_i)$ is not a subset of $L(R_{\text{A}})$ from Lemma 3-(i). Then at least one element of $L(\mathcal{G}_i)$ is not a leaf in R_{A}, hence is in H. This shows that H is a hitting set of $\{L(\mathcal{G}_1), L(\mathcal{G}_2), \ldots, L(\mathcal{G}_c)\}$.

Now change H in the following way: replace each $y_i^j \in H$ (which only hits the set $L(\mathcal{G}_i)$) with any element in X_i. H is then a hitting set of $\mathcal{C}$ of size at most p.

(1) $\Rightarrow$ (2). Given $i \in [1, c]$ and $j \in [1, m_i]$, we denote σ_i^j the $(j-1)$th cyclic shift of the sequence $y_i^1, y_i^2, \ldots, y_i^{m_i}$:

$$\sigma_i^j = y_i^j, y_i^{j+1}, \ldots, y_i^{m_i}, y_i^1, y_i^2, \ldots, y_i^{j-1}.$$

Let H be a hitting set of $\mathcal{C}$ of cardinality at most p. For each $i \in [1, c]$, H contains at least an element of X_i, that we denote $x_i^{j_i}$, with $j_i \in [1, m_i]$. Concatenate the c sequences $\sigma_1^{j_1}$, $\sigma_2^{j_2}$, $\ldots$, $\sigma_c^{j_c}$ in any order: this yields a label sequence z_1, z_2, $\ldots$, z_m of length $m := m_1 + m_2 + \cdots + m_c$. Then form the tree

$$R_{\mathrm{A}} := \mathrm{rake}(z_1, z_2, \ldots, z_m, R^*)$$

where R^* is any rooted tree on $(X_1 \cup X_2 \cup \ldots \cup X_c) - H$.

By construction, R_{A} is a tree on $L(\mathcal{R}) - H$ and thus of size at least $\mathrm{Card}\,(L(\mathcal{R})) - p$. Moreover, for each $i \in [1, c]$, R_{A} (consisting of leaves of the kind y_i^j hanging one by one from internal nodes on the path from the root of R_{A} to the root of subtree R^*) is such that $R_{\mathrm{A}} \,|\, L(\mathcal{G}_i) = \mathrm{rake}(y_i^{j_i}, y_i^{j_i+1}, \ldots, y_i^{m_i}, y_i^1, y_i^2, \ldots, y_i^{j_i-1} R^* \,|\, X_i)$, which is an agreement supertree of $\mathcal{G}_i$ by Lemma 3-(ii) (note that $x_i^{j_i} \in H$ is not a leaf in R^*, hence not in $R^* \,|\, X_i$). Thus, R_{A} is an agreement supertree of $\mathcal{R}$ of size at least $\mathrm{Card}\,(L(\mathcal{R})) - p$.

Unrooted case. We reduce below the version of CSMAST using a collection of rooted triples as input, to the version of CSMAST using a collection of binary

unrooted trees as input. Note that solving CSMAST on binary unrooted trees is equivalent to solving this problem on unrooted quartets (Corollary 4-(iii)).

Let $(\mathcal{R}, p)$ be an instance of CSMAST where $\mathcal{R}$ is a collection of rooted triples and p a non-negative integer.

Let R' be a rooted binary tree containing $\mathrm{Card}\,(L(\mathcal{R}))$ *new* leaves: $|R'| = \mathrm{Card}\,(L(\mathcal{R}))$ and $L(R') \cap L(\mathcal{R}) = \emptyset$. Consider the collection $\mathcal{U}$ of unrooted trees where the trees $U_i \in \mathcal{U}$ are in one to one correspondence with the trees $R_i \in \mathcal{R}$: given a tree $R_i \in \mathcal{R}$, the corresponding tree U_i is the unrooted tree obtained by adding an edge between the root of R_i and the root of a copy of R'. We have $L(\mathcal{U}) = L(R') \cup L(\mathcal{R})$ and $\mathrm{Card}\,(L(\mathcal{U})) = 2\mathrm{Card}\,(L(\mathcal{R}))$.

The instance $(\mathcal{R}, p)$ of CSMAST is transformed into an another instance $(\mathcal{U}, p)$ of CSMAST where all trees in $\mathcal{U}$ are *unrooted*. This transformation is clearly done in polynomial time and preserves parameter p.

We now prove that $(\mathcal{R}, p)$ is a positive instance of CSMAST if and only if $(\mathcal{U}, p)$ is a positive instance of CSMAST.

First, suppose that there is an agreement supertree R_{A} of $\mathcal{R}$ with $|R_{\mathrm{A}}| \geq \mathrm{Card}\,(L(\mathcal{R})) - p$. Then the unrooted tree U_{A}, obtained by connecting the roots of R_{A} and a copy of R' by an edge, is an agreement supertree of $\mathcal{U}$ of size $|R_{\mathrm{A}}| + |R'| \geq \mathrm{Card}\,(L(\mathcal{R})) - p + |R'| = \mathrm{Card}\,(L(\mathcal{U})) - p$.

