Linear-Time Tree Containment in Phylogenetic Networks

Mathias Weller

Linear-Time Tree Containment in Phylogenetic Networks

Mathias Weller[1]

[1]LIRMM, IBC, Montpellier, France

April 17, 2018

Abstract

We consider the NP-hard Tree Containment problem that has important applications in phylogenetics. The problem asks if a given single-rooted leaf-labeled network ("phylogenetic network") N contains a subdivision of a given leaf-labeled tree ("phylogenetic tree") T. We develop a fast algorithm for the case that N is a phylogenetic tree in which multiple leaves might share a label. Generalizing a previously known decomposition scheme lets us leverage this algorithm, yielding linear-time algorithms for so-called "reticulation visible" networks and "nearly stable" networks. While these are special classes of networks, they rank among the most general of the previously considered cases. We also present a dynamic programming algorithm that solves the general problem in $O(3^t \cdot |N| \cdot |T|)$ time, where the parameter t is the maximum number of tree components with unstable roots in any block of the input network. Notably, t is stronger (that is, smaller on all networks) than the previously considered parameter "number of reticulations" and even the popular parameter "level" of the input network.

1 Introduction

The quest to find the infamous "tree of life" has been popular in live sciences since the widespread adoption of evolution as the source of biodiversity on earth. With the discovery of DNA, the task of constructing a history of the evolution of a set of species has become both a blessing and a curse. A blessing because we no longer rely on phenotypical characteristics to distinguish between species and a curse because we are being overwhelmed with data that has to be cleaned, interpreted and visualized in order to draw conclusions. The use of DNA also gave strong support to the realization that trees are not always suited to display ancestral relations, as they fail to model recombination events such as hybridization (occurring frequently in plants) and horizontal gene transfer (a dominating factor in bacterial evolution) [5, 21]. Thus, researchers are more and more interested in evolutionary networks and algorithms dealing with them (see the monographs by Gusfield [16] and Huson et al. [17]).

The particular task that we consider in this work is to tell whether a given evolutionary network "displays" an evolutionary tree, that is, whether the tree-like information that we might have come to believe in the past is consistent with a proposed recombinant evolution. This problem is known as Tree Containment and it has been studied extensively. As it is NP-hard for general networks [18, 22], research focuses on moderately exponential time algorithms [13] and biologically relevant special cases of networks [3, 11, 12, 14, 18, 22]. Prominent among these special classes are the following:

- nearly-stable networks for which an $O(n^2)$-time algorithm is known [12]
- reticulation-visible networks for which $O(n^3)$-time [3, 14] and $O(n^2)$-time algorithms are known [14]. Early this year, a preprint claiming a linear-time algorithm for this type of networks was published [15].

Generalizing the decomposition of Gunawan et al. [14] for reticulation-visible networks to *general* networks, we show that Tree Containment can be solved in $O(|N| \cdot \overleftarrow{\Delta}_N^2 \cdot \vec{\Delta}_T^2)$ time[1] if each tree vertex with a reticulation parent is stable[2] on some leaf. This running time degenerates to linear time for binary N in which the length of a longest "reticulation chain" (directed path consisting only of reticulations) is constant. This latter class of networks comprises both reticulation visible and nearly stable networks and, therefore, subsumes previous work mentioned above. We culminate the ideas that lead to the linear-time algorithms to develop an $O((\vec{\Delta}_T + 1)^{t^*} \cdot (\vec{\Delta}_N + \vec{\Delta}_T^{2.5}) \cdot |V(N)| \cdot |V(T)|)$-time algorithm, where t^* is the maximum number of unstable tree components (see Definition 1) in any biconnected component[3] of N For bifurcating N, this degenerates to $O(3^{t^*} \cdot |V(N)| \cdot |V(T)|)$ time.

Preliminaries. Let N be a weakly connected, directed acyclic graph (DAG) with a single source $\rho(N)$ called the *root* and each of the sinks $\mathcal{L}(N)$ (called *leaves*) carries a label (its "taxon"). Then, we call N an evolutionary (or phylogenetic) *network* (or "network" for short). We call the vertices of in-degree at least two in N *reticulations* and all other vertices *tree vertices* and we demand that all reticulations have out-degree one. If N has no reticulations, then it is called a *tree*. We denote the number of arcs in N by $|N|$.

