Maximum Agreement and Compatible Supertrees ⋆ RR-LIRMM 04045

Vincent Berry and François Nicolas

Équipe *Méthodes et Algorithmes pour la Bioinformatique - L.I.R.M.M.* {vberry,nicolas}@lirmm.fr

Abstract. Given a collection of trees on n leaves with identical leaf set, the MAST, resp. MCT, problem consists in finding a largest subset of the leaves such that all input trees restricted to this set are identical, resp. have a common refinement. For MAST, resp. MCT, on k rooted trees, we give an $O(min\{3^p kn, c^p + kn^3\})$ algorithm, where $c < 3$ and p is the smallest number of leaves whose removal leads to the existence of an agreement subtree, resp. a compatible tree. This improves on [13] for MAST and proves fixed parameter tractability for MCT.
We then extend these problems to the case of supertrees where input trees can have non-identical leaf sets. For the obtained problems, SMAST and SMCT, we give an $O(N + n)$ time algorithm for the special case of two input trees (N is the time bound for solving MAST, resp. MCT, on two $O(n)$-leaf trees). Finally, we show that SMAST and SMCT parametrized in p are W[2]-hard and cannot be approximated in polynomial time within a constant factor unless $\mathrm{P} = \mathrm{NP}$, even when the input trees are rooted triples.
We also extend the above results to the case of unrooted input trees.

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

1 Introduction

Given a set of leaf-labelled trees with identical leaf sets, the *maximum agreement subtree* problem (MAST) consists in finding a subtree homeomorphically included in all input trees and with the largest number of leaves [30, 14, 1, 18, 21, 22, 11]. This pattern matching problem on trees arises in various areas among which is the reconstruction of *evolutionary trees* (or *phylogenies*) whose leaves represent living species and internal nodes represent ancestral species. The shape of the tree describes the speciation pattern from which current species originated.

Motivation. A recent problem in phylogenetics is to infer trees from a collection of input trees on overlapping, but different, sets of leaves. Each input tree is built from a separate data set which does not include all studied species for various reasons. These trees are then given as input to a method that proposes a tree, called a *supertree*, i.e., a tree including all (or most) species according to their relative positions in the input trees. For various reasons, the input trees can disagree on the position of several species. Depending on the way they handle such conflicts, supertree methods can be divided into two main categories: [31]: *(i)* optimization methods which tend to resolve conflicts according to a specified optimization criterion [3, 26, 28, 24, 10] ; *(ii)* consensus methods which output a supertree displaying only parts of the species' history on which the input trees agree. The drawback of approach *(i)* is that output supertrees sometimes contain undesirable or unjustified resolutions of conflicts [25, 27, 5, 33, 24].Approach *(ii)* has been less investigated in the context of supertrees, the two proposed methods being the strict consensus [17], sometimes criticized because of the poor amount of information of the produced supertree [33, 6], and the reduced consensus [31], which usually proposes a set of complementary supertrees as output (see [31, Sect. 4]). Both these methods focus on the *clusters* (groups of leaves under internal nodes). Several authors remarked that an alternative would be to focus on *leaves* themselves, because in many cases removing a few species on the position of which the input trees disagree, could enable to produce a single informative supertree [17, 33, 6]. Here we follow this proposition by extending the MAST problem to the case of input trees on overlapping sets of leaves. We call SMAST the resulting problem concerned with the inference of a supertree. We also extend a variant of MAST called *maximum compatible tree* (MCT) which is of interest when input trees are non-binary [20, 16], to obtain the SMCT problem. MCT and SMCT allow multifurcating nodes (high degree nodes) of input trees to be resolved (split into several nodes) according to other input trees.

Apart from inferring an estimate of the species' history, SMAST and SMCT can play the same role as MAST in the context of supertrees, i.e., measuring the similarity of input trees or identifying species that could be implied in horizontal transfers of genes. Moreover, the supertree they produce is most likely to contain leaves from most input trees (see Lem. 5) and, by definition, is shaped according to all these trees. It is thus a good candidate to strengthen the results of the popular *matrix representation with parsimony* (MRP) method [3, 26]. E.g., [7] explicitly recommended to use such a *seed* tree (i.e., a tree with leaves spanning most input trees) to improve its relatively low accuracy observed for MRP when the input trees overlap moderately.

Previous work. [18] designed a variant of MAST that builds a tree they call a "supertree", but with a different meaning from supertrees considered in phylogenetics and in this paper. MAST is NP-hard on only three rooted trees of unbounded degree [1], and MCT on six rooted trees [20]. Efficient polynomial time algorithms have been recently proposed for MAST on two rooted n-leaf trees: $O(n \log n)$ for binary trees [11], then $O(n^{1.5})$ for trees of unbounded degree [21], and $O(n^{1.5} \log^2 \frac{n}{d})$ for trees of degree bounded by d [22]. The $O(n^{1.5})$ results of [21] also applies to the case of two unrooted trees of unbounded degree. In contrast, MCT is thought to be hard on two trees [29].

When k rooted trees are given as input and have maximum degree d, MAST can be solved in $O(n^d + kn^3)$[14, 1, 8] and MCT in $O(2^{2kd} n^k)$ [16]. MAST is 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 [13]. [13] describe an algorithm in $O(3^p kn \log n)$ and cite an unpublished work which would lead, as we understand it, to an $O(c^p + kn^3)$ algorithm (where $c \approx 2.56$).

Results. Following the work of [13], we obtain an $O(min\{3^p kn, H + kn^3\})$ algorithm for both MCT and MAST, where H is the time required to solve the 3-HITTING SET problem on an instance of size $O(n^3)$ (currently $H = O(c^p + n^3)$ where $c \approx 2.311$ [23] and p is the size of a smallest hitting set). This improves the bound for the MAST problem w.r.t. [13] and is the first result showing that MCT is FPT. Moreover, from this standpoint, MCT has the same complexity as MAST which was not expected w.r.t. previous work (on two general trees, the former is thought to be NP-hard while efficient algorithms exist for the second, and on bounded degree trees, less efficient algorithms exist for MCT). Remark also that the exponential term in the complexity of the obtained FPT algorithm does not depend on the degree or number of input trees, which might be an advantage in practice over the algorithm of [16], though the latter may be faster for trees with a high level of disagreement.

Then, we show how to extend MAST and MCT in a natural way to obtain the problems SMAST and SMCT on supertrees. SMAST and SMCT are NP-hard in general, as they are equivalent to MAST, resp. MCT, in the case of input trees with identical leaf sets. For both SMAST and SMCT, we give an $O(N + n)$ algorithm for the case of two input trees, where N is the time bound for solving MAST, resp. MCT, on two $O(n)$-leaf trees.

Finally, by reduction to the HITTING SET problem, we show that SMAST and SMCT are more difficult than MAST and MCT, as they are W[2]-hard for p. Thus, there is little hope to obtain efficient exact algorithms for these problems on more than two trees, suggesting to resort on heuristic algorithms to solve the problem. However, it will be difficult to prove tight approximation results for such heuristics, as we also show that no polynomial time algorithm can approximate SMAST and SMCT within a constant factor, unless P $=$ NP, even when the input trees are rooted triples (binary trees on only three leaves).

All above results can be extended to the case of unrooted trees (adding an n factor in the case of the FPT algorithm). In the following, Sect. 2 gives definitions, then results are presented for MAST and MCT in Sect. 3, and for SMAST and SMCT in Sect. 4.

2 Definitions and Preliminaries

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 $r(T)$), in which case all internal nodes have at least two children, or *unrooted*, in which case internal nodes have degree at least three. When there is no ambiguity, we will identify leaves with their labels. The size $\#T$ of a tree T is the number of its leaves. Let u be a node in a rooted tree T, we denote by $S_T(u)$ the subtree rooted at u (i.e., u and its descendants) and by $L(u)$ the leaves of this subtree. Given a rooted tree T and a set of leaves $L \subseteq L(T)$, we denote $lca_T(L)$ the lowest common ancestor of leaves L in T.

Definition 1 *Given a set L of labels and a tree T, the* restriction *of T to L, denoted $T|L$, is obtained by taking the smallest subtree of T which connects leaves with label in L and making it homeomorphically irreducible (by suppressing non-root nodes of degree two and identifying the two edges to which they were connected).*

Definition 2 *A tree T is said to be* homeomorphically included *into a tree T', which is denoted $T \sqsubseteq T'$, iff $T = T'|L(T)$, i.e., if there exists a graph isomorphism $T \mapsto T'|L(T)$ preserving leaf labels, and the root if T, T' are rooted. Given a collection $\mathcal{T} = \{T_1, \ldots, T_k\}$ of trees with a common leaf set L, an* agreement subtree *of $\mathcal{T}$ is any tree T with leaves in L s.t. $\forall T_i \in \mathcal{T}, T \sqsubseteq T_i$. The* Maximum Agreement Subtree *problem (*MAST*) consists in finding an agreement subtree of $\mathcal{T}$ with the largest number of leaves. We denote $MAST(\mathcal{T})$ such a tree.*

In phylogenetic analysis, this definition is sometimes considered too stringent when input trees can have *multifurcations* (nodes with more than 2 children). Indeed, such a node can either represent a multi-speciation event, or an uncertainty (irresolution) concerning the relative branching of the child subtrees of the node. The MAST problem is best suited for the first interpretation, as the presence of a multifurcation in a input tree, will imped to resolve the node according to other input trees. For the second interpretation, [20] introduced the MCT problem, a variant of MAST that allows multifurcations to be resolved in the output tree:

Definition 3 *A tree T* refines *a tree T', and we write $T \trianglerighteq T'$, if T' can be obtained by collapsing certain edges of T (i.e., merging their extremities). More generally, a tree T refines a collection $\mathcal{T} = \{T_1, \ldots, T_k\}$, denoted $T \trianglerighteq \mathcal{T}$, whenever T refines all T_i's in $\mathcal{T}$. Given a collection $\mathcal{T} = \{T_1, \ldots, T_k\}$ of input trees with identical leaf set L, a tree T with leaves in L is said to be* compatible with *$\mathcal{T}$ iff $\forall T_i \in \mathcal{T}, T \trianglerighteq T_i|L(T)$. The* MCT *problem consists in finding a tree compatible with $\mathcal{T}$ having the largest number of leaves. We denote $MCT(\mathcal{T})$ such a tree.*

Note that $\#MCT(\mathcal{T}) \geq \#MAST(\mathcal{T})$ and that MCT is equivalent to MAST when input trees are binary.

