Fixed-Parameter Tractability of the Maximum Agreement Supertree Problem[⋆]

Research Report - LIRMM 07005

Sylvain Guillemot[1] and Vincent Berry[1]

Equipe *Méthodes et Algorithmes pour la Bioinformatique*, LIRMM, C.N.R.S.-Université Montpellier II.
{sguillem,vberry}@lirmm.fr

Abstract. Given a ground set L of labels and a collection of trees whose leaves are bijectively labelled by some elements of L, the Maximum Agreement Supertree problem (SMAST) is the following: find a tree T on a largest label set $L' \subseteq L$ that homeomorphically contains every input tree restricted to L'. The problem finds applications in phylogenetics, databases and data mining. In this paper we focus on the parameterized complexity of this NP-hard problem. We consider different combinations of parameters for SMAST as well as particular cases, providing both FPT algorithms and intractability results.

1 Introduction

Motivation. Supertree construction consists in building trees on a large set of labels from smaller trees covering parts of the label set. This task finds application in bioinformatics where trees represent phylogenies, but also in other fields [1, 2]. In phylogenetics, the labels are bijectively associated with the leaves of the trees and represent current organisms, while internal nodes represent hypothetical ancestors. The topological information in the input trees consists in the groupings of labels induced by internal nodes, representing related sets of organisms such as species, orders, families, etc. The goal is to build a supertree complying as much as possible with the topological information of the source trees. The task is relatively easy when the input trees 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, hence that incorporates all topological information provided by the data [1]. However, in practice several input trees usually disagree on the position of some leaves with respect to other leaves.

Related work. Some methods aim at producing supertrees incorporating as much input information as possible under the constraint that they do not contradict any input tree: they avoid disagreements between the input trees by collapsing some of their edges [3, 4] or by excluding some of their leaves, i.e. labels. The Maximum Agreement Supertree (SMAST) method [5–7] is apparented to the latter kind. Given a collection $\mathcal{T}$ of k

⋆ This paper was supported by the *Action incitative* BIOSTIC-LR.

trees of maximum degree d with labels taken in a ground set L of size n, 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 SMAST problem consists in finding an agreement supertree containing the maximum number of labels from L.

This problem is NP-hard in general as it generalizes the MAST problem [8]. SMAST remains NP-hard when d is unrestricted for $k \geq 3$ input trees [6] and for trees of degree $d \geq 2$ when k is unrestricted [5]. Moreover, [6, 5] have also considered the complement problem, which is a minimization problem where the measure is the number p of labels missing in an agreement supertree. This problem can not be approximated in polynomial time within a constant factor, unless P = NP [5]. The corresponding decision problem parameterized in p is W[2]-hard [5].

For the particular case of $d = 2$, [6] gave an $O(n^{3k^2})$ algorithm for SMAST. For $k = 2$ both [6, 5] shown that SMAST can be solved in polynomial time, by reduction to MAST.

Our results. In this paper, we focus on the particular case where $d = 2$. Note that in phylogenetics, the input trees of SMAST will often be binary as a result of the optimization algorithms used to analyze raw molecular data. We improve on previous results in several ways.

First, we show that SMAST on k rooted binary trees on a label set of size n can be solved in $O((2k)^p kn^2)$. This algorithm is only exponential in p, that roughly represents the extent to which the input trees disagree. Thus, the algorithm will be reasonably fast when dealing with collections of trees obtained for genes displaying a low level of homoplasy. Then, we provide an $O((8n)^k)$ algorithm, independent of p, and significantly improving on the $O(n^{3k^2})$ algorithm of [6]. This algorithm shows that SMAST is tractable for a small number of trees, extending in some sense the previously known results for $k = 2$ trees [6, 5, 9]. We also obtain some fixed-parameter intractability results for various combinations of parameters of SMAST.

We then 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 [9], 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 the input trees can be circumvented to small sets of labels, leading to $O(4^p n^3)$ and $O(3.12^p + n^4)$ algorithms.

2 Definitions

We consider rooted trees which are bijectively leaf-labelled. Let T be such a tree, we identify its leaf set with its label set, denoted by $L(T)$. The *size* of T is $|T| := |L(T)|$.

The node set of T is denoted by $N(T)$, and $r(T)$ stands for the root of T. We use a parenthesized notation for trees: if x is a label, then x denotes the tree whose root is a leaf labelled by x; if $T_1, ..., T_k$ are trees, then $(T_1, ..., T_k)$ stands for the tree whose root is connected to the child subtrees $T_1, ..., T_k$.

If x is a node of T, $T(x)$ stands for the subtree of T rooted at x, and $L(x)$ for the label set of this subtree. If x, y are two nodes of T, then $x <_T y$ means that x is a descendant of y in T. The upper bound of two nodes x, y of T w.r.t. $<_T$ is called the lowest common ancestor of x, y, and is denoted by $\mathsf{lca}_T(x, y)$. If x, y are two nodes of T s.t. $x <_T y$, denote by $\mathsf{child}_T(x, y)$ the child of y along the path joining y to x in T. If x is an internal node of T, the set of children of x in T is denoted by $\mathsf{children}_T(x)$.

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 $T \leq T'$, iff $T = T'|L(T)$. We say that T *partially embeds* in T', denoted by $T \bowtie T'$, iff $T|L(T') = T'|L(T)$. 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.

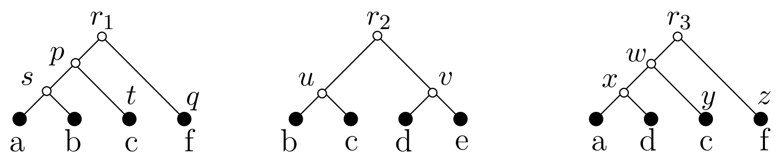

Fig. 1. A collection $\mathcal{T}$ of 3 input trees.

