Embedding Phylogenetic Trees in Networks of Low Treewidth

Leo van Iersel
Delft Institute of Applied Mathematics, Delft University of Technology, The Netherlands

Mark Jones
Delft Institute of Applied Mathematics, Delft University of Technology, The Netherlands

Mathias Weller
CNRS, LIGM (UMR 8049), Champs-s/-Marne, France

Abstract

Given a rooted, binary phylogenetic network and a rooted, binary phylogenetic tree, can the tree be embedded into the network? This problem, called Tree Containment, arises when validating networks constructed by phylogenetic inference methods. We present the first algorithm for (rooted) Tree Containment using the treewidth t of the input network N as parameter, showing that the problem can be solved in $2^{O(t^2)} \cdot |N|$ time and space.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases fixed-parameter tractability, treewidth, phylogenetic tree, phylogenetic network, display graph, tree containment, embedding

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.69

Related Version *Full Version*: https://arxiv.org/abs/2207.00574v2 [40]

Funding *Leo van Iersel*: Partially funded by Netherlands Organization for Scientific Research (NWO) Vidi grant 639.072.602 and KLEIN grant OCENW.KLEIN.125.
Mark Jones: Partially funded by Netherlands Organization for Scientific Research (NWO) KLEIN grant OCENW.KLEIN.125.

Acknowledgements We are extremely grateful to the anonymous reviewers for their many insightful and helpful comments.

1 Introduction

1.1 Background: phylogenetic trees and networks

Phylogenetic trees and networks are graphs used to represent evolutionary relationships. In particular, a *rooted phylogenetic network* is a directed acyclic graph with distinctly labelled leaves, a unique root and no indegree-1 outdegree-1 vertices. The labels of the leaves can, for example, represent a collection of studied biological species, and the network then describes how they evolved from a common ancestor (the root). Here, we will only consider rooted binary phylogenetic networks, which we will call *networks* for short. Vertices with indegree 2 in such a network are called *reticulations* and represent events where lineages combine, for example the emergence of new hybrid species. A network without reticulations is a *phylogenetic tree*.

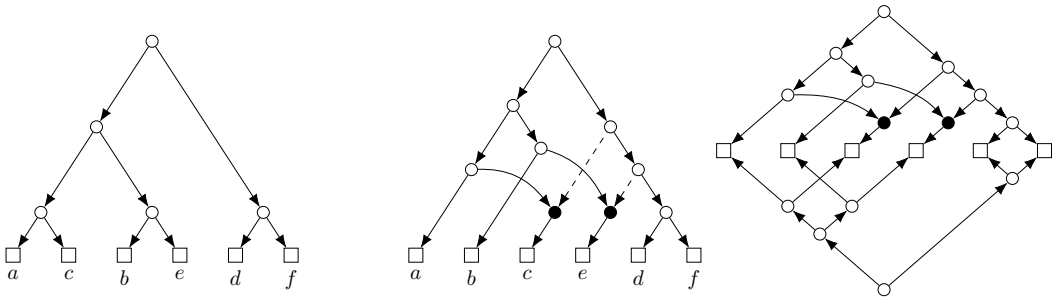

Figure 1 Left: a phylogenetic tree T. **Middle**: a phylogenetic network N displaying T (solid lines indicate an embedding of T; black nodes indicate reticulations). **Right**: the display graph $D(N,T)$ of N and T (see Section 1.5) with the network part drawn on top and the tree part drawn on the bottom. Note that vertices of the display graph are not labelled. In the figure, the leaves (square vertices) are ordered in the same way as in N.

1.2 The Tree Containment problem

The evolutionary history of a small unit of hereditary information (for example a gene, a fraction of a gene or (in linguistics) a word) can often be described by a phylogenetic tree. This is because at each reticulation, each unit is inherited from only one parent. Hence, if we trace back the evolutionary history of such a hereditary unit in the network, we see that its phylogenetic tree can be embedded in the network. This raises the fundamental question: given a phylogenetic network and a phylogenetic tree, can the tree be embedded into the network? This is called the Tree Containment problem (see Figure 1). To formalize this problem, we say that a network N *displays* a tree T if some subgraph of N is a subdivision of T.

Tree Containment (TC)

Input: phylogenetic network N_{IN} and tree T_{IN}, both on the same set of leaf labels
Question: Does N_{IN} display T_{IN}?

1.2.1 Motivation

Apart from being a natural and perhaps one of the most fundamental questions regarding phylogenetic networks, the Tree Containment problem has direct applications in phylogenetics. The main application is the validation of phylogenetic network inference methods. After constructing a network, one may want to verify whether it is consistent with the phylogenetic trees. For example, if a heuristic method is used to generate a network for a genomic data set, and tree inference methods are used to generate trees for each gene, then the quality of the produced network can be assessed by computing the fraction of the gene trees that can be embedded into it. In addition, one may want to find the actual embeddings for visualisation purposes and/or to assess the importance of each network arc.

However, our main motivation for studying Tree Containment is that it is a first step towards the wider application of treewidth based approaches in phylogenetics (see Sections 1.4 and 1.5). The techniques we develop are not exclusively designed for Tree Containment but intended to be useful also for other problems such as Network Containment [31] and Hybridization Number [8, 41, 42, 43]. The former is the natural generalization of Tree Containment in which we have two networks as input and want to decide whether one can be embedded into the other. It can, in particular, be used to decide whether two networks

are isomorphic. In the latter problem, Hybridization Number, the input consists of a set of phylogenetic trees, and the aim is to construct a network with at most k reticulations that embeds each of the input trees. Although this will certainly be non-trivial, we expect that at least part of the approach we introduce here can be applied to those and other problems in phylogenetics.