Definition 4 *Let T be a rooted tree, on any three leaves $a, b, c \in L(T)$, there are only three possible binary shapes for $T|\{a, b, c\}$, denoted $a|bc$, resp. $b|ac$, resp. $c|ab$, depending on their innermost grouping of two leaves (bc, resp. ac, resp ab). These trees are called* rooted triples *(or* resolved triples*). Alternatively $T|\{a, b, c\}$ can be a* fan *(also called* unresolved *tree), connecting the three leaves to a same internal node, which is denoted (a, b, c). We define $tr(T)$, resp.*

$f(T)$, to be the set of rooted triples, resp. fans, induced by the leaves of a tree T. We extend these definitions to define rooted triples and fans of a collection of trees $\mathcal{T} = \{T_1, \ldots, T_k\}$: $tr(\mathcal{T}) := \cap_{T_i \in \mathcal{T}} tr(T_i)$ and $f(\mathcal{T}) := \cap_{T_i \in \mathcal{T}} f(T_i)$. Two trees T, T' are in hard conflict *whenever $\exists a, b, c, \in L(T) \cap L(T')$ s.t. $a|bc \in tr(T)$ and $b|ac \in tr(T')$. T, T' are in* soft conflict *whenever $a|bc \in tr(T)$ and $(a, b, c) \in f(T')$.*

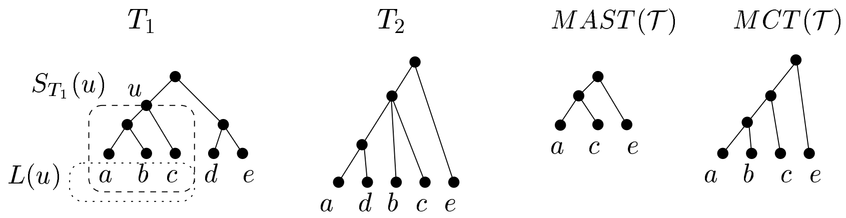

Fig. 1. A collection $\mathcal{T} = \{T_1, T_2\}$, one of the $MAST(\mathcal{T})$ trees, and the $MCT(\mathcal{T})$. T_1, T_2 are in hard conflict on a, c, d (since $d|ac \in tr(T_1)$ while $c|ad \in tr(T_2)$) and in soft conflict on a, b, c (since $c|ab \in tr(T_1)$ while $\{a, b, c\} \in f(T_2)$).

Lemma 1 *Let $\mathcal{T}$ be a collection of trees with identical leaf set L and T, T' two trees.*
(i) T is an agreement subtree of $\mathcal{T}$ iff $tr(T) \subseteq tr(\mathcal{T})$ and $f(T) \subseteq f(\mathcal{T})$.
(ii) T is compatible with $\mathcal{T}$ iff $\forall T_i \in \mathcal{T}, tr(T_i) \subseteq tr(T)$ and $L(T) \subseteq L$.
(iii) T is homeomorphic to T' iff $tr(T) = tr(T')$ and $f(T) = f(T')$.
(iv) A tree T refines a tree T' iff $tr(T') \subseteq tr(T)$ and $L(T) = L(T')$.

Proof. (i) is [8, Lem. 6.6], (ii) is [9, Thm. 1], (iii) derives from (i), and (iv) results from (ii). □

3 An FPT algorithm for MAST and MCT

Considering the MAST problem, [8, 13] remark that if the input trees are in hard or soft conflict on a set of leaves $\{a, b, c\}$, then obviously at least one of the three leaves has to be excluded to obtain an agreement subtree (because of Lem. 1 (i)). If trees in $\mathcal{T}$ are not in conflict w.r.t. any three leaves, then they are all isomorphic and any of them is a maximum agreement subtree. Then, an algorithm for finding an agreement subtree of an initial collection $\mathcal{T}$ consists in identifying a conflict $\{a, b, c\}$, trying alternatively to remove one of a, b, c and iterate on the restricted collections until no conflict remains. Hence, to solve MAST, we need an algorithm to check that all trees in $\mathcal{T}$ are isomorphic or alternatively identify a hard or soft conflict on three leaves. Such an algorithm can be produced by modification of a tree isomorphism algorithm [19, 32]. We consider the case of two input trees (i.e. $\mathcal{T} = \{T_1, T_2\}$), since applying the resulting algorithm at most $k - 1$ times, solves the problem on k input trees. Define a node in a tree to be a *cherry* iff all its children are leaves. We then have:

Lemma 2 ([19]) *Let T_1, T_2 be two isomorphic trees and v_1 a cherry in T_1. Then, there exists a cherry $v_2 \in T_2$ s.t. $L(v_1) = L(v_2)$.*

***Sketch of the algorithm* CHECK-ISOMORPHISM-OR-FIND-CONFLICT.** Lem. 2 suggests a bottom-up process examining cherries $v_1 \in T_1$. Let $l \in L(v_1)$ and v_2 the parent node of leaf l in T_2. If v_2 is not a cherry then it has a child v_i that is not a leaf, and comparing $L(v_i)$ to $L(v_1)$ we can identify two leaves l', l'' s.t. T_1, T_2 are in conflict w.r.t. l, l', l'' (this requires an $O(n)$ search of $S_{T_2}(v_i)$). If v_2 is a cherry but $L(v_1) \neq L(v_2)$ then comparing $L(v_1)$ to $L(v_2)$ we can again easily identify leaves $l', l'' \in L(v_1) \cup L(v_2)$ s.t. T_1, T_2 are in (hard or soft) conflict w.r.t. l, l', l'' (see appendix 4.3 for details). If, however, v_2 is a cherry and $L(v_1) = L(v_2)$, then we can *eat* the cherries in both trees, i.e., delete leaves hanging from v_1 and v_2, turning them into leaves to which a same new label is given. Then the process is repeated for another cherry of T_1 and iterated until both trees are reduced to a single identical leaf (in which case we know that both input trees are isomorphic) or until a conflict is identified. Each edge being processed at most twice in the process, and requiring each time $O(1)$ operations, we have:

Theorem 1 *Let T_1, T_2 be two rooted trees with identical leaf sets, in time $O(n)$ we can either conclude that the trees are isomorphic or otherwise identify three leaves on which T_1, T_2 conflict.*

This improves by a $\log n$ factor on the complexity given in [13] for identifying a conflict or checking isomorphism, and subsequently improves the complexity of the FPT algorithm to solve MAST (see Corol. 1).

Concerning the MCT problem, Lem. 1 (ii) implies that any tree compatible with $\mathcal{T}$ will have to eliminate at least one of $\{a, b, c\}$ if input trees are in *hard* conflict on these three leaves (soft conflicts do not imped the existence of a tree compatible with $\mathcal{T}$). If trees in $\mathcal{T}$ induce no hard conflict then there is a tree T refining $\mathcal{T}$ and thus $T := MCT(\mathcal{T})$ (because it includes all leaves of $\mathcal{T}$). Thus, we are interested in an algorithm that either identifies a hard conflict or returns a refinement of $\mathcal{T}$. As above, we consider case of two input trees. However, to use this algorithm to find a refinement T (if one exists) of k input trees, we also need this refinement to be *minimum*, i.e., such that $\forall T' \trianglerighteq \{T_1, T_2\}, T' \trianglerighteq T$, to not artificially create hard conflicts.

It is no longer true also that a cherry $v_1 \in T_1$ corresponds to a cherry $v_2 \in T_2$ with the same leaf set. We can now allow $L(v_1) \subseteq L(v_2)$ or $L(v_2) \subseteq L(v_1)$. Moreover, given a leaf $l \in L(v_1)$, its parent v_2 in T_2 is not required anymore to be a cherry[1]. However, we can prove the following:

Lemma 3 *Let T_1, T_2 be two trees admitting a common refinement and v_1 a cherry in T_1. Then $L(v_1) = \bigcup_{u \in C} L(u)$ with C set of at least two children of $v_2 := lca_{T_2}(L(v_1))$.*

***Sketch of algorithm* FIND-REFINEMENT-OR-CONFLICT.** Given two input trees T_1, T_2 with same leaf set L, the algorithm gradually erodes T_1 and T_2, repeatedly removing cherries in T_1 and corresponding parts in T_2 until the trees are reduced to a single leaf or a hard conflict is found. The process is guided by cherries v_1 *removed from* T_1. This means that nodes labelled $L(v_1)$ are deleted from T_1, making v_1 a leaf in T_1 that we identify with a new label l^*. The part of T_2 removed according to v_1 is also replaced by a leaf labelled l^*.