A *rooted triple* (or *triple* for short) is a binary tree T s.t. $|L(T)| = 3$; such a tree has the form $T = ((x, y), z)$, 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* iff each set of three labels in $L(\mathcal{R})$ is present in at least one t_i. To a binary tree T, we associate a complete collection of triples $rt(T)$ formed by the triples $t_i \leq T$; 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* iff there exists a tree T s.t. $\mathcal{R} = rt(T)$; then we say that $\mathcal{R}$ *displays* T.

An *agreement supertree* for $\mathcal{T}$ is a tree S s.t. $L(S) \subseteq L(\mathcal{T})$ and for each $i \in [k]$, $S \bowtie T_i$. We say that S is a *total agreement supertree* for $\mathcal{T}$ if additionnally $L(S) = L(\mathcal{T})$. The collection $\mathcal{T}$ is *compatible* iff there exists a total agreement supertree for $\mathcal{T}$. A *conflict* between $\mathcal{T}$ is a set $C \subseteq L(\mathcal{T})$ s.t. $\mathcal{T}|C$ is incompatible. For instance, $T = (((a, b), c), (e, f))$ is an agreement supertree for the collection $\mathcal{T}$ of Figure 1, and $C = \{a, b, c, d\}$ is a conflict between $\mathcal{T}$.

Given a collection $\mathcal{T}$, we define $SMAST(\mathcal{T})$ as the set of agreement supertrees for $\mathcal{T}$. The Maximum Agreement Supertree problem (Smast) asks: given a collection

$\mathcal{T}$, find an agreement supertree for $\mathcal{T}$ with the largest size. Equivalently, it amounts to seek a largest set $L \subseteq L(\mathcal{T})$ s.t. $\mathcal{T}|L$ is compatible. The size of such an optimal solution is denoted by $\#SMAST(\mathcal{T})$. 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 labels?

3 Solving Smast on binary trees

Throughout this section, we consider a fixed collection $\mathcal{T} = \{T_1, ..., T_k\}$ of binary trees, we let n denote the size of the label set and k the number of trees.

If T is a tree, we define $N^{\perp}(T) := N(T) \cup \{\perp\}$. We extend the notation $T(u)$ to $u \in N^{\perp}(T)$, s.t. if $u = \perp$ then $T(u)$ is the empty tree. We extend the relation $\leq_T$ to $N^{\perp}(T)$ s.t. $\perp \leq_T x$ for each $x \in N^{\perp}(T)$.

A *position* in $\mathcal{T}$ is a tuple $\pi = (u_1, ..., u_k)$, where each $u_i \in N^{\perp}(T_i)$. For $i \in [k]$, the ith component of π is denoted $\pi[i]$. We set $I(\pi) = \{i \in [k] : \pi[i]$ is an internal node of T_i $\}$. We define the *initial position* $\pi_{\top} = (r_1, ..., r_k)$, where each r_i is the root of T_i. We define the *final position* $\pi_{\perp} = (\perp, ..., \perp)$. We let $\Pi(\mathcal{T})$ denote the set of positions in $\mathcal{T}$.

3.1 Solving Smast in $O((2k)^p \times kn^2)$ time

In this section, we 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 well-known Build algorithm [1, 10] decides the compatibility of a collection but doesn't give a conflict in case of incompatibility. Like Build, the algorithm presented here builds the supertree using a recursive top-down approach. Each step constructs a graph where the connected components correspond to the subtrees hanging from the root of the supertree. However, we replace the graph used in Build with a graph that when connected yields a conflict of size $\leq 2k$, identified thanks to a spanning tree.

We begin with some additional definitions. A position is *reduced* iff 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. We set $\mathcal{T}(\pi) := \{T_1(u_1), ..., T_k(u_k)\}$. Given a position π in $\mathcal{T}$, we say that π is *compatible* iff $\mathcal{T}(\pi)$ is compatible.

Observe that:

Lemma 1. *Suppose that $\mathcal{T} = \{T_1, ..., T_k\}$ is compatible. Then:*

1. *for each $L \subseteq L(\mathcal{T})$, $\mathcal{T}|L$ is compatible.*
2. *for each position π in $\mathcal{T}$, $\mathcal{T}(\pi)$ is compatible.*

Proof. To prove Point 1, consider $L \subseteq L(\mathcal{T})$. Let S be a total agreement supertree for $\mathcal{T}$. Then for each $i \in [k]$ we have $T_i \leq S \Rightarrow T_i|L \leq S|L$. Hence $S|L$ is a total agreement supertree for $\mathcal{T}|L$.

To prove Point 2, consider a position π in $\mathcal{T}$. Remark that $\mathcal{T}|L(\pi)$ is compatible by Point 1. Let S be a total agreement supertree for $\mathcal{T}|L(\pi)$, we claim that S is a total agreement supertree for $\mathcal{T}(\pi)$. Indeed, we have $L(S) = L(\pi) = L(\mathcal{T}(\pi))$, and given $i \in [k]$ we have $T_i(u_i) \leq T_i|L(\pi) \leq S$. $\square$

In the following, we describe a recursive algorithm to decide the compatibility of π position in $\mathcal{T}$. The base case ($\pi = \pi_\perp$) is obvious, since $\pi_\perp$ is compatible. Moreover, observe that: π is compatible iff $\pi\downarrow$ is compatible. Thus, it is enough to consider reduced positions.