1.3 Previous work

Tree Containment was shown to be NP-hard [32], even for tree-sibling, time-consistent, regular networks [29]. On the positive side, polynomial-time algorithms were found for other restricted classes, including tree-child networks [17, 18, 22, 24, 29, 44]. The first non-trivial FPT algorithm for Tree Containment on general networks had running time $O(2^{k/2}n^2)$, where the parameter k is the number of reticulations in the network [32]. Another algorithm was proposed by [23] with the same parameter, but it is only shown to be FPT for a restricted class of networks. Since the problem can be split into independent subproblems at non-leaf cut-edges [29], the parameterization can be improved to the largest number of reticulations in any biconnected component (block), also called the *level* of the network. Further improving the parameterization, the maximum number t^* of "unstable component-roots" per biconnected component was considered and an algorithm (working also in the non-binary case) was found with running time $O(3^{t^*}|N||T|)$ [44]. Herein, a parameterization "improves" over another if the first is provably smaller than (a function of) the second in any input network.

Several generalizations and variants of the Tree Containment problem have been studied. The more general Network Containment problem asks to embed a *network* in another and has been shown to be solvable in polynomial time on a restricted network class [31]. When allowing multifurcations and non-binary reticulations, two variants of Tree Containment have been considered: In the firm version, each non-binary node ("polytomy") of the tree has to be embedded in a polytomy of the network whereas, in the soft version, polytomies may be "resolved" into binary subtrees in any way [2]. Finally, the unrooted version of Tree Containment was also shown to be NP-hard but fixed-parameter tractable when the parameter is the reticulation number (the number of edges that need to be deleted from the network to obtain a tree) [28]. While this version of the problem is also known to be fixed-parameter tractable with respect to the treewidth of the network [30], the work does not explicitly describe an algorithm and the implied running time depends on Courcelle's theorem [10] which makes practical implementation virtually impossible.

Since the notion of "display" closely resembles that of "topological minor" (with the added constraint that the embedding must respect leaf-labels), Tree Containment can be understood as a special case of a variant of the well-known Topological Minor Containment (TMC) problem for directed graphs. TMC is known to be NP-complete in general by reduction from Hamiltonian Cycle and previous algorithmic results focus on the undirected variant, parameterized by the *size* h of the sought topological minor H (corresponding to the input tree for Tree Containment). In particular, undirected TMC can be solved in $f(h)n^{O(1)}$ time [21, 15]. In the directed case, even the definition of "topological minor" has been contested [19] and we are aware of little to no algorithmic results. In Tree Containment, part of the embedding of the host tree in the guest network is fixed by the leaf-labeling. If the node-mapping is fixed for *all* nodes of the host, then directed TMC generalizes the Disjoint Paths problem [16], which is NP-complete for 2 paths or in case the host network is acyclic. Indeed, one can show Tree Containment to be NP-hard in a similar fashion [32].

1.4 Treewidth

From the overview presented in Section 1.3, we see that the parameters that are most heavily used are the reticulation number and the level. This is true not only for Tree Containment but more generally in the phylogenetic networks literature. Although these parameters are natural, their downside is that they are not necessarily much smaller than the input size. This is why we study a different parameter here.

The *treewidth* of a graph measures its tree-likeness (see definition below), similarly to the reticulation number and level. In that sense, it is also a natural parameter to consider in phylogenetics, where networks are often expected to be reasonably tree-like. A major advantage of treewidth is that it is expected to be much smaller than the reticulation number and level. In particular, there exist classes of networks for which the treewidth is at most a constant factor times the square-root of the level (see [33] for an example). Moreover, a broad range of advanced techniques have been developed for designing FPT algorithms for graph problems when the parameter is the treewidth [4, 12, 6, 13]. For these reasons, the treewidth has recently been studied for phylogenetics problems [30, 34, 33, 38] and related width parameters have been proposed [3]. However, using treewidth as parameter for phylogenetic problems poses major challenges and, therefore, there are still few algorithms in phylogenetics that use treewidth as parameter (see Section 1.5).

It will be convenient to define a *tree decomposition* of a graph $G = (V, E)$ as a rooted tree, where each vertex of the tree is called a *bag* and is assigned a partition (P, S, F) of V, where S is a separator between P and F. We will refer to S as the *present* of the bag. The set P is equal to the union of the presents of all descendant bags (minus the elements of S) and we refer to it as the *past* of the bag. The set $F = V \setminus (S \cup P)$ is referred to as the *future* of the bag. For each edge of the graph, there is at least one bag for which both endpoints of the edge are in the present of the bag. Finally, for each $v \in V$, the bags that have v in the present form a non-empty connected subtree of the tree decomposition. The *width* of a tree decomposition is one less than the maximum size of any bag's present and the *treewidth* $tw(G)$ of a graph G is the minimum width of any tree decomposition of G. The treewidth of a phylogenetic network or other directed graph is the treewidth of the underlying undirected graph.

Our dynamic programming works with *nice* tree decompositions, in which the root is assigned $(V, \varnothing, \varnothing)$ and each bag assigned (P, S, F) has exactly one of four types: *Leaf* bags have $P = S = \varnothing$ (hence $F = V$) and have no child, *Introduce* bags have a single child assigned $(P, S \setminus \{z\}, F \cup \{z\})$ for some $z \in S$, *Forget* bags have a single child assigned $(P \setminus \{z\}, S \cup \{z\}, F)$ for some $z \in P$, and *Join* bags have two children assigned $(L, S, F \cup R)$ and $(R, S, F \cup L)$ respectively, where (L, R) is a partition of P. When the treewidth is bounded by a constant, [5] showed that a minimum-width tree decomposition can be found in linear time and [36] showed that a nice tree decomposition of the same width can be obtained in linear time. Regarding approximation, it is known that, for all graphs G, tree decompositions of width $O(tw(G))$ can be computed in time single-exponential in $tw(G)$ [11, 7, 37] and tree decompositions of width $O(tw(G)\sqrt{\log tw(G)})$ can be computed in polynomial time [14].

1.5 Challenges

One of the main challenges of using treewidth as parameter in phylogenetics is that the central goal in this field is to infer phylogenetic networks and, thus, the network is not known *a priori* so a tree decomposition cannot be constructed easily. A possible strategy to overcome this problem is to work with the *display graph* (see Figure 1). Consider a problem

taking as input a set of trees, such as Hybridization Number. Then, the *display graph* of the trees is obtained by taking all trees and identifying leaves with the same label. Now we have a graph in the input and hence we can compute a tree decomposition. Moreover, in some cases, there is a strong relation between the treewidth of the display graph and the treewidth of an optimal network [20, 33, 30].