Conversely, assume there is an agreement supertree U_{A} of $\mathcal{U}$ with $|U_{\mathrm{A}}| \geq \mathrm{Card}\,(L(\mathcal{U})) - p$. We can assume that $p < \mathrm{Card}\,(L(\mathcal{R}))$ since otherwise, the empty tree is clearly an agreement supertree of $\mathcal{R}$ of size at least $\mathrm{Card}\,(L(\mathcal{R})) - p$. Hence, at most $\mathrm{Card}\,(L(\mathcal{R})) - 1$ leaves appearing in $\mathcal{U}$ are not leaves of U_{A}. Since $\mathrm{Card}\,(L(\mathcal{R}))$ distinct leaves of $\mathcal{U}$ appear in R', there is a leaf ℓ' of R' that is also a leaf of U_{A}. Let $U' := U_{\mathrm{A}} \,|\, (L(\mathcal{R}) \cup \{\ell'\})$. Note that U' is an agreement supertree of $\mathcal{U}$ and, as for U_{A}, contains at least $\mathrm{Card}\,(L(\mathcal{R})) - p$ leaves from $L(\mathcal{R})$. Let R_{A} be the tree with leaves in $L(\mathcal{R})$, obtained by rooting U' at the leaf ℓ', and deleting ℓ' and its incident edge. R_{A} is an agreement supertree of $\mathcal{R}$ and as $\ell' \notin L(\mathcal{R})$, R_{A} has at least $\mathrm{Card}\,(L(\mathcal{R})) - p$ leaves from $L(\mathcal{R})$. □

Theorem 9 CSMAST *is not approximable within a constant factor unless* P = NP*, even for instances $\mathcal{T}$ only composed of rooted triples, respectively unrooted quartets.*

Proof. The reduction from Hitting Set to CSMAST on rooted triples and the reduction from CSMAST on rooted triples to CSMAST on unrooted quartets described in Theorem 8 can be seen as approximation preserving reductions.

Hence, the result of [5] stated in Section 5.1 for Hitting Set also applies to CSMAST on rooted triples and to CSMAST on unrooted quartets. □

The above two intractability and inapproximability results also hold for the CSMCT problem, as the reductions use collections of binary trees, *i.e.* cases in which this problem is equivalent to CSMAST.

References

[1] A. V. Aho, Y. Sagiv, T. G. Szymanski, and J. D. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM Journal on Computing*, 10(3):405–421, 1981.

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

[3] B. R. Baum. Combining trees as a way of combining data sets for phylogenetic inference, and the desirability of combining gene trees. *Taxon*, 41:3–10, 1992.

[4] B.R. Baum and M.A. Ragan. The MRP method. In O.R.P. Bininda-Emonds, editor, *Phylogenetic supertrees: combining information to reveal the Tree of Life*, pages 17–34. Kluwer, 2004.

[5] M. Bellare, S. Goldwasser, C. Lund, and A. Russeli. Efficient probabilistically checkable proofs and applications to approximations. In *Proc. of the 25th Annual A.C.M. Symposium on Theory of Computing (STOC'93)*, pages 294–304, 1993.

[6] T. Berger-Wolf. Online consensus and agreement of phylogenetic trees. In *Proc. of the 4th Workshop on Algorithms in Bioinformatics (WABI'04)*, volume 3240 of *LNCS*, pages 350–361. Springer Verlag, 2004.

[7] V. Berry, S. Guillemot, F. Nicolas, and C. Paul. On the approximation of computing evolutionary trees. In L.Wang, editor, *Proc. of the 11th Annual International Conference on Computing and Combinatorics (COCOON'05)*, volume 3595 of *LNCS*, pages 115–125. Springer, 2005.

[8] V. Berry and F. Nicolas. Maximum agreement and compatible supertrees. In S. C. Sahinalp, S. Muthukrishnan, and U. Dogrusoz, editors, *Proceedings of the 15th Combinatorial Pattern Matching Symposium (CPM'O4)*, volume 3109 of *LNCS*, pages 205–219, 2004.

[9] V. Berry and F. Nicolas. Improved parametrized complexity of the maximum agreement subtree and maximum compatible tree problems. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(3):289–302, 2006.

[10] O. R. P. Bininda-Edmonds, editor. *Phylogenetic supertrees: Combining information to reveal the tree of life*, volume 4 of *Computational Biology series*. Kluwer Academic Publishers, 2004.

[11] O. R. P. Bininda-Edmonds, J. L. Gittleman, and M. A. Steel. The (super)tree of life: procedures, problems, and prospects. *Annual Review of Ecology and Systematics*, 33:265–289, 2002.

[12] O. R. P. Bininda-Edmonds and M. J. Sanderson. Assessment of the accuracy of matrix representation with parsimony analysis supertree construction. *Systematic Biology*, 50(4):565–579, 2001.

[13] D. Bryant. *Building trees, hunting for trees and comparing trees: theory and method in phylogenetic analysis*. PhD thesis, University of Canterbury, Department of Mathemathics, 1997.

[14] D. Bryant and M. A. Steel. Extension operations on sets of leaf-labelled trees. *Advances in Applied Mathematics*, 16(4):425–453, 1995.

[15] 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*, 30(5):1385–1404, 2001.

[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. M.I.T. Press, Cambridge, Massachusetts, second edition, 2001.

[17] U. Feige, M. M. Halldórsson, and G. Kortsarz. Approximating the domatic number. In *Proc. of the 32nd Annual A.C.M. Symposium on Theory of Computing (STOC'00)*, pages 134–143, 2000.

[18] C. R. Finden and A. D. Gordon. Obtaining common pruned trees. *Journal of Classification*, 2:255–276, 1985.

[19] G. Ganapathy and T. J. Warnow. Approximating the complement of the maximum compatible subset of leaves of k trees. In *Proc. of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX'02)*, pages 122–134, 2002.

[20] 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 O. Gascuel and B. M. E. Moret, editors, *Proc. of the 1st International Workshop on Algorithms in Bioinformatics (WABI'01)*, pages 156–163, 2001.

[21] L. Gąsieniec, J. Jansson, A. Lingas, and A. Östlin. On the complexity of constructing evolutionary trees. *Journal of Combinatorial Optimization*, 3(2–3):183–197, 1999.

[22] A. G. Gordon. Consensus supertrees: the synthesis of rooted trees containing overlapping sets of labelled leaves. *Journal of Classification*, 3:335–348, 1986.