From now on, we assume that π is a reduced position in $\mathcal{T}$ different from $\pi_\perp$. We define the graph $G(\mathcal{T}, \pi)$ as follows: (i) its vertex set is $V = \cup_{i \in I(\pi)} \mathsf{children}_{T_i}(\pi[i])$; (ii) two vertices $x, y \in V$ are adjacent iff $L(x) \cap L(y) \neq \emptyset$. In other terms, $G(\mathcal{T}, \pi)$ is the intersection graph of the set system $\{L(x) : x \in V\}$. See Figure 2 for an example of such graphs.

If $V' \subseteq V$, we define the position $Succ_{V'}(\pi)$ as the position π' s.t.

- if $\pi[i] = \perp$, then $\pi'[i] = \perp$;
- if $\pi[i]$ is an internal node of T_i, with children v_i, v'_i, then one of the following holds: (i) $\pi'[i] = v_i$ if $v_i \in V', v'_i \notin V'$, (ii) $\pi'[i] = v'_i$ if $v_i \notin V', v'_i \in V'$, (iii) $\pi'[i] = u_i$ if $v_i \in V', v'_i \in V'$, (iv) $\pi'[i] = \perp$ if $v_i \notin V', v'_i \notin V'$.

We set $succ_{V'}(\pi) = Succ_{V'}(\pi)\downarrow$: this is the successor position of π induced by $V' \subseteq V$. Given $V' \subseteq V$, set $L(V') = \cup_{x \in V'} L(x)$. Set $L(\pi) = L(\mathcal{T}(\pi))$. Given $V_1, V_2 \subseteq V$, we say that V_1, V_2 are *connected* iff $G(\mathcal{T}, \pi)$ contains an edge $\{x, y\}$ with $x \in V_1, y \in V_2$; otherwise, we say that V_1, V_2 are *disconnected*.

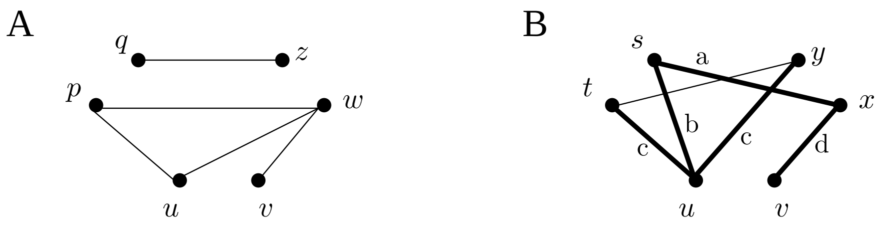

Fig. 2. A. The graph $G(\mathcal{T}, \pi_\top)$ of the position $\pi_\top = (r_1, r_2, r_3)$ for the collection of trees of Figure 1. This graph is disconnected, the two connected components indicate the two successor positions $\pi_1 = (p, r_2, w)$ and $\pi_2 = (q, \perp, z)$ of $\pi_\top$. B. The graph $G(\mathcal{T}, \pi_1)$ 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 $C = \{a, b, c, d\}$.

We will repeatedly use the following simple observations:

Lemma 2. *Let $V' \subseteq V$. Then: $L(Succ_{V'}(\pi)) \subseteq L(V')$.*

Lemma 3. *Two sets $V_1, V_2 \subseteq V$ are connected in $G(\mathcal{T}, \pi)$ iff $L(V_1) \cap L(V_2) \neq \emptyset$.*

The following lemma describes a recursive characterization of compatibility, relying on connectedness properties of the graph $G(\mathcal{T}, \pi)$:

Lemma 4. *Suppose that π is a reduced position $\neq \pi_\perp$. The following are equivalent:*

– *π is compatible;*
– *there exists a partition V_1, V_2 of V s.t. (i) V_1, V_2 are disconnected in $G(\mathcal{T}, \pi)$, (ii) $succ_{V_1}(\pi), succ_{V_2}(\pi)$ are compatible.*

Proof. ($\Rightarrow$). Suppose that π is compatible. Let S be a total agreement supertree for $\mathcal{T}(\pi)$. Since $|L(\pi))| \geq 2$, we have $S = (S_1, S_2)$. Since S is a total agreement supertree for $\mathcal{T}(\pi)$, we have $T_i(\pi[i]) \leq S$ for each $i \in [k]$. Define a partition V_1, V_2 of V as follows. Let $u_i = \pi[i]$, and suppose that u_i is an internal node of T_i, with children v_i, v'_i. Then $T_i(u_i) = (T_i(v_i), T_i(v'_i))$. Together with $T_i(u_i) \leq S$, this yields $T_i(v_i) \leq S_1$ or $T_i(v_i) \leq S_2$: add v_i to V_1 in the first case, to V_2 in the second case. Proceed similarly for v'_i.

We first prove Point (i). To see that V_1, V_2 are disconnected in $G(\mathcal{T}, \pi)$, observe that $L(V_1) \subseteq L(S_1)$ and $L(V_2) \subseteq L(S_2)$. Indeed, if $x \in V_j$ with $x \in \{v_i, v'_i\}$ then $T_i(x) \leq S_j$, hence $L(x) \subseteq L(S_j)$. It follows that V_1, V_2 are disconnected by Lemma 3. We now prove Point (ii). Let $\pi_j = succ_{V_j}(\pi)$, then π_j is a position in $\mathcal{T}(\pi)$, and since $\mathcal{T}(\pi)$ is compatible by assumption, it follows that π_j is compatible by Point 2 of Lemma 1.