A few instances of exploiting (tree decompositions of) the display graph of input networks for algorithm design have been published. Famously, Bryant and Lagergren [9] designed MSOL formulations solving the Tree Consistency problem on display graphs, which have been improved[1] by a concrete dynamic programming on a given tree decomposition of the display graph [1]. Kelk et al. [34] also developed MSOL formulations on display graphs for multiple incongruence measures on trees, based on so-called "agreement forests". For the Tree Containment problem, MSOL formulations acting on the display graph have been used to prove fixed-parameter tractability with respect to the treewidth [30]. Analogously to the work of Baste et al. [1] for Tree Consistency, we develop in this manuscript a concrete dynamic programming algorithm for Tree Containment acting on display graphs.

Tree Containment is conceptually similar to Hybridization Number in the sense that the main challenge is to decide which tree vertices correspond to which vertices of the other trees (for Hybridization Number) or network vertices (for Tree Containment). However, Hybridization Number is even more challenging since the network may contain vertices that do not correspond to any input vertex [41]. Therefore, Tree Containment is a natural first problem to develop techniques for, aiming at extending them to Hybridization Number and other problems in phylogenetics in the long run.

That being said, solving Tree Containment parameterized by treewidth poses major challenges itself. Even though the general idea of dynamic programming on a tree decomposition is clear, its concrete use for Tree Containment is severely complicated by the fact that the tree decomposition does not know the correspondence between tree vertices and network vertices. For example, when considering a certain bag of the tree decomposition, a tree vertex that is in the present of that bag may have to be embedded into a network vertex that is in the past or in the future. It may also be necessary to map vertices from the future of the tree to the past of the network and vice versa. Therefore, it will not be possible to "forget the past" and "not worry about the future". In particular, this makes it much more challenging to bound the number of possible assignments for a given bag. We will do this by bounding the number of "time-travelling" vertices by a function of the treewidth. We will describe these challenges in more detail in Section 2.2.

1.6 Our contribution

In this paper, we present an FPT-algorithm for Tree Containment parameterized by the treewidth of the input network. Our algorithm is one of the first (constructive) FPT-algorithms for a problem in phylogenetics parameterized by treewidth. We believe that this is an important development as the treewidth can be much smaller than other parameters such as reticulation number and level which are easier to work with. We see this algorithm as an important step towards the wide application of treewidth-based methods in phylogenetics.

[1] Commonly, MSOL formulations are used to classify problems as FPT but are considered impractical since the resulting running times are dominated by a tower of exponentials of size bounded in the treewidth.

2 Preliminaries

2.1 Reformulating the problem

A key concept throughout this paper will be *display graphs* [9], which are the graphs formed from the union of a phylogenetic tree and a phylogenetic network by identifying leaves with the same labels. Throughout this paper we will let N_{IN} and T_{IN} denote the respective input network and tree in our instance of Tree Containment. The main object of study will be the "display graph" of N_{IN} and T_{IN}. For the purposes of our dynamic programming algorithm, we will often consider graphs that are not exactly this display graph, but may be thought of as roughly corresponding to subgraphs of it (though they are not exactly subgraphs; see [40, Appendix A.1]). In order to incorporate such graphs as well, we will define display graphs in a slightly more general way than that usually found in the literature. In particular, we allow for the two "sides" of a display graph to be disconnected, and for some leaves to belong to one side but not the other.

▶ **Definition 1** (display graph). *A* display graph *is a directed acyclic graph $D = (V, A)$, with specified subsets $V_T, V_N \subseteq V$ such that $V_T \cup V_N = V$, satisfying the following properties:*

- *The graph $T := D[V_T]$ is an out-forest;*
- *Every vertex has in- and out-degree at most 2 and total degree at most 3;*
- *Any vertex in $V_N \cap V_T$ has out-degree 0 and in-degree at most 1 in both T and $N := D[V_N]$.*

Herein, we call T the tree side *and N the* network side *of D and we will use the term $D(N, T)$ to denote a display graph with network side N and tree side T.*

Given a phylogenetic network N_{IN} and phylogenetic tree T_{IN} with the same leaf-label set, we define $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$ to be the display graph formed by taking the disjoint union of N_{IN} and T_{IN} and identifying pairs of leaves that have the same label. We note that, while the leaves of N_{IN} and T_{IN} were originally labelled, this labelling does not appear in $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$. Labels were used to construct $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$, but in the rest of the paper we will not need to consider them. Indeed, such labels are relevant to the Tree Containment problem only insofar as they establish a relation between the leaves of T_{IN} and N_{IN}, and this relation is now captured by the structure of $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$.

We now reformulate the Tree Containment problem in terms of an *embedding function* on a display graph. Unlike the standard definition of an embedding function (see, e.g., [29]), which is defined for a phylogenetic network N and tree T, our definition of an embedding function applies directly to the display graph $D(N, T)$. Because of our more general definition of display graphs, our definition of an embedding function will also be more general than that found in the literature. The key idea of an embedding function remains the same, however: it shows how a subdivision of T may be viewed as a subgraph of N.

▶ **Definition 2** (embedding function). *Let D be a display graph with network side N and tree side T, and let $\mathcal{P}(N)$ denote the set of all directed paths in N. An* embedding function *on D is a function $\phi : V(T) \cup A(T) \to V(N) \cup \mathcal{P}(N)$ such that:*

- **(a)** *for each $u \in V(T)$, $\phi(u) \in V(N)$ and, for each arc $uv \in A(T)$, $\phi(uv)$ is a directed $\phi(u)$-$\phi(v)$-path in N;*
- **(b)** *for any distinct $u, v \in V(T)$, $\phi(u) \neq \phi(v)$;*
- **(c)** *for any $u \in V(T) \cap V(N)$, $\phi(u) = u$;*
- **(d)** *the paths $\{\phi(uv) \mid uv \in A(T)\}$ are arc-disjoint;*
- **(e)** *for any distinct $p, q \in A(T)$, $\phi(p)$ and $\phi(q)$ share a vertex z only if p and q share a vertex w with $z = \phi(w)$;*

Note that the standard definition of an embedding of a phylogenetic tree T into a phylogenetic network N (see e.g. [29]) coincides with the definition of an embedding function on $D(N,T)$. Property (e) ensures that, while the embeddings of arcs uv, vw_1, vw_2 can all meet at $\phi(v)$, the embeddings of different tree arcs cannot otherwise meet. (In particular, the path $\phi(uv)$ cannot end at a reticulation that is also an internal vertex of $\phi(u'v')$, something that is otherwise allowed by properties (a)–(d).)