We denote the maximum in- and out-degree in N by $\overleftarrow{\Delta}_N$ and $\vec{\Delta}_N$, respectively. Then, we call N *forward-binary* (or *bifurcating*) if $\vec{\Delta}_N \leq 2$ and *binary* if also $\overleftarrow{\Delta}_N \leq 2$. If each label occurs $\leq k$ times in N, we call N *k-labeled* or, if k is unknown or inconsequential, *multi-labeled*. We define the

[1] Herein, $\vec{\Delta}_T$ is the maximum out-degree in T and $\overleftarrow{\Delta}_N$ is the maximum out-degree in the result of contracting all arcs between reticulations in N.

[2] u is stable on ℓ if all root-ℓ-paths contain u. The notion of stability is equivalent to the notion of "dominators" in directed graphs [19].

[3] A biconnected component (or "block") of a network is a subdigraph induced by the vertices of a biconnected component of its underlying undirected graph, that is, a connected component in the result of removing all bridges.

relation $\leq_N$ such that $u \leq_N v \iff u$ is a descendant of v (that is, v is an ancestor of u) in N. Note that $u \leq_N \rho(N)$ for all $u \in V(N)$. For each vertex v of N, we define N_v to be the *subnetwork rooted at* v, that is, the subnetwork of N that contains exactly the vertices u with $u \leq_N v$ and all arcs of N between those vertices. The subnetwork $N|_U$ of N *restricted* to a set U of vertices is the result of first removing all vertices v with $\forall_{u \in U} u \not\leq_N v$ and then contracting all arcs that are outgoing of vertices w with in-degree and out-degree at most one, unless $w \in U$. Note that the least common ancestor (LCA) of any two vertices of U is also in $N|_U$.

We call any vertex v of N *stable on another vertex* u if all $\rho(N)$-u-paths contain v and we call v *stable* if v is stable on a leaf of N. Then, N is called *reticulation visible* if each reticulation r is stable. Further, N is called *nearly stable* if, for each vertex v, either v or its parents are stable. For all k, a k-labeled network N is said to *contain* a tree T if T is a subgraph of N (respecting the leaf-labeling). Further, N is said to *display* T if N contains a subdivision of T (that is, the result of a series of arc-subdivisions in T). In this work, we consider the Tree Containment problem defined below.

We assume that each reticulation path in N is initially contracted to a single reticulation with possibly large in-degree. Clearly, this has no influence on whether or not N displays T.

Tree Containment (TC)
Input: a network N, a tree T
Question: Does N display T?

Assumption 1. *The children and parents of all reticulations are tree vertices.*

2 Multi-Labeled Tree Containment

The following is a simple dynamic programming deciding if a k-labeled tree $\hat{T}$ displays a tree T. To this end, we define a table with entries $[u, v]$ where $u \in V(\hat{T})$ and $v \in V(T)$ such that, for all computed entries $[u, v]$, we have

$$[u, v] = 1 \iff u \in \min_{\leq_{\hat{T}}} \{w \mid \hat{T}_w \text{ displays } T_v\}. \tag{1}$$

While the table $[u, v]$ might have $|T| \cdot |\hat{T}|$ cells, we will not compute all of them, but only those with $[u, v] = 1$ which we show to be at most $|\hat{T}| \cdot k$. We represent the table as a sparse set, allowing efficient enumeration, setting, and querying [4].

Assumption 2. *Given v, we can get $U := \{u \mid [u, v] = 1\}$ in $O(|U|)$ time.*

Note that, if T_v is displayed by subtrees $\hat{T}_u$ and $\hat{T}_w$ of $\hat{T}$ and u and w are incomparable wrt. $\leq_{\hat{T}}$, then for each leaf-label λ in T_v, each of $\hat{T}_u$ and $\hat{T}_w$ contains a different leaf labeled λ. Thus, there cannot be more than k such subtrees.

