Fixed-Parameter Tractability of the Maximum Agreement Supertree Problem

Sylvain Guillemot and Vincent Berry

***Abstract*— Given a set L of labels and a collection of rooted trees whose leaves are bijectively labelled by some elements of L, the Maximum Agreement Supertree problem (SMAST) is as follows: find a tree T on a largest label set $L' \subseteq L$ that homeomorphically contains every input tree restricted to L'. The problem has phylogenetic applications to infer supertrees and perform tree congruence analyses.**

In this paper, we focus on the parameterized complexity of this NP-hard problem, considering different combinations of parameters as well as particular cases. We show that SMAST on k rooted binary trees on a label set of size n can be solved in $O((8n)^k)$ time, which is an improvement with respect to the previously known $O(n^{3k^2})$ time algorithm. In this case, we also give an $O((2k)^p kn^2)$ time algorithm, where p is an upper bound on the number of leaves of L missing in a SMAST solution. This shows that SMAST can be solved efficiently when the input trees are mostly congruent. Then for the particular case where any triple of leaves is contained in at least one input tree, we give $O(4^p n^3)$ and $O(3.12^p + n^4)$ time algorithms, obtaining the first fixed-parameter tractable algorithms on a single parameter for this problem. We also obtain intractability results for several combinations of parameters, thus indicating that it is unlikely that fixed-parameter tractable algorithms can be found in these particular cases.

***Index Terms*— Phylogenetics, maximum agreement supertree, parameterized complexity, algorithms, reductions, rooted triples.**

I. INTRODUCTION

A. Motivation.

Supertree construction consists of building trees on a large set of labels from smaller trees covering parts of the label set. This task is applied in bioinformatics where trees represent phylogenies, but also in other fields such as databases [1] and data mining [2]. In phylogenetics, the tree nodes represent sequences or organisms (*taxa*), and the labels are bijectively associated with the leaves of the trees, representing current organisms, while internal nodes represent hypothetical ancestors. Rooted trees are usually described by their set of *clades*: a clade is the set of labels present under a same internal node. Clades represent related sets of organisms such as species, orders, families, etc. The goal of supertree methods is to infer a tree that complies as closely as possible with the topological information of the source trees. The task is relatively easy when the input trees fully agree on the relative positions of the labels. In this case, it is possible to find, in polynomial time, a supertree that contains any input tree as an induced subtree [1], hence fully respecting the topological information present in the data. However, practical input trees usually conflict with respect to the relative positioning of some labels. These incompatibilities sometimes affect only two input trees, but sometimes result from a combination of more source trees which do not conflict when considered pairwise.

The most used supertree methods focus on clades, e.g., the well-known Matrix Representation with Parsimony (MRP) method [3], [4] and its variants. This is a problem whenever the input trees contain some "rogue" taxa, i.e., labels whose position greatly differs from one input tree to another. Such rogue taxa can result from horizontal gene transfer (HGT) events [5], a phenomenon that commonly arises in bacteria, plants, and to a lesser extent among vertebrates. The presence of rogue taxa can induce tremendous changes in the clade set of an input tree, and hence have a non-negligible impact in the supertree obtained by clade-based methods. The Maximum Agreement Supertree (SMAST) method [6], [7], [8] has been specifically designed to deal with rogue taxa: it infers a supertree from a set of source trees by removing some labels, i.e., taxa, on the position of which the source trees disagree. More precisely, given a collection $\mathcal{T}$ of rooted trees with labels taken in a common set L, an *agreement supertree* for $\mathcal{T}$ is a tree T on a subset $L' \subseteq L$ such that each tree of $\mathcal{T}$ restricted to L' is included in T. The computational problem called SMAST, or sometimes MASP [6], consists of finding an agreement supertree containing the maximum number of labels from L.

The SMAST method, an extension of the maximum agreement subtree method (MAST), specifically allows the input trees to have different, and usually overlapping, label sets. With this flexibility, SMAST is well-adapted to replace MAST in several practical applications where input trees have non-identical label sets. The first such application is tree congruence analysis. Before building a supertree for a set of source trees, it is essential to certify that the source trees are not telling completely different stories on the evolution of studied taxa: if a set of source trees is not congruent enough, then no supertree can accurately represent the set. This can be problematic when subsequent analyses have to be conducted from the supertree, e.g., measuring the influence of geographical or climate factors on speciation events. Several studies have recently proposed procedures to assess the congruence of a set of source trees by randomization tests performed on MAST scores obtained for these trees: the source trees become more congruent as the number of leaves contained in their maximum agreement subtree increases [9], [10], [11]. The study of [9] focuses on the case where the considered source trees have different label sets and no taxon is common to all trees. They propose to divide the congruence analysis into MAST computations on pairs of trees. The obtained values are then normalized and summarized by an average value, for which a p-value is computed by similar MAST computations on random trees. Here, replacing MAST with SMAST copes with the fact that input trees have different label sets, and thus the congruence analysis can be performed directly: the whole set of trees can be considered at once, instead of resorting to separate analyses on

Manuscript submitted July 13, 2007

S. Guillemot and V. Berry are with the LIRMM, CNRS - University Montpellier 2. Emails: {vberry,sguillem}@lirmm.fr

pairs of trees. This is preferable to indirect analysis, since a same average value of the separate MAST computations on tree pairs can be induced by completely different situations: for instance, a small average value can be obtained when: *(i)* the input trees all roughly conflict on the same small label subset, where it is possible to obtain a large supertree complying with the source trees on all other labels plus on part of those involved in the conflict; *(ii)* when each pair of source trees conflict on different label subsets, where a supertree agreeing with all source trees can only contain a small portion of the labels. In contrast, resorting to SMAST instead of MAST can distinguish between these two different situations in the congruence of the source trees, with the SMAST value being large in case (i) and small in case (ii).

A second application where there can be some advantage to replacing MAST with SMAST is HGT detection [12], [13]. [12] show that MAST computations on gene trees enables the successful detection of HGT events. For each pair of trees, the size (i.e., number of labels) of a MAST is computed for the two trees restricted to their common labels. A gene tree is detected to be affected by HGT events depending on the distribution of MAST scores obtained for the pairs to which it belongs. However, here a MAST cannot be computed for a vast majority of pairs simply because the considered genes do not have enough labels in common, which limits the confidence in the final conclusions [12]. Replacing the MAST computations on pairs of gene trees by SMAST computations on more than two trees would undoubtedly increase the proportion of cases where there is enough overlap to conduct the analysis.

B. *Theoretical framework.*

The current paper addresses questions of parameterized complexity for the SMAST problem. The theory of parameterized complexity [14], [15] was developed as a framework to study computational problems which, in spite of being NP-hard, can be efficiently solved when a parameter of the problem is small. This situation occurs in various applied domains such as database querying and computational biology: (a) when answering a query in a database, the size of the query is small with respect to that of the database; (b) when dealing with with biological sequences, the size of the alphabet is small, e.g., 4 in the case of DNA sequences.

In both cases, algorithms with a time complexity that is exponential only within the parameter are practical, with the parameter being the query size in case (a) and the alphabet size in case (b). A well-known example for case (b) is the perfect phylogeny problem, related to character compatibility: given a set S of n sequences of m characters admitting k different states, does a tree exist whose leaves are bijectively labeled by S and internal nodes assigned to sequences of size m such that for each $i \in [m]$ and each state a, the subset of sequences having state a as ith character form a connected subgraph of T. This problem can be solved in $O(2^{3k}(nm^3 + m^4))$ [16].

Traditional computational complexity expresses time complexity of algorithms in terms of the instance size alone, while parameterized complexity considers both the instance size (usually denoted n) and a parameter (usually denoted k). Parameterized complexity theory makes a distinction between: (i) a problem solvable in $O(2^k n)$ time, (ii) a problem solvable in $O(n^k)$ time, and (iii) a problem which is NP-hard for any value of k larger than some constant. In case (i), the corresponding algorithm remains practical for large n values, provided that k is small. In case (ii), the algorithm is still practical for the smallest k values. In case (iii), the problem is not easier for instances where the parameter is small.

The central concept of fixed-parameter tractability has been introduced to deal with case (i). A problem is said to be fixed parameter tractable (fpt) if there is an algorithm that solves the problem on an instance of size n in time $O(f(k)n^c)$, where f is any function of the parameter k, and c is a constant independent of k. In most cases, f has exponential growth. The above definition naturally extends to problems involving a combination of several parameters. In the last 10 years, a large number of NP-hard problems have been shown to be fpt for natural parameters, particularly in the computational biology and graph theory fields.

Tools are also available to distinguish problems that specifically fall into case (ii) above. Parameterized complexity classes and parameterized reduction enables one to show that a parameterized problem is unlikely to be fpt. The ground complexity class is that of fpt problems and is denoted here FPT. The theory defines several other complexity classes, which are conjectured to properly contain the FPT class. Showing that a studied problem is hard for one of these classes is done by a parameterized reduction from an already classified problem, and rules out the possibility of an fpt algorithm (under some complexity-theoretic assumption). We refer the reader to [14], [15] for formal definitions of these concepts.

C. *Results.*

We first detail known theoretical results for the SMAST problem. Complexities for this problem are mainly expressed in terms of the total number n of distinct labels appearing in the input trees, and the number k of input trees. This problem involves several other natural parameters: d, the maximum outer degree (number of children) of a node in an input tree (when considering rooted input trees); l, an upper bound on the maximum size of the input trees; p (resp. q), an upper (resp. lower) bound on the number of input labels that are missing (resp. are present) in a SMAST solution. The SMAST problem is NP-hard as it generalizes the MAST problem [17]. It remains NP-hard when the outer degree d is unrestricted for $k \geq 3$ input trees [6], and for trees with $d \geq 2$ when k is unrestricted [6], [7]. When $k = 2$, SMAST can be solved in polynomial time by reduction to MAST [6], [7]. A sufficient condition for SMAST to be solved by resorting to MAST algorithms is also given in [7]. For such cases, [7] provide an algorithm for solving SMAST in time linear to that needed to solve MAST. For the particular case where $d = 2$, [6] give an $O(n^{3k^2})$ time algorithm for SMAST.

Until now, the only parameterized complexity result related to SMAST has been obtained for a decision version of the complement problem. The SMAST problem parameterized in p has been shown to be W[2]-hard [7], which rules out the possibility of an fpt algorithm for this parameterization of the problem. Several works have also considered the approximability of the corresponding minimization problem, where the measure is the number p of input labels missing in an outputted agreement supertree [6], [7]. The problem cannot be approximated in polynomial time within a constant factor, unless P = NP [7].

In this paper, we focus on the particular case where $d = 2$. Note that in phylogenetics, SMAST input trees will often be binary

Parameters	Complexity of SMAST	Source
q	W[1]-complete (even for $l = 3$)	[6], Thm 6 & 7
q, k	W[1]-complete	Thm 6
p	W[2]-hard (even for $l = 3$)	[7]
k, p	fpt by an $O((2k)^p \times kn^2)$ time algorithm	Thm 2
k	XNL-hard Solvable in $O((8n)^k)$ time	Thm 6 Thm 3

TABLE I
SUMMARY OF PREVIOUS AND NEW RESULTS. n IS THE NUMBER OF DISTINCT LABELS APPEARING IN THE INPUT TREES; k IS THE NUMBER OF INPUT TREES; l IS AN UPPER BOUND ON THE MAXIMUM SIZE OF THE INPUT TREES; p, RESPECTIVELY q, IS AN UPPER BOUND, RESPECTIVELY LOWER BOUND, ON THE NUMBER OF INPUT LABELS THAT ARE MISSING, RESPECTIVELY ARE PRESENT, IN A SMAST SOLUTION.

as a result of the optimization algorithms used to analyze raw molecular data. We improve on previous results in several ways.