($\Leftarrow$). Suppose that there exists a partition V_1, V_2 of V satisfying Points (i), (ii). Let $\pi_j = Succ_{V_j}(\pi)$. Since $succ_{V_j}(\pi)$ is compatible, it follows that π_j is compatible. Hence, there exists a total agreement supertree S_j for $\mathcal{T}(\pi_j)$, which thus satisfies: $T_i(\pi_j[i]) \leq S_j$ for each i. By Lemma 2, we then have $L(S_j) = L(\pi_j) \subseteq L(V_j)$. Since V_1, V_2 are disconnected in $G(\mathcal{T}, \pi)$, it follows that $L(V_1) \cap L(V_2) = \emptyset$ by Lemma 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 $\mathcal{T}(\pi)$: to this end, we need to show 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 holds obviously. 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 $\pi_1[i] = u_i$, therefore we have $T_i(u_i) \leq S_1$, and we conclude that $T_i(u_i) \leq S$. If $v_i, v'_i \in V_2$: then $\pi_2[i] = u_i$, therefore we have $T_i(u_i) \leq S_2$, and 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 $T_i(u_i) = (T_i(v_i), T_i(v'_i)) \leq (S_1, S_2) = S$. □

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

Lemma 5. *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 $l_e \in L(u) \cap L(v)$. Then $C = \{l_e : e \in F\}$ is a conflict between $\mathcal{T}$.*

Proof. We show that $\mathcal{T}' = \mathcal{T}|C$ is incompatible. For each $i \in I(\pi)$, 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 v_i, v_i', u_i there corresponds nodes $\tilde{v}_i, \tilde{v}_i', \tilde{u}_i$ in $T_i|C$. Define the position π' by setting $\pi'[i] = \perp$ if $i \notin I(\pi)$, $\pi'[i] = \tilde{u}_i$ if $i \in I(\pi)$. 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 of these edges is a spanning tree of $G(\mathcal{T}', \pi')$, hence the graph is connected. By Lemma 4, we conclude that π' is an incompatible position of $\mathcal{T}'$, therefore $\mathcal{T}'$ is incompatible (by Point 2 of Lemma 1). □

Lemmas 4 and 5 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, or returns a conflict of size $\leq 2k$.*

Proof. We define a procedure IsCompatible(π) which takes as input a reduced position, decides if π is compatible, or returns a conflict of size $\leq 2k$ in case of incompatibility. The procedure is as follows: (i) answer ("*yes*") if $\pi = \pi_\perp$; (ii) if $\pi \neq \pi_\perp$, test whether $G(\mathcal{T}, \pi)$ is connected.

– If the graph is connected, then let $T = (V, F)$ be a spanning tree of $G(\mathcal{T}, \pi)$, choose $l_e \in L(u) \cap L(v)$ for each edge $e = \{u, v\} \in F$, construct $C = \{l_e : e \in F\}$, and return ("*no*", C).
– If the graph is not connected, then let V_1, V_2 be a partition of V in two disconnected sets, and construct the positions π_1, π_2 where $\pi_i = succ_{V_i}(\pi)$. Call IsCompatible(π_1), let R_1 be its result; if $R_1 =$ ("*yes*") then call IsCompatible(π_2) and return its result R_2, else return R_1.

To decide if $\mathcal{T}$ is compatible, we simply call IsCompatible($\pi_\top \downarrow$).

We now justify the correctness and the running time of the algorithm. The correctness of the procedure IsCompatible follows from lemmas 4 and 5. 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 A for details). By lemmas 2 and 3, the total number of calls to IsCompatible is $O(n)$, therefore the total running time of the algorithm is $O(kn^2)$. □

The algorithm of Theorem 1 yields a simple FPT algorithm for P-Smast using the bounded search tree technique:

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

Proof. The algorithm constructs a search tree of height $\leq p$, where a node of the search tree at depth i is labelled by a set of labels $X \subseteq L$ s.t. $|X| = i$. 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". Otherwise, the algorithm proceeds as follows: if the node is at depth p, then it is labelled by "failure"; if it is at depth $< p$, then the procedure of Theorem 1 has returned a conflict C of size $\leq 2k$, and for each $x \in C$ a child node of u is added, with label $X \cup \{x\}$. The running time follows easily, since the search tree has height $\leq p$, degree $\leq 2k$, and since each node is processed in $O(kn^2)$ time. $\square$

3.2 Solving Smast in $O((8n)^k)$ time

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

We first give an alternative definition of the $\bowtie$ relation in terms of *partial embeddings*. Let T, T' be two trees, say that a partial embedding of T into T' is a function $\phi : N(T) \rightarrow N(T') \cup \{\perp\}$ such that:

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

Then: $T \bowtie T'$ iff there exists a partial embedding of T into T' (or equivalently, a partial embedding of T' into T).

Given a collection $\mathcal{T}$, the algorithm computes values $\#SMAST(\pi)$ for each position π. Let $SMAST(\pi)$ denote the set of trees T s.t. (i) T is an agreement supertree for $\mathcal{T}$, (ii) for each i, the partial embedding $\phi_i : T \rightarrow T_i$ is such that $\phi_i(r(T)) \leq_{T_i} \pi[i]$. We denote by $\#SMAST(\pi)$ the size of a largest tree of $SMAST(\pi)$.

We now define two values $\#SMAST_1(\pi)$ and $\#SMAST_2(\pi)$, from which $\#SMAST(\pi)$ is computed. We first define $\#SMAST_1(\pi)$. Say that a position π' is a *successor* of π iff there exists $i \in [k]$ s.t. $\pi'[i]$ 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:

$$\#SMAST_1(\pi) = \max_{\pi' \in S(\pi)} \#SMAST(\pi'). \tag{1}$$

We now define $\#SMAST_2(\pi)$. Say that a pair of positions (π_1, π_2) is a *decomposition* of π iff (i) $\pi_1 \neq \pi, \pi_2 \neq \pi$, (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 $\{\pi_1[i], \pi_2[i]\} = \{\perp, x\}$ or $\{\pi_1[i], \pi_2[i]\} = \{v, v'\}$.