Observation 1. *For all v, we have $|\{u \mid [u, v] = 1\}| \leq k$.*

We compute the table in a bottom-up manner. If v is a leaf, then we find the $\leq k$ leaves of $\hat{T}$ with the same label as v and set $[u, v] = 1$ for them. If v has children $v_1, v_2, \ldots, v_d$, then we first compute $U := \bigcup_i \{w \mid [w, v_i] = 1\}$ and then compute the subtree $\hat{T}|_U$ of $\hat{T}$ that is restricted to U. Finally, we find the lowest vertices u of $\hat{T}|_U$ such that there is a matching M between the children of v in T and the children of u in $\hat{T}|_U$ such that each $M(v_i)$ has a descendant z_i in $\hat{T}|_U$ with $[z_i, v_i] = 1$. For all these u, we set $[u, v] = 1$.

Lemma 1. *The computation is correct, that is, (1) holds for all entries $[u, v]$.*

Proof. The proof is by induction on the height of v in T. If v is a leaf, then (1) clearly holds. Otherwise, let $v_1, v_2, \ldots, v_d$ be the children of v in T and let $U := \bigcup\{w \mid [w, v_i] = 1\}$.

"$\Leftarrow$": Let u be a lowest vertex in $\hat{T}$ such that $\hat{T}_u$ contains a subdivision S of T_v. Then, $\hat{T}_u$ contains lowest $z_1, z_2, \ldots, z_d$ such that S_{z_i} displays T_{v_i}. Suppose that S is chosen such as to maximize the sum of the distances between u and z_i. By minimality of u, we know that u is the LCA of the z_i in $\hat{T}$. Then, by induction hypothesis, $[z_i, v_i] = 1$ for all i, implying that $z_i \in U$ and, thus, $z_i \in V(\hat{T}|_U)$. Since u is the LCA of the z_i in $\hat{T}$, we also have $u \in V(\hat{T}|_U)$. Moreover, u has children $w_1, w_2, \ldots, w_d$ in $\hat{T}$ such that $z_i \leq_{\hat{T}} w_i$ for all i and, thus, u also has children $w'_1, w'_2, \ldots, w'_d$ in $\hat{T}|_U$ with $z_i \leq_{\hat{T}|_U} w'_i$ for all i. Hence, mapping v_i to w'_i for each i constitutes a matching M as demanded by the above construction, implying $[u, v] = 1$.

"$\Rightarrow$": Suppose that $[u, v] = 1$. By construction, u is a lowest vertex of $\hat{T}|_U$ for which there is a matching M between the children of v in T and the children of u in $\hat{T}|_U$ such that each $M(v_i)$ has a descendant z_i in $\hat{T}|_U$ with $[z_i, v_i] = 1$. By induction hypothesis, each z_i is a minimum wrt. $\leq_{\hat{T}}$ of $\{w \mid \hat{T}_w \text{ displays } T_{v_i}\}$. Thus, each T_{v_i} has a subdivision S_i in $\hat{T}_{z_i}$ and, since M is a matching, each z_i descends from a different child u_i of u in $\hat{T}$. Thus, the S_i together with the unique u-z_i-paths in $\hat{T}$ can be merged to form a subdivision of T_v contained in $\hat{T}_u$. Towards a contradiction, assume that u is not minimal wrt. $\leq_{\hat{T}}$ among such vertices, that is, there is a different lowest $u' <_{\hat{T}} u$ such that $\hat{T}_{u'}$ contains a subdivision S' of T_v. By the argument in the "$\Leftarrow$"-direction, there is a matching M' between the children of v in T and the children of u' in $\hat{T}|_U$ such that each $M'(v_i)$ has a descendant z'_i in $\hat{T}|_U$ with $[z'_i, v_i] = 1$. But then, u is not minimal among such vertices in $\hat{T}|_U$, and we would not have constructed $[u, v] = 1$. □

To show the running time, we assume that $\hat{T}$ is preprocessed to allow translating labels into leaves and leaves of $\hat{T}$ into leaves of T.