First, we give an algorithm that solves SMAST on k rooted binary trees on a label set of size n in $O((2k)^p kn^2)$ time. This algorithm is only exponential in p, the number of input labels that are missing in a SMAST solution. Thus, the algorithm will be reasonably fast when dealing with trees inferred by different methods on a same data set, or with trees inferred from genes displaying a low level of conflict. Then we provide an $O((8n)^k)$ time algorithm, independent of p. This is a significant improvement on the $O(n^{3k^2})$ time algorithm of [6] and shows that SMAST is tractable for a small number of trees, extending the previously known results for $k = 2$ trees [6], [7].

Secondly, we consider SMAST on collections of rooted triples (binary trees on 3 leaves), focusing on the complexity of this variant parameterized in p. Since this problem is equivalent to SMAST in its general setting [7], it is W[2]-hard. However, we show here that an fpt algorithm can be achieved for *complete* collections of rooted triples, i.e., when there is at least one rooted triple for each set of 3 labels in L. This results from the fact that conflicts between input trees can be circumvented to small sets of labels, leading to $O(4^p n^3)$ and $O(3.12^p + n^4)$ time algorithms. Note that this result also applies to input trees of arbitrary size, provided their decomposition in rooted triples yields a complete collection.

Lastly, we obtain some fixed-parameter intractability results, showing that SMAST is hard for several parameterized complexity classes when considering various parameters. The classes of interest here are W[1], W[2] and XNL (introduced in Section V), and the considered parameters are k, p, q and/or l. The intractability results we obtain are detailed in Table I together with other results for the problem. In particular, the W[1]-completeness of SMAST on binary trees regarding parameter p, resp. k, q, contrasts with the results obtained for MAST. Indeed, the latter is polynomial for binary trees [18], [19] and, for trees of unbounded degree, MAST is fpt in p [20], [21] and fpt in k, q [22].

Overall, this paper proposes a number of results on the parameterized complexity of the SMAST problem for binary trees, including two fpt algorithms and an algorithm that runs in polynomial time for a fixed number of input trees.

II. DEFINITIONS

In this paper, we consider rooted trees which are bijectively leaf-labelled. We first define some notations for these trees, their nodes and subtrees.

Definition 1: Let T be a leaf-labelled tree. We identify its leaf set with its label set, denoted by $L(T)$. The *size* of T is the number of its labels, i.e., $|T| = |L(T)|$. The node set of T is denoted by $N(T)$, and $r(T)$ stands for the root of T. We use a recursive parenthesized notation for trees: if ℓ is a label, then ℓ denotes the trivial tree whose root is a leaf labelled by ℓ; if $T_1, ..., T_k$ are trees, then $(T_1, ..., T_k)$ stands for the tree whose root is unlabeled and has $T_1, ..., T_k$ as child subtrees. If u is a node in a tree T, then $T(u)$ stands for the complete subtree of T rooted at u (i.e., the subtree made of all nodes descending from u), and $L(u)$ for the label set of this subtree, i.e., the labels descending from u. If u, v are two nodes of T, then $u <_T v$ means that u is a proper descendant of v in T; we denote by $u \leq_T v$ if and only if $u <_T v$ or $u = v$. The smallest upper bound of two nodes u, v of T with respect to $<_T$ is called the lowest common ancestor of u, v, and is denoted by $\mathsf{lca}_T(u, v)$.

Given a tree T and a label set L, the *restriction* of T to L, denoted by $T|L$, is the tree homeomorphic to the smallest subtree of T connecting leaves of L. Let T, T' be two trees. We say that T *embeds* in T', denoted by $T \leq T'$, if and only if $T = T'|L(T)$. We say that T and T' *agree* if and only if $T|L(T') = T'|L(T)$. A *collection* is a family $\mathcal{T} = \{T_1, ..., T_k\}$ of trees, the label set of the collection is $L(\mathcal{T}) = \cup_{i=1}^{k} L(T_i)$. Given a label set L, the *restriction* of $\mathcal{T}$ to L is the collection $\mathcal{T}|L = \{T_1|L, ..., T_k|L\}$. See Figure 1 for an example of a collection.

We now recall several useful relations on trees.

Definition 2: An *agreement supertree* for a collection $\mathcal{T}$ is a tree S such that $L(S) \subseteq L(\mathcal{T})$ and for each $T_i \in \mathcal{T}$, S and T_i agree. We say that S is a *total agreement supertree* for $\mathcal{T}$ if additionally $L(S) = L(\mathcal{T})$. A collection $\mathcal{T}$ is *compatible* if and only if there exists a total agreement supertree for $\mathcal{T}$. A *conflict* among $\mathcal{T}$ is a set $C \subseteq L(\mathcal{T})$ such that $\mathcal{T}|C$ is incompatible. For instance, $S = (((a, b), c), (e, f))$ is an agreement supertree for the collection $\mathcal{T}$ of Figure 1, and $C = \{a, b, c, d\}$ is a conflict among $\mathcal{T}$.

The MAXIMUM AGREEMENT SUPERTREE problem (SMAST) asks: given a collection $\mathcal{T}$, find an agreement supertree for $\mathcal{T}$ with the largest size. Equivalently, this amounts to seek a largest set $L \subseteq L(\mathcal{T})$ such that $\mathcal{T}|L$ is compatible. The size of such an *optimal* solution is denoted by $\mathsf{smast}(\mathcal{T})$ and $SMAST(\mathcal{T})$ stands for the set of agreement supertrees of $\mathcal{T}$. See Figure 2 for an example on a real data.

We also denote by P-SMAST the parameterized version of SMAST, which asks: given a collection $\mathcal{T}$ and a parameter p, can $\mathcal{T}$ be made compatible by removing at most p distinct labels

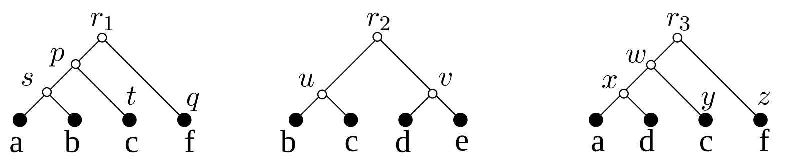

Fig. 1. A collection $\mathcal{T}$ of three input trees on the label set $L = \{a, b, c, d, e, f\}$. Several internal nodes are also given names (but not *labels*) for the purpose of referencing them in the text.

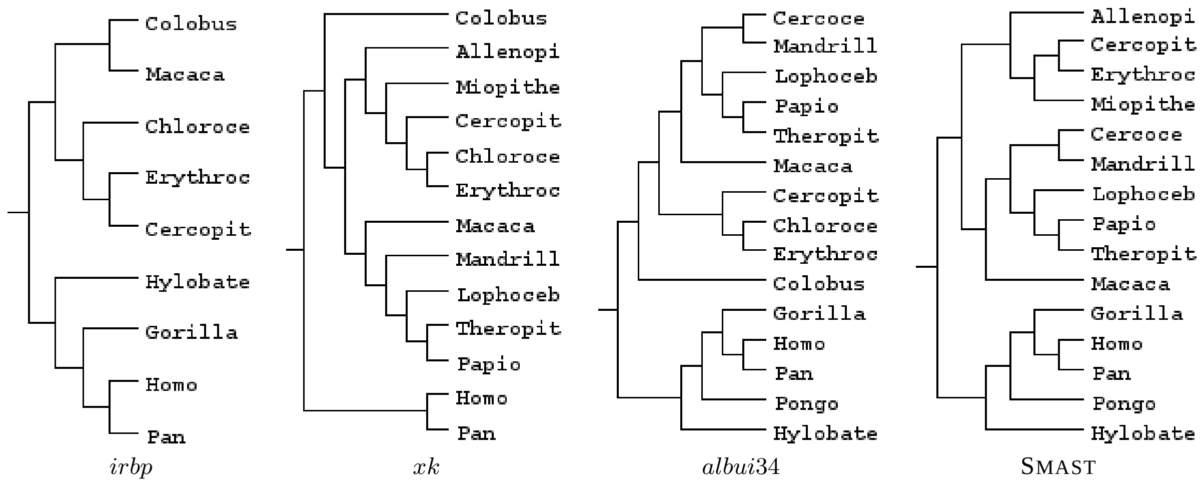

Fig. 2. A collection of three source trees on Old World primates (Catarrhinii) and a maximum agreement supertree, represented as cladograms. Source trees were respectively obtained by PhyML analyses of DNA sequences of Interphotoreceptor Retinoid Binding Protein exon 1 ($irbp$), chromosome Xq13.3 fragment (xk) and Albumin gene introns 3 and 4 ($albui34$). The maximum agreement supertree is obtained by removing leaves Colobus and Chloroce on the relative position of which the source trees disagree.

form the trees in $\mathcal{T}$?

III. ALGORITHMS FOR SOLVING SMAST ON BINARY TREES

Throughout this section, we consider a fixed collection $\mathcal{T} = \{T_1, ..., T_k\}$ of binary trees, let n denote the size of the corresponding label set $L(\mathcal{T})$ and k the number of trees in $\mathcal{T}$. In the following algorithms, we usually consider nodes of input trees as ancestors of label sets. Furthermore, a tree T will sometimes be considered together with a set L of labels, some of which will come from other trees. In such cases, we will be interested in the node of T that is the least common ancestor of the labels in L appearing in T. In the case where no label of L appears in T, we assign the label set with the null node, denoted by the special symbol $\perp$. We adjoin the null node to the node set $N(T)$ of any tree T for the rest of Section III, as this facilitates descriptions. We also extend the notation for complete subtrees, assuming $T(\perp)$ denotes the empty tree, and that for node descendencies in trees, assuming $\perp \leq_T u$ for each $u \in N(T)$.

As several papers describing algorithms for MAST, we will consider tuples of nodes, each tuple containing a node per input tree. We will solve SMAST recursively by considering subtrees of the input trees whose roots will correspond to the nodes in tuples. Formal definitions are given below:

Definition 3: A *position* in $\mathcal{T}$ is a tuple $\pi = (u_1, ..., u_k)$, where each u_i is in the respective set $N(T_i)$ and is called a *component* of π. For any $i \in [k]$, the ith component of π is denoted $\pi[i]$. We define the *initial position* $\pi_\top = (r(T_1), ..., r(T_k))$ and the *final position* $\pi_\perp = (\perp, ..., \perp)$. The set of labels under a position π is denoted $L(\pi) = \cup_{i \in [k]} L(\pi[i])$.

A. Solving SMAST in $O((2k)^p \times kn^2)$ time

In this section, we first describe an algorithm deciding the compatibility of a collection in $O(kn^2)$ time, *and* returning a conflict of size $\leq 2k$ in case of incompatibility. This yields an fpt algorithm for P-SMAST with $O((2k)^p \times kn^2)$ running time.

The compatibility of a collection $\mathcal{T}$ can be decided by the well-known BUILD algorithm [1], [23]. However, in case of incompatibility, this algorithm does not provide a conflict, which is required here to serve as basis for a bounded search fpt algorithm. Like BUILD, the compatibility algorithm presented here progressively builds the supertree using a recursive top-down approach. Each step constructs a graph where the connected components correspond to subtrees of the supertree. Here, we replace the graphs used in BUILD with graphs $G(\mathcal{T}, \pi)$, with varying postitions π. When such a graph is connected, it yields a conflict of size $\leq 2k$, identified thanks to a spanning tree. The recursive steps of the algorithm focus on particular positions π in the collection $\mathcal{T}$. We first define these positions, then we will define the $G(\mathcal{T}, \pi)$ graphs and state associated results.

Definition 4: A position is *reduced* if and only if no component is a leaf (i.e., each component is either $\perp$ or an internal node). To any position π, we associate a reduced position $\pi\!\downarrow$ by replacing by $\perp$ any component of π that is a leaf.

Given a position π in $\mathcal{T}$, we set $\mathcal{T}(\pi) = \{T_1(\pi[1]), ..., T_k(\pi[k])\}$; we say that π is *compatible* if and only if $\mathcal{T}(\pi)$ is compatible.

Observe that:

Lemma 1:

1) $\pi_\perp$ is compatible.
2) π is compatible if and only if $\pi\!\downarrow$ is compatible.

