Efficient FPT Algorithms for (Strict) Compatibility of Unrooted Phylogenetic Trees

Julien Baste, Christophe Paul, Ignasi Sau Valls, Celine Scornavacca

Efficient FPT algorithms for (strict) compatibility of unrooted phylogenetic trees

Julien Baste[1], Christophe Paul[1], Ignasi Sau[1], and Celine Scornavacca[2]

[1] CNRS, LIRMM, Université de Montpellier, Montpellier, France.
{baste, paul, sau}@lirmm.fr
[2] Institut des Sciences de l'Evolution (Université de Montpellier, CNRS, IRD, EPHE), Montpellier, France.
celine.scornavacca@umontpellier.fr

Abstract. In phylogenetics, a central problem is to infer the evolutionary relationships between a set of species X; these relationships are often depicted via a phylogenetic tree – a tree having its leaves univocally labeled by elements of X and without degree-2 nodes – called the "species tree". One common approach for reconstructing a species tree consists in first constructing several phylogenetic trees from primary data (e.g. DNA sequences originating from some species in X), and then constructing a single phylogenetic tree maximizing the "concordance" with the input trees. The so-obtained tree is our estimation of the species tree and, when the input trees are defined on overlapping – but not identical – sets of labels, is called "supertree". In this paper, we focus on two problems that are central when combining phylogenetic trees into a supertree: the compatibility and the strict compatibility problems for unrooted phylogenetic trees. These problems are strongly related, respectively, to the notions of "containing as a minor" and "containing as a topological minor" in the graph community. Both problems are known to be fixed-parameter tractable in the number of input trees k, by using their expressibility in Monadic Second Order Logic and a reduction to graphs of bounded treewidth. Motivated by the fact that the dependency on k of these algorithms is prohibitively large, we give the first explicit dynamic programming algorithms for solving these problems, both running in time $2^{O(k^2)} \cdot n$, where n is the total size of the input.

Keywords: Phylogenetics; compatibility; unrooted phylogenetic trees; parameterized complexity; FPT algorithm; dynamic programming.

1 Introduction

A central goal in *phylogenetics* is to clarify the relationships of extant species in an evolutionary context. Evolutionary relationships are commonly represented via *phylogenetic trees*, that is, acyclic connected graphs where leaves are univocally labeled by a label set X, and without degree-2 nodes. When a phylogenetic tree is defined on a label set X designating a set of genes issued from a gene family, we refer to it as a *gene tree*, while, when X corresponds to a set of extant species, we refer to it as a *species tree*. A gene tree can differ from the species

tree depicting the evolution of the species containing the gene for a number of reasons [15]. Thus, a common way to estimate a species tree for a set of species X is to choose *several* gene families that appear in the genome of the species in X, reconstruct a gene tree per each gene family (see [10] for a detailed review of how to infer phylogenetic trees), and finally combine the trees in a unique tree that maximizes the "concordance" with the given gene trees. The rationale underlying this approach is the confidence that, using several genes, the species signal will prevail and emerge from the conflicting gene trees. If the gene trees are all defined on the same label set, we are in the *consensus* setting; otherwise the trees are defined on overlapping – but not identical – sets of labels, and we are in the *supertree* setting. Several consensus and supertree methods exist in the literature (see [2,3,17] for a review), and they differ in the way the concordance is defined.

In this paper, we focus on a problem that arises in the supertree setting: given a set of gene trees $\mathcal{T} = \{T_1, \ldots, T_k\}$ on label sets $\{X_1, \ldots, X_k\}$, respectively, does there exist a species tree on $X := \cup_{i=1}^{k} X_i$ that *displays* all the trees in $\mathcal{T}$? This is the so-called Compatibility of Unrooted Phylogenetic Trees problem. The notion of "displaying" used by the phylogenetic community, which will be formally defined in Section 2, coincides with that of "containing as a minor" in the graph community. Another related problem is the Strict Compatibility (or Agreement) of Unrooted Phylogenetic Trees problem, where the notion of "displaying" is replaced by that of "strictly displaying". This notion, again defined formally in Section 2, coincides with that of "containing as a topological minor" in the graph community.

Both problems are polynomial-time solvable when the given gene trees are out-branching (or *rooted* in the phylogenetic literature), or all contain some common label [1,16]. In the general case, both problems are NP-complete [19] and fixed-parameter tractable in the number of trees k [5, 18]. The fixed-parameter tractability of these problems has been established via Monadic Second Order Logic (MSOL) together with a reduction to graphs of bounded treewidth. For both problems, it can be checked that the corresponding MSOL formulas [5,18] contain 4 alternate quantifiers, implying by [11] that the dependency on k in the derived algorithms is given by a tower of exponentials of height 4; clearly, this is prohibitively large for practical applications. Therefore, even if the notion of compatibility has been defined quite some time ago [12], at the moment no "reasonable" FPT algorithms exist for these problems, that is, algorithms with running time $f(k) \cdot p(|X|)$, with f a moderately growing function and p a low-degree polynomial. In this paper we fill this lack and we prove the following two theorems.

Theorem 1. *The* Compatibility of Unrooted Phylogenetic Trees *problem can be solved in time* $2^{O(k^2)} \cdot n$*, where* k *is the number of trees and* n *is the total size of the input.*

Theorem 2. *The* Agreement of Unrooted Phylogenetic Trees *problem can be solved in time* $2^{O(k^2)} \cdot n$*, where* k *is the number of trees and* n *is the total size of the input.*

Our approach for proving the two above theorems is to present explicit dynamic programming algorithms on graphs of bounded treewidth. As one could suspect from the fact that the corresponding MSOL formulas are quite involved [5, 18], it turns out that our dynamic programming algorithms are quite involved as well, implying that we are required to use a technical data structure.

This paper is organized as follows. In Section 2 we provide some preliminaries and we define the problems under study. In Section 3 we present our algorithm for the Compatibility of Unrooted Phylogenetic Trees problem. Due to space limitations, its proof of correctness and the analysis of its running time are given in Appendix A and Appendix B, respectively. The algorithm for the Agreement of Unrooted Phylogenetic Trees problem is entirely deferred to Appendix C. Finally, we provide some directions for further research in Section 4.

2 Preliminaries

Basic definitions. Given a positive integer k, we denote by $[k]$ the set of all integers between 1 and k. If S is a set, we denote by 2^S the set of all subsets of S. A *tree* T is an acyclic connected graph. We denote by $V(T)$ its vertex set, by $E(T)$ its edge set, and by $L(T)$ its set of vertices of degree one, called *leaves*. Two trees T and T' are *isomorphic* if there is a bijective function $\alpha : V(T) \cup E(T) \to V(T') \cup E(T')$ such that for every edge $e = \{u, v\} \in E(T)$, $\alpha(e) = \{\alpha(u), \alpha(v)\}$. If T is a tree and S is a subset of $V(T)$, we denote by $T[S]$ the subgraph of T induced by S. *Suppressing* a degree-2 vertex v in a graph G consists in deleting v and adding an edge between the former neighbors of v, if they are not already adjacent. *Identifying* two vertices v and v' of a graph G consists in creating a graph H by removing v and v' and adding a new vertex w such that, for each $u \in V(G) \setminus \{v, v'\}$, there is an edge $\{u, w\}$ in $E(H)$ if and only if $\{u, v\} \in E(G)$ or $\{u, v'\} \in E(G)$. *Contracting* an edge $e = \{u, v\}$ in G consists in identifying u and v. A graph H is a *minor* (resp. *topological minor*) of a graph G if H can be obtained from a subgraph of G by contracting edges (resp. contracting edges with at least one vertex of degree 2). See [9] for more details about the notions of minor and topological minor. If Y is a subset of vertices of a tree T, then $T|_Y$ is the tree obtained from the minimal subtree of T containing Y by suppressing degree-2 vertices. For simplicity, we may sometimes consider the vertices of $T|_Y$ also as vertices of T.

As already mentioned in the introduction, an *unrooted phylogenetic tree* on a label set X is defined as a pair (T, ϕ) with T a tree with no degree-2 vertex along with a bijective function $\phi : L(T) \to X$. We say that a vertex $v \in L(T)$ is *labeled* with label $\phi(v)$. Two unrooted phylogenetic trees (T, ϕ) and (T', ϕ') are *isomorphic* if there exists an isomorphism α from T to T' satisfying that if $v \in L(T)$ then $\phi'(\alpha(v)) = \phi(v)$.

The three graph operations defined above, namely suppressing a vertex, identifying two vertices, and contracting an edge, can be naturally generalized to unrooted phylogenetic trees. In this context, two vertices to be identified are either both unlabeled or both with the same label. In the latter case, the newly

created vertex inherits the label of the identified vertices. Finally, contractions in unrooted phylogenetic trees are restricted to edges incident to two unlabeled vertices. In this case, we speak about *upt-contraction*. If (T, ϕ) is an unrooted phylogenetic tree and Y is subset of leaves of $L(T)$, then $(T, \phi)|_Y$ is the unrooted phylogenetic tree $(T|_Y, \phi|_Y)$ where $\phi|_Y$ is the restriction of ϕ to the label set Y.