Assumption 3. *Given a label λ, we can get the list L of all leaves of $\hat{T}$ with label λ in $O(|L|)$ time. Given a leaf ℓ of $\hat{T}$, we can get the leaf of T with the same label in $O(1)$ time.*

We also assume that we can compute the LCA of two vertices in T or in $\hat{T}$ in constant time (see, for example [1]). This helps us compute the restriction of T and $\hat{T}$ to any ordered (left to right) list U in $O(|U|)$ time (see, for example [7, Section 8]).

Assumption 4. *Given vertices x and y in T or in $\hat{T}$, we can find $\mathrm{LCA}_T(xy)$ and $\mathrm{LCA}_{\hat{T}}(xy)$ in $O(1)$ time. Given an ordered list U, we can find $T|_U$ and $\hat{T}|_U$ in $O(|U|)$ time.*

Lemma 2. *Let $\hat{T}$ be k-labeled. Then, we can find the maximal (wrt. $\leq_T$) vertices v such that $\hat{T}$ displays T_v in $O(|\hat{T}| \cdot k^2 \vec{\Delta}_T^2)$ time.*

Proof. First, we get the set Y of leaves of T whose label occurs in $\hat{T}$ by scanning all leaves of $\hat{T}$ and translating these leaves to T using Assumption 3. This allows us to set $[u, v]$ for all leaves v of T in $O(|\hat{T}|)$ time. Furthermore, we can compute $T|_Y$ in $O(|Y|)$ time using Assumption 4.

Scanning $T|_Y$ in a bottom-up manner, we compute $[u, v]$ for each u and each v with children $v_1, v_2, \ldots, v_d$ as described. To this end, we construct $U_i := \{w \mid [w, v_i] = 1\}$ in $O(k)$ time by Assumption 2 and Observation 1 and $U := \bigcup_i U_i$ in $O(kd)$ time since $i \leq d$. Then, we construct $\hat{T}|_U$ in $O(|U|)$ time using Assumption 4. For each $x \in V(\hat{T}|_U)$, we then compute the set L_x of indices i such that there is some $w <_{\hat{T}|_U} x$ with $[w, v_i] = 1$. With a bottom-up dynamic programming in $\hat{T}|_U$, this can be done in $O(|\hat{T}|_U| \cdot d)$ time since $|L_x| \leq d$ for each x. Then, we produce a list C of all vertices $x \in V(\hat{T}|_U)$ with $L_x = \{1, 2, \ldots, d\}$. Since the subtrees of $\hat{T}$ rooted at each minimum wrt. $\leq_{\hat{T}|_U}$ of C are leaf-disjoint, we know that each such minimum has its own private descendant w with $[w, v_1] = 1$ and, thus, there are at most k such minima, implying $|C| \leq 2k - 1$.

For each vertex $u \in C$, we then construct a bipartite graph B whose two partitions are the children of u in $\hat{T}|_U$ and the children of v in T, respectively, and B contains an edge $\{x, v_i\}$ if and only if $i \in L_x$ (that is, x has a descendant w in $\hat{T}|_U$ with $[w, v_i] = 1$). If B has a size-d matching, we set $[u, v] = 1$. This can be done in $O(\sqrt{d} \cdot \min\{d^2, kd\}) \subseteq O(kd^{1.5})$ time [6] for each u. Note that no vertex $u \notin C$ can have such a matching and, thus, we set $[u, v] = 1$ correctly for all u and v.

Summing up the total time spent and noting that $|Y| \leq |\hat{T}|$, and $|U| \leq kd$, and $|\hat{T}|_U| \leq 2|U| - 1$, and $|C| \leq 2k - 1$, we arrive at a total running time of $O(|\hat{T}| \cdot (k\vec{\Delta}_T^2 + k^2\vec{\Delta}_T^{1.5}))$. Since our algorithm runs bottom-up in $T|_Y$, we can retain the highest v for which there is some w in $\hat{T}$ with $[w, v] = 1$ as claimed in the lemma. □

If we are only interested in whether or not $\hat{T}$ displays T, then we can prepend a size check and refuse the instance if $|T| > |\hat{T}|$. Thus, we can bound all preprocessing in $O(|\hat{T}|)$ time and Lemma 2 implies the following theorem.