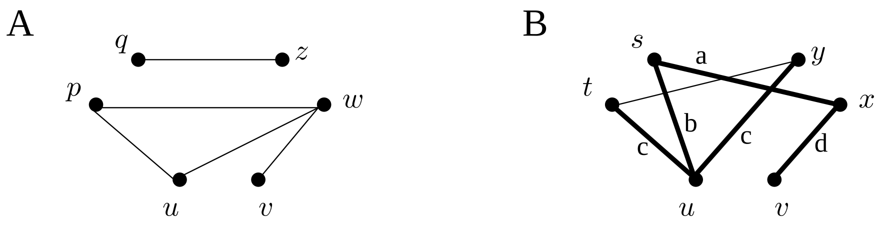

Fig. 3. A. The graph $G(\mathcal{T}, \pi_\top)$ of the position $\pi_\top = (r_1, r_2, r_3)$ for the collection of trees of Figure 1. This graph is disconnected, the two connected components indicate the two successor positions $\pi_1 = (p, r_2, w)$ and $\pi_2 = (q, \perp, z)$ of $\pi_\top$. B. The graph $G(\mathcal{T}, \pi_1)$ is connected. Choosing a spanning tree (bold edges) of the graph and an arbitrary label shared by the two subtrees corresponding to the extremities of each edge of this tree identifies a conflict $C = \{a, b, c, d\}$.

3) if $\mathcal{T}$ is compatible then $\mathcal{T}(\pi)$ is compatible, for any position π in $\mathcal{T}$.

Proof: Points 1 and 2 result from definitions. Point 3 results from the fact that any tree of $\mathcal{T}(\pi)$ is a restriction of a tree in $\mathcal{T}$. The compatibility of $\mathcal{T}(\pi)$ hence follows from that of $\mathcal{T}$. ■

We now turn to the definition of the graphs that will serve as a basis for testing the compatibility of a collection.

Definition 5: Let π be a reduced position, the graph $G(\mathcal{T}, \pi)$ is defined as follows:

(i) its vertex set V is composed of the two child nodes of each node $\pi[i]$ such that $\pi[i] \neq \perp$;
(ii) two vertices $u, v \in V$ are adjacent if and only if $L(u) \cap L(v) \neq \emptyset$.

In other terms, $G(\mathcal{T}, \pi)$ is the intersection graph of the set system $\{L(u) : u \in V\}$.

We are also interested in subsets $V' \subseteq V$ of vertices in a graph $G(\mathcal{T}, \pi)$, to which we can associate respective successor positions of π:

Definition 6: Given $V' \subseteq V$, we define the successor of π with respect to V', denoted by $succ_{V'}(\pi)$, as the position π' such that

- if $\pi[i] = \perp$, then $\pi'[i] = \perp$;
- if $\pi[i]$ is an internal node u_i of T_i, with children v_i, v'_i, then either:

$$\begin{cases} \text{if } v_i \in V' \text{ and } v'_i \notin V' \text{ then } \pi'[i] = v_i, \\ \text{if } v_i \notin V' \text{ and } v'_i \in V' \text{ then } \pi'[i] = v'_i, \\ \text{if } v_i \in V' \text{ and } v'_i \in V' \text{ then } \pi'[i] = u_i, \\ \text{if } v_i \notin V' \text{ and } v'_i \notin V' \text{ then } \pi'[i] = \perp. \end{cases}$$

In other words, $succ_{V'}(\pi)$ is the position whose ith component is the root of the smallest subtree in T_i including nodes in V', or is set to $\perp$ when V' contains no node of T_i. See Figure 3A for an illustration of these definitions.

We now describe a recursive algorithm to decide the compatibility of a given position (see pseudo-code ISCOMPATIBLE). Calling this algorithm with $\pi_\top$ allows us to decide the compatibility of $\mathcal{T}$. Any recursive step of the algorithm is given a position π in $\mathcal{T}$. For the rest of the algorithm, by considering $\pi\!\downarrow$ instead of π, we can assume that π is a reduced position. The base case of the recursion corresponds to $\pi = \pi_\perp$, the algorithm then succeeds since $\pi_\perp$ is known to be compatible. The general case of the recursion corresponds to a reduced position $\pi \neq \pi_\perp$, for which the algorithm tries to identify two successors π_1, π_2, corresponding to child subtrees of an hypothetical agreement supertree for $\mathcal{T}(\pi)$. To that aim, it considers the graph $G(\mathcal{T}, \pi)$, and performs a connectivity test on this graph. If the graph is not connected, then the connectivity test yields a partition of V into two disconnected sets V_1, V_2 (where V_2 can contain several connected components); then the successor positions are $\pi_1 = succ_{V_1}(\pi), \pi_2 = succ_{V_2}(\pi)$, and recursive calls are issued for π_1, π_2. The correctness of this step is precisely stated in Lemma 2. If the graph is connected, then the connectivity test yields a spanning tree of $G(\mathcal{T}, \pi)$ from which a conflict can be obtained by choosing, for each edge (u, v) of the tree, a label present in $L(u) \cap L(v)$ (as shown in Lemma 3). See Figure 3B for an illustration of this process.

To prove these lemmas, we need some intermediary results on parts of a graph $G(\mathcal{T}, \pi)$. Given $V_1, V_2 \subseteq V$, we say that V_1, V_2 are *connected* if and only if $G(\mathcal{T}, \pi)$ contains an edge (u, v) with $u \in V_1$ and $v \in V_2$; otherwise, V_1, V_2 are said to be *disconnected*. Additionally, given $V' \subseteq V$, set $L(V') = \cup_{u \in V'} L(u)$.

Observation 1: Two sets $V_1, V_2 \subseteq V$ are connected in $G(\mathcal{T}, \pi)$ if and only if $L(V_1) \cap L(V_2) \neq \emptyset$.

Proof: V_1, V_2 are connected if and only if there exists $u \in V_1, v \in V_2$ such that $L(u) \cap L(v) \neq \emptyset$, which is equivalent to the statement $L(V_1) \cap L(V_2) \neq \emptyset$. ■

Observation 2: Let $V' \subseteq V$, and let $\pi' = succ_{V'}(\pi)$. Then: $L(\pi') = L(V')$.

Proof: For each $i \in [k]$, let V'_i be the set of nodes of V' belonging to $L(T_i)$. Observe that $L(V') = \cup_{i \in [k]} L(V'_i)$ and that $L(\pi') = \cup_{i \in [k]} L(\pi'[i])$. We conclude by noting that for each $i \in [k]$, $L(V'_i) = L(\pi'[i])$, by definition of the notation $succ$. ■

Observation 3: Let $V_1, V_2 \subseteq V$, and let $\pi_1 = succ_{V_1}(\pi), \pi_2 = succ_{V_2}(\pi)$. Then, $L(\pi_1) \cap L(\pi_2) = \emptyset$ if and only if V_1, V_2 are disconnected in $G(\mathcal{T}, \pi)$.

Proof: Directly follows by applying Observations 1 and 2. ■

Let us now consider a reduced position $\pi \neq \pi_\perp$. We have the following recursive characterization of compatibility:

Lemma 2: Suppose that π is a reduced position such that $\pi \neq \pi_\perp$ and let V be the vertex set of the graph $G(\mathcal{T}, \pi)$. The following statements are equivalent:

- π is compatible;
- there exists a partition V_1, V_2 of V such that
  (i) V_1, V_2 are disconnected in $G(\mathcal{T}, \pi)$, and
  (ii) $\pi_1 = succ_{V_1}(\pi)$ and $\pi_2 = succ_{V_2}(\pi)$ are compatible.

Proof: For a given partition V_1, V_2 of V, and for $j \in \{1, 2\}$, let $\pi_j = succ_{V_j}(\pi)$.

($\Rightarrow$). Suppose that π is compatible. Let S be a total agreement supertree for $\mathcal{T}(\pi)$. Since $\pi \neq \pi_\perp$, then $|L(\pi)| \geq 2$, hence $S = (S_1, S_2)$. Since S is a total agreement supertree for $\mathcal{T}(\pi)$, for each $i \in [k]$ the subtree $T_i(\pi[i])$ embeds in S, which is denoted $T_i(\pi[i]) \leq S$. Define a partition V_1, V_2 of V by considering all components $i \in [k]$ of π as follows. Suppose that $\pi[i]$ is an internal node of T_i with children v_i, v'_i. Then $T_i(\pi[i]) = (T_i(v_i), T_i(v'_i))$ using bracket notation for trees. Together with $T_i(\pi[i]) \leq S$, this yields either $T_i(v_i) \leq S_1$ or $T_i(v_i) \leq S_2$: add v_i to V_1 in the first case, and to V_2 in the second case. Proceed similarly for v'_i. In the end, V_1 and V_2 are a partition of V such that $L(V_1) \subseteq L(S_1)$ and $L(V_2) \subseteq L(S_2)$, i.e., such that $L(V_1) \cap L(V_2) = \emptyset$. Thus, we obtain Point (i) by applying Observation 1. We now prove Point (ii): π_1 and π_2 are positions in $\mathcal{T}(\pi)$, and since $\mathcal{T}(\pi)$ is compatible by hypothesis, it follows that π_1 and π_2 are compatible by Point 3 of Lemma 1.

($\Leftarrow$). Suppose that there exists a partition V_1, V_2 of V satisfying Points (i) and (ii). Since π_1 and π_2 are compatible by hypothesis, there exists a total agreement supertree S_1 for $\mathcal{T}(\pi_1)$ and similarly S_2 for $\mathcal{T}(\pi_2)$. We then have $L(\pi_1) = L(S_1)$ and $L(\pi_2) = L(S_2)$. Since V_1, V_2 are disconnected, $L(\pi_1) \cap L(\pi_2) = \emptyset$ by Observation 3. Therefore, we have $L(S_1) \cap L(S_2) = \emptyset$, and we can define the tree $S = (S_1, S_2)$. We show that S is a total agreement supertree for $\mathcal{T}(\pi)$ by showing that $T_i(\pi[i]) \leq S$ for each $i \in [k]$.

Fix such an i, let $u_i = \pi[i]$. If $u_i = \perp$, then the relation obviously holds. Suppose now that u_i is an internal node of T_i, and let v_i, v'_i be its two children. We consider three cases. If $v_i, v'_i \in V_1$: then $\pi_1[i] = u_i$, which together with the definition of S_1 implies $T_i(u_i) \leq S_1$; we conclude that $T_i(u_i) \leq S$. If $v_i, v'_i \in V_2$: in a similar way, we conclude that $T_i(u_i) \leq S$. If $v_i \in V_1, v'_i \in V_2$: then $\pi_1[i] = v_i$, which implies that $T_i(v_i) \leq S_1$, and $\pi_2[i] = v'_i$, which implies that $T_i(v'_i) \leq S_2$. It is easy to see that $T_i(v_i) \leq S_1$ and $T_i(v'_i) \leq S_2$ imply that $T_i(u_i) = (T_i(v_i), T_i(v'_i)) \leq (S_1, S_2) = S$. ■

We now prove that if the graph $G(\mathcal{T}, \pi)$ turns out to be connected, a spanning tree of this graph yields a small conflict among $\mathcal{T}$:

Lemma 3: Let π be a reduced position such that $\pi \neq \pi_\perp$. Suppose that $G(\mathcal{T}, \pi)$ is connected, and let $T = (V, F)$ be a spanning tree of $G(\mathcal{T}, \pi)$. For each edge $e = (u, v) \in F$, choose $\ell_e \in L(u) \cap L(v)$. Then $C = \{\ell_e : e \in F\}$ is a conflict among $\mathcal{T}$.