[23] S. Guillemot and F. Nicolas. Tight bounds on the complexity of the maximum agreement subtree and maximum compatible tree problems. Technical report, LIRMM, Univ. of Montpellier, 2005.

[24] A. Gupta and N. Nishimura. Finding largest subtrees and smallest supertrees. *Algorithmica*, 21(2):183–210, 1998.

[25] A. M. Hamel and M. A. Steel. Finding a maximum compatible tree is NP-hard for sequences and trees. *Applied Mathematics Letters*, 9(2):55–59, 1996.

[26] D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestor. *SIAM Journal on Computing*, 13(2):338–355, 1984.

[27] J. Hein, T. Jiang, L. Wang, and K. Zhang. On the complexity of comparing evolutionary trees. *Discrete Applied Mathematics*, 71(1–3):153–169, 1996.

[28] J. Jansson, A. Lingas, and E.-L. Lundell. A triplet approach to approximations of evolutionary trees. In *Proc. of the 8th Annual International Conference on Research in Computational Molecular Biology (RECOMB'04)*, 2004.

[29] J. Jansson, J. H.-K. Ng, K. Sadakane, and W.-K. Sung. Rooted maximum agreement supertrees. *Algorithmica*, 43(4):293–307, 2005.

[30] 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 *Proc. of the 7th Annual European Symposium on Algorithms (ESA'99)*, pages 438–449, 1999.

[31] 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*, 40(2):212–233, 2001.

[32] D. T. J. Littlewood and R. A. Bray, editors. *Interrelationships of the Platyhelminthes*, chapter 27, Towards a phylogenetic supertree of Platyhelminthes? Number 60 in Systematics Association Special Volumes. CRC Press, London, 2000.

[33] R. D. M. Page. Modified mincut supertrees. In R. Guigó and D. Gusfield, editors, *Proc. of the 2nd International Workshop on Algorithms in Bioinformatics (WABI'02)*, pages 537–552, 2002.

[34] Z. S. Peng and H. F. Ting. An $O(n \log n)$-time algorithm for the maximum constrained agreement subtree problem for binary trees. In *Proc. of the 15th International Symposium on Algorithms and Computations (ISAAC'04)*, pages 754–765, 2004.

[35] M. A. Ragan. Matrix representation in reconstructing phylogenetic relationships among the eukaryots. *Biosystems*, 28(1–3):47–55, 1992.

[36] V. Ranwez, V. Berry, S. Guillemot, A. Criscuolo, and E.J.P. Douzery. Vote or veto: desirable properties for supertree methods. submitted, LIRMM,ISEM, Université Montpellier 2, 2006.

[37] C. Semple and M. Steel. *Phylogenetics*, volume 24 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, 2003.

[38] J. L. Thorley and M. Wilkinson. A view of supertrees methods. In M. F. Janowitz, F.-J. Lapointe, F. R. McMorris, and F. S. Roberts, editors, *Bioconsensus*, volume 61 of *Discrete Mathematics and Theoretical Computer Science*, pages 185–194. DIMACS, 2003.

[39] `http://tolweb.org/tree/phylogeny.html`.

A Determining anchors between two rooted trees in linear time

In this section, we show how *anchors* from nodes of R_I to nodes of R_A can be determined in $O(n)$ where $n := \mathrm{Card}\,(L(R_I) \cup L(R_\mathrm{A}))$.

Note that nodes of R_I for which an anchor has to be determined are nodes $v_i := \mathrm{lca}_{R_I}(S)$ for some set $S \subseteq L_\cap(\{R_\mathrm{A}, R_I\})$. Here such nodes are called *anchorable*.

The *leaf*-nodes in R_I whose label is in $L_\cap(\{R_\mathrm{A}, R_I\})$ are anchorable. Their corresponding set S contains the single leaf of R_A having the same label. A single traversal of R_i and one of R_A is thus enough to establish the anchors of leaves of R_i. This costs $O(n)$.

Now consider anchorable *internal* nodes (*i.e.* for which $\mathrm{Card}\,(S) > 1$). To anchor these nodes, we proceed by considering specific couples of leaves in $L_\cap(\{R_\mathrm{A}, R_I\})$. Let $\mathcal{O}$ be the left-right order with which the leaves of $L_\cap(\{R_\mathrm{A}, R_I\})$ appear in the tree R_I, and denote the ℓth element in $\mathcal{O}$ as $\mathcal{O}_\ell$. Let m be the cardinality of $L_\cap(\{R_\mathrm{A}, R_I\})$. Then consider the $m = O(n)$ pairs $(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$ of consecutive leaves in $\mathcal{O}$. For each such pair, two lca queries are performed to identify $v_i := \mathrm{lca}_{R_I}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$ and $v_a := \mathrm{lca}_{R_\mathrm{A}}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$. For some of these pairs, v_a is the anchor of v_i. As lca relationships can be pre-processed in trees R_I and R_A in $O(n)$ time to answer each lca query in $O(1)$ [26], considering the $O(n)$ pairs $(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$ leads to an $O(n)$ running time. The correctness of this approach is demonstrated below. More precisely, it is shown

that considering all pairs $(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$, $1 \leq \ell < m$, is sufficient to determine the anchor of all anchorable internal nodes of R_I.

First, by definition, any anchorable internal node $v_i \in R_I$ is the lca of a set $S \subseteq L_\cap(\{R_\mathrm{A}, R_I\})$ with $\mathrm{Card}\,(S) \geq 2$, hence v_i has leaves of S in $s \geq 2$ child subtrees, denoted $c_1, \ldots, c_s$. Now, by definition of $\mathcal{O}$, for all couples (c_j, c_j+1), $1 \leq j < s$, the right-most leaf of $L(c_j) \cap S$ just precedes in $\mathcal{O}$ the left-most leaf of $S \cap L(c_{j+1})$, and v_i is the lca of these two leaves. Hence, examining the lca in R_I of all couples $(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$, $1 \leq \ell < m$ of consecutive leaves in $\mathcal{O}$ ensures that v_i is considered at some step.