Theorem 1. *Let $\hat{T}$ be a k-labeled tree and let T be a tree with maximum out-degree $\vec{\Delta}_T$. Then, we can decide if $\hat{T}$ displays T in $O(|\hat{T}| \cdot k^2\vec{\Delta}_T^2)$ time ($O(|\hat{T}| \cdot k^2)$ time if T is binary).*

3 Tree Containment in Special Networks

In this section, we move from multi-labeled trees to single-labeled networks, that is, in what follows, each label occurs exactly once (the leaf-labelling function is bijective).

Network Decomposition. Gunawan et al. [14] introduced a decomposition for reticulation-visible networks which we apply to arbitrary networks. To this end, we have to do some initial cleanup using the following reduction.

Rule 1. *Let ab be a* cherry *(that is, a pair of leaves sharing a common parent) in N. If ab is not a cherry in T, then reject (N, T) and, otherwise, delete a in both N and T and contract the arc incoming to b in N and in T.*

Definition 1 (See [14])**.** *Let N be reduced wrt. Rule 1 and let F be the forest that results from removing all reticulations from N. Then, each tree of F is called* tree component *of N. A tree component of N is called* trivial *if it contains only a leaf of N and* stable *if its root is stable. Let Γ be the set of roots of the non-trivial tree components of N. The restriction of "$\leq_N$" to Γ forms a DAG Q and we call it the* component DAG *of N. More formally, $Q := (\Gamma, (\leq_N) \cap (\Gamma \times \Gamma))$.*

The goal will be to repeatedly find a leaf γ of Q and the best possible v of T such that N_γ displays T_v. Then, we shrink both N_γ and T_v to a single leaf and remove γ from Q. We make use of the special structure of N_γ, implied by the fact that all tree nodes with a reticulation ancestor in N_γ are leaves of N (otherwise, they are in a tree component below γ, contradicting γ being a leaf).

Definition 2. *Let γ be a leaf of Q. Then, $P := N_\gamma$ consists of a tree with root $\rho(P) := \gamma$, some reticulations and some leaves of N. Further, P can be divided into "layers" (see Figure 1) and we call P a* pyramid *with a* tip P^Δ *(layer of tree vertices), a* base P^B *(layer of reticulations) and a* foundation P^F *(layer of leaves below reticulations).*

Algorithm. In this section, we show how Lemma 2 can be applied to pyramids. Given a pyramid P in N, our goal is to display as much of T as possible in P and reduce N and T using this information. To this end, we consider only the tip P^Δ of P and replace each arc xy from the tip to the base by an arc to a copy of the child ℓ of y. By Definition 2, ℓ is a leaf of N. Recall that $\overleftarrow{\Delta}_P$ is the maximum in-degree in P and $\vec{\Delta}_T$ is the maximum out-degree in T.