Proof: We show that $\mathcal{T}' = \mathcal{T}|C$ is incompatible. For each i such that $\pi[i] \neq \perp$, let $u_i = \pi[i]$, and let v_i, v'_i be its two children in T_i. By definition of C, the sets $L(v_i) \cap C, L(v'_i) \cap C$ are not empty, hence to the nodes u_i, v_i, v'_i there corresponds nodes $\tilde{u}_i, \tilde{v}_i, \tilde{v}'_i$ in $T_i|C$, where $\tilde{u}_i$ is the least common ancestor in $T_i|C$ of $L(u_i) \cap C$, and the other two nodes are defined similarly. Let π' be the reduced position in $\mathcal{T}'$ obtained by setting $\pi'[i] = \perp$ if $\pi[i] = \perp$, or $\pi'[i] = \tilde{u}_i$ $\pi[i] \neq \perp$. Consider the graph $G(\mathcal{T}', \pi')$, then by definition of C for each edge (x, y) of T, the edge $(\tilde{x}, \tilde{y})$ is present in $G(\mathcal{T}', \pi')$, therefore the tree T' formed by these edges is a spanning tree of $G(\mathcal{T}', \pi')$, hence the graph is connected. By Lemma 2, we conclude that π' is an incompatible position of $\mathcal{T}'$, therefore $\mathcal{T}'$ is incompatible (by Point 3 of Lemma 1). ■

Lemmas 2 and 3 give rise to an algorithm for deciding the compatibility of a collection, and obtaining a conflict of small size in case of incompatibility.

Theorem 1: There is an algorithm which, in $O(kn^2)$ time, decides if $\mathcal{T}$ is compatible, and returns a conflict of size $\leq 2k$ in case of incompatibility.

Proof: We rely on the procedure IsCompatible(π) which takes as input a position π in $\mathcal{T}$, decides if π is compatible and returns a conflict of size $\leq 2k$ in case of incompatibility. The procedure is formally stated in the pseudo-code called Algorithm 1. To decide if $\mathcal{T}$ is compatible, the procedure is called with the argument $\pi_\top$.

The correctness of the procedure IsCompatible follows from Lemmas 2 and 3. For the running time, we rely on the fact that using appropriate data structures, we can ensure that a call to IsCompatible takes $O(kn)$ time (see Appendix I for details). Moreover, when a call IsCompatible(π) issues two recursive calls for positions π_1, π_2, then $L(\pi_1), L(\pi_2)$ are disjoint (by Observation 3) and included in $L(\pi)$ (as a consequence of Observation 2). Hence the total number of calls to IsCompatible is $O(n)$, therefore the total running time of the algorithm is $O(kn^2)$. ■

Algorithm 1: IsCompatible(π)

Input: A position π in a collection $\mathcal{T}$ of trees.

Result: A tuple (B, C) where

- B is a boolean indicating whether π is compatible
- C is a conflict among π when the position π is not compatible.

```
if $\pi = \pi_\perp$ then return (true, $\emptyset$)
if $\pi$ is not reduced then $\pi \leftarrow \pi\downarrow$
if $G(\mathcal{T}, \pi)$ is connected then
    $C \leftarrow \emptyset$ ; Let $T = (V, F)$ be a spanning tree of $G$
    foreach edge $(u, v) \in F$ do
        choose a label $\ell \in L(u) \cap L(v)$ ; $C \leftarrow C \cup \{\ell\}$
    return (false, $C$);
else
    Let $V_1$ be a connected component of $G(\mathcal{T}, \pi)$ ; $V_2 \leftarrow V - V_1$
    $\pi_1 \leftarrow succ_{V_1}(\pi)$ ; $\pi_2 \leftarrow succ_{V_2}(\pi)$
    $(B_1, C_1) \leftarrow$ IsCompatible($\pi_1$) ; if $B_1$ is false then return (false, $C_1$)
    $(B_2, C_2) \leftarrow$ IsCompatible($\pi_2$) ; if $B_2$ is false then return (false, $C_2$)
    return (true, $\emptyset$)
```

On the basis of this compatibility algorithm, we can design a simple fpt algorithm for solving P-Smast on a collection $\mathcal{T}$ with parameter p (see end of section II for the formal definition of this problem). Algorithm 2 contains the pseudo-code for this procedure, called RecSMAST, that uses the well-known bounded search tree technique. Note that the third argument mentioned in the heading of the procedure, namely X, is only present in order to know a set of leaves to remove from the input trees in case of success. The initial call to the algorithm uses $X = \emptyset$.

Theorem 2: The P-Smast problem can be solved in $O((2k)^p \times kn^2)$ time.

Proof: A run of the algorithm follows a search tree of height $\leq p$, whose nodes at depth i are each labelled by a set of labels $X \subseteq L$ such that $|X| = i$. At a given node u labelled by a set X, the algorithm determines in $O(kn^2)$ time if $\mathcal{T}|(L \backslash X)$ is compatible, using the procedure of Theorem 1. If the answer is positive, the node is labelled by "success", and is then a leaf of the search tree. Otherwise, the algorithm proceeds as follows: if the node is at depth p, then it is labelled by "failure" and becomes a leaf of the search tree; if it is at depth $< p$, then the procedure of Theorem 1 has returned a conflict C of size $\leq 2k$, and for each $\ell \in C$ a child node of u is added in the search tree, with label $X \cup \{x\}$. The running time follows easily, since the search tree

Algorithm 2: RECSMAST($\mathcal{T}, p, X$)

Input: A collection $\mathcal{T}$ of rooted binary trees, an integer $p \geq 0$ and a set $R \subseteq L(\mathcal{T})$.

Result: A tuple (B, X) where

- B is a boolean stating whether a solution to SMAST for $\mathcal{T}$ can be obtained by removing at most p leaves from $L(\mathcal{T})$;
- X is a set of leaves to remove from $L(\mathcal{T})$ to obtain such a solution when it exists, otherwise $X = \emptyset$.

```
$\pi_\top \leftarrow$ the initial position of $\mathcal{T}$, ie $(r(T_1), r(T_2), \ldots, r(T_k))$
$(compatible, C) \leftarrow$ ISCOMPATIBLE($\pi_\top$)
if $compatible$ is true then return (True, $X$)
if $p > 0$ then
    foreach label $\ell \in C$ do
        $(r_c, X_c) \leftarrow$RECSMAST$(\mathcal{T}|(L(\mathcal{T}) - \{\ell\}), p-1, X \cup \{\ell\})$
        if $r_c$ is true then return (true, $X_c$)
return (false, $\emptyset$)
```

has height $\leq p$, degree $\leq 2k$, and since each node is processed in $O(kn^2)$ time. ■

B. *Solving* SMAST *in* $O((8n)^k)$ *time*

In this section, we describe an algorithm to solve SMAST in $O((8n)^k)$ time. The algorithm uses dynamic programming, and is somewhat similar in spirit to the algorithm described in [24] for solving MAST on two trees.

For the needs of this section, it is convenient to characterize the agreement relation on trees in terms of *partial embeddings*. First define $\mathsf{child}_T(u, v)$ as the child of v along the path joining v to u in T, where u, v are two nodes of T such that $u <_T v$. Let T, T' be two trees, say that a partial embedding of T into T' is a function $\phi : N(T) \to N(T')$ such that:

- for any u leaf of T, we have $\phi(u) = \perp$ if $u \notin L(T')$, or $\phi(u) = u$ otherwise,
- for any u internal node of T with children $u_1, \ldots, u_p$, let $V = \{j : \phi(u_j) \neq \perp\}$, then (i) either $V = \emptyset$, and $\phi(u) = \perp$, (ii) either $V = \{i\}$ and $\phi(u) = \phi(u_i)$, (iii) or $|V| \geq 2$ and $\phi(u_i) <_T \phi(u)$ for each $i \in V$, and the nodes $\{\mathsf{child}_T(\phi(u), \phi(u_i)) : i \in V\}$ are pairwise distinct.

Then T and T' agree if and only if there exists a partial embedding of T into T' (and equivalently a partial embedding of T' into T).

Let $\mathcal{T}$ be a collection and π a position in $\mathcal{T}$. Let $SMAST(\pi)$ denote the set of trees T such that (i) T is an agreement supertree for $\mathcal{T}$, (ii) for each i, the partial embedding $\phi_i : T \to T_i$ is such that $\phi_i(r(T)) \leq_{T_i} \pi[i]$. We denote by $\mathsf{smast}(\pi)$ the size of a largest tree of $SMAST(\pi)$.

The algorithm computes values $\mathsf{smast}(\pi)$ for each position π using a recurrence relation whose base case is stated in Lemma 4 and general case is stated in Lemma 5. The recurrence relation relies on a partial order $\leq_{\mathcal{T}}$ on positions, which is defined below. Given a position π, $\mathsf{smast}(\pi)$ will be computed from values $\mathsf{smast}(\pi')$ with $\pi' <_{\mathcal{T}} \pi$. At the end of the algorithm, $\mathsf{smast}(\mathcal{T})$ is obtained as $\mathsf{smast}(\pi_\top)$.

We define the relation $\leq_{\mathcal{T}}$ on positions in $\mathcal{T}$ by: $\pi \leq_{\mathcal{T}} \pi'$ if and only if for each $i \in [k]$, $\pi[i] \leq_{T_i} \pi'[i]$. We denote by $<_{\mathcal{T}}$ its strict counterpart, where $\pi <_{\mathcal{T}} \pi'$ if and only if for each $i \in [k]$, $\pi[i] \leq_{T_i} \pi'[i]$, and one of these relations is strict.

The following observation states that agreement supertrees of restricted parts of the input trees (identified in a position π) are also agreement supertrees of wider parts of the input trees (associated with a position π with $\pi' \subseteq \pi$).

Observation 4: If $\pi' \leq_{\mathcal{T}} \pi$, then $SMAST(\pi') \subseteq SMAST(\pi)$.

The base case of the recurrence corresponds to *terminal* positions: a position π is *terminal* if and only if for each $i \in [k]$, $\pi[i]$ is a leaf or $\perp$. For a terminal position π, note that $L(\pi)$ is the set of labels occuring as components in π. Moreover, in this case, say that an element $x \in L(\pi)$ is *maximally present* if and only if for each $i \in [k]$, $x \in L(T_i)$ implies $\pi[i] = x$. Let $P(\pi)$ denote the set of maximally present elements of $L(\pi)$. Then:

Lemma 4: Suppose that π is terminal. Then: $\mathsf{smast}(\pi) = |P(\pi)|$.

Proof: First, let T be any binary tree on the label set $P(\pi)$, then $T \in SMAST(\pi)$. Indeed, for each $i \in [k]$ define ϕ_i as follows:

- if $\pi[i] = \perp$, then $\phi_i(u) = \perp$ for each $u \in L(T)$;
- if $\pi[i]$ is a leaf x of T_i, then $\phi_i(u) = x$ if $x \in T(u)$, otherwise $\phi_i(u) = \perp$.

Then ϕ_i is a partial embedding of T into T_i satisfying $\phi_i(r(T)) \leq_{T_i} \pi[i]$. We conclude that $T \in SMAST(\pi)$.

We now show that for each $T \in SMAST(\pi)$, we have $L(T) \subseteq P(\pi)$. Indeed, consider such a tree T, and for each $i \in [k]$ consider the partial embedding $\phi_i : T \to T_i$. Fix an element $x \in L(T)$, and consider $i \in [k]$ such that $x \in L(T_i)$, we show that $\pi[i] = x$. By definition of a partial embedding we have: $\phi_i(x) = x$. Since $\phi_i(x) \leq_{T_i} \pi[i]$ and $\pi[i]$ is a leaf (because π is terminal), it follows that $\pi[i] = x$. We conclude that $x \in P(\pi)$. ■

We now describe the general case of the recurrence relation, corresponding to nonterminal positions. If π is nonterminal, then $\mathsf{smast}(\pi)$ is computed from two values $\mathsf{smast}_1(\pi), \mathsf{smast}_2(\pi)$.