▶ **Lemma 3.** *A phylogenetic network N displays a phylogenetic tree T if and only if there is an embedding function on $D(N,T)$.*

In light of Lemma 3, we may henceforth view Tree Containment as the following problem:

Tree Containment (TC)
Input: phylogenetic network N_{IN} and phylogenetic tree T_{IN} with the same leaf-label set
Task: Find an embedding function on $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$.

2.2 Overview of our approach

We study Tree Containment parameterized by the treewidth of the input network N_{IN}. A key tool will be the following theorem.

▶ **Theorem 4** ([30]). *Let N and T be an unrooted binary phylogenetic network and tree, respectively, with the same leaf-label set. If N displays T then $tw(D(N,T)) \leq 2tw(N)+1$.*

By Theorem 4, we suppose that the display graph $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$ has treewidth at most $2k+1$, where k is the treewidth of the underlying undirected graph N of N_{IN} as, otherwise, N does not display the unrooted version T of T_{IN}, implying that N_{IN} does not display T_{IN}.

As is often the case for treewidth parameterizations, we will proceed via a dynamic programming on a tree decomposition, in this case a tree decomposition of $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$. Recall that we view a bag (P,S,F) in the tree decomposition as partitioning the vertices of $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$ into past, present and future. A typical dynamic programming approach is to store, for each bag, some set of information about the present, while forgetting most information about the past, and not yet caring about what happens in the future. The resulting information is stored in a "signature", and the algorithm works by calculating which signatures are possible on each bag, in a bottom-up manner. This approach is complicated by the fact that the sought-for embedding of T_{IN} into N_{IN} may not map the past/present/future of T_{IN} into the past/present/future (respectively) of N_{IN}. Vertices from the past of T_{IN} may be embedded in the future of N_{IN}, or vice versa. Thus, we have to store more information than we might at first think. In particular, it is not enough to store information about which present vertices of T_{IN} are embedded in which present vertices of N_{IN} (indeed, depending on the bag, it may be that none of them are). As such, our notion of a "signature" has to track how vertices from the past of T_{IN} are embedded in the present and future of N_{IN}, and which vertices from the past of N_{IN} contain vertices from the present or future of T_{IN}. Vertices of the past which are mapped to vertices of the past, on the other hand, can mostly be forgotten about.

2.3 An informal guide to (compact) signatures

Roughly speaking, a *signature* σ for a bag (P,S,F) in the tree decomposition of $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$ consists of the following items (see Figure 3 for an example):

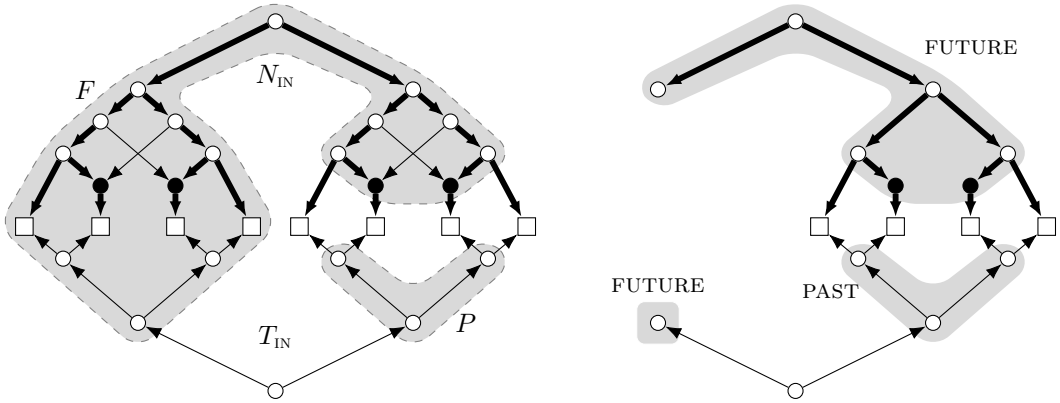

Figure 2 Left: An example of a display graph $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$ for which N_{IN} displays T_{IN} as witnessed by the embedding function ϕ that is indicated by bold edges. Highlighting with dashed border represents the sets P and F, for some bag (P, S, F) in a tree decomposition of $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$. **Right**: A representation of the (compact) signature for (P, S, F) derived from this solution. Vertices labelled PAST or FUTURE are highlighted in gray without border.

1. a display graph $D(N, T)$, some of whose vertices correspond (isomorphically) to $S \subseteq V(D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}}))$, and the rest of which are labeled PAST or FUTURE (which we may think of as vertices corresponding to some vertex of $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$ in P or F, respectively). We use a function ι on $V(D(N, T))$ to capture both this correspondence and labelling, where ι maps each vertex to an element of S or a label from $\{\text{PAST}, \text{FUTURE}\}$.
2. an embedding ϕ of T in N such that, for no arc uv, all of $V(\phi(uv)) \cup \{u, v\}$ have the same label $y \in \{\text{PAST}, \text{FUTURE}\}$ under ι.

Signatures may be seen as "partial embedding functions" on parts of $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$ in a straightforward way. In particular, we call σ *valid* for (P, S, F) if, roughly speaking, ϕ corresponds (via ι) to something that can be extended to an embedding function on the subgraph of $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$ induced by the vertices $P \cup S$ introduced below (P, S, F). In our dynamic programming algorithm, we build valid signatures for a bag x from valid signatures of the child bag(s) of x (in particular, validity for x is implied by validity for the child bag(s)).