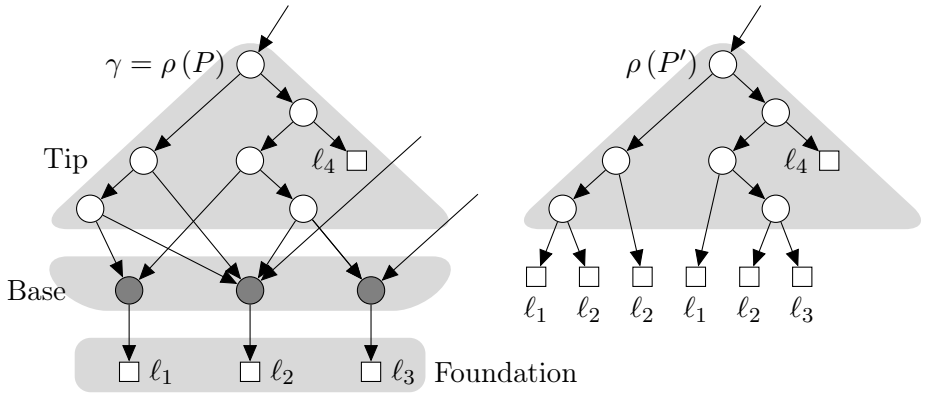

Figure 1: **Left:** A leaf γ of the component DAG Q of N implies a layering of the *pyramid* $P = N_\gamma$ into its *tip* P^Δ (tree nodes ○), its *base* P^B (reticulations ●), and its *foundation* P^F (leaves □ below reticulations). Note that leaves may also be in the tip of P. **Right:** The multi-labeled tree P' computed from the pyramid on the left in the proof of Lemma 3.

Lemma 3. *In $O(|P| \cdot \overleftarrow{\Delta}_P^2 \cdot \vec{\Delta}_T^2)$ time, we can find all maximal v (wrt. $\leq_T$) s.t. P displays T_v.*

Proof. Let P' denote the multi-labeled tree that results from P^Δ by, for each arc $xy \in V(P^\Delta) \times V(P^B)$, hanging a leaf onto x that is labeled with the same label as the unique child ℓ of y in P (see Figure 1). Note that P' is indeed $\overleftarrow{\Delta}_P$-labeled, its size is at most $|P|$, and it can be constructed in $O(|P|)$ time. Having constructed P', we compute the maximal (wrt. $\leq_T$) vertices v such that P' displays T_v. By Lemma 2, this can be done in $O(|P'| \cdot \overleftarrow{\Delta}_P^2 \cdot \vec{\Delta}_T^2)$ time. It remains to show for all v of T that P displays T_v if and only if P' does (see also [14]).

"⇒": Let P contain a subdivision S of T_v. Let S' result from S by contracting all arcs that are incoming to a vertex of the base P^B of P. Since S is a tree, all vertices of P^B have indegree one and outdegree one in S and, thus, S' is also a subdivision of T_v. To show that P' contains S', assume that S' contains an arc xy that is not in P'. If xy is in S, then neither x nor y is a reticulation in N, implying that xy is in P^Δ and, thus, in P'. Otherwise, S contains a path (x, r, y), where $r \in P^B$ and y is a (copy of a) leaf in the foundation of P. Then, xr is an arc in $V(P^\Delta) \times V(P^B)$, implying that P' contains a copy of y hanging from x.

"⇐": Let P' contain a subdivision S' of T_v. Let $x\ell$ be an arc of S' that is not in P. Then, ℓ is a leaf of P and its parent r is in P^B. Let S result from S' by replacing each such arc $x\ell$ by the path (x, r, ℓ). Clearly, S is a subdivision of S' and, thus, of T_v. To show that P contains S, it suffices to show that none of the new paths p introduces vertices that were already in S' or in any previously added path. For the first claim, note that all newly added vertices are in P^B and, thus, not in P'. For the second claim, note that each label of P' occurs at most once in S' and each vertex of P^B is parent of a unique leaf in P. Thus, P contains S and, therefore, P displays T_v. □

It is noteworthy that Lemma 3 might return many vertices v such that T_v is displayed by P and, without any more assumptions regarding N, the number of possible combinations grows exponentially. Thus, we restrict the class of networks that we are considering by demanding that each tree vertex of N that has a reticulation parent is stable. Hence, $\rho(P)$ is stable for all tree components P^Δ which form the tips of the pyramids P that we are seeing in the algorithm. In the following, let c denote the leaf that $\rho(P)$ is stable on and observe that the set of all vertices v such that P displays T_v and $c \leq_T v$ has a unique maximum wrt. $\leq_T$. Thus, at most one of the maxima obtained by Lemma 3 is an ancestor of c in T and we can find it in $O(|P|)$ time. We then apply the following reduction that places T_v into P and removes all arcs that disagree (see Figure 2).