The tree refining both T_1 and T_2 is built as a forest T of trees (initially containing a leaf-tree for each leaf in L) that are gradually connected to exactly mimic clusters of leaves induced by the internal nodes of T_1 and T_2. This guarantees the minimality of the computed refinement (invariant 1). Trees in T are built according to subtrees of T_1, T_2 that are not in conflict (invariant 2). Trees are agglomerated in T just before subtrees of T_1 and T_2 are removed and replaced by a new leaf l^*. The root of the tree in forest T resulting from this agglomeration is accordingly labelled l^*.

When a new cherry $v_1 \in T_1$ is chosen, it is matched to node $v_2 := lca_{T_2}(L(v_1))$. Then, either

- v_2 has the same leaf set as v_1 (and is then the part of T_2 to remove);
- a proper subset of v_2's children have their leaf sets spanning exactly $L(v_1)$ (these subtrees are thus the part of T_2 to remove);
- we identify a conflict involving two leaves $l, l' \in L(v_1)$ and a leaf $l'' \in L(v_2) - L(v_1)$ (see appendix 4.4 for details).

Note that in the second case above, v_1 induces a refinement of node v_2, in the sense that it leads to agglomerate in T a proper subset of v_2's subtrees, while v_2 may also refines v_1 in the same time if these subtrees are not all restricted to single leaf (i.e., there is a subtree of v_2 agglomerating a proper subset of $L(v_1)$ in T).

After the successful processing of a cherry v_1, leaves $L(v_1)$ have been removed from both T_1 and T_2, which also have a new common leaf l^*. Hence, their leaf sets are conserved identical (invariant 3 below). If no conflict is met, both trees end as identical leaf-trees. The pseudo-code FIND-REFINEMENT-OR-CONFLICT(T_1, T_2) (see algorithm 1) details this process that either identifies a hard conflict between T_1 and T_2, either returns a tree minimally refining them.

Lemma 4 *The following invariants hold at the beginning and after each cherry $v_1 \in T_1$ has been processed:*

1. *Let v be a node in a tree of forest T, then there is a node u in T_1 or T_2 s.t. $L(v) = L(u)$.*
2. *T_1, T_2 are not in hard conflict w.r.t. three leaves of a same tree in the forest T.*
3. *$L(T_1) = L(T_2)$.*

The algorithm is traversing T_1, T_2 a constant number of times, spending a constant time at each of the $O(n)$ nodes and edges. Nodes v_2 are identified in $O(n)$ globally by using the dynamic data structures proposed by [12], the list L of cherries in T_1 is maintained in $O(n)$ globally, sets R of subtrees $S_{T_2}(v_l)$ corresponding to processed cherries of T_1 are identified and removed in $O(n)$ globally.

[1] as shown by the example where, using parenthetical notation, $T_1 = (l_1, l_2, (l_3, l_4))$ and $T_2 = (l_3, l_4, (l_1, l_2))$, admitting $((l_1, l_2), (l_3, l_4))$ as common refinement.

Algorithm 1: FIND-REFINEMENT-OR-CONFLICT(T_1, T_2)

Input: Two rooted trees T_1, T_2 on the same leaf set L.
Result : Three leaves on which T_1 and T_2 are in hard conflict, or a tree T on L minimally refining T_1 and T_2.
$T \leftarrow L$ /* T is a forest of trees (initially leaf-trees) */
Let C be the list of cherries in T_1
while $C \neq \emptyset$ **do**
- Choose v_1 in C
- Let l^* be a new label, v a new node in T labelled l^* and let $v_2 = lca_{T_2}(L(v_1))$
- $R \leftarrow \emptyset$ /* R are the subtrees of v_2 to agglomerate because of v_1 */
- $P \leftarrow L(v_1)$ /* P are the leaves that lead to identify new subtrees of v_2 to agglomerate */
- **while** $P \neq \emptyset$ **do**
  - Choose l in P
  - Let v_l be the child of v_2 s.t. $l \in L(v_l)$
  - **foreach** *leaf* $l'' \in L(v_l)$ **do**
    - **if** $l'' \in L(v_1)$ **then** $P \leftarrow P - \{l''\}$
    - **else**
      - Let l' a leaf in $L(v_1) - L(v_l)$
      - Replace l, resp. l' and l'', by any leaf in $S_T(l)$, resp. $S_T(l')$ and $S_T(l'')$
      - **return** conflict on l, l', l''
  - Assemble trees in T to form a copy of $S_{T_2}(v_l)$ grafted as a new child of v.
  - Add v_l to R
- Replace $S_{T_1}(v_1)$ in T_1 by a new leaf labelled l^* and add its parent to C if it becomes a cherry
- **foreach** *subtree rooted at* $v_l \in R$ **do** Remove $S_{T_2}(v_l)$ from T_2
- **if** v_2 *has become a leaf* **then** label v_2 by l^* /* v_2 corresponds to v_1 */
- **else** graft a new leaf labelled l^* under v_2 /* to replace its subtrees connected because of v_1 */

return T /* the forest T has been reduced to a single tree that is returned. */

Theorem 2 *Let T_1, T_2 be two rooted trees with identical leaf sets, in time $O(n)$ algorithm* FIND-REFINEMENT-OR-CONFLICT(T_1, T_2) *either finds three leaves on which T_1, T_2 conflict, either returns a tree T minimally refining T_1 and T_2.*

Using this algorithm at most $k-1$ times enables to solve the same problem for k input trees in $O(kn)$. Thus, both in the case of MAST and MCT we have an algorithm that can be used over and over in the FPT algorithm described in [13] to solve MAST in time $O(3^p kn)$. The same FPT algorithm can be used for MAST (to the difference that only hard conflicts generate branch points in the search tree). [13] also remark that this FPT algorithm implies an indirect resolution of the 3-HITTING-SET (3HS) problem where the parameter p is preserved (for 3HS, p is the size of the smallest hitting set). Using this problem as an explicit subproblem of MAST and MCT provides an alternative approach with running time $O(H + kn^3)$, where H is the time to solve 3HS on an instance of size $O(n^3)$. As a whole, we obtain the following result:

Corollary 1 *Given a collection of k input trees of unbounded degree,* MAST *and* MCT *can be solved in time $O(min\{3^p kn, H + kn^3\})$, where H is the time to solve* 3HS *on an instance of size $O(n^3)$.*

The proof is given in appendix 4.6. Note that currently, $H = O(2.311^p + n^3)$ [23].

This improves on the result of [13] for MAST by a $\log n$ factor. Moreover, this shows that MCT is FPT, more precisely that the burden of the complexity can depend only on the level of disagreement of the input trees. When considering a collection of trees disagreeing on few species, we obtain an efficient algorithm, whatever the number and degree of the input trees. If input trees are unrooted, the same approach as above can be used by trying successively all possible rooting of input trees at a leaf. This only adds an extra n factor to the complexity bound.

4 Extending problems to the supertree context

We now consider the case of supertree inference, where input trees are allowed to have different (but overlapping) sets of leaves. We show how to extend MAST and MCT to this context.

Definition 5 *Given a collection $\mathcal{T} = \{T_1, \ldots, T_k\}$ of input trees, we denote by $L(\mathcal{T}) := \cup_{T_i \in \mathcal{T}} L(T_i)$ the set of all input leaves. Leaves appearing in only one input tree are called* specific*, and we denote $\mathcal{L}(T_i)$ the specific leaves of tree T_i and $\mathcal{L}(\mathcal{T}) := \cup_{T_i \in \mathcal{T}} \mathcal{L}(T_i)$ the set of specific leaves of the whole collection. If u is a node in a tree T_i, let $\mathcal{L}(S_{T_i}(u)) := L(S_{T_i}(u)) \cap \mathcal{L}(T_i)$ be the specific leaves of the subtree rooted at u. A* specific subtree *is a subtree containing only specific leaves.*

Though definitions would allow this, in the following we assume that any input tree shares at least two leaves with other input trees, otherwise there is no purpose in taking such a tree into account to build a supertree.

Definition 6 *Given a collection $\mathcal{T} = \{T_1, \ldots, T_k\}$ of trees on overlapping sets of leaves, an* agreement supertree *of $\mathcal{T}$ is a tree T with $L(T) \subseteq L(\mathcal{T})$ and s.t. $\forall T_i \in \mathcal{T}, T|L(T_i) = T_i|L(T)$. The* Maximum Agreement Supertree *problem consists in finding an agreement supertree with the largest number of leaves. Such a tree is denoted $SMAST(\mathcal{T})$. In a similar way, we define the* Maximum Compatible Supertree *problem (*SMCT*) as finding a largest tree T with $L(T) \subseteq L(\mathcal{T})$ s.t. $\forall T_i \in \mathcal{T}, T|L(T_i) \trianglerighteq T_i|L(T)$. Such a tree is denoted $SMCT(\mathcal{T})$.*

The two problems stated above are natural extensions of the problems defined in Sect. 2, as SMAST, resp. SMCT, is equivalent to MAST, resp. MCT, when the input trees have equal leaf set.