We first define $\mathsf{smast}_1(\pi)$. Say that a position π' is a *successor* of π if and only if there exists $i \in [k]$ such that $\pi'[i]$ is a child of $\pi[i]$ and $\pi'[j] = \pi[j]$ for each $j \neq i$. Let $S(\pi)$ denote the set of successors of π. Then define:

$$\mathsf{smast}_1(\pi) = \max_{\pi' \in S(\pi)} \mathsf{smast}(\pi'). \tag{1}$$

We now define $\mathsf{smast}_2(\pi)$. Say that a pair (π_1, π_2) of positions is a *decomposition* of π if and only if (i) $\pi_1 \neq \pi, \pi_2 \neq \pi$ and (ii) for each $i \in [k]$, the following holds:

- either $\pi[i] = \perp$, in which case $\pi_1[i] = \pi_2[i] = \perp$;
- either $\pi[i]$ is a leaf x, in which case we have $\{\pi_1[i], \pi_2[i]\} = \{\perp, x\}$;
- either $\pi[i]$ is an internal node u with two children v, v', in which case we have either $\{\pi_1[i], \pi_2[i]\} = \{\perp, u\}$ or $\{\pi_1[i], \pi_2[i]\} = \{v, v'\}$.

Let $D(\pi)$ denote the set of decompositions of π. Then define:

$$\mathsf{smast}_2(\pi) = \max_{(\pi_1, \pi_2) \in D(\pi)} (\mathsf{smast}(\pi_1) + \mathsf{smast}(\pi_2)). \tag{2}$$

Note that computing the values $\mathsf{smast}_1(\pi)$ and $\mathsf{smast}_2(\pi)$ only involves values $\mathsf{smast}(\pi')$ with $\pi' <_{\mathcal{T}} \pi$, by the following observation:

Observation 5:

(i) If $\pi' \in S(\pi)$, then $\pi' <_{\mathcal{T}} \pi$;

(ii) If $(\pi_1, \pi_2) \in D(\pi)$ then $\pi_1 <_{\mathcal{T}} \pi$ and $\pi_2 <_{\mathcal{T}} \pi$.

We are now ready to state the relation for nonterminal positions:

Lemma 5: Suppose that π is not terminal. Then: $\mathsf{smast}(\pi) = \max(\mathsf{smast}_1(\pi), \mathsf{smast}_2(\pi))$.

Proof: We first prove that $\mathsf{smast}_1(\pi) \le \mathsf{smast}(\pi)$. Let $S \in SMAST(\pi')$ for some $\pi' \in S(\pi)$, such that $|S|$ is maximal. Since $\pi' <_{\mathcal{T}} \pi$ by Lemma 5, we have $S \in SMAST(\pi)$ by Observation 4, and the result follows.

We now prove that $\mathsf{smast}_2(\pi) \le \mathsf{smast}(\pi)$. Let $(\pi_1, \pi_2) \in D(\pi)$, and let S_1, S_2 such that $S_j \in SMAST(\pi_i)$, $|S_j|$ maximal. If one of the S_j's is empty, say S_1, then $\mathsf{smast}(\pi_1) = 0$, and we obtain $\mathsf{smast}_2(\pi) = |S_2| = \mathsf{smast}(\pi_2) \le \mathsf{smast}(\pi)$ by Observations 4 and 5. Suppose now that S_1, S_2 are not empty. For $j \in \{1, 2\}$, since $S_j \in SMAST(\pi_j)$, there exists partial embeddings $\phi_{j,i} : S_j \to T_i$ such that $\phi_{j,i}(r(S_i)) \le_{T_i} \pi_j[i]$ for each $i \in [k]$. Let $S = (S_1, S_2)$, we claim that $S \in SMAST(\pi)$. Indeed, define $\phi_i : S \to T_i$ as follows. Set $\phi_i(x) = \phi_{j,i}(x)$ if x is a node of S_j, and $\phi_i(x) = \mathsf{lca}_{T_i}(\phi_{1,i}(r(S_1)), \phi_{2,i}(r(S_2))$ if x is the root of S. Then: (i) $L(S_1) \cap L(S_2) = \emptyset$, hence S is well-defined, (ii) ϕ_i is a partial embedding of S into T_i, (iii) $\phi_i(r(S)) \le_{T_i} \pi[i]$ (see Appendix II-A for a proof). We conclude that $\mathsf{smast}_2(\pi) = |S_1| + |S_2| = |S| \le \mathsf{smast}(\pi)$.

Finally, we show that $\mathsf{smast}(\pi) \le \max(\mathsf{smast}_1(\pi), \mathsf{smast}_2(\pi))$. Let $S \in SMAST(\pi)$ such that $|S|$ is maximal. Then there exists partial embeddings $\phi_i : S \to T_i$ such that $\phi_i(r(S)) \le_{T_i} \pi[i]$ for each $i \in [k]$. Let $u_i = \phi_i(r(S))$ for each i. We consider two cases.

First case: there exists $i \in [k]$ such that $u_i <_{T_i} \pi[i]$. This case holds in particular if $|S| \le 1$. Define π' from π by setting the ith component to $\mathsf{child}_{T_i}(u_i, \pi[i])$, then $\pi' \in S(\pi)$. We verify that $S \in SMAST(\pi')$: indeed, ϕ_i is a partial embedding of S into T_i such that $\phi_i(r(S)) \le_{T_i} \pi'[i]$. We conclude that $|S| = \mathsf{smast}(\pi) \le \mathsf{smast}(\pi') \le \mathsf{smast}_1(\pi)$.

Second case: $u_i = \pi[i]$ for each $i \in [k]$. In this case, we have $|S| \ge 2$, hence $S = (S_1, S_2)$. Let u be the root of S, let v_j be the root of S_j in S, then $\pi = (\phi_1(u), ..., \phi_k(u))$. For $j \in \{1, 2\}$, define π_j as follows: given $i \in [k]$, (i) if $\phi_i(v_j) = \phi_i(u)$, set $\pi_j[i] = \phi_i(u)$, (ii) if $\phi_i(v_j) = \perp$, set $\pi_j[i] = \perp$, (iii) if $\phi_i(v_j) <_{T_i} \phi_i(u)$, set $\pi_j[i] = \mathsf{child}_{T_i}(\phi_i(v_j), \phi_i(u))$. Then $(\pi_1, \pi_2) \in D(\pi)$ (see Appendix II-B for a proof). We now show that $S_j \in SMAST(\pi_j)$: indeed, ϕ_i is a partial embedding of S_j into T_i, and by definition of π_j we have $\phi_i(r(S_j)) \le_{T_i} \pi_j[i]$ for each $i \in [k]$. We conclude that $|S| = \mathsf{smast}(\pi) = |S_1| + |S_2| \le \mathsf{smast}(\pi_1) + \mathsf{smast}(\pi_2) \le \mathsf{smast}_2(\pi)$. ■

Lemmas 4 and 5 yield an algorithm for computing $\mathsf{smast}(\mathcal{T})$:

Theorem 3: $\mathsf{smast}(\mathcal{T})$ can be computed in $O((8n)^k)$ time and $O((2n)^k)$ space.

Proof: Using dynamic programming, the algorithm computes the values $\mathsf{smast}(\pi)$ for each π position in $\mathcal{T}$, using the recurrence relations stated in Lemmas 4 and 5. The correctness of the algorithm follows from the lemmas, and the termination of the algorithm is ensured by Observation 5 and the fact that $<_{\mathcal{T}}$ is an order relation on positions in $\mathcal{T}$.

We now consider the space and time requirements for the algorithm. First observe that the number of positions π in $\mathcal{T}$ is $\le (2n)^k$: a component $\pi[i]$ has $\le 2n$ possible values (one of the $\le 2n - 1$ nodes of T_i, or the value $\perp$). It follows that the space complexity is $O((2n)^k)$. We claim that the time complexity is $O((8n)^k)$. Indeed, consider the time required to compute $\mathsf{smast}(\pi)$, assuming that the values $\mathsf{smast}(\pi')$ for $\pi' <_{\mathcal{T}} \pi$ are available. Testing if π is terminal requires $O(k)$ time. If π is terminal, computing $|P(\pi)|$ takes $O(k)$ time. If π is nonterminal, then we need to compute $\mathsf{smast}_1(\pi)$ and $\mathsf{smast}_2(\pi)$, which respectively require $O(k)$ and $O(4^k)$ time. Thus, $\mathsf{smast}(\pi)$ is computed in $O(4^k)$ time, hence the total running time of the algorithm is $O((8n)^k)$. ■

We note that after the first version of this paper was submitted, an $O((6n)^k)$ algorithm was provided in [25]. However, applying a finer mathematical analysis of the subcases encountered by our algorithm, similar to that of [25], also yields an $O((6n)^k)$ time complexity. We refer the reader to [25] for mathematical details.

IV. Algorithm for solving Smast on complete collections of triples

Recall that P-SMAST is the version of SMAST parameterized in the number p of distinct labels to remove from the input trees to obtain an agreement. We consider in this section the restriction of P-SMAST to complete collections of rooted triples.

A *rooted triple* (or *triple* for short) is a binary tree T such that $|L(T)| = 3$; such a tree has the form $T = ((x, y), z)$ in parenthetical notation, and will be denoted by $xy|z$. A *collection of triples* is a collection $\mathcal{R} = \{T_1, ..., T_k\}$ where each T_i is a triple. $\mathcal{R}$ is *complete* if each set of three labels in $L(\mathcal{R})$ is present in at least one T_i. To a binary tree T of arbitrary size, we associate a complete collection of triples $rt(T)$ formed by the triples $T_i \le T$; to a collection $\mathcal{T}$, we associate a collection of triples $rt(\mathcal{T}) = \cup_{T \in \mathcal{T}} rt(T)$. For a complete collection of triples $\mathcal{R}$, we say that $\mathcal{R}$ is *treelike* if there exists a tree T such that $\mathcal{R} = rt(T)$; then we say that $\mathcal{R}$ *displays* T.

We consider the following parameterized problem, denoted P-SMASTCR: given a complete collection of triples $\mathcal{R}$ and a parameter p, can $\mathcal{R}$ be made treelike by removing at most p distinct labels? Observe that this problem is the restriction of P-SMAST to complete collections of triples, since for such collections treelikeness is equivalent to compatibility, as defined in Section II.

This section presents an fpt-algorithm to solve P-SMASTCR, which contrasts with the fact that P-SMAST is W[2]-hard on non-complete collections of triples. This algorithm also applies to collections of general trees such that any triple of labels is present in at least one input tree. Indeed, recall that any tree can be equivalently described by the triples it contains.

It is possible to show that non-treelike complete collections of triples have conflicts of size ≤ 4, a result similar to that known on quartets [26]. This allows to solve P-SMASTCR in $O(n^4 + 3.12^p)$ time by reduction to 4-HITTING SET [27], and also in $O(4^p n^4)$ time by bounded search (similar to the work of [28] for the minimum quartet inconsistency problem). In the following, we describe a faster algorithm with $O(4^p n^3)$ running time. We first present an algorithm to decide treelikeness in linear $O(n^3)$ time (Proposition 1 and Theorem 4).

Proposition 1: There is an algorithm INSERT-LABEL-OR-FIND-CONFLICT$(\mathcal{R}, X, x, T)$ which takes a complete collection of triples $\mathcal{R}$, a set $X \subseteq L(\mathcal{R})$, an element $x \in L(\mathcal{R}) \backslash X$ and a tree T such that $\mathcal{R}|X$ displays T, and in $O(n^2)$ time decides if $\mathcal{R}' = \mathcal{R}|(X \cup \{x\})$ is treelike. Additionally, the algorithm returns the tree T' displayed by $\mathcal{R}'$ in case of positive answer, or returns a conflict C among $\mathcal{R}'$ with $|C| \le 4$ in case of negative answer.

Proof: In a first step, the algorithm checks whether $\mathcal{R}$ contains two different triples on the same set of three labels

x, ℓ, ℓ'. In such a case, they form a conflict of size 3 which is then returned by the algorithm.