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

$$\#SMAST_2(\pi) = \max_{(\pi_1,\pi_2)\in D(\pi)} (\#SMAST(\pi_1) + \#SMAST(\pi_2)). \tag{2}$$

We define the relation $\leq_{\mathcal{T}}$ on $\Pi(\mathcal{T})$ by: $\pi \leq_{\mathcal{T}} \pi'$ iff for each $i \in [k]$, $\pi[i] \leq_{T_i} \pi'[i]$. We observe that:

Lemma 6. *(i) If $\pi' \in S(\pi)$, then $\pi' <_{\mathcal{T}} \pi$, (ii) if $(\pi_1, \pi_2) \in D(\pi)$ then $\pi_i <_{\mathcal{T}} \pi$ for $i \in \{1, 2\}$.*

Lemma 7. *If $\pi' \leq_{\mathcal{T}} \pi$, then $SMAST(\pi') \subseteq SMAST(\pi)$.*

We now give a recurrence relation for computing $\#SMAST(\pi)$. Say that a position π is *terminal* if for each $i \in [k]$, $\pi[i]$ is a leaf or $\perp$.

We first consider terminal positions. Given $x \in L(\mathcal{T})$, let $Ind(x) = \{i \in [k] : x \in L(T_i)\}$. Given a terminal position π, and given $x \in L(\pi)$, define $Ind(x, \pi) = \{i \in [k] : \pi[i] = x\}$; observe that $Ind(x, \pi) \subseteq Ind(x)$. Say that an element $x \in L(\pi)$ is *nice* iff $Ind(x, \pi) = Ind(x)$, and let $Nice(\pi)$ denote the set of nice elements of $L(\pi)$. Then:

Lemma 8. *Suppose that π is terminal. Then: $\#SMAST(\pi) = |Nice(\pi)|$.*

Lemma 9. *Suppose that π is not terminal. Then:*
$\#SMAST(\pi) = \max(\#SMAST_1(\pi), \#SMAST_2(\pi))$.

Proof. We first prove that $\#SMAST_1(\pi) \leq \#SMAST(\pi)$. Let $S \in SMAST(\pi')$ for some $\pi' \in S(\pi)$, s.t. $|S|$ is maximal. Since $\pi' <_{\mathcal{T}} \pi$ by Lemma 6, we have $S \in SMAST(\pi)$ by Lemma 7, and the result follows.

We now prove that $\#SMAST_2(\pi) \leq \#SMAST(\pi)$. Let $(\pi_1, \pi_2) \in D(\pi)$, and let S_1, S_2 s.t. $S_j \in SMAST(\pi_i)$, $|S_j|$ maximal. If one of the S_j's is empty, say S_1, then $\#SMAST(\pi_1) = 0$, and we obtain $\#SMAST_2(\pi) = |S_2| = \#SMAST(\pi_2) \leq \#SMAST(\pi)$ by lemmas 6 and 7. 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 $\phi_{j,i} : S_j \to T_i$ s.t. $\phi_{j,i}(r(S_i)) \leq_{T_i} \pi_j[i]$ 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) $\phi_i(r(S)) \leq_{T_i} \pi[i]$ (proof in Appendix B.1). We conclude that $\#SMAST_2(\pi) = |S_1| + |S_2| = |S| \leq \#SMAST(\pi)$.

Finally, we show that $\#SMAST(\pi) \leq \max(\#SMAST_1(\pi), \#SMAST_2(\pi))$. Let $S \in SMAST(\pi)$ s.t. $|S|$ is maximal. Then there exists partial embeddings $\phi_i : S \to T_i$ s.t. $\phi_i(r(S)) \leq_{T_i} \pi[i]$ for each $i \in [k]$. Let $u_i = \phi_i(r(S))$ for each i. We consider two cases.

First case: there exists $i \in [k]$ s.t. $u_i <_{T_i} \pi[i]$. This case holds in particular if $|S| \leq 1$. 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 s.t. $\phi_i(r(S)) \leq_{T_j} \pi'[j]$ for each j. We conclude that $|S| = \#SMAST(\pi) \leq \#SMAST(\pi') \leq \#SMAST_1(\pi)$.

Second case: $u_i = \pi[i]$ for each $i \in [k]$. In this case, we have $|S| \geq 2$, hence $S = (S_1, S_2)$. Let u be the root of S, let v_i be the root of S_i 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 $(\pi_1, \pi_2) \in D(\pi)$ (proof in Appendix B.2). 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 $\phi_i(r(S_j)) \leq_{T_i} \pi_j[i]$ for each $i \in [k]$. We conclude that $|S| = \#SMAST(\pi) = |S_1| + |S_2| \leq \#SMAST(\pi_1) + \#SMAST(\pi_2) \leq \#SMAST_2(\pi)$. □

Lemmas 8 and 9 yield an algorithm for computing $\#SMAST(\mathcal{T})$:

Theorem 3. *$\#SMAST(\mathcal{T})$ can be computed in $O((8n)^k)$ time.*

Proof. Using dynamic programming, the algorithm computes the values $\#SMAST(\pi)$ for each position π, using the recurrence relations stated in lemmas 8 and 9. The correctness of the algorithm follows from the lemmas, and the termination of the algorithm is ensured by Lemma 6 and the fact that $<_{\mathcal{T}}$ is an order relation on $\Pi(\mathcal{T})$.