Proposition 1 *Given a node v_i in R_I such that $v_i = \mathrm{lca}_{R_I}(S)$ with $S \subseteq L_\cap(\{R_\mathrm{A}, R_I\})$ and $\mathrm{Card}\,(S) \geq 2$. Let $\mathcal{O}_{s_1}, \ldots, \mathcal{O}_{s_j}$ be the leaves of S, with $1 \leq s_1 < s_2 < \ldots < s_j \leq n$. Let v_a be the node of R_A such that $v_a := \mathrm{lca}_{R_\mathrm{A}}(S)$. Then there is an integer ℓ, such that $s_1 \leq \ell < s_j$ and $v_a = lca_{R_\mathrm{A}}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$.*

Proof. First note that the elements of S are consecutive in $\mathcal{O}$. Let c be the child subtree of v_a that contains $\mathcal{O}_{s_1}$. Let ℓ be the smallest integer such that $s_1 \leq \ell < s_j$ and $\mathcal{O}_{\ell+1} \notin L(c)$. Note that ℓ exists, since by definition of v_a, this node has leaves of S in at least two different child subtrees. Since $\mathcal{O}_\ell, \mathcal{O}_{\ell+1}$ are in different child subtrees of v_a, then $v_a = \mathrm{lca}_{R_\mathrm{A}}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$. □

This proposition ensures that while examining all pairs of leaves $(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$ with $1 \leq \ell < m$, all nodes of R_A that are anchors of nodes in R_I are considered.

Now we know that all anchorable nodes v_i of R_I will be considered, as will all anchor nodes v_a of R_A. The following shows that, at some point, each such node in R_I is considered *at the same time* as its anchor v_a in R_A.

Proposition 2 *Let* $S = \{\mathcal{O}_{s_1}, \ldots, \mathcal{O}_{s_j}\} \subseteq L_\cap(\{R_\mathrm{A}, R_I\})$ *be a set of consecutive leaves in* $\mathcal{O}$ *with* $\mathrm{Card}\,(S) \geq 2$ *such that there exists a node* v_i *in* R_I *with* $v_i = \mathrm{lca}_{R_I}(S)$. *Let* v_a *be the node of* R_A *such that* $v_a = \mathrm{lca}_{R_\mathrm{A}}(S)$.

Then ℓ *exists,* $s_1 \leq \ell < s_j$, *such that* $v_a = \mathrm{lca}_{R_\mathrm{A}}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$ *and* $v_i = \mathrm{lca}_{R_I}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$.

Proof. If $m = 2$ then S contains only one pair of leaves. Thus, for $\ell = 1$ we have $S = \{\mathcal{O}_\ell, \mathcal{O}_{\ell+1}\}$, so $\mathrm{lca}_{R_I}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1}) = \mathrm{lca}_{R_I}(S)$ and $\mathrm{lca}_{R_\mathrm{A}}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1}) = \mathrm{lca}_{R_\mathrm{A}}(S)$.

Consider now the case where $m > 3$. Let ℓ with $s_1 \leq \ell \leq s_j$ be the smallest integer such that $\mathcal{O}_\ell$ and $\mathcal{O}_{\ell+1}$ belong to two different child subtrees of $v_a = \mathrm{lca}_{R_\mathrm{A}}(S)$. Such an ℓ exists because of Prop. 1. Now, $\{\mathcal{O}_\ell, \mathcal{O}_{\ell+1}\} \in L(v_i)$ by definition of ℓ and $S = \{\mathcal{O}_{s_1}, \ldots, \mathcal{O}_{s_j}\}$. More precisely, it can be shown that $v_i = \mathrm{lca}_{R_I}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$. Indeed, if this is not the case, *i.e.* these two leaves are in the same child subtree of v_i, then let $\ell'' \subseteq L_\cap(\{R_\mathrm{A}, R_I\})$ be a leaf in S belonging to another child subtree of v_i. We have $v_i = \mathrm{lca}_{R_I}(\ell'', \mathcal{O}_\ell) < \mathrm{lca}_{R_I}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$. Since $R_a|L(R_i) \trianglerighteq R_i|L(R_A)$, this implies $\mathrm{lca}_{R_\mathrm{A}}(\ell'', \mathcal{O}_\ell) < \mathrm{lca}_{R_\mathrm{A}}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$ by Observation 1. This in turn implies $\ell'' \notin L(v_a)$ because $v_a = \mathrm{lca}_{R_\mathrm{A}}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$. Hence, $L(v_a) \cap L_\cap(\{R_\mathrm{A}, R_I\}) \neq L(v_i) \cap L_\cap(\{R_\mathrm{A}, R_I\})$, which is in contradiction with

the definition of v_a and Remark 3. This shows that $\mathcal{O}_\ell$ and $\mathcal{O}_{\ell+1}$ are in different child subtrees of v_i, hence that $v_i = \text{lca}_{R_I}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$. By definition of v_a and v_i, this shows that for this precise ℓ, we have both $\text{lca}_{R_I}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1}) = \text{lca}_{R_I}(S)$ and $R_{\text{A}}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1}) = \text{lca}_{R_{\text{A}}}(S)$. □

Now, given that $R_{\text{A}}|L(R_I)$ refines $R_I|L(R_{\text{A}})$, the same anchorable internal node $v_i \in R_I$ might be considered in several pairs $(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$, sometimes together with a node $v_a = \text{lca}_{R_{\text{A}}}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$ that is not its anchor in R_{A}. However, the anchor of v_i can be easily identified as it is the closest to the root (*i.e.* the one with the minimum depth), among the examined candidate nodes v_a in R_{A}. More formally:

Remark 4 *Let v_i be a node of R_I such that $v_i = \text{lca}_{R_I}(S)$ for a set $S \subseteq L_\cap(\{R_{\text{A}}, R_I\})$ with $\text{Card}\,(S) \geq 2$. Suppose $v_i = \text{lca}_{R_I}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$ and $v_i = \text{lca}_{R_I}(\mathcal{O}_{\ell'}, \mathcal{O}_{\ell'+1})$ with $\ell \neq \ell'$. Let $v_a^\ell = \text{lca}_{R_{\text{A}}}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$ and $v_a^{\ell'} = \text{lca}_{R_{\text{A}}}(\mathcal{O}_{\ell'}, \mathcal{O}_{\ell'+1})$. If $depth(v_a^\ell) > depth(v_a^{\ell'})$ then v_a^ℓ is not the anchor of v_i.*

Indeed, since $depth(v_a^\ell) > depth(\text{lca}_{R_{\text{A}}}(\mathcal{O}_{\ell'}, \mathcal{O}_{\ell'+1}))$, then $\mathcal{O}_{\ell'}$ or $\mathcal{O}_{\ell'+1}$ is not in $L(v_a^\ell)$. As both $\mathcal{O}_{\ell'}$ and $\mathcal{O}_{\ell'+1}$ belong to $L(v_i)$, this means that $L(v_a^\ell) \cap L_\cap(\{R_{\text{A}}, R_I\}) \neq L(v_i) \cap L_\cap(\{R_{\text{A}}, R_I\})$, *i.e.* that v_a^ℓ is not the anchor of v_i (from Remark 3).

Based on the previous remarks, the pseudo-code ANCHORS shows how the anchors for all anchorable nodes of R_I are determined in $O(n)$ time. The artificial anchor of the root of R_I (when needed, see Section 3.1) is also described.

Algorithm 3: ANCHORS (R_I, R_A)

Input: Two rooted trees R_I and R_A s.t $R_\mathrm{A}|L(R_I) \trianglerighteq R_I|L(R_\mathrm{A})$.

Result: Determines the nodes of R_A that are anchors for nodes in R_I.

***(i)* Anchoring leaves:**

1 **for** *each leaf node* $\ell \in R_I$ *with a label in* $L_\cap(\{R_\mathrm{A}, R_I\})$ **do**
  set $\mathsf{Anchor}(\ell)$ to the leaf-node of R_A having the same label.

***(ii)* Anchoring internal nodes:**

for *each internal node* $v_i \in R_I$ **do** $\mathsf{Anchor}(v_i) \leftarrow \emptyset$

Let $\mathcal{O}$ be the left-right ordering in the tree R_I of the leaves in set $L_\cap(\{R_\mathrm{A}, R_I\})$

2 **for** *each pair* $(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$ *of consecutive leaves in* $\mathcal{O}$ **do**
  Let $v_i := \mathrm{lca}_{R_I}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$ and $v_a := \mathrm{lca}_{R_\mathrm{A}}(\mathcal{O}_\ell, \mathcal{O}_{\ell+1})$
  if $\mathsf{Anchor}(v_i) = \emptyset$ *or* $depth(\mathsf{Anchor}(v_i)) > depth(v_a)$ **then**
    $\mathsf{Anchor}(v_i) \leftarrow v_a$

***(iii)* Anchor roots of R_I and R_A:**

if $\mathsf{Anchor}(\mathrm{root}(R_I)) = \emptyset$ **then**
  if $\mathrm{root}(R_\mathrm{A})$ *has been anchored with a node of* R_I **then**
    add a new node as parent of $\mathrm{root}(R_\mathrm{A})$ and consider it as the new root of R_A
  $\mathsf{Anchor}(\mathrm{root}(R_I)) \leftarrow \mathrm{root}(R_\mathrm{A})$

B Complement for the proof of Theorem 1

We give here the proof that (4) holds for the special case where $\mathrm{Card}\,(\{\ell, \ell', \ell''\} \cap L(R)) = 2$. This completes the proof of Theorem 1.

Without loss of generality, let ℓ', ℓ'' be the two leaves present in R before the graft of the copy of S_p and let ℓ be the leaf in S_p. If $\mathrm{lca}_{R_I}(\ell, \ell') = \mathrm{lca}_{R_I}(\ell, \ell'') = \mathrm{lca}_{R_I}(\ell', \ell'')$, then (4) is verified by default. The three remaining possibilities for the relative positions of ℓ, ℓ, ℓ'' in R_i are as follows:

(1) $\mathrm{lca}_{R_I}(\ell, \ell') = \mathrm{lca}_{R_I}(\ell, \ell'') < \mathrm{lca}_{R_I}(\ell', \ell'')$.

(2) $\mathrm{lca}_{R_I}(\ell, \ell') = \mathrm{lca}_{R_I}(\ell', \ell'') < \mathrm{lca}_{R_I}(\ell, \ell'')$.

(3) $\mathrm{lca}_{R_I}(\ell, \ell'') = \mathrm{lca}_{R_I}(\ell', \ell'') < \mathrm{lca}_{R_I}(\ell, \ell')$.

The proof is only detailed below for case 1, as a very similar reasoning applies for cases 2 and 3.

Let v_i be the node from which S_p is hanging in R_I. Note that $\text{lca}_{R_I}(\ell, \ell') \leq v_i$, otherwise $\{\ell, \ell', \ell''\} \in L(S_p)$, a contradiction with $\text{Card}\left(\{\ell, \ell', \ell''\} \cap L(R)\right) = 2$.

(I) Suppose $\text{lca}_{R_I}(\ell, \ell') = v_i$. Note that this does not imply that v_i has an anchor in R because $\ell \notin R$.