The following observation states that the supertrees defined above are likely to contain a non-trivial number of leaves, and moreover, leaves from many, if not all, input trees. This suggests that these supertrees might be good *seed* trees for the MRP method [3, 26] or its variant MRF [10].

Lemma 5 *Let $\mathcal{T} = \{T_1, \ldots, T_k\}$ be a collection of rooted trees with overlapping sets of leaves, any tree $T := SMAST(\mathcal{T})$ or $T := SMCT(\mathcal{T})$, is such that $\mathcal{L}(\mathcal{T}) \subseteq L(T)$.*

The same lemma can be proved for unrooted trees.

4.1 Solving SMAST and SMCT on 2 Trees in polynomial time

Given two trees T_1, T_2 on overlapping sets of leaves, let $L_\cap := L(T_1) \cap L(T_2)$.

Lemma 6 *Let $\mathcal{T} = \{T_1, T_2\}$ be a collection of two rooted trees on overlapping sets of leaves,*

$$(i)\ \forall T' := MAST(T_1|L_\cap, T_2|L_\cap),\ \exists T := SMAST(T_1, T_2) \text{ s.t. } T|L_\cap = T' .$$
$$(ii)\ \forall T' := MCT(T_1|L_\cap, T_2|L_\cap),\ \exists T := SMCT(T_1, T_2) \text{ s.t. } T|L_\cap = T' .$$

This result is easily extended to the case of unrooted trees.

Sketch of the algorithm. The algorithm proceeds as suggested by Lem. 6, i.e., grafting specific parts of the input trees to a precomputed maximum agreement subtree T_m of $\mathcal{T}|L_\cap$. The approach is similar to the $O(n^3)$ algorithm of [17] for computing a strict consensus supertree, which attaches one by one specific leaves to a common *backbone* tree (sensu [17]). Similarly, we can take advantage here of the fact that T_m is a backbone tree common to T_1 and T_2. Indeed, once restricted to leaves which are either specific or in T_m, input trees have the same underlying topology as T_m ($T_m \sqsubseteq T_i|L_\cap \sqsubseteq T_i$), from which *specific* subtrees are hanging (see Fig. 2). However here, the algorithm proceeds by successively attaching whole specific subtrees at a time to T_m. Child subtrees of a node $n_i \in T_i$ are partitioned into *specific* subtrees, whose roots are stored in a list called $\mathsf{SpecificChild}(n_i)$, and *common* subtrees (i.e., containing leaves from T_m), whose roots are stored in a list called $\mathsf{CommonChild}(n_i)$. For each node $n_m \in T_m$, there is a *twin* node in T_1, denoted $\mathsf{Twin}_1(n_m)$, and a twin node in T_2, denoted $\mathsf{Twin}_2(n_m)$, in the sense that these three nodes are the least common ancestors (in their respective trees) of the same set of leaves from $L(T_m)$.

The algorithm proceeds by a simultaneous *postorder* recursive traversal of T_m, T_1, T_2 (coordinated by T_m and the Twin_1 and Twin_2 data structures), during which specific subtrees of T_1 and T_2 are grafted to a copy of the backbone tree. The data structures Twin_1 and Twin_2 are initialized for leaves of T_m during a first traversal of the trees (line 15 in pseudo-code COMPSMASTTREE), and are completed before being used for other nodes during the joint traversal of the three trees performed by the procedure GRAFTSPECIFIC: though the algorithm uses a top-down recursive approach, the joint traversal is *post-ordered*, i.e., subtrees of a node will be processed before the node itself. The processing of the children n_c of a node $n_m \in T_m$ determines the twins of n_m (line 25 in call issuing from line 20) before we need to refer to them (lines 21,23).

Algorithm 2: COMPSMASTTREE (T_1, T_2)

Input: Two rooted trees T_1, T_2 on overlapping sets of leaves
Result : A tree $T := SMAST(T_1, T_2)$
Determine $L_\cap, \mathcal{L}(T_1), \mathcal{L}(T_2), T_1' := T_1|L_\cap, T_2' := T_2|L_\cap$
13 $T_m \leftarrow MAST(T_1', T_2'), T_1'' \leftarrow T_1|(L(T_m) \cup \mathcal{L}(T_1)), T_2'' \leftarrow T_2|(L(T_m) \cup \mathcal{L}(T_2))$
/* Add a *common* root to handle specific subtrees branching above $r(T_m)$ */
14 $T_m \leftarrow (l^*, T_m), T_1'' \leftarrow (l^*, T_1''), T_2'' \leftarrow (l^*, T_2'')$, where l^* is a new leaf
15 Traverse T_1'', T_2'' in postorder to compute for each $l \in L(T_m)$ and $n_i \in T_i$
   Twin$_1(l)$, Twin$_2(l)$, CommonChild(n_i) and SpecificChild(n_i)
16 $T' \leftarrow$ GRAFTSPECIFIC$(r(T_m))$
17 **return** (T) where T is s.t. $T' = (l^*, T)$

Note that COMPSMASTTREE adds an artificial root and an artificial leaf l^* to the trees T_1, T_2, T_m before the initial call to GRAFTSPECIFIC (this operation is described on line 14 using the parenthetical notation to code trees). This enables specific subtrees branching above the original root of T_m to be handled by GRAFTSPECIFIC. The artificial node and root are removed at the end of the algorithm (line 17, also using parenthetical notation).

The algorithm computes T_m in $O(N)$, then perform a constant number of traversals of the trees, during which each of the $O(n)$ edges gives rise to a constant number of operations. Hence:

Theorem 3 *Let $\mathcal{T}$ be a collection of two rooted trees with overlapping leaf sets drawn from a set of n labels. The algorithm* COMPSMASTTREE (T_1, T_2) *returns a tree $T := SMAST(\mathcal{T})$ in time $O(N + n)$, where N is the time needed to compute $MAST(T_1, T_2)$ on two n-leaf rooted trees.*

Currently, $N = min\{O(n^{1.5}) \ , \ O(\sqrt{d}n \log^2 \frac{n}{d})\}$, where d is the maximum degree of the input trees [21, 22].

Algorithm 3: GRAFTSPECIFIC(n_m)

Input: A node $n_m \in T_m$
Result : the subtree corresponding to $S_T(n_m)$ in a tree $T := SMAST(T_1, T_2)$
18 **if** *n_m is a leaf l* **then** $T \leftarrow \{l\}$
else
   /* Computing recursively subtrees sharing common leaves then grafting specific subtrees hanging from the twin nodes of n_m in T_1, T_2 */
19    $T \leftarrow$ a new node having as child subtrees:
20   1 - subtrees resulting from GRAFTSPECIFIC(n_c), $\forall n_c \in$ Children(n_m)
21   2 - subtrees SpecificChild(Twin$_1(n_m)$) and SpecificChild(Twin$_2(n_m)$)
if $n_m = r(T_m)$ **then return** (T)
/* Graft specific subtrees branching above n_m in T_1 and T_2 */
22 **for** $i \leftarrow 1$ ***to*** 2 **do**
23    $n_i \leftarrow$ Twin$_i(n_m)$; $p_i \leftarrow$parent(n_i)
   while $\#$*CommonChild*$(p_i) < 2$ **do**
24       $T \leftarrow$ a new node having as child subtrees T and SpecificChild(p_i)
      $p_i \leftarrow$parent(p_i)
25   Twin$_i(n_m) \leftarrow p_i$
return T

Minor Modifications for Solving Related Problems. The case of unrooted input trees is handled in the same way, as lemma 6 is still valid in that setting. An unrooted MAST is then necessary (line 13) and the rooting at an artificial leaf is replaced by a rooting at a leaf in $L(T_m)$. This gives an $O(N' + n)$ algorithm for computing $SMAST$ for two unrooted trees. Currently, $N' = O(n^{1.5})$ [21].

In a similar way, replacing the call to MAST (line 13) by a call to MCT, allows COMPSMASTTREE to solve the SMCT problem in time $O(N'' + n)$, where N'' is the time required to solve MCT on two $O(n)$-leaf trees. Currently,

N'' is $O(min\{2^{4d}n^2, 3^p n, H + n^3\})$ [16, and this paper] for rooted trees. Considering unrooted trees adds an extra n factor.

When aiming at producing an estimate of a phylogeny underlying the input trees, it is not satisfactory that the supertree can contain some arbitrary edges. This happens whenever a same edge in T_m corresponds to paths in T_1 and T_2 from which specific subtrees are hanging in both trees. Instead of grafting subtrees of T_2 above those of T_1 (loop 22), we can transpose the lack of information of the input collection concerning the relative order of these subtrees by connecting them (or their leaves[2]) to a single multifurcating node m in T. This implies a minor modification of algorithm 3 which does not change its running time. Note that the produced tree might not be a maximum agreement supertree anymore, as some internal edges of the input trees can be collapsed to m in T.