We now consider the space and time requirements for the algorithm. First observe that the number of positions π in $\mathcal{T}$ is $\leq (2n)^k$: indeed, a component $\pi[i]$ has $\leq 2n$ possible values (one of the $\leq 2n-1$ 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 $\#SMAST(\pi)$, assuming that the values $\#SMAST(\pi')$ for $\pi' <_{\mathcal{T}} \pi$ are available. Testing if π is terminal requires $O(k)$ time. If π is terminal, computing $|Nice(\pi)|$ takes $O(k)$ time. If π is nonterminal, then we need to compute $\#SMAST_1(\pi)$ and $\#SMAST_2(\pi)$, which respectively require $O(k)$ and $O(4^k)$ time. Thus, $\#SMAST(\pi)$ is computed in $O(4^k)$ time, hence the total running time of the algorithm is $O((8n)^k)$. □

3.3 Hardness results

The parameterized complexity of the SMAST problem on binary trees is considered w.r.t. 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. Our complexity results for several combinations of the parameters are summarized in Theorem 4:

Theorem 4. *We have the following hardness results for* Smast*:*

Parameters	*Complexity of* Smast
q	W[1]-*complete (even for $l = 3$)*
q, k	W[1]-*complete (even for $l = 3$)*
p	W[2]-*hard (even for $l = 3$)*
k, p	*FPT by a $O((2k)^p \times kn^2)$ time algorithm*
k	XNL-*hard, solvable in $O((8n)^k)$ time*

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:

- Clique: given a graph G and a parameter q, decide if G has a clique of size $\geq q$;
- Dominating Set: given a graph G and a parameter q, decide if G has a dominating set of size $\leq q$;
- 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, does M accept 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 [11, 12], note that the class we call XNL is indeed the class [Uniform-Xnl]^{FPT} of [11].

We now give some elements of proof for Theorem 4. The third hardness result was proven in [9]. The first, second and fifth result follow from similar results for the Slcs problem [13], which is defined as follows.

A *sequence* 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 appearing in s. We define the relation $<_s$ on $L(s)$ by: $u <_s v$ iff u precedes v 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 *embeds* in s', denoted $s \leq s'$, if $s = s'|L(s)$; we say that s *partially embeds* in s', denoted $s \bowtie s'$, if $s|L(s') = s'|L(s)$. A *compatible sequence* for a collection $\mathcal{C} = \{s_1, ..., s_k\}$ is a sequence s s.t. $L(s) \subseteq L(\mathcal{C})$ and for each $i \in [k]$, $s \bowtie s_i$.

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 $\#SMAST(\mathcal{T}) \geq q$. We denote by Slcs-D$[k, q]$ (resp. Smast-D$[k, q]$) the problem Slcs-D (resp. Smast-D) parameterized by k, q.

We rely on a parameter-preserving reduction from Slcs-D to Smast-D (see Appendix C for a proof):

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

We obtain:

Proposition 2. *The following results hold for* SMAST*:*

– $\mathsf{W}[1]$*-completeness for* q *and* q, k*;*
– XNL*-hardness for* k.

Proof. The hardness results follow from similar results for SLCS, and from the parameter-preserving reduction given by Proposition 1.

In addition, we can prove that SMAST parameterized in q is in $\mathsf{W}[1]$, see Appendix D for details. □

4 Solving SMAST on complete collection of triples

Let P-SMAST-CR denote the restriction of P-SMAST to complete collections of triples. We can show that non-treelike collections have conflicts of size ≤ 4, a result similar to that known on quartets [14]. This allows to solve P-SMAST-CR in $O(n^4 + 3.12^p)$ time by reduction to 4-HITTING SET [15]. and also in $O(4^p n^4)$ time by bounded search (see, e.g. [16]). 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 3 and Theorem 5).

Proposition 3. *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 *s.t.* $\mathcal{R}|X$ *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 *between* $\mathcal{R}'$ *with* $|C| \leq 4$ *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. Suppose now that no such conflict is found.

Let u be an internal node of T, and let v, v' be the two children of u. An *u-fork* is a pair $\{l, l'\}$ where $l \in L(v), l' \in L(v')$. Each u-fork $\{l, l'\}$ will propose a status $s_{l,l'}$ for the positioning of x w.r.t. u in T, where $s_{l,l'}$ is computed from $\mathcal{R}$ as follows: $s_{l,l'} = L$ if $lx|l' \in \mathcal{R}$, $s_{l,l'} = R$ if $l'x|l \in \mathcal{R}$, $s_{l,l'} = U$ if $ll'|x \in \mathcal{R}$ (L, R, U respectively stand for left, right and up).

The second step of the algorithm consists in successively considering each internal node u. For a given node u, it checks that the different u-forks propose the same status. This verification is performed as follows: (i) for each u-fork $\{l, l'\}$, determine the status

$s_{l,l'}$; (ii) if there exists l, l'_1, l'_2 s.t. $s_{l,l'_1} \neq s_{l,l'_2}$, then $C = \{x, l, l'_1, l'_2\}$ is a conflict; (iii) if there exists l_1, l_2, l' s.t. $s_{l_1,l'} \neq s_{l_2,l'}$, then $C = \{x, l_1, l_2, l'\}$ is a conflict; (iv) else, all values $s_{l,l'}$ are identical, let s_u denote this value.