Since iterating over all signatures for a bag (P, S, F) (in order to check their validity) exceeds FPT time, we will instead consider "compact" signatures, whose number and size are bounded in the width $|S|$ of the bag (P, S, F). If $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$ admits an embedding function ϕ^*, then a compact signature corresponding to this embedding function exists. In the following, we informally describe the compaction process for this hypothetical solution ϕ^*, thus giving a rough idea of the definition of a "compact" signature. At all times, the (tentative) signature will contain a display graph $D(N, T)$ (initially $D(N, T) = D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$), and an embedding function of T into N (initially ϕ^*). For a more complete description of our approach, see Appendix A in [40], for the proofs and remaining details, see Appendix B, and for an illustration, see Figure 2.

Step 1 After initiallization with ϕ^*, we assign a label FUTURE to all vertices of F, and a label PAST to all vertices in P (Observe that no vertex labeled PAST will be adjacent to a vertex labeled FUTURE, since S separates P from F in $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$). Then, we "forget" which vertices of $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$ the vertices labelled PAST or FUTURE correspond to. Our preliminary signature now contains (1) a display graph $D(N, T)$ whose vertices are either labelled FUTURE or PAST or correspond (isomorphically) to vertices in $S \subseteq V(D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}}))$ (we refer the reader to [40, Appendix A.1] for a more formal description), as well as (2) an embedding function for $D(N, T)$ into N.

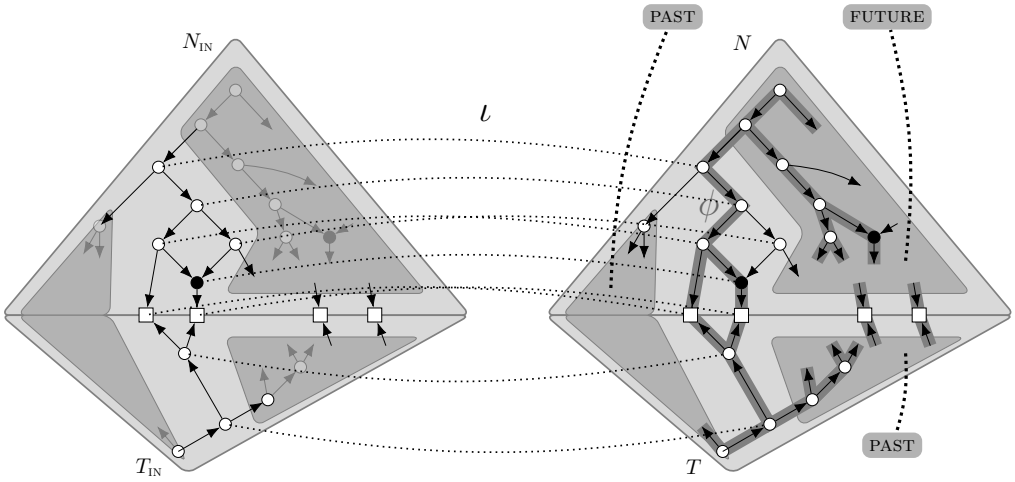

Figure 3 Example of a signature of a bag (P, S, F). The S-part of $D(N_{\text{IN}}, T_{\text{IN}})$ is solid while the non-S part is faded. The embedding ϕ (right, indicated with gray edge-highlight) maps T into N. The dotted arcs labelled ι show the isomorphism between part of $D(N, T)$ and $S \subseteq V(D(N_{\text{IN}}, T_{\text{IN}}))$. Note that the part of $D(N, T)$ that is not mapped to S is not necessarily isomorphic to anything in $D(N_{\text{IN}}, T_{\text{IN}})$.

Step 2 We now simplify the structure of the preliminary signature. The main idea is that, if a is an arc of T with both endpoints labelled PAST and all vertices in the path $\phi(a)$ are also labelled PAST, then we can safely forget a and all the arcs in $\phi(a)$. Intuitively, the information that a will be embedded in $\phi(a)$ does not have any effect on the possible solutions one could construct on the part of $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$ that is "above" the bag (P, S, F). Similarly, we can forget any arc a of T whose endpoints, as well as every vertex in $\phi(a)$, are assigned the label FUTURE. Intuitively, this is because this information should have no bearing on whether a solution exists with this signature for $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$ restricted to $P \cup S$. In a similar way, we forget any vertex $u \in V(T)$ and its embedding $\phi(u)$ if they are assigned the same label, provided that all their incident arcs can also be forgotten. We will call the vertices and arcs fulfilling these conditions "redundant" and we remove them from our tentative signature. We can also safely delete the vertices and arcs of N that are labelled $y \in \{\text{PAST}, \text{FUTURE}\}$ but are not part of the image of ϕ. As a result, we now have that for any remaining vertex $u \in T$, either one of $\{u, \phi(u)\}$ is labelled PAST and the other labelled FUTURE, or some vertex element of S must appear either one of $\{u, \phi(u)\}$, a neighbor of u, or a vertex in the path $\phi(a)$ for an incident arc a of u. Thus, we have "forgotten" all the aspects of the embedding except those that involve vertices from the present in some way, or those where the embedding "time-travels" between the past and future (see [40, Appendix A.3] for a more formal description of this process).

Step 3 Finally, we may end up with long paths of vertices with in-degree and out-degree 1 that are labelled PAST or FUTURE in N (for example, if u and v are labelled PAST, then $\phi(uv)$ may be a long path in N with all vertices labelled FUTURE). Such long paths do not contain any useful information to us, we therefore compress these by suppressing vertices with in-degree and out-degree 1 (This gives the *compact signature*, see [40, Appendix A.8]).

2.4 Bounding the number of signatures

We now outline the main arguments for why the number of possible (compact) signatures for a given bag (P, S, F) can be bounded in $|S|$. Such a bound on the number of signatures ensures that the running time of the algorithm is FPT, because the number of calculations required for each bag is bounded by a function of the treewidth.

The main challenge is to bound the size of the display graph $D(N, T)$ in a given signature for (P, S, F). Once such a bound is achieved, this immediately implies upper bounds (albeit quite large) for the number of possible display graphs and the number of possible embeddings, and hence on the number of possible signatures. We will focus here on bounding the size of the tree part T. Once a bound is found for $|T|$ it is relatively straightforward to use that to give a bound on $|N|$ (because the arcs of N that are not used by the embedding of T into N are automatically deleted, unless they are themselves incident to a vertex in S, and because isolated vertices are deleted and long paths suppressed).