4.2 Intractability of SMAST and SMCT

The algorithm of the previous section cannot be extended to collections of $k > 2$ input trees. This stems from the fact that $L(\mathcal{T})$ now also hosts leaves belonging to several but not all input trees. There are cases where no $MAST(\mathcal{T}|L_\cap)$, resp. $MCT(\mathcal{T}|L_\cap)$ tree can be completed to obtain a $SMAST(\mathcal{T})$, resp. $SMCT(\mathcal{T})$, tree (see Fig. 3 of appendix 4.11 for an example). Indeed, there is an important gap in complexity between the case of 2 input trees, and an arbitrary number of trees as we now show. If not specified, trees in this section are to be considered rooted. Formulating SMAST as the following decision problem:

> **Maximum Agreement Supertree** (SMAST)
> **instance:** A collection $\mathcal{T} = \{T_1, \ldots, T_k\}$ of trees on overlapping sets of leaves and an integer $p \geq 0$.
> **question:** Is there an agreement supertree T of $\mathcal{T}$ with at least $\#L(\mathcal{T}) - p$ leaves?

we prove, by reduction to the general HITTING SET problem (HS), that SMAST is NP-complete and W[2]-hard for the parameter p, even for instances consisting only of rooted triples Note that MAST on rooted triples is solved in time $O(k)$ and is FPT for parameter p in the general case. As MAST is the special case of SMAST where input trees have identical sets of leaves, the NP-hardness result for only three input trees obtained by [1] for MAST also holds for SMAST. Similarly, SMCT in NP-hard for 6 trees from the result of [20].

Definition 7 ([2, 9, 28]) *Let $\mathcal{T}$ be a collection of trees with leaves in S, define $[\mathcal{T}, S]$ to be the undirected graph with vertices S and with edge set $\{(u, v) \mid \exists T_i \in \mathcal{T}$ with $x|uv \in tr(T_i)\}$.*

Lemma 7 *If $[\mathcal{T}, S]$ is connected then there exists no agreement supertree of $\mathcal{T}$ on $L(\mathcal{T})$.*

Proof. A direct consequence of theorem 2 of [9]. □

Definition 8 *We recursively define the function* rake *associating a tree to an non-empty ordered sequence of trees with non-intersecting leaf sets:*

- $\mathrm{rake}(T) = T$ *for any tree T and*
- $\mathrm{rake}(T_1, T_2, \ldots, T_k) = (T_1, rake(T_2, \ldots, T_k))$ *for any sequence of trees $T_1, T_2, \ldots, T_k$ of length $k \geq 2$ s.t. $L(T_i) \cap L(T_j) = \emptyset, \forall i, j \in [1, k], i \neq j$.*

The second item above uses the parenthetical notation for trees. Appendix 4.10 includes a figure illustrating this definition.

Definition 9 (Gadget) *Let $m \geq 1$ and a set of distinct labels $x^1, x^2, \ldots, x^m, y^1, y^2, \ldots, y^m$, we define $\mathcal{G}(x^1, x^2, \ldots, x^m, y^1, y^2, \ldots, y^m)$ to be the following collection of rooted triple trees:*

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

where we set $x^{m+1} := x^1$, $x^{m+2} := x^2$ et $y^{m+1} := y^1$.

[2] though, when aiming at a *seed* tree for the MRP method, we guess it might be better to conserve the topology of the specific subtrees.

The following lemma shows that there is no agreement supertree of $\mathcal{G}$ on $L(\mathcal{G})$, however removing any x^j element leads to the existence of such a tree, thus to the existence of a maximum agreement supertree on leaves $L(\mathcal{G}) - \{x^j\}$. Moreover, the restriction of this tree to leaves $\{x^1, x^2, \ldots, x^m\} - \{x^j\}$ is indifferent.

Lemma 8 *Let* $\mathcal{G} := \mathcal{G}(x^1, x^2, \ldots, x^m, y^1, y^2, \ldots, y^m)$.

1. *there is no agreement supertree of* $\mathcal{G}$ *on* $L(\mathcal{G})$
2. *the following tree is an agreement supertree of* $\mathcal{G}$*:*

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

where $j \in [1, m]$ *and* T^* *is any tree with leaves* $\{x^1, x^2, \ldots, x^m\} - \{x^j\}$.

Theorem 4 *The* SMAST *problem is* NP*-complete and* W[2]*-hard for the parameter* p*, even for instances were trees in* $\mathcal{T}$ *are rooted triples.*

The proof (Appendix 4.10) relies on a reduction to HS. As this is an L-reduction, considering SMAST-OPT and HS-OPT, the optimization versions of problems SMAST and HS, and using the result of [4] for HS-OPT we obtain:

Theorem 5 SMAST-OPT *is not approximable within a constant factor unless* $\mathrm{P} = \mathrm{NP}$.

The two above results of intractability and inapproximability also hold for SMCT, as the reduction uses binary trees, case in which SMCT is equivalent to SMAST. These results hold as well for the unrooted versions of SMAST and SMCT, to which the rooted versions can be easily reduced (see Appendix 4.10).

References

1. Amir A. and D. Keselman. Maximum agreement subtree in a set of evolutionary trees: metrics and efficient algorithm. *SIAM J. on Comp.*, 26(3):1656–1669, 1997.
2. 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.
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. M. Bellare, S. Goldwasser, C. Lund, and A. Russeli. Efficient probabilistically checkable proofs and applications to approximations. In *Proceedings of the Twenty-Fifth Annual A.C.M. Symposium on Theory of Computing*, pages 294–304, 1993.
5. O.R.P. Bininda-Edmonds and H.N. Bryant. Properties of matrix representation with parsimony analyses. *Syst. Biol.*, 47:497–508, 1998.
6. O.R.P. Bininda-Edmonds, J.L. Gittleman, and M.A. Steel. The (super)tree of life: procedures, problems, and prospects. *Ann. Rev. Ecol. Syst.*, 2002.
7. O.R.P. Bininda-Edmonds and M.J. Sanderson. Assessment of the accuracy of matrix representation with parsimony analysis supertree construction. *Syst. Biol.*, 50(4):565–579, 2001.
8. D. Bryant. *Building trees, hunting for trees and comparing trees*. PhD thesis, University of Canterbury, Department of Math., 1997.
9. D. Bryant and M.A. Steel. Extension operations on sets of leaf-labelled trees. *Adv. Appl. Math.*, 16:425–453, 1995.
10. D. Chen, L. Diao, O. Eulenstein, and D. Fernandez-Baca. Flipping: a supertree construction method. *DIMACS Series in Disc. Math. and Theor. Comp. Sci.*, 61:135–160, 2003.
11. R. Cole, M. Farach, R. Hartigan, Przytycka T., and M. Thorup. An $o(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees. *SIAM J. on Computing*, (to appear).
12. R. Cole and R. Hariharan. Dynamic lca queries on trees. In *Proc. of the 10th ann. ACM-SIAM symp. on Disc. alg. (SODA'99)*, pages 235 – 244, 1999.
13. R.G. Downey, M.R. Fellows, and U. Stege. Computational tractability: The view from mars. *Bull. of the Europ. Assoc. for Theoret. Comp. Sci.*, 69:73–97, 1999.
14. M. Farach, T. Przytycka, and M. Thorup. Agreement of many bounded degree evolutionary trees. *Inf. Proc. Letters*, 55(6):297–301, 1995.
15. U. Feige, M. M. Halldórsson, and G. Kortsarz. Approximating the domatic number. In *Proceedings of the Thirty-Second Annual A.C.M. Symposium on Theory of Computing*, pages 134–143, 2000.
16. G. Ganapathysaravanabavan and T. 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 Workshop on Algorithms for Bioinformatics (WABI'01)*, volume 2149 of *LNCS*, pages 156–163, 2001.

17. A.G. Gordon. Consensus supertrees: the synthesis of rooted trees containing overlapping sets of labelled leaves. *J. of Classif.*, 3:335–346, 1986.
18. A. Gupta and N. Nishimura. Finding largest subtrees and smallest supertrees. *Algorithmica*, 21(2):183–210, 1998.
19. D. Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, 21:19–28, 1991.
20. A.M. Hamel and M.A. Steel. Finding a maximum compatible tree is np-hard for sequences and trees. *Appl. Math. Lett.*, 9(2):55–59, 1996.
21. 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 8th Ann. Europ. Symp. Alg. (ESA)*, pages 438–449. Springer-Verlag, New York, NY, 1999.
22. M.Y. Kao, T.W. Lam, W.K. Sung, and H.F. Ting. A faster unifying algorithm for comparing trees. In *Proc. of the 11th Ann. Symp. on Comb. Patt. Match. (CPM'00)*, LNCS, pages 129–142. Springer-Verlag, 2000.
23. R. Niedermeier and P. Rossmanith. An efficient fixed parameter algorithm for 3-Hitting Set. *Journal of Discrete Algorithms*, 1:89–102, 2003.
24. R. Page. Modified mincut supertrees. In O. Gascuel and M.-F. Sagot, editors, *Proc. of the Workshop on Algorithms for Bioinformatics (WABI'02)*, LNCS, pages 538–551. Springer-Verlag, 2002.
25. A. Purvis. A modification to Baum and Ragan's method for combining phylogenetic trees. *Syst. Biol.*, 44:251–255, 1995.
26. M.A. Ragan. Matrix representation in reconstructing phylogenetic relationships among the eukaryots. *Biosystems*, 28:47–55, 1992.
27. F. Ronquist. Matrix representation of trees, redundancy, and weighting. *Syst. Biol.*, 45:247–253, 1996.
28. C. Semple and M.A. Steel. A supertree method for rooted trees. *Disc. Appl. Math.*, 105:147–158, 2000.
29. M.A. Steel. personnal communication, 2003.
30. M.A. Steel and T. Warnow. Kaikoura tree theorems: Computing the maximum agreement subtree. *Information Processing Letters*, 48:77–82, 1993.
31. J.L. Thorley and M. Wilkinson. A view of supertrees methods. In *Bioconsensus, DIMACS*, volume 61, pages 185–194. Amer. Math. Soc. Pub., 2003.
32. T. J. Warnow. Tree compatibility and inferring evolutionary history. *Journal of Algorithms*, 16:388–407, 1994.
33. M. Wilkinson, J. Thorley, D.T.J. Littlewood, and R.A. Bray. *Interrelationships of the Platyhelminthes*, chapter 27, Towards a phylogenetic supertree of Platyhelminthes. Taylor and Francis, London, 2001.