(I-A) Suppose that v_i has an anchor v_a in R. Let c be the child of v_i such that $\{\ell', \ell''\} \in S(c)$. Either (i) $L(c)$ contains no leaf of $L_\cap(\{R_\text{A}, R_I\})$, or (ii) it contains at least such a leaf, say x. Case (i) occurs only when $S(c)$ is a specific subtree of R_i with respect to R_A, which means that a copy of $S(c)$ (containing $\{\ell', \ell''\}$) is grafted under v_a by loop of line 2, just before S_p is grafted. Hence, ℓ', ℓ'' belong to a same subtree $S(c')$ of a child c' of v_a. We now show that the same holds in case (ii). Indeed, in that case, $x \in L(v_i)$ and because v_i is an anchored node, then $\exists y \in L(v_i)$ such that $y \in L_\cap(\{R_\text{A}, R_I\})$ and $v_i = \text{lca}_{R_I}(x, y)$. Since $\{x, \ell', \ell''\} \in S(c)$, we have

$$v_i = \text{lca}_{R_I}(x, y) < \text{lca}_{R_I}(x, \ell') \text{ and } v_i = \text{lca}_{R_I}(x, y) < \text{lca}_{R_I}(x, \ell'')$$

Since $\{x, \ell', \ell''\} \in L(R)$, we have by induction that

$$\text{lca}_R(x, y) < \text{lca}_R(x, \ell') \text{ and } \text{lca}_R(x, y) < \text{lca}_R(x, \ell'').$$

Moreover, by definition of v_a, we know $\{x, y\} \in L(v_a)$, thus $v_a \leq \text{lca}_R(x, y)$,

and then from the previous equation

$$v_a < \mathrm{lca}_R(x, \ell') \text{ and } v_a < \mathrm{lca}_R(x, \ell'').$$

This means that $\{x, \ell', \ell''\}$ belong to a same subtree $S(c')$ in R, with c' being a child of v_a.

Thus, in both (i) and (ii), when a copy of S_p (containing ℓ) is inserted as a new child subtree of v_a, by loop of line 2, then equation (4) holds.

(I-B) The other sub-case is when v_i is not an anchored node of R_i. This means that a copy of S_p is grafted in R as a subtree of a node v_{new} by loop of line 4. This loop inserts subtrees at nodes above the anchor v_a of a node v of R_I. Note that in R_I, $v_i < v$, and that there is no node with an anchor on the path from v to v_i (because the loop of line 4 would end before reaching v_i). There are several possible places for ℓ' and ℓ'' relative to v_i and v.

First, ℓ' and ℓ'', can be in the same specific subtree S'_p hanging from v_i. Then $\ell', \ell'' \in L(R)$ ensures that a copy of S'_p is inserted as a subtree of v_{new} by loop of line 5, before the same loop inserts a copy of S_p. Thus in that case, ℓ', ℓ'' are in the same subtree of v_{new} before S_p is inserted. The second possibility for ℓ', ℓ'' is that they are in the subtree of v_i that contains v. Then ℓ', respectively ℓ'', can be in a specific subtree S'_p, respectively S''_p, hanging from a node on the path between v and v_i, (with possibly $S'_p = S''_p$). But then, the loop of line 4, proceeding in a bottom-up way, ensures that S'_p, respectively S''_p, is inserted in the subtree of v_{new} that contains v_a. Alternatively, ℓ' and ℓ'' (or just one

of them) can be in $S(v)$. A similar argument [1] as in (I-A) shows that in this case they are inserted in the subtree $S(v_a)$ of R. Thus, in this case also, they belong to the subtree of v_{new} containing v_a.

Thus, in all possible positions of ℓ', ℓ'' relative to v_i and v, they are grafted in R the same subtree of v_{new}. This means that when a copy of S_p (containing ℓ) is inserted as a different child subtree of v_{new} to obtain R', then (4) holds.

(II) The other main sub-case arises when $\text{lca}_{R_I}(\ell, \ell') < v_i$. Let $u := \text{lca}_{R_I}(\ell, \ell') = \text{lca}_{R_I}(\ell, \ell'')$. Note that ℓ' and ℓ'' belong to the same child subtree of u, differing from that containing ℓ.

Let v be the node v_i, if v_i is anchored, otherwise it is the closest descendant of v_i that is anchored (v exists otherwise, $S(v_i)$ would be a specific subtree, contradicting the maximality of the specific subtree S_p). Moreover, there is a unique closest descendant of v_i that is anchored because, if two nodes are anchored, then their least common ancestor is also anchored).

Let v_a be the anchor of v in R. By definition of v and v_a, $x, y \in L_\cap(\{R_\text{A}, R_I\})$ exist such that $v = \text{lca}_{R_I}(x, y)$ and $v_a = \text{lca}_{R_\text{A}}(x, y)$. Note that $u = \text{lca}_{R_I}(\ell', x) = \text{lca}_{R_I}(\ell'', x) < \text{lca}_{R_I}(x, y)$ because x, y are in the child subtree of u containing v_i, different from the one containing ℓ', ℓ''. Hence,

$$\begin{aligned} &\text{lca}_{R_I}(\ell', \ell'') > \text{lca}_{R_I}(\ell', x) = \text{lca}_{R_I}(\ell'', x) \\ \text{and } &\text{lca}_{R_I}(x, y) > \text{lca}_{R_I}(\ell', x) = \text{lca}_{R_I}(\ell'', x) \end{aligned}$$