If no such conflict is found, the algorithm proceeds to a second step during which it determines for each internal node u of T, the relative subtree in which u would accept to insert x: its left subtree (denoted L), its right subtree (denoted R), or the subtree *above* it, i.e., the part of the tree excluding $T(u)$ (denoted A), namely the part of the tree that is not below u. To that aim, the algorithm checks that the triples x, ℓ, ℓ', with ℓ, ℓ' labels under u in T, all indicate the same subtree relative to u. More formally, let v, v' be the two children of u. An *u-fork* is a pair $\{\ell, \ell'\}$ where $\ell \in L(v), \ell' \in L(v')$. Each u-fork $\{\ell, \ell'\}$ gives an opinion $o_{\ell,\ell'}$ on the positioning of x with respect to u in T, where $o_{\ell,\ell'}$ is computed from $\mathcal{R}$ as follows: if $\ell x|\ell' \in \mathcal{R}$ then $o_{\ell,\ell'}$ is set to L, if $\ell' x|\ell \in \mathcal{R}$ then $o_{\ell,\ell'}$ is set to R, otherwise, $\ell\ell'|x \in \mathcal{R}$ and $o_{\ell,\ell'}$ is set to A. The algorithm considers each internal node u in turn and computes the opinions $o_{\ell,\ell'}$ of the u-forks $\{\ell, \ell'\}$. If two u-forks indicate a different subtree for x, then the algorithm easily identifies a conflict. In such a case, it can be shown that there exist ℓ, ℓ_1, ℓ_2 such that $o_{\ell,\ell_1} \neq o_{\ell,\ell_2}$ (or $o_{\ell_1,\ell} \neq o_{\ell_2,\ell}$), in which case $C = \{x, \ell_1, \ell_2, \ell\}$ is a conflict, which is then returned by the algorithm. Otherwise, all u-forks indicate the same subtree for x, and the opinion of u, denoted o_u is defined to be this direction (L, R or A).

In a third step, the algorithm checks that the opinions of the different nodes u in T consistently indicate a single position to insert x in T. The opinions are compatible if and only if for each edge u, v of T with u above v, we have: (i) if v is the left child of u, then $o_u = R \Rightarrow o_v = A$, (ii) if v is the right child of u, then $o_u = L \Rightarrow o_v = A$, (iii) if v is a child of u, then $o_u = A \Rightarrow o_v = A$. If one pair of nodes u, v does not meet the above requirements, then by considering $\{\ell, \ell'\}$ v-fork and $\{\ell, \ell''\}$ u-fork, we obtain a conflict $C = \{x, \ell, \ell', \ell''\}$. Otherwise, consider the sets of nodes u such that $o_u \neq A$, they form a (possibly empty) path in T starting at the root and ending at a node v. Then $\mathcal{R}|(X \cup \{x\})$ is treelike, and displays the tree obtained from T by inserting x above v, which is returned by the algorithm.

We now justify the running time of the algorithm. The first step trivially takes $O(n^2)$ time. Consider the second step. Given a node u, let F_u be the set of u-forks, then an internal node u is processed in time $O(|F_u|)$. Therefore, the time required by the second step is $\sum_u O(|F_u|) = O(n^2)$. Now consider the third step. The algorithm checks that for each edge u, v of T, Conditions (i)-(ii)-(iii) hold: for a given edge, checking the conditions or finding a conflict is done in constant time, hence the time required by this step is $O(n)$. It follows that the total time required by the algorithm is $O(n^2)$. ■

Theorem 4: There is an algorithm FIND-TREE-OR-CONFLICT($\mathcal{R}$) which takes a complete collection of triples $\mathcal{R}$, and in $O(n^3)$ time decides if $\mathcal{R}$ is treelike, returns a tree T displayed by $\mathcal{R}$ in case of positive answer, or a conflict C among $\mathcal{R}$ with $|C| \leq 4$ in case of negative answer.

Proof: We use the procedure INSERT-LABEL-OR-FIND-CONFLICT to decide treelikeness as follows. We iteratively insert each label, starting from an empty tree, until: (i) either every label has been inserted, in which case the collection is treelike and the displayed tree is returned, (ii) or a conflict is found and returned. ■

Using bounded search, we obtain:

Theorem 5: The P-SMASTCR problem can be solved in $O(4^p n^3)$ time.

V. HARDNESS RESULTS

The parameterized complexity of the SMAST problem on binary trees is considered with respect to the following parameters: k denotes the number of input trees, l denotes an upper bound on the maximum size of the input trees, p (resp. q) denotes an upper (resp. lower) bound on the number of labels to remove (resp. conserve) in order to obtain compatibility of the collection. After having obtained an fpt algorithm when the SMAST problem is parameterized in k, p, we now turn to intractability results. We remind the reader that W[1], W[2] and XNL are parameterized complexity classes which are conjectured to properly contain FPT. They have the respective complete problems:

- W[1]: CLIQUE: given a graph G and a parameter q, decide if G has a clique of size $\geq q$;
- W[2]: DOMINATING SET: given a graph G and a parameter q, decide if G has a dominating set of size $\leq q$;
- XNL: BOUNDED SPACE TURING MACHINE COMPUTATION: given a nondeterministic Turing machine M with a binary tape alphabet, an integer n in unary, and a parameter q, decide if M accepts the empty string using space $\leq q \log_2 n$.

The class XNL is a parameterized analogue of the class NL; it has been introduced in [29], [15], note that the class we call XNL is the class $[\text{UNIFORM-XNL}]^{FPT}$ of [29].

The intractability results we prove here mainly follow from similar results for the SLCS problem [30], which we now define. A *p-sequence* (after [31], or sequence for short) s is a word without repetition on an alphabet L. We denote by $L(s) \subseteq L$ the *label set* of s, i.e., the set of letters (or *labels*) appearing in s. We define the relation $<_s$ on $L(s)$ by: $x <_s y$ if and only if x precedes y in s. A *collection* (of sequences) is a family $\mathcal{C} = \{s_1, ..., s_k\}$, where the s_is are sequences. The *label set* of $\mathcal{C}$ is $L(\mathcal{C}) = \cup_{i \in [k]} L(s_i)$.

Given a sequence s and a label set L', we denote $s|L'$ the *restriction* of s to L'. Given two sequences s, s', we say that s and s' *agree* if $s|L(s') = s'|L(s)$. A *compatible sequence* for a collection $\mathcal{C} = \{s_1, ..., s_k\}$ is a sequence s such that $L(s) \subseteq L(\mathcal{C})$ and for each $i \in [k]$, s and s_i agree.

The SLCS problem consists in finding a largest compatible sequence of a collection $\mathcal{C}$ (the size of such a sequence is denoted by $\#SLCS(\mathcal{C})$). While the SLCS and SMAST problems are optimization problems, for the need of the proofs we consider their decision version SLCS-D and SMAST-D, which are defined as follows. SLCS-D takes a collection $\mathcal{C}$ of k sequences and an integer q, and asks if $\#SLCS(\mathcal{C}) \geq q$. SMAST-D takes a collection $\mathcal{T}$ of k trees and an integer q, and asks if $\mathsf{smast}(\mathcal{T}) \geq q$. We denote by P-SLCS-D (resp. P-SMAST-D) the problem SLCS-D (resp. SMAST-D) parameterized by k, q.

We rely on a parameter-preserving reduction from P-SLCS-D to P-SMAST-D. For the sake of clarity, the reduction is performed in two steps.

First step: a parameter-preserving reduction from P-SLCS-D to a variant called P-COLORED-SLCS. This problem is defined as follows. Given a label set L partitioned in q sets $L_1, ..., L_q$, and a collection $\mathcal{C}$ on L, a *colored sequence* is a sequence $a_1 ... a_q$ with

$a_i \in L_i$. The problem P-COLORED-SLCS asks: given parameters k, q, a collection $\mathcal{C}$ of k sequences on a label set partitioned in q sets, does $\mathcal{C}$ have a colored compatible sequence? We show:

Lemma 6: There is a polynomial-time reduction from P-SLCS-D to P-COLORED-SLCS which maps an instance $(\mathcal{C}, k, q)$ of P-SLCS-D to an instance $(\mathcal{C}', 2k, q)$ of P-COLORED-SLCS.

Proof: Given an instance $I = (\mathcal{C}, k, q)$ of P-SLCS-D, we construct an instance $I' = (\mathcal{C}', 2k, q)$ of P-COLORED-SLCS as follows. Suppose that $\mathcal{C} = \{s_1, ..., s_k\}$ has label set L. For each $x \in L$ we create new labels $x^1, ..., x^q$, we set $L'^i = \{x^i : x \in L\}$ and $L' = L'^1 \cup ... \cup L'^q$. Consider the morphisms of free monoids ϕ, ϕ', from L^* to L'^*, defined as follows: for each $x \in L$,

$$\begin{cases} \phi(x) = x^1 ... x^q \\ \phi'(x) = x^q ... x^1 \end{cases}$$

For each sequence $s_i \in \mathcal{C}$, define $s'_i = \phi(s_i)$ and $s''_i = \phi'(s_i)$. Then $\mathcal{C}' = \{s'_1, s''_1, ..., s'_k, s''_k\}$.
Note that $\mathcal{C}'$ contains $2k$ sequences. Thus, the reduction is parameter-preserving. The correction of the reduction is detailed in Appendix III-A. ■

Second step: we give a parameter-preserving reduction from P-COLORED-SLCS to P-SMAST-D. If $T_1, ..., T_m$ are trees, the notation $rake(T_1, ..., T_m)$ is defined as follows using the parenthesized notation for trees:

$$\begin{cases} rake(T_1) & = T_1 \\ rake(T_1, ..., T_m) & = (rake(T_1, ..., T_{m-1}), T_m) \end{cases}$$

In other words, $rake(T_1, ..., T_m)$ is a caterpillar tree whose leaves are replaced by the trees $T_1, ..., T_m$ hanging in increasing order from the bootom to the root the tree. We show:

Lemma 7: There is a polynomial-time reduction from P-COLORED-SLCS to P-SMAST-D which maps an instance $(\mathcal{C}, k, q)$ of P-COLORED-SLCS to an instance $(\mathcal{T}, k+2, 2q+1)$ of P-SMAST-D.

Proof: Let $I = (\mathcal{C}, k, q)$ be an instance of P-COLORED-SLCS, where $\mathcal{C} = \{s_1, ..., s_k\}$ is a collection on a label set L, partitioned in q sets $L_1, ..., L_q$. We construct an instance $I' = (\mathcal{T}, k', q')$ of SMAST$[k, q]$ as follows.

- we first define the label set L': we create new labels $z_0, z_1, ..., z_q$. For each $i \in [q]$, we set $L'_i = L_i \cup \{z_i\}$, and we define $L' = \{z_0\} \cup L'_1 \cup ... \cup L'_q$.
- we define $\mathcal{T} = \{S, S'\} \cup \{T_1, ..., T_k\}$ as follows. For each $i \in [q]$, we define R_i, R'_i as follows: consider an enumeration of $L'_i = \{x_1, ..., x_m\}$, then $R_i = rake(x_1, ..., x_m)$ and $R'_i = rake(x_m, ..., x_1)$. We then set $S = rake(z_0, R_1, ..., R_q)$ and $S' = rake(z_0, R'_1, ..., R'_q)$. For each sequence $s_i = y_1 ... z_n$ in $\mathcal{C}$, we create a tree $T_i = rake(z_0, y_1, ..., y_n)$.
- we set $k' = k+2$ and $q' = 2q+1$.

Note that $\mathcal{T}$ contains $k+2$ trees. Thus, the reduction is parameter-preserving. The correction of the reduction is detailed in Appendix III-B. ■

Combining the two above results we obtain:

Proposition 2: There is a polynomial-time reduction from P-SLCS-D to P-SMAST-D which maps an instance $(\mathcal{C}, k, q)$ of P-SLCS-D to an instance $(\mathcal{T}, 2k+2, 2q+1)$ of P-SMAST-D.