It can be seen that a vertex $u \in V(T)$ is redundant (and so would be deleted from the signature) unless one of the following properties holds:

(1) $u \in S$,
(2) $\phi(u) \in S$,
(3) u is incident to an element of S
(4) for some arc a incident to u, the path $\phi(a)$ contains a vertex from S or
(5) u and $\phi(u)$ have different labels from $\{\text{PAST}, \text{FUTURE}\}$.

Essentially if none of (1)–(4) holds, then all the vertices mentioned in those properties have the same label as either u or $\phi(u)$, using the fact that S separates the vertices labelled PAST from the vertices labelled FUTURE. If u and $\phi(u)$ have the same label, then all these vertices have the same label, which is enough to show that u is redundant. It remains to bound the number of vertices satisfying one of these properties. For the first four properties, it is straightforward to find a bound in terms of $|S|$. The vertices satisfying the final property are "time-travelling" (in the sense that either u is labelled PAST and $\phi(u)$ FUTURE, or u is labelled FUTURE and $\phi(u)$ PAST). Because of the bounds on the other types of vertices, it is sufficient to provide a bound on the number of *lowest* time-travelling vertices in T.

To see the intuition why this bound should hold: consider some full solution on the original input, i.e. an embedding function on $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$, and suppose $u \in V(T_{\text{IN}})$ is a lowest tree vertex for which $u \in P, \phi(u) \in F$ (thus in the corresponding signature, u has label PAST and $\phi(u)$ has label FUTURE). Let $x \in V(N_{\text{IN}}) \cap V(T_{\text{IN}})$ be some leaf descendant of u. Then there is path in T_{IN} from u to x, and a path in N_{IN} from $\phi(u)$ to $\phi(x) = x$. Thus $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$ has an (undirected) path from u to $\phi(u)$. As this is a path between a vertex in P and a vertex in F, some vertex on this path must be in S (since S separates P from F). Such a path must exist for every lowest time-travelling vertex u, and these paths are distinct. The existence of these paths can then be used to bound the number of lowest time-travelling vertices.

3 Algorithm and running time

The final algorithm first computes (or constant-factor approximates) the treewidth of the display graph $D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})$ and concludes non-containment if this computation already implies $tw(D_{\text{IN}}(N_{\text{IN}}, T_{\text{IN}})) > 2tw(N_{\text{IN}}) + 1$. Otherwise, we proceed with a bottom-up dynamic programming on a nice low-width tree-decomposition, which computes for each bag $x = (P, S, F)$ a set CV_x of "compact-valid signatures" for x (see Section 2.3 for a rough definition). We use "compact-restrictions" to convert a compact signature of one bag into a compact

signature for a different bag. Basically, such a restriction works by mapping certain vertices to a label PAST or FUTURE, removing redundant parts of the display graph and collapsing long paths (see Section 2.3; also see [40, Appendix A.3] for the formal definition).

Leaf bag. If x is a leaf bag, then $P = S = \varnothing$ and all compact signatures $\sigma = (D(N,T), \phi, \iota)$ for x with $\iota^{-1}(\text{PAST}) = \varnothing$ are valid for x (and, thus, included in CV_x).

Introduce-z bag. If x is an introduce bag with child $y = (P, S \setminus \{z\}, F \cup \{z\})$ in $\mathcal{T}$, then all compact signatures σ for x whose compact-$\{z \to \text{FUTURE}\}$-restriction is valid for y (that is, contained in CV_y) are valid for x.

Forget-z bag. If x is a forget bag with child $y = (P \setminus \{z\}, S \cup \{z\}, F)$ in $\mathcal{T}$, then all compact-$\{z \to \text{PAST}\}$-restrictions of compact-valid signatures σ for y are valid for x.

Join bag. For join bags, we use "reconciliations" which are, basically, 3-way analogues of signatures, using labels $\{\text{LEFT}, \text{RIGHT}, \text{FUTURE}\}$ instead of $\{\text{PAST}, \text{FUTURE}\}$ (see [40, Appendix A.7]). In particular, if x is a join bag with children $y_L = (L, S, R \cup F)$ and $y_R = (R, S, L \cup F)$, then we compute all compact reconciliations μ for x and check whether

- the compact-$\{\text{LEFT} \to \text{PAST}, \text{RIGHT} \to \text{FUTURE}\}$-restriction of μ is valid for y_L and
- the compact-$\{\text{RIGHT} \to \text{PAST}, \text{LEFT} \to \text{FUTURE}\}$-restriction of μ is valid for y_R.

Then, the compact-$\{\{\text{LEFT}, \text{RIGHT}\} \to \text{PAST}\}$-restriction of each μ verifying these conditions is valid for x.

Correctness

The correctness of the computation of the sets CV_x follows from [40, Lemmas 15–17 and 19]. After having computed CV_x for the root bag r of the decomposition, we conclude that N_{IN} displays T_{IN} if and only if there is a compact-valid signature $(D(N,T), \phi, \iota)$ for the root bag with $\iota^{-1}(\text{FUTURE}) = \varnothing$. The correctness of this follows from [40, Lemma 10].

Running Time

To show that the running time is bounded in a function in the treewidth of N, the main challenge is to bound the number of compact signatures for a bag (P, S, F) by a function of $|S|$ (which, by Theorem 4, we may assume is at most $2tw(N) + 1$). In order to do this, we first bound the size of the display graph $D(N,T)$ in a signature by a function of $|S|$, straightforwardly implying bounds on the number of possible display graphs, embedding functions and isolabellings. Full details are given in [40]; to give a flavor of the proofs, we present the argument bounding the number of arcs in T.