[1] based on leaves $\{x, y\} \in L(v_a) \cap L_\cap(\{R_\text{A}, R_I\})$

As $\{\ell', \ell'', x, y\} \in L(R)$, induction applies to obtain

$$\begin{aligned} \mathrm{lca}_{R_\mathrm{A}}(\ell', \ell'') &> \mathrm{lca}_{R_\mathrm{A}}(\ell', x) = \mathrm{lca}_{R_\mathrm{A}}(\ell'', x) \\ \text{and } \mathrm{lca}_{R_\mathrm{A}}(x, y) = v_a &> \mathrm{lca}_{R_\mathrm{A}}(\ell', x) = \mathrm{lca}_{R_\mathrm{A}}(\ell'', x) \end{aligned}$$

Let v'_a be the node $\mathrm{lca}_{R_\mathrm{A}}(\ell', x)$ of R_A. Previous equations indicate that v'_a has different children $c_{\ell'}$ and c_x, such that $\{\ell', \ell''\} \in S(c_{\ell'})$ and $x, y \in S(c_x)$. The node $v_a = \mathrm{lca}_{R_\mathrm{A}}(x, y)$, anchor of v, is thus in $S(c_x)$. Moreover, by definition of v, the copy of S_p is inserted either as a new child subtree of v_a (in the case where $v = v_i$), either between v_a and its parent in R. In both cases, ℓ is inserted in $S(c_x)$. Thus in tree R', ℓ is in a subtree of v'_a differing from that containing ℓ', ℓ'', hence (4) holds.

C Proof of Lemma 1

Proof. The three statements of the Lemma are related to the call to MERGETREES issued in line 2 of the loop of BUILDSMCT, statement (A) applying before the execution of a call, while statements (B) and (C) apply to the result of this call. Thus, the three statements are strongly inter-dependent and we prove them by a joint induction. However, note that there is no circularity as

- the proof of (A) uses inductive hypothesis on previous iterations of statements (B) and (C), except for the basic step which is proved independently;
- the proof of (B) uses statement (A) of the same iteration and, except for the basic step, inductive hypothesis on the previous iteration of (B);
- the proof of (C) uses statements (A) and (B) of the same iteration and,

except for the basic step, inductive hypothesis on the previous iteration of (C).

The basic step ($i = 1$) of each statement is proved as following:

(A) When $i = 1$, $R_I = R_1|(L(R_M^0) \cup L_{\mathrm{S}}(R_1))$ and $R_A = R_M^0$. As $R_M^0 = MCT(\mathcal{R}|L(\mathcal{R}))$, we have

$$L(R_M^0) \subseteq L_{\cap}(\mathcal{R}) \tag{C.1}$$
$$\text{and} \quad R_M^0 \trianglerighteq R_1|L_{\cap}(\mathcal{R})\Big|L(R_M^0) \tag{C.2}$$

From (C.1) we have $L(R_M^0) \subseteq L(R_1)$, thus

$$R_M^0 = R_M^0|L(R_1) \tag{C.3}$$
$$\text{and} \quad (R_1|L_{\cap}(\mathcal{R}))|L(R_M^0) = R_1|L(R_M^0). \tag{C.4}$$

Rewriting (C.2) thanks to (C.3) and (C.4) gives $R_A|L(R_I) \trianglerighteq R_I|L(R_M^0) = R_I|L(R_A)$, the last equality resulting from the remark following Definition 1.

(B) From the basic step of (A), we know that Theorem 1 applies when MergeTrees is called. Thus, the tree R_M^1 returned by this call is such that $L(R_M^1) = \Big(L(R_1) \cap \Big(L(R_M^0) \cup L_{\mathrm{S}}(R_1)\Big)\Big) \cup L(R_M^0)$, which simplifies into $L(R_M^0) \cup L_{\mathrm{S}}(R_1)$ since $L_{\Delta}(\mathcal{R}) = \emptyset$.

(C) From the basic step of statement (A) and Theorem 1 we know R_M^1 is a tree $SMCT(R_I, R_A)$, i.e.

$$R_M^1|L(R_I) \trianglerighteq R_I|L(R_M^1) \quad \text{and} \quad R_M^1|L(R_A) \trianglerighteq R_A|L(R_M^1) \tag{C.5}$$

with $R_I = R_1|\Big(L(R_M^0) \cup L_{\mathrm{S}}(R_1)\Big)$ and $R_A = R_M^0$. Moreover, from the basic step of statement (B), $L(R_M^1) = L(R_M^0) \cup L_{\mathrm{S}}(R_1)$, thus for all $R_j \in \mathcal{R}$,

$j > 1$, we have from (C.5) that

$$R_M^1|L(R_M^0) \trianglerighteq R_M^0|L(R_M^1) = R_M^0 \trianglerighteq R_j|L(R_M^0), \tag{C.6}$$

where the rightmost refinement relation results from the definition of R_M^0. Statement (B), $L_\Delta(\mathcal{R}) = \emptyset$, and $L(R_M^0) \subseteq L_\cap(\mathcal{R})$ imply $L(R_M^1) \cap L(R_M^0) = L(R_M^0) = L(R_j) \cap L(R_M^0) = L(R_M^1) \cap L(R_j)$ for all $R_j \in R, j > 1$. Thus, (C.6) rewrites as

$$R_M^1|L(R_j) \trianglerighteq R_j|L(R_M^1), \forall j > 1. \tag{C.7}$$

From (C.5) we also have

$$R_M^1|L(R_1) \cap (L(R_M^0) \cup L_\mathrm{S}(R_1)) \trianglerighteq R_1|L(R_M^0) \cup L_\mathrm{S}(R_1)|L(R_M^1)$$

which rewrites as $R_M^1|L(R_1) \trianglerighteq R_1|L(R_M^1)$ using statement (B). Together with (C.7), this proves the basic step of statement (C).

Now suppose statements (A), (B) and (C) hold for the first $i-1$ iterations of the loop in line 2 and consider its ith iteration, with $i > 1$.