(Strictly) Compatible supertree. Let $\mathcal{T} = \{(T_1, \phi_1), (T_2, \phi_2), \ldots, (T_k, \phi_k)\}$ be a collection of unrooted phylogenetic trees, not necessarily on the same label set. We say that an unrooted phylogenetic tree (T, ϕ) is a *compatible supertree* of $\mathcal{T}$ if for every $i \in [k]$, $(T_i, \phi_i) \in \mathcal{T}$ can be obtained from $(T, \phi)|_{L(T_i)}$ by performing upt-contractions. The phylogenetic tree (T, ϕ) is a *strictly compatible supertree* of $\mathcal{T}$ if for every $i \in [k]$, $(T_i, \phi_i) \in \mathcal{T}$ is isomorphic to $(T, \phi)|_{L(T_i)}$. If a collection $\mathcal{T}$ of unrooted phylogenetic trees admits a (strictly) compatible supertree, then we say that $\mathcal{T}$ is *(strictly) compatible*. The two definitions are equivalent when $\mathcal{T}$ contains only binary phylogenetic trees, that is, unrooted trees in which every vertex that is not a leaf has degree 3. Note that, as mentioned in the introduction, the notions of "being a compatible supertree" and "being a strictly compatible supertree" correspond, modulo the conditions on the labels, to the notions of "containing as a minor" and "containing as a topological minor", respectively.

In this paper we consider the following problem:

Compatibility of Unrooted Phylogenetic Trees
Instance: A set $\mathcal{T}$ of k unrooted phylogenetic trees.
Parameter: k.
Question: Does there exist an unrooted phylogenetic tree (T, ϕ) that is a compatible supertree of $\mathcal{T}$?

The Agreement (or Strict Compatibility) of Unrooted Phylogenetic Trees problem is defined analogously, just by replacing "compatible supertree" with "strictly compatible supertree". For notational simplicity, we may henceforth drop the function ϕ from an unrooted phylogenetic tree (T, ϕ), and just assume that each leaf of T comes equipped with a label.

Assume that $\widehat{T}$ is a compatible supertree of $\mathcal{T}$. Then, according to the definition of minor, for every $i \in [k]$, every vertex $v \in V(T_i)$ can be mapped to a subtree of $\widehat{T}$, in such a way that the subtrees corresponding to the vertices of the same tree are pairwise disjoint. We call the set of vertices of that subtree the *vertex-model* of v. Observe that by the definition of the upt-contraction operation, the vertex-model of a leaf is a singleton. Hereafter, we denote by $\widehat{\varphi}(v)$ the subset of vertices belonging to the vertex-model of v. Moreover, if $u, v \in V(T_i)$ are two adjacent vertices in T_i, then there is *exactly one* edge in $\widehat{T}$ that connects the vertex-model of u to the vertex-model of v. We call such an edge of $\widehat{T}$ the *edge-model* of $\{u, v\} \in E(T_i)$. Observe that a vertex of $\widehat{T}$ may belong to several vertex-models, but then these vertex-models correspond to vertices from different trees of $\mathcal{T}$. Also, an edge of $\widehat{T}$ may be the edge-model of edges of different trees of $\mathcal{T}$.

Similarly, if $\widehat{T}$ is a strictly compatible supertree of $\mathcal{T}$, then according to the definition of topological minor, for every $i \in [k]$, every vertex $v \in V(T_i)$

can be mapped to a vertex of $\widehat{T}$, called the *vertex-model* of v, in such a way that this mapping is injective when restricted to every $i \in [k]$. In this case, if $u, v \in V(T_i)$ are two adjacent vertices in T_i, then there is exactly one *path* in $\widehat{T}$ that connects the vertex-model of u to the vertex-model of v called the *edge-model* of $\{u, v\} \in E(T_i)$. Similarly to the vertex-models, the edge-models of the same tree need to be pairwise disjoint, except possibly for their endvertices.

Treewidth. A *tree-decomposition* of width w of a graph $G = (V, E)$ is a pair $(\mathsf{T}, \mathcal{B})$, where T is a tree and $\mathcal{B} = \{B_t \mid B_t \subseteq V, t \in V(\mathsf{T})\}$ such that

- $\bigcup_{t \in V(\mathsf{T})} B_t = V$,
- for every edge $\{u, v\} \in E$ there is a $t \in V(\mathsf{T})$ such that $\{u, v\} \subseteq B_t$,
- $B_i \cap B_k \subseteq B_j$ for all $\{i, j, k\} \subseteq V(\mathsf{T})$ such that j lies on the unique path from i to k in T, and
- $\max_{t \in V(\mathsf{T})} |B_t| = w + 1$.

To avoid confusion, we speak about the *nodes* of a tree-decomposition and the *vertices* of a graph. The sets of $\mathcal{B}$ are called *bags*. The *treewidth* of G, denoted by $\mathbf{tw}(G)$, is the smallest integer w such that there is a tree-decomposition of G of width w.

Theorem 3 (Bodlander *et al.* [4]). *Let G be a graph and k be an integer. In time $2^{O(k)} \cdot n$, we can either decide that $\mathbf{tw}(G) > k$ or construct a tree-decomposition of G of width at most $5k + 4$.*

A tree-decomposition $(\mathsf{T}, \mathcal{B})$ rooted at a distinguished node t_r is *nice* if the following conditions are fulfilled:

- $B_{t_r} = \emptyset$ and this is the only empty bag,
- each node has at most two children,
- for each leaf $t \in V(\mathsf{T})$, $|B_t| = 1$,
- if $t \in V(\mathsf{T})$ has exactly one child t', then either
  - $B_t = B_{t'} \cup \{v\}$ for some $v \notin B_{t'}$ and t is called an *introduce-vertex* node, or
  - $B_t = B_{t'} \setminus \{v\}$ for some $v \in B_{t'}$ and t is called a *forget-vertex* node, or
  - $B_t = B_{t'}$, t is associated with an edge $\{x, y\} \in E(G)$ with $x, y \in B_t$, and t is called an *introduce-edge* node. We add the constraint that each edge of G labels exactly one node of T.
- and if $t \in V(\mathsf{T})$ has exactly two children t' and t'', then $B_t = B_{t'} = B_{t''}$. Then t is called a *join* node.

Note that we follow closely the definition of nice tree-decomposition given in [7], which slightly differs from the usual one [13]. Given a tree-decomposition, then we can build a nice tree-decomposition of G with the same width in polynomial time [7,13].

Let $(\mathsf{T}, \mathcal{B})$ be a nice tree-decomposition of a graph G. For each node $t \in V(\mathsf{T})$, we define the graph $G_t = (V_t, E_t)$ where V_t is the union of all bags corresponding to the descendant nodes of t, and E_t is the set of all edges introduced by the descendant nodes of t. Observe that the graph G_t may be disconnected.

The display graph. Let $\mathcal{T} = \{(T_1, \phi_1), (T_2, \phi_2), \ldots, (T_k, \phi_k)\}$ be a collection of unrooted phylogenetic trees. The *display graph* $D_{\mathcal{T}} = (V_D, E_D)$ of $\mathcal{T}$ is the graph obtained from the disjoint union of the trees in $\mathcal{T}$ by iteratively identifying every pair of labeled vertices with the same label. We denote by L_D the set of vertices of $D_{\mathcal{T}}$ resulting from these identifications. The elements of L_D are called the *labeled vertices*. Observe that every vertex of $V_D \setminus L_D$ (resp. every edge of E_D) is also a vertex (resp. an edge) of some tree $T_i \in \mathcal{T}$. If v is a vertex of L_D, then we will say, with a slight abuse of notation, that v is a vertex of T_i if it results from the identification of some leaf of T_i. Finally, the display graph $D_{\mathcal{T}}$ is equipped with a coloring function $c : V_D \cup E_D \to \{0, \ldots, k\}$ defined as follows. If $v \in L_D$, then we set $c(v) = 0$; if $v \in (V_D \setminus L_D) \cup E_D$ belongs to the tree T_i, we set $c(v) = i$. Observe that if a vertex $v \in L_D$ is incident to an edge e such that $c(e) = i$, then v belongs to T_i. Suppose that $\widehat{T}$ is a (strictly) compatible supertree of $\mathcal{T}$. Then we extend the definition of vertex-model and edge-model for the vertices and edges of the T_i's to the vertices and edges of the display graph $D_{\mathcal{T}}$.