▶ **Lemma 5.** *Any compact signature $(D(N,T), \phi, \iota)$ for a bag (P, S, F) has $|A(T)| \leq 6|S|$.*

Proof. Let A_S contain all arcs uv of $D(N,T)$ with $\iota(u) \in S$ or $\iota(v) \in S$. As there is only one vertex u with $\iota(u) = s$ for each $s \in S$ and every vertex in $D(N,T)$ has total degree at most 3, we have that $|A_S| \leq 3|S|$. As $\phi(uv)$ and $\phi(u'v')$ are arc-disjoint for any distinct tree arcs uv and $u'v'$, there are at most $|A_S \cap A(N)|$ arcs uv of T for which $\phi(uv)$ contains an arc in A_S. Further, at most $|A_S \cap A(T)|$ arcs in T are incident with a vertex in $\iota^{-1}(S)$. Thus, there are at most $|A_S|$ arcs uv in T for which $\{u, v\} \cup V(\phi(uv))$ contains a vertex of $\iota^{-1}(S)$.

Every remaining arc uv in T has $\iota(\{u, v\} \cup V(\phi(uv))) \subseteq \{\text{PAST}, \text{FUTURE}\}$. Further, we may assume the signature to be "well-behaved" (see [40, Sections A.4 and B.4]), which implies among other things that vertices mapped to PAST and FUTURE cannot be adjacent in $D(N,T)$, and that no arcs or vertices are redundant. Then we have $\iota(u) = \iota(v)$ and $\iota(u') = \iota(v')$ for every arc $u'v'$ in the path $\phi(uv)$. Then, for all but at most $|A_S| \leq 3|S|$ arcs uv of T, we have $\iota(u) = \iota(v) \in \{\text{PAST}, \text{FUTURE}\}$ as well as one of $\iota(V(\phi(uv))) = \{\text{PAST}\}$

and $\iota(V(\phi(uv))) = \{\text{FUTURE}\}$. If $\iota(u) = \iota(v) = \text{PAST}$ and $\iota(V(\phi(uv))) = \{\text{PAST}\}$, then uv is redundant w.r.t. $\{\text{PAST}\}$, a contradiction, and, similarly in case $\iota(u) = \iota(v) = \text{FUTURE}$ and $\iota(\phi(uv)) = \{\text{FUTURE}\}$. So, for all but at most $3|S|$ tree arcs uv, either $\iota(u) = \iota(v) = \text{PAST}$ and $\iota(\phi(uv)) = \{\text{FUTURE}\}$ or $\iota(u) = \iota(v) = \text{FUTURE}$ and $\iota(\phi(uv)) = \{\text{PAST}\}$. In particular, we have that $\iota(\phi(v)) \neq \iota(v)$, and for such vertices we may assume v has out-degree 2 (see Definition in [40, Appendix A.2]). Hence, any lowest arc uv in T is one of the at most $|A_S|$ many arcs uv for which $\{u, v\} \cup V(\phi(uv))$ contain a vertex of $\iota^{-1}(S)$. Thus, in total, T has at most $2|A_S| \leq 6|S|$ arcs. ◀

▶ **Theorem 6.** *Tree Containment can be solved in* $2^{O(tw(N_{\text{IN}})^2)} \cdot |A(N_{\text{IN}})|$ *time.*

4 Future work

Before implementing our dynamic programming algorithm, one should first try to reduce the constant in the bound on the number of possible signatures as much as possible. Such reductions may be possible for instance by imposing further structural constraints on the signatures that need to be considered. If this constant can be reduced, possibly including heuristic improvements, it would be interesting to implement the algorithm and test it on practical data.

From a theoretical point of view, there are many opportunities for future work. First, there are multiple variants and generalizations of Tree Containment that deserve attention: non-binary inputs, unrooted inputs and inputs consisting of two networks. Indeed, in order to decide if a network is contained in a second network, our approach would have to be extensively modified, since our size-bound on the signatures heavily relies on T_{IN} being a tree.

Second, a major open problem is whether the Hybridization Number problem is FPT with respect to the treewidth of the output network. Again there are different variants: rooted and unrooted, binary and non-binary, a fixed or unbounded number of input trees. For some applications, the definition of an embedding has to be relaxed (allowing, for example, multiple tree arcs embedded into the same network arc) [26, 25]. Other interesting candidate problems for treewidth-based algorithms include phylogenetic network drawing [35], orienting phylogenetic networks [27] and phylogenetic tree inference with duplications [39].

Finally, we believe that the approach taken in this paper (applying dynammic programming techniques on a tree decomposition of single graph representing all the input data, with careful attention given to the interaction between past and future) could potentially have applications outside of phylogenetics, in any context where the input to a problem consists of two or more distinct partially-labelled graphs that need to be reconciled.

References

1 Julien Baste, Christophe Paul, Ignasi Sau, and Scornavacca Celine. Efficient FPT algorithms for (strict) compatibility of unrooted phylogenetic trees. *Bulletin of Mathematical Biology*, 79:920–938, 2017.

2 Matthias Bentert, Josef Malík, and Mathias Weller. Tree containment with soft polytomies. In *SWAT'18*, volume 101, pages 9–1, 2018.

3 Vincent Berry, Celine Scornavacca, and Mathias Weller. Scanning phylogenetic networks is NP-hard. In *SOFSEM'20*, pages 519–530. Springer, 2020.

4 Hans L Bodlaender. Dynamic programming on graphs with bounded treewidth. In *ICALP'88*, pages 105–118. Springer, 1988.

5 Hans L Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996.

6 Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inf. Comput.*, 243:86–111, 2015.

7 Hans L Bodlaender, Pål Grønås Drange, Markus S Dregi, Fedor V Fomin, Daniel Lokshtanov, and Michał Pilipczuk. A $c^k n$ 5-approximation algorithm for treewidth. *SIAM Journal on Computing*, 45(2):317–378, 2016.

8 Magnus Bordewich and Charles Semple. Computing the minimum number of hybridization events for a consistent evolutionary history. *Discrete Applied Mathematics*, 155(8):914–928, 2007.

9 David Bryant and Jens Lagergren. Compatibility of unrooted phylogenetic trees is FPT. *Theoretical Computer Science*, 351(3):296–302, 2006. Parameterized and Exact Computation.

10 Bruno Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In *Handbook Of Graph Grammars And Computing By Graph Transformation: Volume 1: Foundations*, pages 313–400. World Scientific, 1997.

11 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer, 2015.

12 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Joham MM van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *FOCS'11*, pages 150–159. IEEE, 2011.