Proof: Direct consequence of Lemma 6 and Lemma 7. ■

Proposition 2 allows us to transfer to SMAST known hardness results for SLCS [30]:

Theorem 6: The following results hold for SMAST:

- W[1]-hardness for q and for q, k;
- XNL-hardness for k.

Proof: The hardness results follow from similar results for SLCS [30], and from the parameter-preserving reduction given by Proposition 2. ■

In addition, we now show membership in W[1] for SMAST parameterized by q (Thm 7) by resorting to triples (see Section IV for related definitions). We rely on two preliminary lemmas.

Proposition 3: Let T be a tree such that $L(T) \subseteq L(\mathcal{T})$. The following statements are equivalent:

- T is an agreement supertree for $\mathcal{T}$;
- $rt(\mathcal{T})|L(T) \subseteq rt(T)$.

Proof: Observe that for each $T_i \in \mathcal{T}$, there is equivalence between: (i) T and T_i agree, (ii) $rt(T_i)|L(T) \subseteq rt(T)$. ■

We are now ready to show:

Theorem 7: SMAST parameterized in q is in W[1].

Proof: We use a parameterized reduction to SHORT TURING MACHINE COMPUTATION [14]. Let $I = (\mathcal{T}, q)$ be an instance of SMAST, where $\mathcal{T}$ is a collection and q an integer. We define a nondeterministic Turing machine M which accepts the empty string in q' steps if and only if $\mathcal{T}$ has an agreement supertree of size $\geq q$.

The tape alphabet of M consists of the following symbols:

- a symbol p_x for each $x \in L$;
- a symbol $r_{xy|z}$ for each $x, y, z \in L$, $x < y$ and $xz|y, yz|x \notin rt(\mathcal{T})$.

In a first step, M guesses q symbols p_x, and $\binom{q}{3}$ symbols $r_{xy|z}$. The idea is that for a consistent solution, the symbols p_x will correspond to a label set L, and the symbols $r_{xy|z}$ will form a complete collection of triples $\mathcal{R}$, such that: (i) $L(\mathcal{R}) = L$, (ii) $\mathcal{R}$ is treelike. Then $\mathcal{R} = rt(T)$ for some tree T, and since $rt(\mathcal{T})|L \subseteq rt(T)$ by definition of the symbols $r_{xy|z}$, it will follow that T is an agreement supertree for $\mathcal{T}$ by Proposition 3.

In a second step, M checks that the labels p_x and $r_{xy|z}$ are consistent. First, it checks that the symbols $p_{x_1}, ..., p_{x_q}$ are such that $x_1 < ... < x_q$, which requires $O(q)$ steps. Let $L = \{x_1, ..., x_q\}$, then M verifies that for each $x, y, z \in L$ distinct with $x < y < z$, one of $r_{xy|z}, r_{xz|y}, r_{yz|x}$ is present. The machine needs to examine $O(q^3)$ triples, and each triple is checked in $O(q^3)$ time by scanning the tape. Now, $\mathcal{R} = \{xy|z : r_{xy|z}$ guessed $\}$ is a complete collection of triples without direct contradiction (i.e., such that for all elements $x, y, z \in L(\mathcal{R})$, not $xy|z$ and $xz|y$ are both in $\mathcal{R}$). Finally, M verifies that $\mathcal{R}$ satisfies property (P): there are $O(q^4)$ quadruples to examine, and each check takes time $O(q^3)$. Overall, the machine performs $q' = O(q^7)$ steps. ■

REFERENCES

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

[2] Xia, Y., Yang, Y.: Mining Closed and Maximal Frequent Subtrees from Databases of Labeled Rooted Trees. IEEE Transactions on Knowledge and Data Engineering **17**(2) (2005) 190–202

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

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

[5] Daubin, V., Gouy, M., Perrière, G.: A phylogenomic approach to bacterial phylogeny: evidence of a core of genes sharing a common history. Genome Research **12**(7) (2002) 1080–1090

[6] Jansson, J., Ng, J.H.K., Sadakane, K., Sung, W.K.: Rooted Maximum Agreement Supertrees. Algorithmica **43**(4) (2005) 293–307

[7] Berry, V., Nicolas, F.: Maximum Agreement and Compatible Supertrees. Journal of Discrete Algorithms **5**(3) (2007) 564–591

[8] Kao, M.Y.: Encyclopedia of Algorithms. http://refworks.springer.com/algorithms/ (2007)

[9] Lapointe, F.J., Rissler, L.J.: Congruence, consensus and the comparative phylogeography of codistributed species in California. Am. Nat. **166**(2) (2005) 290–299

[10] de Vienne, D.M., Giraud, T., Martin, O.C.: A congruency index for testing topological similarity between trees. Bioinformatics **23**(23) (2007) 3119–3124

[11] Jousselin, E., van Noort, S., Berry, V., Rasplus, J.Y., Ronsted, N., Erasmus, J., Greeff, J.: One fig to binf them all: host conservatism in a fig wasp community unraveled by cospeciation analyses among pollinating and nonpollinating fig wasps. Evolution (2008)

[12] Ge, F., Wang, L.S., Kim, J.: The cobweb of life revealed by genome-scale estimates of horizontal gene transfer. PLOS Biology **3**(10) (2005) 1709–1718

[13] Nakhleh, L., Ruths, D., Wang, L.S.: RIATA-HGT: a fast and accurate heuristic for reconstructing horizontal-gene transfer. In Wang, L., ed.: Computating and Combinatorics, 11th Ann. Int. Conf. (COCOON). Volume 3595 of LNCS., Springer (2005) 84–93

[14] Downey, R., Fellows, M.: Parameterized Complexity. Springer-Verlag (1999)

[15] Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer-Verlag (2006)

[16] Agarwala, R., Fernandez-Baca, D.: A polynomial time algorithm for the perfect phylogeny problem when the number of character states is fixed. SIAM Journal on Computing **23** (1994) 1216–1224

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

[18] Farach, M., Przytycka, T.M., Thorup, M.: On the agreement of many trees. Information Processing Letters **55**(6) (1995) 297–301

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

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

[21] Berry, V., Nicolas, F.: Improved parametrized complexity of the Maximum Agreement Subtree and Maximum Compatible Tree problems. IEEE/ACM Transactions on Computational Biology and Bioinformatics **3**(3) (2006) 289–302

[22] Guillemot, S., Nicolas, F.: Parameterized complexity of the MAST and MCT problems. Technical report, LIRMM, Univ. Montpellier 2 LIRMM, Univ. Montpellier 2 LIRMM, Univ. Montpellier 2, (extended version of the CPM'06 paper) (2007)

[23] Henzinger, M., King, V., Warnow, T.: Constructing a Tree from Homeomorphic Subtrees, with Applications to Computational Evolutionary Biology. Algorithmica **24**(1) (1999) 1–13

[24] Steel, M., Warnow, T.: Kaikoura tree theorems: computing the maximum agreement subtree. Information Processing Letters **48**(2) (1993) 77–82

[25] Hoang, V., Sung, W.K.: Fixed parameter polynomial time algorithms for maximum agreement and compatible supertrees. In: Proceedings of STACS'08. (2008) 361–372

[26] Bandelt, H., Dress, A.: Reconstructing the shape of a tree from observed dissimilarity data. Advances in Applied Mathematics **7** (1986) 309–343

[27] Fernau, H.: Parameterized Algorithmics: A Graph-Theoretic Approach. Habilitationsschrift, Universität Tübingen, Germany (2005)

[28] Gramm, J., Niedermeier, R.: A fixed-parameter algorithm for minimum quartet inconsistency. Journal of Computer and System Sciences **67**(4) (2003) 723–741

[29] Chen, Y., Flum, J., Grohe, M.: Bounded Nondeterminism and Alternation in Parameterized Complexity Theory. In: Proceedings of CCC'03. (2003) 13–29

[30] Guillemot, S.: Parameterized complexity and approximability of the SLCS problem. In Grohe, M., Niedermeier, R., eds.: Proceedings of IWPEC'08. Volume 5018 of LNCS., Springer (2008) 115–128

[31] Fellows, M., Hallett, M., Stege, U.: Analogs & duals of the MAST problem for sequences & trees. Journal of Algorithms **49**(1) (2003) 192–216

APPENDIX I
COMPLEMENTS OF PROOF OF THEOREM 1.

We show that the procedure ISCOMPATIBLE can be implemented as a $O(kn)$ time algorithm. Obviously, (i) testing if $\pi = \pi_\perp$ is done in $O(k)$ time, (ii) given T spanning tree of $G(\mathcal{T}, \pi)$, constructing C is done in $|T| = O(k)$ time, provided we have stored a label l_e for each edge of T, (iii) given V_1, V_2 partition of V, constructing the positions π_1, π_2 is done in $O(k)$ time. We now justify that in $O(kn)$ time we can perform a connexity test on $G(\mathcal{T}, \pi)$.

The crucial point is that the algorithm tests the connexity of the graph, by working on the intersection model of $G := G(\mathcal{T}, \pi)$ provided by the sets $\{L(x) : x \in V\}$. In this way, we avoid constructing the adjacency matrix of G, which would require $O(k^2 n)$ time. We thus need to describe a connexity test for a graph $G = (V, E)$ given by an intersection model $\{S_v : v \in V\}$, where the S_v are subsets of a base set S. We will justify that the algorithm has running time $O(kn)$, where $k = |V|$ and $n = |S|$.

The algorithm proceeds as follows. It performs a traversal of the graph, by starting at an arbitrary vertex $u \in V$, and maintains the following information during the traversal: (i) the set U of nodes already visited, (ii) a set F of edges forming a spanning tree of $G[U]$. At each step, the algorithm seeks a *transversal edge*, which is an edge $e = (u, v) \in E$ with $u \in U, v \in \overline{U}$. If such an edge is found, then v is added to U, and e is added to F. If no such edge exists, the algorithm stops, and the graph is connected if and only if $U = V$.

We show that using appropriate data structures, each step of the algorithm can be done in $O(n)$ time. For each $x \in S$, let $V_x = \{v \in V : x \in S_v\}$. We maintain for each $x \in S$, two lists representing the sets $U_x = V_x \cap U$ and $\overline{U}_x = V_x \cap \overline{U}$. Initializing these lists at the beginning of the algorithm is done in $O(kn)$ time. Moreover, at a given step of the algorithm: (i) we can find a tranversal edge in $O(n)$ time, (ii) we can update the structures in $O(n)$ time. To justify Point (i), observe that finding a transversal edge amounts to find an element $x \in S$ such that both U_x and $\overline{U_x}$ are non empty; if such an x is found then by choosing $u \in U_x, v \in \overline{U_x}$ we obtain a transversal edge (u, v); clearly, these operations can be performed in $O(n)$ time. To justify Point (ii), observe that when adding a new vertex v to U, we need, for each $x \in S_v$, to add v to U_x and to remove v from $\overline{U}_x$, which can be performed in $O(n)$ time by using appropriate linkage in the data structure storing elements in a set.

Pseudo-code of algorithm 3, called CONNECTIVITYTEST, describes this routine in detail, in the context where the vertives of V are subtrees corresponding to a position π in trees $\mathcal{T}$ and the sets $\{L(x) : x \in V\}$ are the leaves of these subtrees. Note that the forest F could be obtained by adding edge (u, v) to F after line 2, but the forest itself is not useful as for each of its edges, a label f shared by the extremities of that edge is already identified and put into the conflict set at line 1. For ease of implementation we also give in Algorithm 4 the variant ISCOMPATIBLE* of Algorithm 1 that resorts to CONNECTIVITYTEST for deciding if a position in a collection $\mathcal{T}$ is compatible.

APPENDIX II
COMPLEMENTS OF PROOF OF LEMMA 5.

A.