The following theorem provides a bound on the treewith of the display graph of a (strictly) compatible family of unrooted phylogenetic trees:

Theorem 4 (Bryant and Lagergren [5]). *Let $\mathcal{T} = \{(T_1, \phi_1), (T_2, \phi_2), \ldots, (T_k, \phi_k)\}$ be a collection of (strictly) compatible unrooted phylogenetic trees, not necessarily on the same label set. The display graph of $\mathcal{T}$ has treewidth at most k.*

3 Compatibility version

This section provides a proof of Theorem 1. Let $D = (V_D, E_D)$ be the display graph of a collection $\mathcal{T} = \{(T_1, \phi_1), (T_2, \phi_2), \ldots, (T_k, \phi_k)\}$ of unrooted phylogenetic trees, and let $n = |V(D)|$. By Theorem 3 and Theorem 4, we may assume that we are given a nice tree-decomposition $(\mathsf{T}, \mathcal{B})$ of D of width at most $5k+4$, as otherwise we can safely conclude that $\mathcal{T}$ is not compatible. Let t_r be the root of T, and recall that $B_{t_r} = \emptyset$.

Our objective is to build a compatible supertree $\widehat{T}$ of $\mathcal{T}$, if such exists. (We would like to note that there could exist an exponential number of compatible supertrees; we are just interested in constructing *one* of them.) As it is usually the case of dynamic programming algorithms on tree-decompositions, for building $\widehat{T}$ we process $(\mathsf{T}, \mathcal{B})$ in a bottom-up way from the leaves to the root, where we will eventually decide whether a solution exists or not. We first describe the data structure used by the algorithm along with a succinct intuition behind the defined objects, and then we proceed to the description of the dynamic programming algorithm itself.

Description of the data structure. Before defining the dynamic-programming table associated with every node t of $(\mathsf{T}, \mathcal{B})$, we need a few more definitions.

Definition 1. *Given a node t of $(\mathsf{T}, \mathcal{B})$, its graph $G_t = (V_t, E_t)$, and a subset $Z \subseteq V_t$, a* (Z, t)-supertree *is a tuple $\mathfrak{T} = (T, \varphi, \psi, \rho)$ such that*

- *T is a tree containing at most $|B_t| + |Z|$ vertices,*
- *$\varphi : Z \to 2^{V(T)}$, called the* vertex-model function*, associates every $v \in Z$ with a subset $\varphi(v)$ such that*
  - *$T[\varphi(v)]$ is connected and if v is a labeled vertex, then $|\varphi(v)| = 1$, and*
  - *if u and v are two vertices of Z such that $c(u) = c(v)$, then $\varphi(u) \cap \varphi(v) = \emptyset$,*
- *$\psi : E(T) \to 2^{[k]}$, called* the edge-model function*, associates a subset of colors with every edge of T, and*
- *$\rho : Z \to V(T)$, called the* vertex-representative function*, selects, for each vertex $v \in Z$, a* representative *$\rho(v)$ in the vertex-model $\varphi(v) \subseteq V(T)$.*

Moreover, we say that a (Z,t)-supertree (T, φ, ψ, ρ) is valid *if*

- *for every $\{u, v\} \in E_t$ such that $u, v \in Z$, then the unique edge e between $\varphi(u)$ and $\varphi(v)$ exists in T and satisfies $c(\{u, v\}) \in \psi(e)$.*

For a node t of $(\mathsf{T}, \mathcal{B})$, we define a B_t-supertree *as a (B_t, t)-supertree and a* V_t-supertree *as a (V_t, t)-supertree.*

To give some intuition on why (Z,t)-supertrees capture partial solutions of our problem, let us assume that $\widehat{T}$ is a compatible supertree of $\mathcal{T}$ and consider a node t of $(\mathsf{T}, \mathcal{B})$. Then we can define a B_t-supertree $\mathfrak{T} = (T, \varphi, \psi, \rho)$ as follows:

- For every vertex $v \in B_t$, $\rho(v)$ can be chosen as any element in the set $\widehat{\varphi}(v)$,
- $T = \widehat{T}|_Y$, where $Y = \bigcup_{v \in B_t} \rho(v)$,
- for every vertex $v \in B_t$, $\varphi(v) = V(T) \cap \widehat{\varphi}(v)$, where $\widehat{\varphi}(v)$ is the vertex-model of v in $\widehat{T}$, and
- for every edge $e \in E(T)$, $i \in \psi(e)$ if there exist an edge $\{u, v\} \in E_t$, with $c(\{u, v\}) = i$, and an edge $f \in E(\widehat{T})$ such that f is incident to a vertex of $\widehat{\varphi}(u)$ and to a vertex of $\widehat{\varphi}(v)$, and f is on the unique path in $\widehat{T}$ between the vertices incident to e.

The edge-model function ψ introduced in Definition 1 allows to keep track, for every edge $e \in E(T)$, of the set of trees in $\mathcal{T}$ containing an edge having e as an edge-model. Observe that the size of a vertex-model $\widehat{\varphi}(v)$ in $\widehat{T}$ of some vertex $v \in V_D$ may depend on n (so, a priori, we may need to consider a number of vertex-models of size exponential in n). We overcome this problem via the vertex-representative function ρ, which allows us to store a tree T of size at most $2k$. This tree T captures how the vertex-models in $\widehat{T}$ "project" to the current bag, namely B_t, of the tree-decomposition of the display graph.

Before we describe the information stored at each node of the tree-decomposition, we need three more definitions.

Definition 2. *A tuple $\mathfrak{T}_s = (T_s, \varphi_s, \psi_s, \rho_s)$ is called a* shadow B_t*-supertree if there exists a B_t-supertree $\mathfrak{T} = (T, \varphi, \psi, \rho)$ such that*

- *T_s is a tree obtained from T by subdividing every edge once, called* shadow tree*. The new vertices are called* shadow vertices *and denoted by $S(T_s)$, while the original ones, that is, $V(T_s) \setminus S(T_s)$, are denoted by $O(T_s)$,*

- *for every* $v \in B_t$, $\varphi_s(v)$ *is a subset of* $V(T_s)$ *such that* $T_s[\varphi_s(v)]$ *is connected and such that* $\varphi(v) = \varphi_s(v) \cap O(T_s)$, *where we licitly consider the vertices in* $\varphi(v)$ *as a subset of* $O(T_s)$. *Furthermore, if* $u, v \in B_t$ *with* $c(u) = c(v)$, *then* $\varphi_s(u) \cap \varphi_s(v) = \emptyset$,
- $\psi_s : E(T_s) \to 2^{[k]}$ *such that for every* $s \in S(T_s)$, *if* x *and* y *are the neighbors of* s *in* T_s, *then* $\psi_s(\{x, s\}) = \psi_s(\{s, y\}) = \psi(\{x, y\})$, *and*
- $\rho_s : B_t \to V(T_s)$ *such that for every* $v \in B_t$, $\rho_s(v) = \rho(v)$.

We say that $\mathfrak{T}_s$ is a *shadow* of $\mathfrak{T}$. Note that $\mathfrak{T}$ may have more than one shadow satisfying Definition 2.

Definition 3. *Let* $\mathfrak{T} = (T, \varphi, \psi, \rho)$ *be a* (Z, t)*-supertree. The* restriction *of* $\mathfrak{T}$ *to a subset of vertices* $Y \subseteq V_t$ *is defined as the* (Y, t)*-supertree* $\mathfrak{T}|_Y = (\tilde{T}, \tilde{\varphi}, \tilde{\psi}, \tilde{\rho})$, *where*

- $\tilde{T} = T|_Z$, *where* $Z = \{\rho(v) \mid v \in Y\}$,
- *for every* $v \in Y$, $\tilde{\varphi}(v) = \varphi(v) \cap V(T|_Y)$,
- *for every* $e \in E(\tilde{T})$, $\tilde{\psi}(e) = \bigcup_{f \in E(P_e)} \psi(f)$, *where* P_e *is the unique path in* T *between the vertices incident to* e, *and*
- *for every* $v \in Y$, $\tilde{\rho}(v) = \rho(v)$.

If $\mathfrak{T}$ *is a* (Z, t)*-supertree and* $B_t \subseteq Z$, *we define a* shadow restriction of $\mathfrak{T}$ to B_t *as a shadow of* $\mathfrak{T}|_{B_t}$, *and we denote it by* $\mathfrak{T}|^s_{B_t}$.

Definition 4. *Two* (Z, t)*-supertrees* $\mathfrak{T} = (T, \varphi, \psi, \rho)$ *and* $\mathfrak{T}' = (T', \varphi', \psi', \rho')$ *are* equivalent, *and we denote it by* $\mathfrak{T} \simeq \mathfrak{T}'$, *if there exists an isomorphism* α *from* T *to* T' *such that*

- $\forall v \in Z$, $\forall a \in \varphi(v)$, $\alpha(a) \in \varphi'(v)$,
- $\forall e \in E(T)$, $\psi(e) = \psi'(\alpha(e))$, *and*
- $\forall v \in Z$, $\alpha(\rho(v)) = \rho'(v)$.