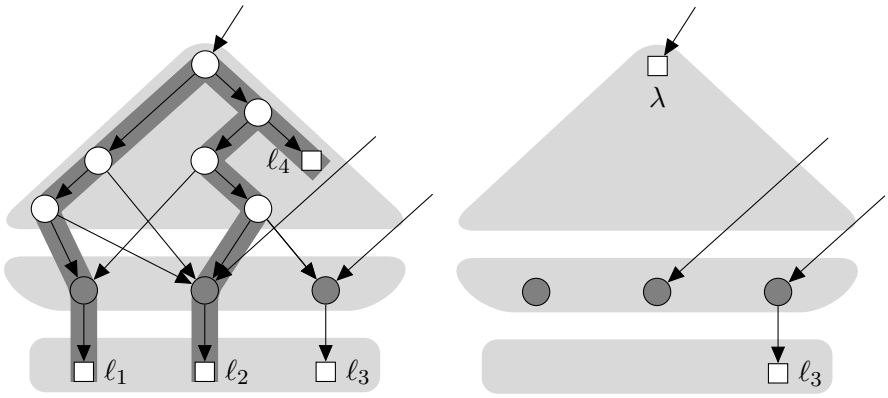

Figure 2: An example of an application of Rule 2 to the network N of Figure 1 with a subdivision of T_v shown in dark gray on the left.

Rule 2. *Let $\rho(P)$ be stable on a leaf c and let v be the unique maximum wrt. $\leq_T$ such that $c \leq_T v$ and P displays T_v. Then, remove all leaves of N whose label occurs in T_v, remove all vertices in the tip of P except $\rho(P)$, remove all arcs outgoing of $\rho(P)$, remove all vertices of T_v except v, and label v and $\rho(P)$ with the same new label λ.*

For correctness of Rule 2, see [14, Proposition 5] or our proof in the appendix. To apply Rule 2 in $O(|P|)$ time, we have to find a leaf c that $\rho(P)$ is stable on, in $O(|P|)$ time. This is easy if P^Δ contains a leaf of N. Otherwise, we mark all arcs between the tip and the base of P and check if any vertex in the base has all its incoming arcs marked. For a vertex r with m incoming marked arcs, this check can be done in $O(m)$ time. Thus, we can produce c in $O(|P|)$ time.

Observation 2. *We can produce a leaf c that $\rho(P)$ is stable on in $O(|P|)$ time.*

Further, note that Rule 2 might leave former reticulations as isolated vertices or pending leaves without label. Clearly, such a vertex is created by the deletion of an incoming or outgoing arc. To remove them, we mark such vertices as garbage upon removal of this incident arc in $O(1)$ time per removed arc. Then, we run a cleanup phase after Rule 2 that removes garbage in constant time per removed vertex, that is $O(|N|)$ overall.

Observation 3. *N does not contain isolated vertices or unlabeled leaves.*

Each time Rule 2 is applied to a leaf γ of the component DAG Q, it will replace the tip of N_γ by a single leaf in N. To keep Q up to date we just need to delete γ from Q at that point (since the tree component of γ is no longer non-trivial), but none of the other tree component roots are affected.

Observation 4. *We can produce a leaf of Q in constant time.*

The algorithm terminates when Rule 2 has been applied to the last pyramid of N and we return yes if and only if both $\rho(N)$ and $\rho(T)$ have the same label. By Lemma 3, the overall running time can be bounded by $O(\sum_i |P_i| \cdot \overleftarrow{\Delta}_N^2 \cdot \overrightarrow{\Delta}_T^2)$, where the summation is over all applications of Rule 2. Since no arc outgoing of P^Δ survives an application of Rule 2 to P, we conclude $\sum_i |P_i| \leq |N|$.