In a third step, the algorithm checks that the different statuses are compatible. They are compatible iff for each edge u, v of T with u above v, we have: (i) if v is the left child of u, then $s_u = R \Rightarrow s_v = U$, (ii) if v is the right child of u, then $s_u = L \Rightarrow s_v = U$, (iii) if v is a child of u, then $s_u = U \Rightarrow s_v = U$. If one pair of nodes u, v does not meet the above requirements, then by considering $\{l, l'\}$ v-fork and $\{l, l''\}$ u-fork, we obtain a conflict $C = \{x, l, l', l''\}$. Otherwise, consider the sets of nodes u s.t. $s_u \neq U$, 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 5. *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 between $\mathcal{R}$ 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 6. *The* P-SMAST-CR *problem can be solved in $O(4^p n^3)$ time.*

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

4. Ranwez, V., Berry, V., Criscuolo, A., Guillemot, S., Douzery, E.: Vote or veto: desirable properties for supertree methods. submitted to Syst. Biol., LIRMM (2007)
5. Berry, V., Nicolas, F.: Maximum Agreement and Compatible Supertrees. In Sahinalp, S.C., Muthukrishnan, S., Dogrusoz, U., eds.: Proceedings of CPM. Volume 3109 of LNCS. (2004) 205–219
6. Jansson, J., Ng, J.H.K., Sadakane, K., Sung, W.K.: Rooted Maximum Agreement Supertrees. Algorithmica **4**(43) (2005) 293–307
7. Kao, M.Y.: Encyclopedia of algorithms. http://refworks.springer.com/algorithms/ (2007)
8. Steel, M., Warnow, T.: Kaikoura tree theorems: computing the maximum agreement subtree. Information Processing Letters **48**(2) (1993) 77–82
9. Berry, V., Nicolas, F.: Maximum Agreement and Compatible Supertrees. Journal of Discrete Algorithms (in press) (2007)
10. Henzinger, M., King, V., Warnow, T.: Constructing a Tree from Homeomorphic Subtrees, with Applications to Computational Evolutionary Biology. Algorithmica **24**(1) (1999) 1–13
11. Chen, Y., Flum, J., Grohe, M.: Bounded nondeterminism and alternation in parameterized complexity theory. In: Proceedings of 18th IEEE Annual Conference on Computational Complexity. (2003) (13–29)
12. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer-Verlag (2006)
13. Guillemot, S.: Parameterized complexity and approximability of the SLCS problem. Technical report, LIRMM (2007) In preparation.
14. Bandelt, H., Dress, A.: Reconstructing the shape of a tree from observed dissimilarity data. Advances in Applied Mathematics **7** (1986) 309–343
15. Fernau, H.: Parameterized algorithmics: A graph-theoretic approach. Habilitationsschrift, Universität Tübingen, Germany (2005)
16. Gramm, J., Niedermeier, R.: A fixed-parameter algorithm for minimum quartet inconsistency. Journal of Computer and System Sciences **67**(4) (2003) 723–741

5 Appendix

5.1 Appendix A. 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 *transverse edge*, which is an edge $e = \{u, v\} \in E$ with $u \in U, v \in \bar{U}$. If such an edge is found, then y is added to U, and e is added to F. If no such edge exists, the algorithm stops, and the graph is connected iff $U = V$.

We show that using appropriate data structures, a 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 $\bar{U}_x = V_x \cap \bar{U}$. 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 $x \in S$ s.t. $U_x, \bar{U_x}$ are non empty; if such an x is found then by choosing $u \in U_x, v \in \bar{U_x}$ we obtain a transverse edge $\{u, v\}$; clearly, these operations can be performed in $O(n)$ time. To justify Point (ii), observe that when visiting a new vertex v, we need, for each $x \in S_v$, to add v to U_x and to remove v from $\bar{U}_x$, which can be performed in $O(n)$ time by using appropriate linkage. □

5.2 Appendix B. Complements of proof of Lemma 9.

Appendix B.1. (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 $x = \phi_{j,i}(x) \leq_{T_i} \pi_j[i]$. 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'' s.t. $y' \leq_{T_i} \pi_1[i]$, $y'' \leq_{T_i} \pi_2[i]$. 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 $\phi_i(r(S))$ and from the fact that $\phi_i(r(S_j)) \leq_{T_i} \pi_j[i]$.

Appendix B.2. 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 s.t. $x \in L(T_i)$, 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.

□

5.3 Appendix C. Proof of Proposition 1

For the sake of clarity, we choose to perform the reduction in two steps.

First step: we give a parameter-preserving reduction from Slcs-D to a variant called 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 Colored-Slcs asks: does $\mathcal{C}$ have a colored compatible sequence?

We show:

Lemma 10. *There is a polynomial-time reduction from* Slcs-D$[k, q]$ *to* Colored-Slcs$[k, q]$ *which maps an instance* $(\mathcal{C}, k, q)$ *of* Slcs-D$[k, q]$ *to an instance* $(\mathcal{C}', 2k, q)$ *of* Colored-Slcs$[k, q]$.

Proof. Given an instance $I = (\mathcal{C}, k, q)$ of Slcs-D$[k, q]$, we construct an instance $I' = (\mathcal{C}', 2k, q)$ of Colored-Slcs$[k, q]$ 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. The correctness of the reduction follows by proving that: I is a positive instance of Slcs-D$[k, q]$ iff I' is a positive instance of Colored-Slcs$[2k, q]$.

($\Rightarrow$): suppose that s is a compatible sequence for $\mathcal{C}$ with $|s| = q$. Then $s = a_1...a_q$. Let $s' = a_1^1...a_q^q$, 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 $x, y \in L(s') \cap L(s'_p)$ s.t. $x <_{s'} y$, then $x = a^i_i, y = a^j_j$ with $i < j$, and since $a_i <_s a_j$ and $s \bowtie s_p$ it follows that $a_i <_{s_p} a_j$, thus $a^i_i <_{s'_p} a^j_j$, 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 $s' = x_1...x_q$ with $x_i \in L'^i$ for each i. Since $x_i \in L'^i$, there exists $a_i \in L$ s.t. $x_i = a^i_i$.