Every node t of $(\mathsf{T}, \mathcal{B})$ is associated with a set $\mathcal{R}_t$ of pairs $(\mathfrak{T}, \gamma)$, called *colored shadow* B_t*-supertrees*, where $\mathfrak{T} = (T, \varphi, \psi, \rho)$ is a shadow B_t-supertree and $\gamma : V(T) \to 2^{[k]}$ is the so-called *coloring function*. The dynamic programming algorithm will maintain the following invariant:

Invariant 1 *A colored shadow* B_t*-supertree* $(\mathfrak{T} = (T, \varphi, \psi, \rho), \gamma)$ *belongs to* $\mathcal{R}_t$ *if and only if there exists a valid* V_t*-supertree* $\mathfrak{T}_{\mathsf{ps}} = (T_{\mathsf{ps}}, \varphi_{\mathsf{ps}}, \psi_{\mathsf{ps}}, \rho_{\mathsf{ps}})$ *such that*

(1) $\mathfrak{T} \simeq \mathfrak{T}_{\mathsf{ps}}|^s_{B_t}$,
(2) *for every* $a \in V(T)$, *a color* $i \in \gamma(a)$ *if and only if there exists* $u \in V_t$ *with* $c(u) = i$ *such that* $a \in \varphi_{\mathsf{ps}}(u)$, *and*
(3) *for every* $z \in S(T)$ *with neighbors* x *and* y *in* $V(T)$, *a color* $i \in \gamma(z)$ *if there exists* $u \in V_t$ *with* $c(u) = i$ *and* $x, y \notin \varphi_{\mathsf{ps}}(u)$ *such that the unique path between* x *and* y *in* T_{ps} *uses at least one vertex of* $\varphi_{\mathsf{ps}}(u)$.

Intuitively, condition **(2)** of Invariant 1 guarantees that for every vertex $v \in V(T)$, we can recover the set of trees for which v has already appeared in a vertex-model of a vertex of $V_t \setminus B_t$. On the other hand, condition **(3)** of

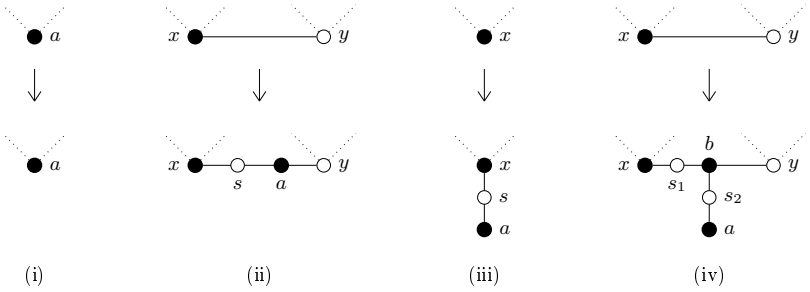

Fig. 1. The four possible cases (i-iv) in the dynamic programming algorithm. The configurations above correspond to T', while the ones below correspond to T. Full dots correspond to vertices in $O(T)$, the other ones being in $S(T)$.

Invariant 1 is useful for the following reason. When a vertex is forgotten in the tree-decomposition, we need to keep track of its "trace", in the sense that the colors given to the corresponding shadow vertex guarantee that the algorithm will construct vertex-models appropriately. If γ is a coloring function satisfying conditions **(2)** and **(3)**, we say that γ is *consistent* with $\mathfrak{T}_{\mathsf{ps}}$.

For $Z = \emptyset$, we denote by $\oslash$ the unique colored shadow (Z,t)-supertree. From the above description, it follows that the collection $\mathcal{T}$ is compatible if and only if $\oslash \in \mathcal{R}_{t_r}$. Indeed, for $t = t_r$ we have that $B_{t_r} = \emptyset$ and $V_{t_r} = V_D$. In that case, the only condition imposed by Invariant 1 is the existence of a valid V_D-supertree. Then, by Definition 1, the existence of such a supertree is equivalent to the existence of a compatible supertree $\widehat{T}$ of $\mathcal{T}$ in which the vertex-models and edge-models are given by the functions φ and ψ, respectively. Finally, note that the first condition of Definition 1, namely that $|\widehat{T}| \leq |B_{t_r}| + |V_{t_r}| = |V_D|$, is not a restriction on the set of solutions, as we may clearly assume that the size of a compatible supertree is always at most the size of the display graph.

Description of the dynamic programming algorithm. Let $(\mathsf{T}, \mathcal{B})$ be a nice tree-decomposition of the display graph D of $\mathcal{T}$. We proceed to describe how to compute the set $\mathcal{R}_t$ for every node $t \in \mathsf{T}$. For that, we will assume inductively that, for every descendant t' of t, we have at hand the set $\mathcal{R}_{t'}$ that has been correctly built. We distinguish several cases depending on the type of node t:

1. t **is a leaf with** $B_t = \{v\}$**:** $\mathcal{R}_t = \{((T, \varphi, \psi, \rho), \gamma)\}$, where T is a tree with only one vertex a, $\rho(v) = a$, $\varphi(v) = \{a\}$, $\psi : \emptyset \to 2^{[k]}$, and $\gamma(a) = \{c(v)\}$.
2. t **is an introduce-vertex node such that the introduced vertex** v **is unlabeled:** For every element $(\mathfrak{T}' = (T', \varphi', \psi', \rho'), \gamma')$ of $\mathcal{R}_{t'}$, we add to $\mathcal{R}_t$ the elements of the form $(\mathfrak{T} = (T, \varphi, \psi, \rho), \gamma)$ that can be built according to one of the following four cases. For all of them, we define the vertex-representative function such that $\rho(v) = a$ for some vertex $a \in V(T)$, and for every $u \in B_{t'}$, $\rho(u) = \rho'(u)$. The different cases depend on this vertex a.

(i) $\rho(v) = a$ **such that** $a \in V(T')$ **and** $c(v) \notin \gamma'(a)$**.** See Figure 1(i) for an example. We define $T = T'$. Let us define φ, ψ, and γ.
- *Definition of the vertex-model function*: $T[\varphi(v)]$ is connected, contains a, and for every $z \in \varphi(v)$, $c(v) \notin \gamma'(z)$. For every $u \in B_{t'}$, $\varphi(u) = \varphi'(u)$.
- *Definition of the edge-model function*: For every $e \in E(T)$, $\psi(e) = \psi'(e)$.
- *Definition of the coloring function*: For every $z \in V(T)$, $\gamma(z) = \gamma'(z) \cup \{c(v) \mid z \in \varphi(v)\}$.

(ii) $\rho(v) = a$ **and** a **subdivides an edge** $\{x, y\}$ **of** T' **with** $c(v) \notin \psi'(\{x, y\})$. See Figure 1(ii) for an example. Since T' is a shadow tree, assume w.l.o.g. that $x \in O(T')$ and $y \in S(T')$. Then T is obtained from T' by removing the edge $\{x, y\}$, adding two vertices $a \in O(T)$ and $s \in S(T)$ and three edges $\{x, s\}$, $\{s, a\}$, and $\{a, y\}$. Let us define φ, ψ, and γ.
- *Definition of the vertex-model function*: $T[\varphi(v)]$ is connected, contains a, and for each $z \in \varphi(v)$, $c(v) \notin \gamma'(z)$. For each $u \in B_{t'}$, $T[\varphi(u)]$ is connected, $\varphi'(u) \subseteq \varphi(u) \subseteq \varphi'(u) \cup \{a\} \cup S(T)$, and if u is unlabeled, then $\varphi(u) = \varphi'(u)$. For each $u, u' \in B_t$ with $c(u) = c(u')$, $\varphi(u) \cap \varphi(u') = \emptyset$.
- *Definition of the edge-model function*: For each $e \in E(T) \setminus \{\{x, s\}, \{s, a\}, \{a, y\}\}$, $\psi(e) = \psi'(e)$. Also, $\psi(\{x, s\}) = \psi(\{s, a\}) = \psi(\{a, y\}) = \psi'(\{x, y\})$.
- *Definition of the coloring function*: For each $z \in O(T) \setminus \{a\}$, $\gamma(z) = \gamma'(z) \cup \{c(v) \mid z \in \varphi(v)\}$. $\gamma(a) = \{i \mid \exists u \in B_t : c(u) = i$ and $a \in \varphi(u)\} \cup \psi'(\{x, y\})$. For each $z \in S(T')$, $\gamma(z) = \gamma'(z) \cup \{i \mid \exists u \in B_t : c(u) = i$ and $z \in \varphi(u)\}$. Finally, $\gamma(s) = \{i \mid \exists u \in B_t : c(u) = i$ and $s \in \varphi(u)\} \cup \psi'(\{x, y\})$.