(i) $L(S_1) \cap L(S_2) = \emptyset$: indeed, if there was $x \in L(S_1) \cap L(S_2)$ then we would have $x \in L(T_i)$ for some $i \in [k]$; then

Algorithm 3: CONNECTIVITYTEST(π,V)

```
Input: A position $\pi$ in a collection $\mathcal{T}$ of trees and the set $V$
       of vertices of the graph $G(\mathcal{T},\pi)$.
Result: A tuple $(B,R)$ where
   • $B$ is a boolean indicating whether $G(\mathcal{T},\pi)$ is disconnected
   • $R$ a set $U \subseteq V$ of nodes forming a connected component when
     the graph is disconnected, otherwise $R$ is a conflict among $\pi$.

/* Note that the graph $G(\mathcal{T},\pi)$ is never explicitly built */
Choose an arbitrary vertex $u$ in $V$ and set $U \leftarrow \{u\}$
foreach $\ell \in L(\pi)$ do
    if $\ell \in L(u)$ then $U_\ell \leftarrow \{u\}$ else $U_\ell \leftarrow \emptyset$
    $\overline{U}_\ell \leftarrow \emptyset$
    foreach $v \in V$ with $v \neq u$ do
        if $\ell \in L(v)$ then $\overline{U}_\ell \leftarrow \overline{U}_\ell \cup \{v\}$
$C \leftarrow \emptyset$
while $U \neq V$ do
    /* Look for a transversal edge, i.e., $(u,v)$ such that $u \in U, v \notin U$ */
    Choose $f \in L(\pi)$ such that $U_f \neq \emptyset$ and $\overline{U}_f \neq \emptyset$
    if no such $f$ exists then
        return $(\texttt{true},U)$   /* $U$ is a connected component of $G(\mathcal{T},\pi)$ */
    else
1       $C \leftarrow C \cup \{f\}$
2       Choose $u \in U_f$ and $v \in \overline{U}_f$
        $U \leftarrow U \cup \{v\}$
        foreach $\ell \in L(\pi)$ do
            if $v \in \overline{U}_\ell$ then $\overline{U}_\ell \leftarrow \overline{U}_\ell - \{v\}$ ; $U_\ell \leftarrow U_\ell \cup \{v\}$
return $(\texttt{false},C)$   /* $C$ is a conflict */
```

$x = \phi_{j,i}(x) \leq_{T_i} \pi_j[i]$. Since $x \in L(T_i)$, we must have $\pi_1[i], \pi_2[i] \neq \perp$; then they are equal to distinct children of $\pi[i]$, impossible.

(ii) ϕ_i is a partial embedding of S into T_i:

- if $x \in L(S)$, then $x \in L(S_j)$. We conclude using the fact that $\phi_{j,i}$ is a partial embedding and that $\phi_i(x) = \phi_{j,i}(x)$.
- if x is an internal node of S with children x', x'', then:
  - if $x \in N(S_j)$, we conclude using the fact that $\phi_{j,i}$ is a partial embedding and that $\phi_i(x) = \phi_{j,i}(x)$.
  - if $x = r(S)$, with children $x' = r(S_1), x'' = r(S_2)$: then
    - either $\phi_{1,i}(x') = \phi_{2,i}(x'') = \perp$, in which case $\phi_i(x) = \perp$;
    - either $\phi_{1,i}(x') \neq \perp$, $\phi_{2,i}(x'') = \perp$, in which case $\phi_i(x) = \phi_{1,i}(x')$;
    - either $\phi_{1,i}(x') = \perp$, $\phi_{2,i}(x'') \neq \perp$, in which case $\phi_i(x) = \phi_{2,i}(x'')$;
    - either $\phi_{1,i}(x') \neq \perp$, $\phi_{2,i}(x'') \neq \perp$, in which case these are nodes y', y'' such that $y' \leq_{T_i} \pi_1[i]$, $y'' \leq_{T_i} \pi_2[i]$. Since $(\pi_1, \pi_2) \in D(\pi)$, it follows that $\pi_1[i], \pi_2[i]$ are $\neq \perp$ and are distinct children of $\pi[i]$, hence $\phi_i(x) = \pi[i]$, which implies that $\phi_{1,i}(x') <_{T_i} \phi_i(x)$, $\phi_{2,i}(x'') <_{T_i} \phi_i(x)$.

(iii) $\phi_i(r(S)) \leq_{T_i} \pi[i]$: follows from the definition of $\phi_i(r(S))$

Algorithm 4: ISCOMPATIBLE*(π)

```
Input: A position $\pi$ in a collection $\mathcal{T}$ of trees.
Result: A tuple $(B,C)$ where
   • $B$ is a boolean indicating whether $\pi$ is compatible
   • $C$ is a conflict among $\pi$ when the position $\pi$ is not compatible.

if $\pi = \pi_\perp$ then return $(\texttt{true},\emptyset)$
if $\pi$ is not reduced then $\pi \leftarrow \pi\downarrow$
$V \leftarrow \bigcup_i \texttt{children}_{T_i}(\pi[i])$
$(disconnected, R_{CT}) \leftarrow$ CONNECTIVITYTEST($\pi$,$V$)
if $disconnected$ is $\texttt{false}$ then
    return $(\texttt{false},R_{CT})$   /*$R_{CT}$ is a conflict among $\pi$*/
else
    /* $R_{CT}$ is a connected component of $G(\mathcal{T},\pi)$ */
    $V_1 \leftarrow R_{CT}$ ; $V_2 \leftarrow V - R_{CT}$
    $\pi_1 \leftarrow succ_{V_1}(\pi)$ ; $\pi_2 \leftarrow succ_{V_2}(\pi)$
    $(B_1,C_1) \leftarrow$ ISCOMPATIBLE*($\pi_1$) ;
    if $B_1$ is $\texttt{false}$ then return $(\texttt{false},C_1)$
    $(B_2,C_2) \leftarrow$ ISCOMPATIBLE*($\pi_2$) ;
    if $B_2$ is $\texttt{false}$ then return $(\texttt{false},C_2)$
    return $(\texttt{true},\emptyset)$
```

and from the fact that $\phi_i(r(S_j)) \leq_{T_i} \pi_j[i]$.

B.

We show that $(\pi_1, \pi_2) \in D(\pi)$. Indeed, (i) we have $\pi_j \neq \pi$ since if we had $\pi_1[i] = \pi[i]$ for each i, this would imply $\pi_2[i] = \perp$ for each i, but given $x \in L(S_2)$ there exists i such that $x \in L(T_i)$, impossible; (ii) fix $i \in [k]$:

- if $\pi[i] = \perp$, then $\phi_i(u) = \perp$, and we then have $\phi_i(v_1) = \phi_i(v_2) = \perp$ by definition of a partial embedding, hence $\pi_1[i] = \pi_2[i] = \perp$;
- if $\pi[i] \neq \perp$, then $\phi_i(u)$ is a node of T_i, and we have:
  - either $\phi_i(v_1), \phi_i(v_2) \neq \perp$, in which case the nodes $\mathsf{child}_{T_i}(\phi_i(v_1), \phi_i(u))$, $\mathsf{child}_{T_i}(\phi_i(v_2), \phi_i(u))$ are distinct, which implies that $\pi_1[i], \pi_2[i]$ are distinct children of $\pi[i]$;
  - or one of $\phi_i(v_1), \phi_i(v_2)$ is equal to $\perp$, in which case the other must be equal to $\phi_i(u)$, which implies that $\pi_1[i] = \pi[i]$, $\pi_2[i] = \perp$ or the symmetric case.

APPENDIX III
COMPLEMENTARY PROOFS FOR INTRACTABILITY RESULTS

Below, given two trees T, T', we use the notation $T \bowtie T'$ to denote that T and T' agree. We use a similar notation for sequences.

A. Complement for the proof of Lemma 6

Proof: The correctness of the reduction given in the main text follows by proving that: I is a positive instance of P-SLCS-D if and only if I' is a positive instance of P-COLORED-SLCS.

($\Rightarrow$): suppose that s is a compatible sequence for $\mathcal{C}$ with $|s| = q$. Then $s = z_1...z_q$. Let $s' = z_1^1...z_q^q$, we show that s' is a colored compatible sequence for $\mathcal{C}'$. Clearly s' is a colored sequence. To prove that s' is a compatible sequence for $\mathcal{C}'$, we need to show that:

- $s' \bowtie s'_p$: consider $x, y \in L(s') \cap L(s'_p)$ such that $x <_{s'} y$, then $x = z_i^i, y = z_j^j$ with $i < j$, and since $z_i <_s z_j$ and $s \bowtie s_p$ it follows that $z_i <_{s_p} z_j$, thus $z_i^i <_{s'_p} z_j^j$, and we obtain that $x <_{s'_p} y$.
- $s' \bowtie s''_p$: the reasoning is similar.

($\Leftarrow$): suppose that s' is a colored compatible sequence for $\mathcal{C}'$. Then $s' = y_1 ... y_q$ with $y_i \in L'^i$ for each i. Since $y_i \in L'^i$, there exists $z_i \in L$ such that $y_i = z_i^i$.

Note that the labels $z_1, ..., z_q$ are pairwise distinct: if $z_j, z_{j'}$ were equal (to a label x) with $j < j'$, then by considering a sequence s_i such that $x \in L(s_i)$, we would obtain $z_j^j <_{s'_i} z_{j'}^{j'}$ but $z_{j'}^{j'} <_{s''_i} z_j^j$, impossible.

Let us now define $s = z_1 ... z_q$, we show that s is a compatible sequence for $\mathcal{C}$. We need to show that $s \bowtie s_p$. Consider $x, y \in L(s) \cap L(s_p)$ such that $x <_s y$, then $x = z_i, y = z_j$ with $i < j$. Since $z_i^i <_{s'} z_j^j$ and since $s' \bowtie s'_p$, we obtain $z_i^i <_{s'_p} z_j^j$, and since z_i, z_j are distinct this implies $z_i <_{s_p} z_j$, and thus $x <_{s_p} y$. ■

B. Complement for the proof of Lemma 7

Proof: The correctness of the reduction given in the main text follows by proving that: I is a positive instance of P-COLORED-SLCS if and only if I' is a positive instance of P-SMAST-D.

($\Rightarrow$): suppose that s is a colored compatible sequence s for $\mathcal{C}$, with $|s| = q$. Then $s = y_1 ... y_q$, with $y_i \in L_i$. Let $T = rake(z_0, (z_1, y_1), ..., (z_q, y_q))$, then T is an agreement supertree for $\mathcal{T}$, with $|T| = q'$. Clearly, we have $T \bowtie S$ and $T \bowtie S'$, since $R_i | \{z_i, y_i\} = R'_i | \{z_i, y_i\} = (z_i, y_i)$ for each $i \in [q]$. Moreover, we have $T \bowtie T_i$ for each $i \in [k]$: indeed, if $s | L(s_i) = s_i | L(s) = y_{i_1} ... y_{i_m}$ with $i_1 < ... < i_m$, then $T | L(T_i) = T_i | L(T) = rake(z_0, y_{i_1}, ..., y_{i_m})$.

($\Leftarrow$): suppose that T is an agreement supertree for $\mathcal{T}$, with $|T| = q'$. First observe that $|L(T) \cap L'_i| \leq 2$ for each $i \in [q]$, since otherwise one of $T \bowtie R_i, T \bowtie R'_i$ would fail. Since $|T| = q'$, it follows that we have $|L(T) \cap L'_i| = 2$ for each $i \in [q]$, and thus $z_0 \in L(T)$. Now, for each $i \in [q]$ choose $y_i \in L(T) \cap L'_i$ distinct from z_i, and let $s = y_1 ... y_q$. Then s is a colored compatible sequence for $\mathcal{C}$. Indeed, consider $i \in [k]$, since $T \bowtie T_i$ we have $T | L(T_i) = T_i | L(T) = rake(z_0, y_{i_1}, ..., y_{i_m})$ with $i_1 < ... < i_m$, it follows that $s | L(s_i) = s_i | L(s) = y_{i_1} ... y_{i_m}$, hence $s \bowtie s_i$. ■

ACKNOWLEDGMENT

This paper was supported by the *Action incitative* BIOSTIC-LR, the *Action Jeunes Chercheurs* of University Montpellier II and the french ANR *PhylAriane*.