(A) We first rewrite $L(R_A), L(R_M^i)$ and $L(R_I)$:

$$L(R_A) = L(R_M^{i-1}) = L(R_M^0) \cup \bigcup_{j<i} L_\mathrm{S}(R_i) \tag{C.8}$$

$$L(R_i) = L_\cap(\mathcal{R}) \cup L_\mathrm{S}(R_i) \tag{C.9}$$

$$L(R_I) = L(R_i) \cap (L(R_M^0) \cup L_\mathrm{S}(R_i)) = L(R_M^0) \cup L_\mathrm{S}(R_i) \tag{C.10}$$

where (C.8) results by inductive hypothesis from statement (B), (C.9) results from $L_\Delta(\mathcal{R}) = \emptyset$, and (C.10) follows from the fact that $L(R_M^0) \subseteq L_\cap(\mathcal{R})$ and $L_\mathrm{S}(R_i)$ are both included into $L(R_i)$, as (C.9) shows. This three

equations show that

$$L(R_I) \cap L(R_A) = L(R_M^0) = L(R_i) \cap L(R_A). \tag{C.11}$$

By inductive hypothesis, statement (C) says that R_M^{i-1} is a supertree compatible with $\mathcal{R}$, i.e. $R_M^{i-1}|L(R_i) \trianglerighteq R_i|L(R_M^{i-1})$, which rewrites as $R_A|L(R_I) \trianglerighteq R_I|L(R_A)$ by definition of R_A, R_I and use of (C.11).

(B) From statement (A) and Theorem 1, the call to MergeTrees (R_I, R_A) performed at iteration i of the loop in BuildSMCT with trees $R_I = R_i\Big|(L(R_M^{i-1}) \cup L_{\mathrm{S}}(R_i))$ and $R_A = R_M^{i-1}$ returns a tree R_M^i such that

$$L(R_M^i) = L(R_I) \cup L(R_A). \tag{C.12}$$

It is easy top see that $L(R_I) = L(R_M^0) \cup L_{\mathrm{S}}(R_i)$ and that, by inductive hypothesis of (B), $L(R_A) = L(R_M^{i-1}) = L(R_M^0) \cup \bigcup_{j<i} L_{\mathrm{S}}(R_j)$. Thus, from (C.12) we obtain $L(R_M^i) = L(R_M^0) \bigcup_{j\leq i} L_{\mathrm{S}}(R_j)$.

(C) From statement (A) for the ith iteration, we know Theorem 1 applies to the call MergeTrees (R_I, R_A) performed at that iteration with trees $R_I = R_i|\Big(L(R_M^0) \cup L_{\mathrm{S}}(R_i)\Big)$ and $R_A = R_M^{i-1}$. Thus, the tree R_M^i is a $SMCT(R_I, R_A)$, i.e.

$$R_M^i|L(R_I) \trianglerighteq R_I|L(R_M^i) \quad \text{and} \quad R_M^i|L(R_A) \trianglerighteq R_A|L(R_M^i). \tag{C.13}$$

Moreover, statement (B) used for the ith iteration says that

$$L(R_M^i) = L(R_M^0) \cup \bigcup_{j\leq i} L_{\mathrm{S}}(R_i) = L(R_M^{i-1}) \cup L_{\mathrm{S}}(R_i). \tag{C.14}$$

Consider the case of trees $R_j \in \mathcal{R}$ with $j \neq i$, for which

$$L(R_j) \cap L(R_M^i) = L(R_j) \cap L(R_M^{i-1}), \tag{C.15}$$

from (C.14) and the fact that specific leaves of R_i do not appear in other input trees. By inductive hypothesis of statement (C),

$$R_M^{i-1}|L(R_j) \trianglerighteq R_j|L(R_M^{i-1}) = R_j|L(R_M^i). \tag{C.16}$$

From the second part of (C.13) we know another refinement relation:

$$R_M^i|L(R_M^{i-1}) \trianglerighteq R_M^{i-1}|L(R_M^i) = R_M^{i-1}, \tag{C.17}$$

the equality resulting from $L(R_M^{i-1}) \subseteq L(R_M^i)$ (from C.14). Thus, reducing both sides of (C.17) to leaves of a tree R_j, $j \neq i$, we obtain

$$(R_M^i|L(R_M^{i-1}))\Big|L(R_j) \trianglerighteq R_M^{i-1}\Big|L(R_j),$$

the left part of which rewrites as $(R_M^i|L(R_j))\Big|L(R_M^{i-1}) = R_M^i|L(R_j)$ from (C.15). Combining this refinement relation with the one stated in (C.16) we obtain by transitivity that

$$R_M^i|L(R_j) \trianglerighteq R_j|L(R_M^i), \forall j \neq i. \tag{C.18}$$

Now consider the case of the input tree $R_i \in \mathcal{R}$. Since $L_\Delta(\mathcal{R}) = \emptyset$, i.e. $L(R_i) = L_\cap(\mathcal{R}) \cup L_\mathrm{S}(R_i)$, and $L(R_M^0) \subseteq L_\cap(\mathcal{R})$, we obtain

$$L(R_i) \cap L(R_M^i) = L(R_M^0) \cup L_\mathrm{S}(R_i) \tag{C.19}$$

from (C.14). The refinement relation stated in the first part of (C.13) can

be rewritten as

$$R_M^i|L(R_i) \cap (L(R_M^0) \cup L_{\mathrm{S}}(R_i)) \trianglerighteq R_i|L(R_M^0) \cup L_{\mathrm{S}}(R_i)|L(R_M^i)$$

which can be simplified into $R_M^i|L(R_i) \trianglerighteq R_i|L(R_M^i)$ using (C.19). Together with (C.18), this proves statement (C) for the ith iteration.

□