(iii) $\rho(v) = a$ **with** $a \notin V(T')$ **and** a **is connected to a vertex** $x \in V(T')$. See Figure 1(iii) for an example. T is obtained from T' by adding two vertices $a \in O(T)$ and $s \in S(T)$ and two edges $\{a, s\}$ and $\{s, x\}$. Let us define φ, ψ, and γ.
- *Definition of the vertex-model function*: $T[\varphi(v)]$ is connected, contains a, and for each $z \in \varphi(v)$, $c(v) \notin \gamma'(z)$. For each $u \in B_{t'}$, $T[\varphi(u)]$ is connected, $\varphi'(u) \subseteq \varphi(u) \subseteq \varphi'(u) \cup \{a\} \cup S(T)$, and if u is unlabeled, then $\varphi(u) = \varphi'(u)$. For each $u, u' \in B_t$ with $c(u) = c(u')$, $\varphi(u) \cap \varphi(u') = \emptyset$.
- *Definition of the edge-model function*: For each $e \in E(T) \setminus \{\{a, s\}, \{s, x\}\}$, $\psi(e) = \psi'(e)$, and $\psi(\{a, s\}) = \psi(\{s, x\}) = \emptyset$.
- *Definition of the coloring function*: For each $z \in V(T) \setminus \{a, s\}$, $\gamma(z) = \gamma'(z) \cup \{c(v) \mid z \in \varphi(v)\}$. For each $z \in \{a, s\}$, $\gamma(z) = \{i \mid \exists u \in B_t : c(u) = i$ and $z \in \varphi(u)\}$.

(iv) $\rho(v) = a$ **with** $a \notin V(T')$ **and** a **subdivides an edge** $\{x, y\}$ **of** T'. See Figure 1(iv) for an example. Again, we may assume that $x \in O(T')$ and $y \in S(T')$. Then T is obtained from T' by removing the edge $\{x, y\}$, adding four vertices $a, b \in O(T)$ and $s_1, s_2 \in S(T)$, and five edges $\{x, s_1\}$, $\{s_1, b\}$, $\{b, y\}$, $\{a, s_2\}$, and $\{s_2, b\}$. Let us define φ, ψ, and γ.

- *Definition of the vertex-model function*: $T[\varphi(v)]$ is connected, contains a and, for every $z \in \varphi(v)$, $c(v) \notin \gamma'(z)$. For each $u \in B_{t'}$, $T[\varphi(u)]$ is connected, $\varphi'(u) \subseteq \varphi(u) \subseteq \varphi'(u) \cup \{a, b\} \cup S(T)$, and if u is unlabeled, then $\varphi(u) = \varphi'(u)$. For each $u, u' \in B_t$ with $c(u) = c(u')$, $\varphi(u) \cap \varphi(u') = \emptyset$.
- *Definition of the edge-model function*: For each edge $e \in E(T) \setminus \{\{a, s_2\}, \{s_2, b\}, \{x, s_1\}, \{s_1, b\}, \{b, y\}\}$, $\psi(e) = \psi'(e)$. $\psi(\{x, s_1\}) = \psi(\{s_1, b\}) = \psi(\{b, y\}) = \psi'(\{x, y\})$, and $\psi(\{a, s_2\}) = \psi(\{s_2, b\}) = \emptyset$.
- *Definition of the coloring function*: For every $z \in O(T) \setminus \{a, b\}$, $\gamma(z) = \gamma'(z) \cup \{c(v) \mid z \in \varphi(v)\}$. For every $z \in \{a, s_2\}$, $\gamma(z) = \{i \mid \exists u \in B_t : c(u) = i \text{ and } z \in \varphi(u)\}$. For every $z \in \{b, s_1\}$, $\gamma(z) = \{i \mid \exists u \in B_t : c(u) = i \text{ and } z \in \varphi(u)\} \cup \psi'(\{x, y\})$. For every $z \in S(T')$, $\gamma(z) = \gamma'(z) \cup \{i \mid \exists u \in B_t : c(u) = i \text{ and } z \in \varphi(u)\}$.

3. t **is an introduce-vertex node such that the introduced vertex** v **is labeled:** This case is very similar to Case 2 but, as vertex v is a leaf, only Case 2(iii) and Case 2(iv) can be applied. In both cases, we further impose that $\varphi(v) = \{a\}$ and $\gamma(v) = \{i \in [k] \mid v \in L(T_i), T_i \in \mathcal{T}\}$.
4. t **in an introduce-edge node for an edge** $\{v, w\}$ **with** $c(\{v, w\}) = i$**:** Let $(\mathfrak{T}' = (T', \varphi', \psi', \rho'), \gamma')$ be an element of $\mathcal{R}_{t'}$ such that there exist $a \in \varphi'(v)$ and $b \in \varphi'(w)$ such that $\{a, b\} \in E(T)$ and $i \notin \psi'(\{a, b\})$. We construct $(\mathfrak{T} = (T, \varphi, \psi, \rho), \gamma)$ as an element of $\mathcal{R}_t$ as follows: $T = T'$. For every $v \in B_t$, $\varphi(v) = \varphi'(v)$. For every $e \in E(T) \setminus \{\{a, b\}\}$, $\psi(e) = \psi'(e)$. $\psi(\{a, b\}) = \psi'(\{a, b\}) \cup \{i\}$. For every $v \in V(T)$, $\gamma(v) = \gamma'(v)$.
5. t **is a forget-vertex node for a vertex** v**:** Let $(\mathfrak{T}' = (T', \varphi', \psi', \rho'), \gamma')$ be an element of $\mathcal{R}_{t'}$. We construct $(\mathfrak{T} = (T, \varphi, \psi, \rho), \gamma)$ as an element of $\mathcal{R}_t$ as follows: $\mathfrak{T} = \mathfrak{T}'|^z_{B_{t'}}$. For every $a \in O(T)$, $\gamma(a) = \gamma'(a)$. For every $z \in S(T)$, if x and y are the neighbors of z in T, then $\gamma(z) = \{i \mid \exists a \in V(T') \text{ on the path between } x \text{ and } y \text{ in } T' : (i \in \gamma'(a)) \text{ and } (\forall u \in B_t : a \notin \varphi'(u))\}$.
6. t **is a join node:** Let $(\mathfrak{T}' = (T, \varphi, \psi', \rho), \gamma')$ be an element of $\mathcal{R}_{t'}$ and let $(\mathfrak{T}'' = (T, \varphi, \psi'', \rho), \gamma'')$ be an element of $\mathcal{R}_{t''}$ such that for every $z \in V(T)$, $\gamma'(z) \cap \gamma''(z) = \emptyset$ and for every $e \in E(T)$, $\psi'(z) \cap \psi''(z) = \emptyset$. We construct $(\mathfrak{T} = (T, \varphi, \psi, \rho), \gamma)$ as an element of $\mathcal{R}_t$ as follows: For every $e \in E(T)$, $\psi(e) = \psi'(e) \cup \psi''(e)$, and for every $z \in V(T)$, $\gamma(z) = \gamma'(z) \cup \gamma''(z)$.

4 Further research

In this paper we give the first "reasonable" FPT algorithms for the Compatibility and the Agreement problems for unrooted phylogenetic trees. Even though this is, from a theoretical point of view, a big step further toward solving this problem in reasonable time, our running times are still prohibitive to be of any use in real-life phylogenomic studies, where k can go up very quickly [8]. One possibility to design a practical algorithm is to devise reduction rules to keep k small. Another possibility would be to design an FPT algorithm with respect to a parameter that is smaller than the number of gene trees in phylogenomic studies.

From a more theoretical perspective, a natural question is whether the function $2^{O(k^2)}$ in the running times of our algorithms can be improved. It would also be interesting to prove lower bounds for algorithms parameterized by treewidth to solve these problems, assuming the Exponential Time Hypothesis [14].