Appendix

4.3 Finding three leaves implied in a hard or soft conflict in algorithm CHECK-ISOMORPHISM-OR-FIND-CONFLICT

Let v_1 be a cherry of T_1, $l \in L(v_1)$, v_2 the parent of l in T_2. If v_1 is not a cherry or $L(v_1) \neq L(v_2)$ then the trees are in conflict. First consider the case where v_2 is not a cherry and let v_i be non-leaf child of v_2. There are three alternatives:

1. $L(v_i) \cap L(v_1) = \emptyset$: pick $l' \in L(v_1), l' \neq l$ and $l'' \in L(v_i)$ (thus $l'' \notin L(v_1)$), then l, l', l'' is a conflict (soft or hard depending on $l' \in L(v_2)$);
2. $L(v_i) \subseteq L(v_1)$: choose $l', l'' \in L(v_i)$, then l, l', l'' is a soft conflict;
3. $L(v_i) \cap L(v_1) \neq \emptyset$ and $L(v_i) \not\subseteq L(v_1)$: pick $l' \in L(v_i) \cap L(v_i)$ and $l'' \in L(v_i) - L(v_1)$, then l, l', l'' is a hard conflict;

If v_2 is a cherry but $L(v_1) \neq L(v_2)$ then there are also three alternatives:

1. $\#L(v_1) < \#L(v_2)$: Pick $l' \in L(v_1), l'' \in L(v_2) - L(v_1)$ s.t. $l'' \neq l' \neq l$ then l, l', l'' is a conflict (soft or hard depending on $l' \in L(v_2)$);
2. $\#L(v_1) > \#L(v_2)$: symmetrical of the preceding case;
3. $\#L(v_1) = \#L(v_2)$: pick $l' \in L(v_1) - L(v_2)$ and $l'' \in L(v_2) - L(v_1)$, then l, l', l'' is a hard conflict.

Let n be the number of leaves in T_1 and T_2. In each case above we identify a set of three leaves on which T_1, T_2 conflict by only comparing $L(v_1)$ to $L(v_2)$, which requires $O(n)$. Collecting $L(v_1)$ and $L(v_2)$ at first is also in $O(n)$ as it involves the search of a subtree of T_1 and a subtree of T_2.

4.4 Finding three leaves implied in a hard conflict in algorithm FIND-REFINEMENT-OR-CONFLICT

Given a cherry $v_1 \in T_1$ and $v_2 := lca_{T_2}(L(v_1))$, there is a hard conflict iff there is a child subtree $S_{T_2}(v_l)$ of v_2 that contains both a leaf $l \in L(v_1)$ and a leaf $l'' \notin L(v_1)$ (Lem. 3). Indeed, there exists at least one leaf $l' \in L(v_1)$ in a subtree of v_2 different from $S_{T_2}(v_l)$ (by definition of v_l) and $ll''|l' \in tr(T_2)$ conflicts with $ll'|l'' \in tr(T_1)$. Note that if $x \in \{l, l', l''\}$ is not a label in L, (i.e. is an artificial label l^* resulting from the removing of a previous cherry), then we replace x by a leaf of L found in the tree of T with root labelled l^*.

4.5 Proof of Theorem 2

Correction of the algorithm results from Lem. 3 and 4. The $O(n)$ running time can be shown by a successive examination of data structures and operations they support:

- Trees T_1 and T_2 are stored by usual pointers.
  Each edge of T_1 is processed once, when its higher node is the cherry v_1 processed by the algorithm. Its higher node is either a cherry at start, either becomes a cherry because of the repeated process of replacing cherries of T_1 by a new leaf each.
  Edges of T_2 are each examined a constant number of times: when considering a node v_2, the edges of each subtree $S_{T_2}(v_l)$ containing leaves in $L(v_1)$ will be considered once when traversing it (line 6), plus once for some of them, when v_l has to be identified (line 5: edges of the path from leaf f to the first ascendant v_l that is a child of v_2). In case of conflict, edges of a subtree can be traversed a final time to identify a leaf l' (line 7) before stopping the algorithm. That's the reason why subtrees $S_{T_2}(v_l)$ are not readily removed from T_2 when processed (hence the reason for list R).
  Edges of other subtrees will be considered only in case of conflict (for finding l'), i.e. before stopping the algorithm. When all leaves of $L(v_1)$ have been processed successfully, each edge of a subtree $S_{T_2}(v_l)$ has been traversed twice and will be traversed a last time (before being removed, line 11) to make a copy of the subtree in T (line 9.
- The forest T consists of a set of nodes arranged in a set of non-overlapping trees that are subtrees of the finally output tree on n leaves. Thus, at any step the forest contains $O(n)$ nodes.
  Assembling some trees in T (line 11) consists in identifying nodes with labels corresponding to leaves in a subtree of $S_{T_2}(v_2)$, and connecting them according to the topology of this subtree of T_2. For this purpose, an additionnal array can be easily maintained to find in $O(1)$ each needed node of T.
  Creating a new node v with a given label (line 1) is done $O(n)$ times and costs $O(1)$ each time.
  In case of conflict, at most three different trees of T, i.e., $O(n)$ nodes, are traversed (line 8) for finding leaves of L (i.e., leaves of trees in T).
- List R is a simple linked list of root nodes of child subtrees of v_2 to be removed from T_2 after all leaves of P have been processed. Each element is added in $O(1)$ and removed in $O(1)$ when the list is emptied (line 11). Actuallyu removing each subtree from the list of child subtrees of v_2 is performed in $O(1)$ when coding its children as a bidirectionnal linked list.
- The list P of leaves is managed as an array of $2n-2$ bits, one for each possible label (the n original leaves, plus the at most $2n-2$ corresponding to internal nodes of the final tree T), Initially, all entries are zeroed, indicating the absence of any leaf in P. When considering leaves $L(v_1)$ of a cherry $v_1 \in T_1$, only the bits corresponding to these labels are set (line 2). Then successively taking leaves put in P (line 4) until none remains (loop line 3) is done in listing all leaves that are child of v_1 from left to right, finding for each execution of the loop, the next one that has its bit set to 1 in P. When a leaf is removed from P when traversing a subtree $S(v_l)$ of T_2, then the corresponding bit in P is set to 0. Thus, using P to know readily which leaves of $L(v_1)$ must be used to identify new subtrees of v_2 that are to be removed, costs a time corresponding to $\#L(v_1)$, hence $O(n)$ over the whole execution.
- For lca queries, we use the dynamic structure of [12], which enables to obtain the lca of any two nodes in $O(1)$ worst case time, and is updated in the same time when a leaf is deleted from the tree.
  Globally, $O(n)$ lca queries issue from line 1: queries issue from set of leaf labels $L(v_1)$ taken from a cherry in $v_1 \in T_1$ and concern nodes in $v_2 \in T_2$. To identify $v_2 := lca_{T_2}(L(v_1))$, we need $\#L(v_1) - 1$ queries. But then these leaves are removed from the trees and v_1 becomes a leaf, that will be implied in a cherry at a latter step (if no conflict arise) and thus give rise to one lca query in turn. Thus, each node of T_1 will be used in at most one lca query, hence the algorithm performs $O(n)$ lca queries, each in $O(1)$.
  The data structure maintaining lcas relationships has also to be updated during the algorithm, but this requires $O(n)$ insertions and deletions of leaves, hence $O(n)$ globally: the number of leaves inserted (line 12) is bounded by the number cherries $v_1 \in T_1$ processed , hence is $O(n)$. Removing a subtree from T_2 (line 11) costs a number of leaf deletions proportionnal to the number of its nodes (performing a postorder traversal). There $O(n)$ nodes initially in T_2 and $O(n)$ will be added (line 12), thus $O(n)$ deletions are performed, each costing $O(1)$. Hence, deletions will cost $O(n)$ time to update the lca structure.

4.6 Proof of Corollary 1

Proof. Given a collection $\mathcal{T} = \{T_1, \ldots, T_k\}$ and an integer p, the algorithm to solve MAST, resp. MCT, follows the description given in [13]: recursively explore the different possible sets L' (initially $L' = \emptyset$) of leaves to remove from L in order for $\mathcal{T}|(L - L')$ to have an isomorphic subtree (MAST), resp. a refining subtree (MCT). Given a set L', algorithm 1 is applied to two input trees. If this gives a tree, the tree is reused in another run of algorithm 1 with another input tree. This process is repeated until either all k input trees have been considered (incorporated), in which case the last tree obtained is returned, either a set of leaves $\{a, b, c\}$ responsible for a conflict between some input trees is identified. In this case, branch in the search tree according to the three possibilities, i.e., adding a, b, or c to L', and proceed recursively with $\mathcal{T}|(L - L')$ and $p' = p - 1$. The exploration of a branch stops at worst when $p' = 0$. This brute-force algorithm either outputs a tree T on $L - L'$ in agreement with $\mathcal{T}$, resp. compatible with $\mathcal{T}$, and s.t. $\#(L-L') \leq p$, either answers that no such tree exists. There will be at most 3^p nodes in the search tree, each requiring k executions of algorithm 1, which costs $O(kn)$, and the removal of a given leaf from the k input trees, which is clearly $O(kn)$. Hence this algorithm requires $O(3^p kn)$ time.