Note that the labels $a_1, ..., a_q$ are pairwise distinct: if $a_j, a_{j'}$ were equal (to a label x) with $j < j'$, then by considering a sequence s_i s.t. $x \in L(s_i)$, we would obtain $a^j_j <_{s'_i} a^{j'}_{j'}$ but $a^{j'}_{j'} <_{s''_i} a^j_j$, impossible.

Let us now define $s = a_1...a_q$, we show that s is a compatible sequence for $\mathcal{C}$. We need to show that $s \bowtie s_p$. Consider $x, y \in L(s) \cap L(s_p)$ s.t. $x <_s y$, then $x = a_i, y = a_j$ with $i < j$. Since $a^i_i <_{s'} a^j_j$ and since $s' \bowtie s'_p$, we obtain $a^i_i <_{s'_p} a^j_j$, and since a_i, a_j are distinct this implies $a_i <_{s_p} a_j$, and thus $x <_{s_p} y$. □

Second step: we give a parameter-preserving reduction from Colored-Slcs to Smast-D. If $T_1, ..., T_m$ are trees, the notation $rake(T_1, ..., T_m)$ is defined by:

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

We show:

Lemma 11. *There is a polynomial-time reduction from* Colored-Slcs$[k, q]$ *to* Smast-D$[k, q]$ *which maps an instance* $(\mathcal{C}, k, q)$ *of* Colored-Slcs$[k, q]$ *to an instance* $(\mathcal{T}, k+2, 2q+1)$ *of* Smast-D$[k, q]$.

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

– we first define the label set L': we create new labels $l_0, l_1, ..., l_q$. For each $i \in [q]$, we set $L'_i = L_i \cup \{l_i\}$, and we define $L' = \{l_0\} \cup L'_1 \cup ... \cup L'_q$.
– 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 $S = rake(l_0, R_1, ..., R_q)$ and $S' = rake(l_0, R'_1, ..., R'_q)$. For each sequence $s_i = a_1...a_m$ in $\mathcal{C}$, we create a tree $T_i = rake(l_0, a_1, ..., a_m)$.
– we set $q' = 2q + 1$ and $k' = k + 2$.

Note that $\mathcal{T}$ contains $k + 2$ trees. The correctness of the reduction follows by proving that: I is a positive instance of Colored-Slcs$[k, q]$ iff I' is a positive instance of Smast-D$[k', q']$.

($\Rightarrow$): suppose that s is a colored compatible sequence s for $\mathcal{C}$, with $|s| = q$. Then $s = a_1 ... a_q$, with $a_i \in L_i$. Let $T = rake(l_0, (l_1, a_1), ..., (l_q, a_q))$, then T is an agreement supertree for $\mathcal{T}$, with $|T| = q'$. Clearly, we have $T \bowtie S$ and $T \bowtie S'$, since $R_i|\{l_i, a_i\} = R'_i|\{l_i, a_i\} = (l_i, a_i)$ for each $i \in [q]$. Moreover, we have $T \bowtie T_i$ for each $i \in [k]$: indeed, if $s|L(s_i) = s_i|L(s) = a_{i_1} ... a_{i_m}$ with $i_1 < ... < i_m$, then $T|L(T_i) = T_i|L(T) = rake(l_0, a_{i_1}, ..., a_{i_m})$.

($\Leftarrow$): 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 $l_0 \in L(T)$. Now, for each $i \in [q]$ choose $a_i \in L(T) \cap L'_i$ distinct from l_i, and let $s = a_1 ... a_q$. Then s is a colored compatible sequence for $\mathcal{C}$. Indeed, consider $i \in [k]$, since $T \bowtie T_i$ we have $T|L(T_i) = T_i|L(T) = rake(l_0, a_{i_1}, ..., a_{i_m})$ with $i_1 < ... < i_m$, it follows that $s|L(s_i) = s_i|L(s) = a_{i_1} ... a_{i_m}$, hence $s \bowtie s_i$. $\square$

The proof of Proposition 1 follows from Lemmas 10 and 11.

5.4 Appendix D. Complements of proof of Proposition 2

To complete the proof of Proposition 2, we now show membership in W[1] for Smast parameterized by q (Lemma 14). We rely on the following lemmas.

Lemma 12. *Let T be a tree. The following are equivalent:*

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

Let $\mathcal{R}$ be a complete collection of triples. A *direct contradiction* in $\mathcal{R}$ is a set $a, b, c \in L(\mathcal{R})$ s.t. $ab|c \in \mathcal{R}, ac|b \in \mathcal{R}$.

Lemma 13. *The following are equivalent:*

- *$\mathcal{R}$ is treelike;*
- *$\mathcal{R}$ does not contain direct contradictions, and the following property holds: (P) for each $a, b, c, d \in L(\mathcal{R})$, $ab|c \in \mathcal{R} \wedge bc|d \in \mathcal{R} \Rightarrow ab|d \in \mathcal{R} \wedge ac|d \in \mathcal{R}$.*

We are now ready to show:

Lemma 14. Smast *parameterized in q is in* W[1].

Proof. We use a parameterized reduction to Short Turing Machine Computation. 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 iff $\mathcal{T}$ 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 Lemma 12.

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 s.t. $x_1 < ... < x_q$, 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, $\mathcal{R} = \{xy|z : r_{xy|z} \text{ guessed }\}$ is a complete collection of triples without direct contradiction. 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. □