References

1. A. V. Aho, Y. Sagiv, T. G. Szymanski, and J. D. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM Journal of Computing*, 10(3):405–421, 1981.
2. O. R. Bininda-Emonds. *Phylogenetic supertrees: combining information to reveal the tree of life*, volume 4. Springer Science & Business Media, 2004.
3. O. R. Bininda-Emonds, J. L. Gittleman, and M. A. Steel. The (super) tree of life: procedures, problems, and prospects. *Annual Review of Ecology and Systematics*, pages 265–289, 2002.
4. H. L. Bodlaender, P. G. Drange, M. S. Dregi, F. V. Fomin, D. Lokshtanov, and M. Pilipczuk. An $O(c^k n)$ 5-Approximation Algorithm for Treewidth. In *Proc. of the IEEE 54th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 499–508, 2013.
5. D. Bryant and J. Lagergren. Compatibility of unrooted phylogenetic trees is FPT. *Theoretical Computer Science*, 351(3):296–302, 2006.
6. A. Cayley. A theorem on trees. *Quarterly Journal of Mathematics*, 23:376–378, 1889.
7. M. Cygan, J. Nederlof, M. Pilipczuk, M. Pilipczuk, J. M. M. van Rooij, and J. O. Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *Proc. of the IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 150–159, 2011.
8. F. Delsuc, H. Brinkmann, and H. Philippe. Phylogenomics and the reconstruction of the tree of life. *Nature Reviews Genetics*, 6(5):361–375, 2005.
9. R. Diestel. *Graph Theory*, volume 173. Springer-Verlag, 4th edition, 2010.
10. J. Felsenstein. *Inferring Phylogenies*. Sinauer Associates, Incorporated, 2004.
11. M. Frick and M. Grohe. The complexity of first-order and monadic second-order logic revisited. *Annals of Pure and Applied Logic*, 130(1-3):3–31, 2004.
12. A. D. Gordon. Consensus supertrees: the synthesis of rooted trees containing overlapping sets of labeled leaves. *Journal of classification*, 3(2):335–348, 1986.
13. T. Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994.
14. D. Lokshtanov, D. Marx, and S. Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011.
15. W. Maddison. Reconstructing character evolution on polytomous cladograms. *Cladistics*, 5(4):365–377, 1989.
16. M. Ng and N. C. Wormald. Reconstruction of rooted trees from subtrees. *Discrete Applied Mathematics*, 69(1-2):19–31, 1996.
17. C. Scornavacca. *Supertree methods for phylogenomics*. PhD thesis, Université Montpellier II-Sciences et Techniques du Languedoc, 2009.
18. C. Scornavacca, L. van Iersel, S. Kelk, and D. Bryant. The agreement problem for unrooted phylogenetic trees is fpt. *Journal of Graph Algorithms and Applications*, 18(3):385–392, 2014.
19. M. Steel. The complexity of reconstructing trees from qualitative characters and subtrees. *Journal of Classification*, 9:91–116, 1992.

A Correctness of the algorithm for the compatibility version

Let t be a node of $(\mathsf{T}, \mathcal{B})$. Our objective is to prove that, on the one hand, the elements $(\mathfrak{T}, \gamma)$ generated by the algorithm indeed belong to the set $\mathcal{R}_t$ (that is, that they satisfy Invariant 1) and, on the other hand, that all the elements of the set $\mathcal{R}_t$ are constructed by the algorithm. We will assume inductively that both claims are true for every descendant t' of t.

Our approach for proving that the generated elements belong to $\mathcal{R}_t$ is the following. We distinguish again the cases of the algorithm. For each of them, the assumption that $\mathcal{R}_{t'}$ has been correctly built for every descendant t' of t guarantees the existence, for every element $(\mathfrak{T}', \gamma')$ of $\mathcal{R}_{t'}$, of the corresponding certificate $\mathfrak{T}'_{\mathsf{ps}}$ that implies by Invariant 1 that $(\mathfrak{T}', \gamma') \in \mathcal{R}_{t'}$. We will then use $\mathfrak{T}'_{\mathsf{ps}}$ to prove, for each of the elements $(\mathfrak{T}, \gamma)$ constructed by the algorithm, that there *exists* a certificate $\mathfrak{T}_{\mathsf{ps}}$ implying that $(\mathfrak{T}, \gamma) \in \mathcal{R}_t$.

We would like to stress that, in order to prove that $(\mathfrak{T}, \gamma) \in \mathcal{R}_t$, we only need to worry about the *existence* of such a certificate $\mathfrak{T}_{\mathsf{ps}}$, and not about how it can be *constructed*. However, if we are interested in constructing a compatible supertree (and not only knowing whether it exists or not), we can easily do it as well. Indeed, starting from the leaves of the tree-decomposition, by using the operations described below we can inductively grow the certificates $\mathfrak{T}'_{\mathsf{ps}}$ of $\mathcal{R}_{t'}$ to get the certificates $\mathfrak{T}_{\mathsf{ps}}$ of $\mathcal{R}_t$, within the same running time of the algorithm.

We now proceed to distinguish the different cases of the dynamic programming algorithm presented in Section 3:

1. t **is a leaf with** $B_t = \{v\}$**:** T_{ps} is a tree with only one vertex a, $\rho_{\mathsf{ps}}(v) = a$, $\varphi_{\mathsf{ps}}(v) = \{a\}$, and $\psi_{\mathsf{ps}} : \emptyset \to 2^{[k]}$.
2. t **is an introduce-vertex node such that the introduced vertex** v **is unlabeled:** Given an element $(\mathfrak{T}', \gamma')$ of $\mathcal{R}_{t'}$ with the corresponding certificate $\mathfrak{T}'_{\mathsf{ps}}$, we distinguish the different cases of the algorithm that create elements of the form $(\mathfrak{T}, \gamma)$, and we define for each case a certificate $\mathfrak{T}_{\mathsf{ps}}$ of $\mathfrak{T}$, which implies that $(\mathfrak{T}, \gamma) \in \mathcal{R}_t$.

   (i) $\rho(v) = a$ **such that** $a \in V(T')$ **and** $c(v) \notin \gamma'(a)$**.** Then $T_{\mathsf{ps}} = T'_{\mathsf{ps}}$. Let us define ρ_{ps}, φ_{ps}, and ψ_{ps}.
   - *Definition of the vertex-representative function*:
     - $\rho_{\mathsf{ps}}(v) = \rho(v) = a$ and
     - for every $u \in V_{t'}$, $\rho_{\mathsf{ps}}(u) = \rho'_{\mathsf{ps}}(u)$.
   - *Definition of the vertex-model function*:
     - $T_{\mathsf{ps}}[\varphi_{\mathsf{ps}}(v)]$ is connected and contains a, $\varphi(v) \cap O(T) = \varphi_{\mathsf{ps}}(v) \cap O(T)$,
     - for every $u \in B_{t'}$, $\varphi_{\mathsf{ps}}(u) = \varphi'_{\mathsf{ps}}(u)$, and
     - for every $u, u' \in V_t$ with $c(u) = c(u')$, $\varphi_{\mathsf{ps}}(u) \cap \varphi_{\mathsf{ps}}(u') = \emptyset$.
   - *Definition of the edge-model function*:
     - for every $e \in E(T)$, $\psi_{\mathsf{ps}}(e) = \psi'_{\mathsf{ps}}(e)$.

(ii) $\rho(v) = a$ **and** a **subdivides an edge** $\{x, y\}$ **of** T' **with** $c(v) \notin \psi'(\{x, y\})$. T_{ps} is obtained from T'_{ps} by removing an edge $\{x_{\mathsf{ps}}, y_{\mathsf{ps}}\}$ on the path between x and y, and adding a vertex a and two edges $\{x_{\mathsf{ps}}, a\}$ and $\{a, y_{\mathsf{ps}}\}$. Let us define ρ_{ps}, φ_{ps}, and ψ_{ps}.

- *Definition of the vertex-representative function*:
  - $\rho_{\mathsf{ps}}(v) = \rho(v) = a$ and
  - for every $u \in V_{t'}$, $\rho_{\mathsf{ps}}(u) = \rho'_{\mathsf{ps}}(u)$.
- *Definition of the vertex-model function*:
  - $T_{\mathsf{ps}}[\varphi_{\mathsf{ps}}(v)]$ is connected and contains a,
  - for every $u \in V_{t'}$, $T_{\mathsf{ps}}[\varphi_{\mathsf{ps}}(u)]$ is connected, $\varphi'_{\mathsf{ps}}(u) \subseteq \varphi_{\mathsf{ps}}(u) \subseteq \varphi'_{\mathsf{ps}}(u) \cup \{a\}$, and if u is unlabeled, then $\varphi_{\mathsf{ps}}(u) = \varphi'_{\mathsf{ps}}(u)$,
  - for every $u \in B_t$, $\varphi(u) \cap O(T) = \varphi_{\mathsf{ps}}(u) \cap O(T)$,
  - for every $u, u' \in V_t$ with $c(u) = c(u')$, $\varphi_{\mathsf{ps}}(u) \cap \varphi_{\mathsf{ps}}(u') = \emptyset$, and
  - for every $\{u, u'\} \in E_t$, there exist $w \in \varphi_{\mathsf{ps}}(u)$ and $w' \in \varphi_{\mathsf{ps}}(u')$ such that $\{w, w'\} \in E(T_{\mathsf{ps}})$.
- *Definition of the edge-model function*:
  - for every $e \in E(T) \setminus \{\{x_{\mathsf{ps}}, a\}, \{a, y_{\mathsf{ps}}\}\}$, $\psi_{\mathsf{ps}}(e) = \psi'_{\mathsf{ps}}(e)$ and
  - $\psi_{\mathsf{ps}}(\{x_{\mathsf{ps}}, a\}) = \psi_{\mathsf{ps}}(\{a, y_{\mathsf{ps}}\}) = \psi'_{\mathsf{ps}}(\{x_{\mathsf{ps}}, y_{\mathsf{ps}}\})$.