The other term in the complexity bound comes from the use of 3HS as a subproblem. Indeed, in time $O(kn^3)$ it is possible to know the triples and fans induced by all input trees, hence the set $\mathcal{C}$ of three-leaves sets on which the conflicts arise. $\mathcal{C}$ can be seen as an instance of 3HS. Each hard conflicts will give rise to a set in $\mathcal{C}$, both in the case of MAST and MCT. Soft conflicts will add sets to $\mathcal{C}$ only in the case of MAST. Any hitting set of $\mathcal{C}$ is a set of leaves whose removal eliminates all conflicts. Thus, MCT and MAST can be solved in this way in time $O(H + kn^3)$, where H is the complexity needed to compute a hitting set of $\mathcal{C}$. □

4.7 Proof of Lemma 5

Proof. We consider the case where $T := SMAST(\mathcal{T})$, the proof for $T := SMCT(\mathcal{T})$ is similar. Suppose $\exists l \in \mathcal{L}(\mathcal{T}) - L(T)$ and let $T_i \in \mathcal{T}$ s.t. $l \in \mathcal{L}(T_i)$. Case (i): if $L(T) \cap L(T_i) < 2$ then consider the tree T' obtained in grafting a leaf with label l as a new child of $r(T)$.

Otherwise ($L(T) \cap L(T_i) \geq 2$), let $T_i' := T|L(T_i)$ and $T_i'' := T_i|(L(T) \cup \{l\})$. Since $T|L(T_i) \sqsubseteq T_i$, T_i'' differs from T_i' only in the fact that is has l as an extra leaf connected by an external edge to the rest of the tree. There are two cases depending on where this external edge (leading to leaf l) could be grafted in T_i' to obtain T_i'': (ii) by grafting it at a node $v \in T_i'$; (iii) by grafting it in the middle of an edge (u, v), and in this case, let v denote the lower extremity of the edge. In both case (ii) and (iii), if we define $L' := L(v)$, the node v is s.t. $v = lca_{T_i'}(L')$, with $L' \in L(T) \cap L(T_i)$ (note that L' can be a single leaf). Let $v' = lca_T(L')$.

We now consider the tree T' obtained by grafting l via a new external edge in T: if case (ii) applies, the new edge is grafted at v' (which then has a new leaf-subtree), else in case (iii) it is grafted on the edge above v' (which is then split into two edges).

In the three cases above, the tree T' is s.t. $L(T') = L(T) \cup \{l\} \subseteq L(\mathcal{T})$ and $\forall T_j \in \mathcal{T}, T_j \neq T_i$ we have $T'|L(T_j) = T|L(T_j) \sqsubseteq T_j$ (by definition of T and l) and we have $T'|L(T_i) \sqsubseteq T_i$ (in cases (ii) and (iii) we have $T'|L(T_i) = T_i'' \sqsubseteq T_i$), thus T' is a maximum agreement subtree of $\mathcal{T}$ of size greater than T, a contradiction with the definition of T.

□

The same lemma is valid for unrooted trees. Proceed by first rooting both T_i and T on the edge leading to a leaf in $L(T_i) \cap L(T)$.

4.8 Proof of Lemma 6

Proof. (i) Add all leaves $l \in \mathcal{L}(\mathcal{T})$ to T' as in proof of Lem. 5 to obtain a tree T that is an agreement supertree of $\mathcal{T}$. We claim that T is a maximum agreement supertree of $\mathcal{T}$ otherwise there is an agreement supertree T'' of $\mathcal{T}$ s.t. $\#T'' > \#T$. Since $\mathcal{L}(T) \subseteq L(T'')$ (Lem. 5) and $\mathcal{L}(\mathcal{T}) \subseteq L(T)$ by construction, this implies that

$$\#(T''|L_\cap) > \#(T|L_\cap) \tag{1}$$

(as $L(\mathcal{T})$ is partitioned into $L_\cap$ and $\mathcal{L}(\mathcal{T})$). By definition of T'', we have $T''|L(T_1) \sqsubseteq T_1$, thus $T''|L_\cap \sqsubseteq T_1|L_\cap$ and, similarly, $T''|L_\cap \sqsubseteq T_2|L_\cap$. Then $T''|L_\cap$ is an agreement subtree of $\{T_1|L_\cap, T_2|L_\cap\}$, as is T' by definition. But then

(1) implies that $T''|L_\cap$ is of larger size than $T' = T|L_\cap$, a contradiction with the definition of T'. *(ii)* is proved in the same way. □

4.9 Proof of Theorem 3

Proof. The correctness of the algorithm results from Lem. 6 and the fact that T is built from $T_m := MAST(\mathcal{T}|L_\cap)$ in a way ensuring that $T|L(T_m) = T_m, T|L(T_1) \sqsubseteq T_1$ and $T|L(T_2) \sqsubseteq T_2$. Specific subtrees of T_1 (resp. T_2) are always added to their respective place in the input trees: line 21, n_m has a twin node in T_1 and T_2, thus the specific subtrees hanging from these twin nodes are added to the corresponding node in T; line 24, if specific nodes branch above the twin node of n_m in T_1 ($i = 1$), then in T_2 ($i = 2$), they are grafted above the corresponding node in T, and in the same order. Note that subtrees of T_1 are arbitrarily grafted before those of T_2 (no constraint is given in the input trees for the respective order of these *specific* subtrees).

Concerning the complexity bound, the algorithm relies on a backbone tree T_m, computed in $O(N)$, the time required for solving the MAST problem on two rooted trees having $O(n)$ leaves (in the worst case all leaves of T_1, T_2 are common). All other operations performed by COMPSMASTTREE and GRAFTSPECIFIC cost $O(n)$, because they rely on a constant number of traversals of the trees, during which each of the $O(n)$ edges gives rise to a constant number of operations. The following shows in detail that the running time is $O(N + n)$:

- T_m is computed in $O(N)$, the complexity of MAST on two $O(n)$-leaf trees.
- COMPSMASTTREE : determining the sets of leaves and performing the required tree restrictions is clearly linear in n, the total number of labels. Handling the artificial root and leaf (lines 14,17) requires $O(1)$ pointer operations. Initializing Twin_1 and Twin_2 for leaves (traversal on line 15) is $O(1)$ for each leaf encountered. The subtrees of a node $n_i \in T_i$ are easily split into the lists $\mathsf{CommonChild}(n_i)$ and $\mathsf{SpecificChild}(n_i)$ in $O(1)$ depending on the fact that they contain or not a leaf from $L(T_m)$. This is handled easily by examining a flag returned by the recursive call issued for each subtree during the *postorder* traversal.
- GRAFTSPECIFIC: this procedure performs a recursive joint traversal of T_m and (a restriction of) T_1, T_2, during which each of the $O(n)$ edges of the three trees is examined once and supports $O(1)$ operations: (i) edges of T_m just guide the traversal; (ii) edges of T_1, T_2 corresponding to a path between two *twin* nodes (of some node $n_m \in T_m$ and its parent), are examined only once: in the `while` loop of the recursive call corresponding to n_m (the `for` loop handles successively the case of T_1 and T_2); edges leading to a specific subtree, or in a specific subtree, are examined during the only traversal of a subtree of this kind that copies its topology in the constructed tree T (lines 21, 24). Concerning T the built tree, its $O(n)$ nodes and edges are created directly from those of T_m and by adding copies of specific subtrees (whose total size is $O(n)$) of the input trees.

□

4.10 Proofs of intractability results

Proof of Lemma 8:

Proof. The graph $[\mathcal{G}, L(\mathcal{G})]$ associated with $\mathcal{G}$ is connected (cf Fig. 5). Hence applying Lem. 7 shows assertion 1. To prove assertion 2 assume w.l.o.g. that $j = 1$ ($\mathcal{G}$ is not altered by a common circular permutation of the two sequences $x^1, x^2, \ldots, x^m$ and $y^1, y^2 \ldots, y^m$). Fixing an arbitrary tree T^* on $\{x^2, x^3, \ldots, x^m\}$, we have to show that the tree $T := \mathrm{rake}(y^1, y^2, \ldots, y^m, T^*)$ on leaves $L(\mathcal{G}) - \{x^1\}$ is an agreement supertree of $\mathcal{G}$. For this purpose, we distinguish trees in $\mathcal{G}$ that do not contain x^1 from those who does:

- $\forall h \in [1, m-1]$, let $T_h := y^h|y^{h+1}x^{h+1} \in \mathcal{G}$, by definition of T it is easily seen that $T|L(T_h) \sqsubseteq T_h$ (in fact $T|L(T_h) = T_h$). The same results holds $\forall h \in [1, m-2]$ for trees $T_h := y^h|x^{h+1}x^{h+2} \in \mathcal{G}$.
- $x^1 = x^{m+1}$ is a leaf in tree $T_m := y^m|y^{m+1}x^{m+1} \in \mathcal{G}$ and in trees $T'_h := y^h|x^{h+1}x^{h+2} \in \mathcal{G}$ for $h \in \{m-1, m\}$. Hence, restricting these trees to $L(\mathcal{G}) - \{x^1\}$ reduces them to only two leaves, belonging to T, hence $T|L(T_h) \sqsubseteq T_h$ and $T|L(T'_h) \sqsubseteq T'_h$.