13 Eduard Eiben, Robert Ganian, Thekla Hamm, and O-joung Kwon. Measuring what matters: A hybrid approach to dynamic programming with treewidth. *Journal of Computer and System Sciences*, 121:57–75, 2021.

14 Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM Journal on Computing*, 38(2):629–657, 2008.

15 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. *Hitting Topological Minors is FPT*, pages 1317–1326. Association for Computing Machinery, New York, NY, USA, 2020.

16 Steven Fortune, John Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111–121, 1980.

17 Philippe Gambette, Andreas D. M. Gunawan, Anthony Labarre, Stéphane Vialette, and Louxin Zhang. Solving the tree containment problem for genetically stable networks in quadratic time. In *IWOCA'15*, pages 197–208, 2015.

18 Philippe Gambette, Andreas DM Gunawan, Anthony Labarre, Stéphane Vialette, and Louxin Zhang. Solving the tree containment problem in linear time for nearly stable phylogenetic networks. *Discrete Applied Mathematics*, 246:62–79, 2018.

19 Robert Ganian, Petr Hliněný, Joachim Kneis, Daniel Meister, Jan Obdržálek, Peter Rossmanith, and Somnath Sikdar. Are there any good digraph width measures? *Journal of Combinatorial Theory, Series B*, 116:250–286, 2016.

20 A Grigoriev, S Kelk, and L Lekic. On low treewidth graphs and supertrees. *Journal of Graph Algorithms and Applications*, 19(1):325–343, 2015.

21 Martin Grohe, Ken-ichi Kawarabayashi, Dániel Marx, and Paul Wollan. Finding topological subgraphs is fixed-parameter tractable. In *STOC'11*, pages 479–488, 2011.

22 Andreas DM Gunawan. Solving the tree containment problem for reticulation-visible networks in linear time. In *AlCoB'18*, pages 24–36. Springer, 2018.

23 Andreas DM Gunawan, Bingxin Lu, and Louxin Zhang. A program for verification of phylogenetic network models. *Bioinformatics*, 32(17):i503–i510, 2016.

24 Andreas DM Gunawan, Hongwei Yan, and Louxin Zhang. Compression of phylogenetic networks and algorithm for the tree containment problem. *Journal of Computational Biology*, 26(3):285–294, 2019.

25 Katharina T Huber, Simone Linz, and Vincent Moulton. The rigid hybrid number for two phylogenetic trees. *Journal of Mathematical Biology*, 82(5):1–29, 2021.

26 Katharina T Huber, Vincent Moulton, Mike Steel, and Taoyang Wu. Folding and unfolding phylogenetic trees and networks. *Journal of Mathematical Biology*, 73(6):1761–1780, 2016.

27 Katharina T Huber, Leo van Iersel, Remie Janssen, Mark Jones, Vincent Moulton, Yukihiro Murakami, and Charles Semple. Rooting for phylogenetic networks. *arXiv preprint*, 2019. arXiv:1906.07430.

28 Leo van Iersel, Steven Kelk, Georgios Stamoulis, Leen Stougie, and Olivier Boes. On unrooted and root-uncertain variants of several well-known phylogenetic network problems. *Algorithmica*, 80(11):2993–3022, 2018.

29 Leo van Iersel, Charles Semple, and Mike Steel. Locating a tree in a phylogenetic network. *Information Processing Letters*, 110(23):1037–1043, 2010.

30 R. Janssen, M. Jones, S. Kelk, G. Stamoulis, and T. Wu. Treewidth of display graphs: bounds, brambles and applications. *Journal oof Graph Algorithms and Applications*, 23:715–743, 2019.

31 Remie Janssen and Yukihiro Murakami. On cherry-picking and network containment. *Theoretical Computer Science*, 856:121–150, 2021.

32 Iyad A Kanj, Luay Nakhleh, Cuong Than, and Ge Xia. Seeing the trees and their branches in the network is hard. *Theoretical Computer Science*, 401(1-3):153–164, 2008.

33 Steven Kelk, Georgios Stamoulis, and Taoyang Wu. Treewidth distance on phylogenetic trees. *Theoretical Computer Science*, 731:99–117, 2018.

34 Steven Kelk, Leo van Iersel, Celine Scornavacca, and Mathias Weller. Phylogenetic incongruence through the lens of monadic second order logic. *Journal of Graph Algorithms and Applications*, 20(2):189–215, 2016.

35 Jonathan Klawitter and Peter Stumpf. Drawing tree-based phylogenetic networks with minimum number of crossings. *arXiv preprint*, 2020. arXiv:2008.08960.

36 Ton Kloks. *Treewidth: computations and approximations*. Springer, 1994.

37 Tuukka Korhonen. A single-exponential time 2-approximation algorithm for treewidth. In *FOCS'21*, pages 184–192, 2021.

38 Celine Scornavacca and Mathias Weller. Treewidth-based algorithms for the small parsimony problem on networks. In *WABI'21*, 2021.

39 Leo van Iersel, Remie Janssen, Mark Jones, Yukihiro Murakami, and Norbert Zeh. Polynomial-time algorithms for phylogenetic inference problems involving duplication and reticulation. *IEEE/ACM transactions on computational biology and bioinformatics*, 17(1):14–26, 2019.

40 Leo van Iersel, Mark Jones, and Mathias Weller. Embedding phylogenetic trees in networks of low treewidth. *arXiv preprint*, 2022. arXiv:2207.00574v2.

41 Leo van Iersel, Steven Kelk, Nela Lekic, Chris Whidden, and Norbert Zeh. Hybridization number on three rooted binary trees is ept. *SIAM Journal on Discrete Mathematics*, 30(3):1607–1631, 2016.

42 Leo van Iersel, Steven Kelk, and Celine Scornavacca. Kernelizations for the hybridization number problem on multiple nonbinary trees. *Journal of Computer and System Sciences*, 82(6):1075–1089, 2016.

43 Leo van Iersel and Simone Linz. A quadratic kernel for computing the hybridization number of multiple trees. *Information Processing Letters*, 113(9):318–323, 2013.

44 Mathias Weller. Linear-time tree containment in phylogenetic networks. In *RECOMB CG'18*, pages 309–323. Springer, 2018.