(iii) $\rho(v) = a$ **with** $a \notin V(T')$ **and** a **is connected to a vertex** $x \in V(T')$. T_{ps} is obtained from T'_{ps} by adding a vertex a and an edge $\{a, x\}$. Let us define ρ_{ps}, φ_{ps}, and ψ_{ps}.

- *Definition of the vertex-representative function*:
  - $\rho_{\mathsf{ps}}(v) = \rho(v) = a$ and
  - for every $u \in V_{t'}$, $\rho_{\mathsf{ps}}(u) = \rho'_{\mathsf{ps}}(u)$.
- *Definition of the vertex-model function*:
  - $T_{\mathsf{ps}}[\varphi_{\mathsf{ps}}(v)]$ is connected and contains a,
  - for every $u \in V_{t'}$, $T_{\mathsf{ps}}[\varphi_{\mathsf{ps}}(u)]$ is connected, $\varphi'_{\mathsf{ps}}(u) \subseteq \varphi_{\mathsf{ps}}(u) \subseteq \varphi'_{\mathsf{ps}}(u) \cup \{a\}$, and if u is unlabeled, then $\varphi_{\mathsf{ps}}(u) = \varphi'_{\mathsf{ps}}(u)$,
  - for every $u \in B_t$, $\varphi(u) \cap O(T) = \varphi_{\mathsf{ps}}(u) \cap O(T)$,
  - for every $u, u' \in V_t$ with $c(u) = c(u')$, $\varphi_{\mathsf{ps}}(u) \cap \varphi_{\mathsf{ps}}(u') = \emptyset$, and
  - for every $\{u, u'\} \in E_t$, there exist $w \in \varphi_{\mathsf{ps}}(u)$ and $w' \in \varphi_{\mathsf{ps}}(u')$ such that $\{w, w'\} \in E(T_{\mathsf{ps}})$.
- *Definition of the edge-model function*:
  - for every $e \in E(T) \setminus \{\{a, x\}\}$, $\psi_{\mathsf{ps}}(e) = \psi'_{\mathsf{ps}}(e)$ and
  - $\psi(\{a, x\}) = \emptyset$.

(iv) $\rho(v) = a$ **with** $a \notin V(T')$ **and** b **subdivides an edge** $\{x, y\}$ **of** T'. T_{ps} is obtained from T'_{ps} by removing an edge $\{x_{\mathsf{ps}}, y_{\mathsf{ps}}\}$ on the path between x and y, and adding two vertices a and b and three edges $\{x_{\mathsf{ps}}, b\}$, $\{b, y_{\mathsf{ps}}\}$, and $\{a, b\}$. Let us define ρ_{ps}, φ_{ps}, and ψ_{ps}.

- *Definition of the vertex-representative function*:
  - $\rho_{\mathsf{ps}}(v) = \rho(v) = a$ and
  - for every $u \in V_{t'}$, $\rho_{\mathsf{ps}}(u) = \rho'_{\mathsf{ps}}(u)$.
- *Definition of the vertex-model function*:
  - $T_{\mathsf{ps}}[\varphi_{\mathsf{ps}}(v)]$ is connected and contains a,

  - for every $u \in V_{t'}$, $T_{\mathsf{ps}}[\varphi_{\mathsf{ps}}(u)]$ is connected, $\varphi'_{\mathsf{ps}}(u) \subseteq \varphi_{\mathsf{ps}}(u) \subseteq \varphi'_{\mathsf{ps}}(u) \cup \{a, b\}$, and if u is unlabeled, then $\varphi_{\mathsf{ps}}(u) = \varphi'_{\mathsf{ps}}(u)$,
  - for every $u \in B_t$, $\varphi(u) \cap O(T) = \varphi_{\mathsf{ps}}(u) \cap O(T)$,
  - for every $u, u' \in V_t$ with $c(u) = c(u')$, $\varphi_{\mathsf{ps}}(u) \cap \varphi_{\mathsf{ps}}(u') = \emptyset$, and
  - for every $\{u, u'\} \in E_t$, there exist $w \in \varphi_{\mathsf{ps}}(u)$ and $w' \in \varphi_{\mathsf{ps}}(u')$ such that $\{w, w'\} \in E(T_{\mathsf{ps}})$.
- *Definition of the edge-model function*:
  - for every $e \in E(T) \setminus \{\{a, b\}, \{x_{\mathsf{ps}}, b\}, \{b, y_{\mathsf{ps}}\}\}$, $\psi(e) = \psi'(e)$,
  - $\psi_{\mathsf{ps}}(\{x_{\mathsf{ps}}, b\}) = \psi_{\mathsf{ps}}(\{b, y_{\mathsf{ps}}\}) = \psi'_{\mathsf{ps}}(\{x_{\mathsf{ps}}, y_{\mathsf{ps}}\})$, and
  - $\psi_{\mathsf{ps}}(\{a, b\}) = \emptyset$.

3. t **is an introduce-vertex node such that the introduced vertex** v **is labeled:** As explained in the description of the algorithm, this case is very similar to Case 2, taking into account that only Case 2(iii) and Case 2(iv) can be applied, and by adding the following constraints:
   - $\varphi_{\mathsf{ps}}(v) = \{a\}$ and
   - $\gamma_{\mathsf{ps}}(v) = \{i \in [k] \mid v \in L(T_i), T_i \in \mathcal{T}\}$.

In the next two cases, let $(\mathfrak{T}', \gamma')$ be the element of $\mathcal{R}_{t'}$ from which the algorithm has started, let $\mathfrak{T}'_{\mathsf{ps}}$ be a certificate of $(\mathfrak{T}', \gamma')$, and let $(\mathfrak{T}, \gamma)$ be the element created by the algorithm. In both cases, we construct a certificate $\mathfrak{T}_{\mathsf{ps}}$ of $(\mathfrak{T}, \gamma')$ showing that $(\mathfrak{T}, \gamma) \in \mathcal{R}_t$.

4. t **in an introduce-edge node for an edge** $\{v, w\}$ **with** $c(\{v, w\}) = i$**:** We construct $\mathfrak{T}_{\mathsf{ps}} = (T_{\mathsf{ps}}, \varphi_{\mathsf{ps}}, \psi_{\mathsf{ps}}, \rho_{\mathsf{ps}})$ as follows:
   - $T_{\mathsf{ps}} = T'_{\mathsf{ps}}$,
   - for every $v \in V_t$, $\varphi_{\mathsf{ps}}(v) = \varphi'_{\mathsf{ps}}(v)$,
   - for every $e \in E(T) \setminus \{\{a, b\}\}$, $\psi_{\mathsf{ps}}(e) = \psi'_{\mathsf{ps}}(e)$, and
   - $\psi_{\mathsf{ps}}(\{a, b\}) = \psi'_{\mathsf{ps}}(\{a, b\}) \cup \{i\}$.
5. t **is a forget-vertex node for a vertex** v**:** In this case, we just define $\mathfrak{T}_{\mathsf{ps}} = \mathfrak{T}'_{\mathsf{ps}}$.
6. t **is a join node:** Let $(\mathfrak{T}', \gamma')$ be the element of $\mathcal{R}_{t'}$ and let $(\mathfrak{T}'', \gamma'')$ be the element of $\mathcal{R}_{t''}$ from which the algorithm has started, and let $\mathfrak{T}'_{\mathsf{ps}} = (T_{\mathsf{ps}}, \varphi_{\mathsf{ps}}, \psi'_{\mathsf{ps}}, \rho_{\mathsf{ps}})$ and $\mathfrak{T}''_{\mathsf{ps}} = (T_{\mathsf{ps}}, \varphi_{\mathsf{ps}}, \psi''_{\mathsf{ps}}, \rho_{\mathsf{ps}})$ be their certificates, respectively. We define $\mathfrak{T}_{\mathsf{ps}} = (T_{\mathsf{ps}}, \varphi_{\mathsf{ps}}, \psi_{\mathsf{ps}}, \rho_{\mathsf{ps}})$, that is, a certificate of $(\mathfrak{T}, \gamma)$ showing that $(\mathfrak{T}, \gamma) \in \mathcal{R}_t$, just by setting, for every $e \in E(T)$, $\psi_{\mathsf{ps}}(e) = \psi'_{\mathsf{ps}}(e) \cup \psi''_{\mathsf{ps}}(e)$. Note that T_{ps}, φ_{ps}, and ρ_{ps} are those given by $(\mathfrak{T}', \gamma')$ (or by $(\mathfrak{T}'', \gamma'')$).

Finally, let us argue that all the elements of the set $\mathcal{R}_t$ are indeed constructed by the algorithm. Let $(\mathfrak{T}, \gamma)$ be an element of $\mathcal{R}_t$, with $\mathfrak{T} = (T, \varphi, \psi, \rho)$, and our objective is to show that the algorithm indeed generates this element $(\mathfrak{T}, \gamma)$. In order to do this, we need to consider each case of the algorithm separately. We will only detail the arguments for Case 2, which is the most involved one, and the other ones follow by using a similar argumentation.