We have shown that $\forall T_h \in \mathcal{G}, T|L(T_h) \sqsubseteq T_h$. Remarking that $L(T) \subseteq L(\mathcal{G})$, this shows assertion 2. □

Proof of Theorem 4:

Proof. Given an instance $(\mathcal{T}, p)$ of SMAST and a tree T, checking whether T is an agreement supertree of $\mathcal{T}$ of size at least $\#L(\mathcal{T}) - p$ is done in polynomial time (by taking appropriate restrictions of trees, then resorting on a tree homeomorphism algorithm [19, 32] this takes polynomial time $O(nk)$). This shows that MAST is in NP.
We now reduce SMAST polynomially and preserving the parameter p to the hitting set (HS) problem:

Hitting Set (HS)
instance: A finite collection $\mathcal{C}$ of finite non-empty sets and an integer $p \geq 0$.
question: Does the collection $\mathcal{C}$ admit a hitting set of cardinal at most p?

HS is an alternate formulation of SET COVER and is NP and W[2]-hard for parameter p [15, Prop. 10].Let $(\mathcal{C}, p)$ be an instance of HS,

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

where $c := \#\mathcal{C}$ and $m_i := \#X_i$. 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 couples (i, j) with $i \in [1, c], j \in [1, m_i]$. We build an additional collection of non-intersecting sets

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

which is similar to $\mathcal{C}$ and 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 $\mathcal{T} := \mathcal{G}_1 \cup \mathcal{G}_2 \cup \ldots \cup \mathcal{G}_c$.

From the definition of gadgets (Def. 9), the transformation of instance $(\mathcal{C}, p)$ of HS to instance $(\mathcal{T}, p)$ of SMAST can be seen to take a polynomial time ($\mathcal{T}$ is of cardinal $2(m_1 + m_2 + \cdots + m_c)$) and is parameter preserving for p. By construction, all trees in $\mathcal{T}$ are on three leaves only. It remains to be proved that the following two assertions are equivalent:

(1) $\mathcal{C}$ admits a hitting set of size at most p
(2) there exists an agreement supertree of $\mathcal{T}$ whose size is at least $\#L(\mathcal{T}) - p$.

$(2) \Rightarrow (1)$. Let $T := SMAST(\mathcal{T})$ of size at least $\#L(\mathcal{T}) - p$. Thus $H := L(\mathcal{T}) - L(T)$ is a set of size at most p. Moreover, for any $i \in [1, c]$, we know that $L(\mathcal{G}_i) \not\subseteq L(T)$ because of assertion 1 of Lem. 8. Thus, at least one element of $L(\mathcal{G}_i)$ is not a leaf in T, hence is in H. This shows that H is an 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 an hitting set of $\mathcal{C}$ of size at most p.

$(1) \Rightarrow (2)$. For any $i \in [1, c], j \in [1, m_i]$ we denote σ_i^j the sequence corresponding to the $(j-1)$th cyclic shift of 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 size at most p. For each $i \in [1, c]$, H contains at least an element of X_i, written $x_i^{j_i}$, with $j_i \in [1, m_i]$. Concatenate sequences to define

$$T := \operatorname{rake}(\sigma_1^{j_1}, \sigma_2^{j_2}, \ldots, \sigma_c^{j_c}, T^*) .$$

where T^* is any tree on leaves $(X_1 \cup X_2 \cup \ldots \cup X_c) - H$. By construction, T is a tree on $L(\mathcal{T}) - H$ and thus of size at least $\#L(\mathcal{T}) - p$. Moreover, for each $i \in [1, c]$, T (consisting of leaves of the kind y_i^j hanging one by one from internal nodes on the path from the root of T to the root of subtree T^*) is s.t. $T|L(\mathcal{G}_i) = \operatorname{rake}(\sigma_i^{j_i}, T^* \mid X_i)$, which is an agreement supertree of $\mathcal{G}_i$ by assertion 2 of Lem. 8 (remark that $x_i^{j_i} \in H$ is not a leaf in T, hence not in $T^* \mid X_i$). Thus, T is an agreement supertree of $\mathcal{T}$ of size at least $\#L(\mathcal{T}) - p$. □

This results hold as well for the unrooted versions of SMAST (and SMCT), as the rooted versions can be easily reduced to the unrooted ones: let T_g be a binary subtree made of $\#L(\mathcal{T})$ new leaves, a collection $\mathcal{T}'$ of unrooted trees is obtained from the rooted trees of $\mathcal{T}$, unrooting each by grafting T_g to its root. This forces $T_g \sqsubseteq T$ for any $T := SMAST(\mathcal{T}')$, resp. $T := SMCT(\mathcal{T}')$. Then rooting T at the internal node to which T_g is connected in T ensures that $T|L(\mathcal{T})$ is a (rooted) maximum agreement supertree, resp. a (rooted) maximum compatible supertree, of $\mathcal{T}$.

4.11 Figures

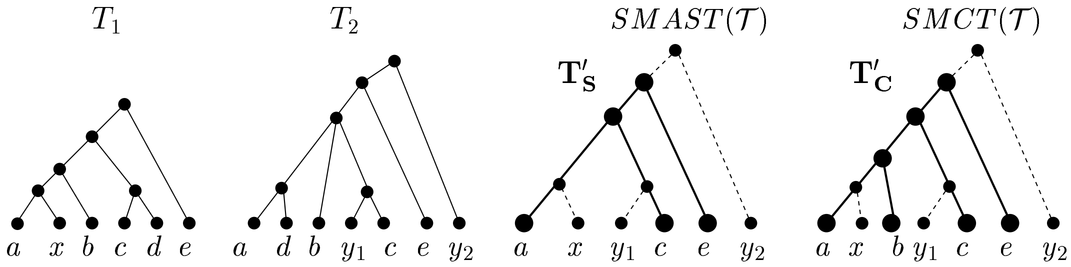

Fig. 2. A collection $\mathcal{T} = \{T_1, T_2\}$ of input trees ($L_\cap = \{a, b, c, d, e\}$), the trees $\mathbf{T'_S} := MAST(\mathcal{T}|L_\cap)$ and $\mathbf{T'_C} := MCT(\mathcal{T}|L_\cap)$ (in bold lines) and the supertrees $SMAST(\mathcal{T})$ and $SMCT_t(\mathcal{T})$.

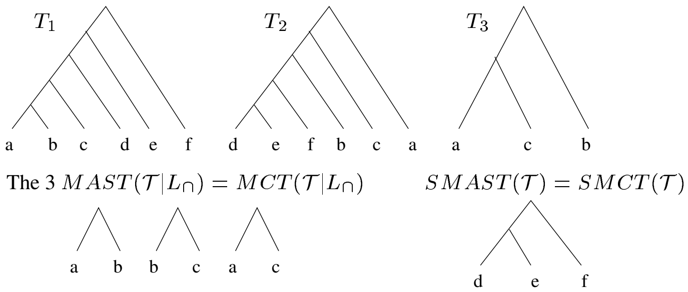

Fig. 3. A collection $\mathcal{T} = \{T_1, T_2, T_3\}$ of input trees for which the MAST and MCT problems cannot be used as a way to obtain $SMAST(\mathcal{T})$ and $SMCT(\mathcal{T})$. Here, $L_\cap = \{a, b, c\}$.

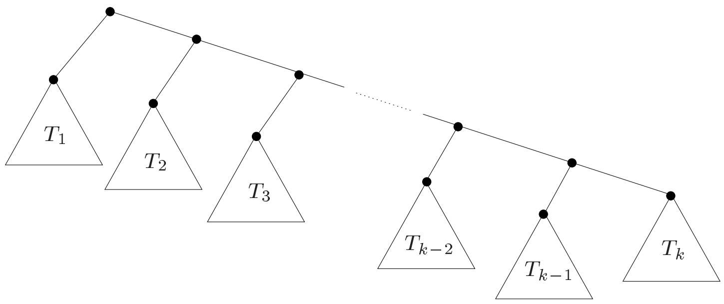

Fig. 4. The tree $\mathrm{rake}(T_1, T_2, \ldots, T_k)$ for a sequence $T_1, T_2, \ldots, T_k$ of trees.

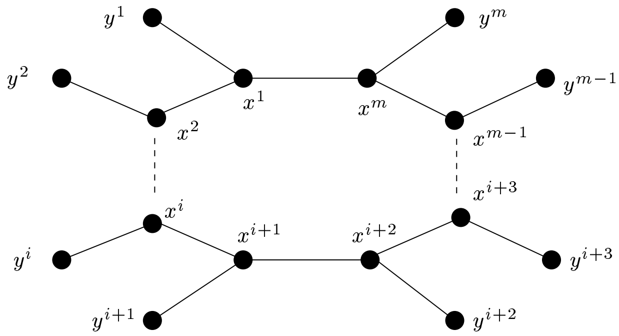

Fig. 5. The graph $[\mathcal{G}, L(\mathcal{G})]$ induced by the gadget $\mathcal{G}$.