By definition of the set $\mathcal{R}_t$, there exists a valid V_t-supertree $\mathfrak{T}_{\mathsf{ps}}$ such that $\mathfrak{T} = \mathfrak{T}_{\mathsf{ps}}|^s_{B_t}$ and such that γ is consistent with $\mathfrak{T}_{\mathsf{ps}}$. Let $\mathfrak{T}'_{\mathsf{ps}} = \mathfrak{T}_{\mathsf{ps}}|_{V_{t'}}$. It can be easily checked that $\mathfrak{T}'_{\mathsf{ps}}$ is a valid $V_{t'}$-supertree. Let $\mathfrak{T}' = \mathfrak{T}'_{\mathsf{ps}}|^s_{B_{t'}}$ and let γ' be the coloring function consistent with $\mathfrak{T}'_{\mathsf{ps}}$. Then, as Invariant 1 is satisfied, $(\mathfrak{T}', \gamma')$ is an element of $\mathcal{R}_{t'}$. Note that $\mathfrak{T}' = \mathfrak{T}|_{B_{t'}}$. As the sets B_t and $B_{t'}$ differ by just one vertex, the elements $\mathfrak{T}$ and $\mathfrak{T}'$ are quite close to each other. Indeed, the way they differ is mainly given by the value of $\rho(v)$, in the sense that we consider all the possible ways to add a vertex $\rho(v)$ to a tree T'. It appears that there are four different ways to add $\rho(v)$ to T'. Indeed, $\rho(v)$ can either be an already existing vertex of T', or a new vertex that subdivides an edge, or a new vertex connected to an already existing vertex, or a new vertex connected to another new vertex that subdivides an edge. Our algorithm precisely explore these four possibilities for $\rho(v)$, and then updates T, φ, ψ, and γ in all the possible ways such that the resulting element is still in $\mathcal{R}$. So in particular, the algorithm necessarily created the element $(\mathfrak{T}, \gamma)$ of $\mathcal{R}_t$, as we wanted to show.

B Running time analysis of the algorithm for the compatibility version

Let us now discuss the running time of the dynamic programming algorithm described in Section 3. Let w be the width of $(\mathsf{T}, \mathcal{B})$, so we have that $w \leq 5k+4$. For each $t \in V(\mathsf{T})$, we bound the size of $\mathcal{R}_t$ as follows. Each element in $\mathcal{R}_t$ is of the form $(\mathfrak{T} = (T, \varphi, \psi, \rho), \gamma)$. Note that T has at most $3w$ nodes, and that there are at most $(3w)^{3w-2} = 2^{\mathcal{O}(k \log k)}$ distinct trees on $3w$ vertices [6]. There are at most $2^{|V(T)|\cdot|B_t|} \leq 2^{3w \cdot w}$ possible functions φ, $2^{|E(T)|\cdot k} \leq 2^{3w \cdot k}$ possible functions ψ, $|V(T)|^{|B_t|} \leq (3w)^w$ possible functions ρ, and $2^{|V(T)|\cdot k} \leq 2^{3w \cdot k}$ possible functions γ. Thus, it holds that $|\mathcal{R}_t| = 2^{\mathcal{O}(k^2)}$ for every node t of $(\mathsf{T}, \mathcal{B})$.

Concerning the complexity of computing $\mathcal{R}_t$, we distinguish several cases. This computation is trivial in Case 1 of the algorithm, that is, when t is a leaf. In Cases 2, 3, 4, and 5, the set $\mathcal{R}_t$ can be clearly computed in time polynomial in $|\mathcal{R}_{t'}|$, where t' is the child of t. Finally, in Case 6, that is, when t is a join node, the set $\mathcal{R}_t$ can also be clearly computed in time polynomial in $|\mathcal{R}_{t'}|$ and $|\mathcal{R}_{t''}|$, where t' and t'' are the two children of t. Finally, as we can assume that $|V(\mathsf{T})| = \mathcal{O}(n)$ [13], the running time claimed in Theorem 1 follows.

C Agreement version

In this section we provide a proof of Theorem 2. Again, by Theorem 3 and Theorem 4, we may assume that we are given a nice tree-decomposition $(\mathsf{T}, \mathcal{B})$ of D of width at most $5k + 4$.

The algorithm follows closely the one described in Section 3 for the compatibility version, so we will just describe the changes to be done to deal with the agreement version. Intuitively, these changes appear because now we are looking for a supertree containing each of the trees in $\mathcal{T}$ as a *topological minor*, instead

of a *minor*, and this forces us to redefine the notions of vertex-model and edge-model functions. Namely, each vertex-model becomes a *single vertex* (instead of a set of vertices), and for guaranteeing the existence of the appropriate topological minors, we have to keep track of the existence of pairwise disjoint *paths* among the vertex-models of each color (instead of just edges).

We first proceed to partially redefine the data structure, and then we will focus on the changes in the dynamic programming algorithm.

Changes in the data structure. For a node t of the tree-decomposition, our tables $\mathcal{R}_t$ store again elements of the form $(\mathfrak{T}, \gamma)$ satisfying the same invariant as in Section 3, namely Invariant 1, the difference is that we update some definitions of the data structure. Namely, the vertex-model function in the definition of (Z,t)-supertree, cf. Definition 1, is updated as follows:

- $\varphi : Z \to V(T)$ is such that if u and v are two vertices of Z with $c(u) = c(v)$, then $\varphi(u) \neq \varphi(v)$,

We also modify slightly the definition of "valid supertrees" and say that a (Z,t)-supertree (T, φ, ψ, ρ) is *valid* if

- for every $\{u,v\} \in E_t$ such that $u, v \in Z$, every edge e on the path between $\varphi(u)$ and $\varphi(v)$ in T satisfies $c(\{u,v\}) \in \psi(e)$ and
- if $i \in \psi(e)$ for some $i \in [k]$, then there exists a unique pair $\{u,v\} \in E_t$ with $u, v \in Z$ with $c(\{u,v\}) = i$ such that e lies on the path between $\varphi(u)$ and $\varphi(v)$.

It is worth noting that the dynamic programming algorithm described below satisfies that, for every vertex $v \in Z$, $\varphi(v) = \rho(v)$, and therefore the vertex-representative function ρ becomes superfluous. Nevertheless, in order for the notation to deviate as little as possible to that of Section 3, we keep ρ in the tuple $\mathfrak{T}$.

Changes in the dynamic programming algorithm. The fact that the image of the vertex-model function φ is now a single vertex allows us to substantially simplify the algorithm. In particular, in the subcases of the two cases where t is an introduce-vertex node (namely, Cases 2 and 3), we do not have to worry anymore about how the image of φ grows when introducing a new vertex, except, naturally, for this newly introduced vertex. The latter simplification implies that we do not need to update the coloring function γ either, except again for the newly introduced vertex. Finally, as the function ρ is now redundant, we may omit it from the description of the algorithm.

More precisely, Cases 1, 2, 3, 5, and 6 of the algorithm from Section 3 remain unchanged, just by taking into account that $\varphi(v)$ returns just one element, namely $\varphi(v) = a$. The changes occur in Case 4, which becomes as follows:

4. *t* **in an introduce-edge node for an edge** *$\{v,w\}$* **with** *$c(\{v,w\}) = i$***:** Let $(\mathfrak{T}' = (T', \varphi', \psi', \rho'), \gamma')$ be an element of $\mathcal{R}_{t'}$ such that for each $e \in P_{v,w}$, $i \notin \psi'(e)$, where $P_{v,w} = \{e \in E(T) \mid e \text{ lies on the path between } \varphi(v) \text{ and } \varphi(w)\}$. We construct $(\mathfrak{T} = (T, \varphi, \psi, \rho), \gamma)$ as an element of $\mathcal{R}_t$ as follows:

- $T = T'$,
- for every $v \in B_t$, $\varphi(v) = \varphi'(v)$,
- for every $e \in E(T) \setminus P_{v,w}$, $\psi(e) = \psi'(e)$,
- for every $e \in P_{v,w}$, $\psi(e) = \psi'(e) \cup \{i\}$, and
- for every $v \in V(T)$, $\gamma(v) = \gamma'(v)$.

The correctness of the algorithm can be proved analogously to the proof given in Appendix A. Finally, note that the analysis of the running time carried out in Appendix B also applies to this case, as the size of the objects stored in the tables is upper-bounded by the size of those used in the algorithm of Section 3. Furthermore, the performed operations incur the same time complexity, except for the case of an introduce-edge node, for which in the previous algorithm we looked for the existence of an appropriate *edge* in T, whereas in the current one we look for the existence of an appropriate *path* in T, which can be performed in time $O(|V(T)|)$. This additional running time is clearly dominated by the overall running time of the algorithm, namely $2^{O(k^2)} \cdot n$.