Theorem 2. *Let T be a tree with maximum out-degree $\overrightarrow{\Delta}_T$, let N be a network with maximum in-degree $\overleftarrow{\Delta}_N$ (after contraction of arcs whose both endpoints are reticulations) and let each tree vertex of N that has a reticulation parent be stable. Then, we can determine if N displays T in $O(|N| \cdot \overleftarrow{\Delta}_N^2 \cdot \overrightarrow{\Delta}_T^2)$ time.*

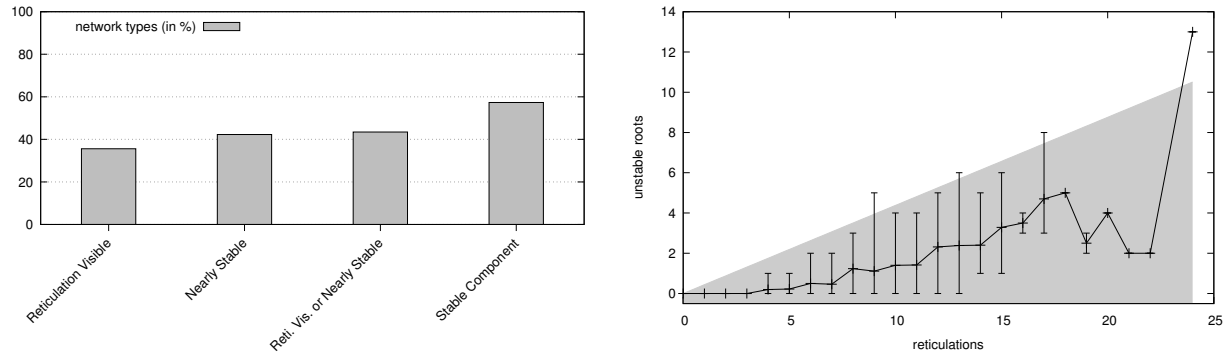

Figure 3: Comparison of 250 networks generated under the coalescent with recombination model (10 taxa, recombination rate 4, see [20]). **Left:** Percentages of network types. Here "Stable Component" refers to the condition that every tree component is stable. **Right:** Comparison of the parameter t to the number r of reticulations. For the gray area, we have $1.618^r > 3^t$. The refined parameter t^* relates similarly to the level.

Consider the special case that T and N are binary. If N is reticulation-visible, it already verifies Assumption 1, implying $\overleftarrow{\Delta}_N \leq 2$ and, as each reticulation is stable, each tree vertex with a reticulation parent is also stable. If N is nearly-stable, it cannot have reticulation paths of length 3, implying $\overleftarrow{\Delta}_N \leq 4$ after the contraction operation of Assumption 1 and, as each node is either stable or has a stable parent, each tree vertex with a reticulation parent is stable.

Corollary 1. *Let T be a binary tree and let N be forward-binary and reticulation-visible or nearly stable. Then, we can decide if N displays T in $O(|N|)$ time.*

See also Figure 3 for an estimation of the probability to encounter the three discussed types of networks when simulating recombinant evolution.

4 Tree Containment in General Networks

In this section, we present an algorithm, based on the ideas of the previous section, that solves Tree Containment in $O((\vec{\Delta}_T + 1)^t \cdot (\vec{\Delta}_N + \vec{\Delta}_T^{2.5}) \cdot |V(N)| \cdot |V(T)|)$, where $\vec{\Delta}_N$ and $\vec{\Delta}_T$ are the respective maximal out-degrees of N and T and t is the number of unstable tree components of N (see Definition 1). For bifurcating N and T, this simplifies to $O(3^t \cdot |V(N)| \cdot |V(T)|)$ time. Remarkably, as each root of a tree component (except $\rho(N)$) has a private reticulation parent, we know that t is always smaller than the number of reticulations in N (plus 1) which has been considered as parameter [13]. Indeed, we prove that we can check all biconnected components of N independently, so the parameter can be improved to the maximum number t^* of unstable tree components in any biconnected component of N. For large classes of networks, t^* is arbitrarily small compared to the number of reticulations or even the level[4] of N. Figure 3 shows a preliminary comparison of these parameters in networks generated by simulating recombinant evolution.

The main difficulty of applying the presented algorithm to general networks is that the roots of the tree components are not necessarily stable on any leaf in N. Upon finding such a root γ, we thus have to keep track of all the (maximal) vertices v of T such that N_γ displays T_v. This brings further difficulties: picture two roots γ_1 and γ_2 of tree components of N with $\gamma <_N \gamma_1, \gamma_2$.

[4]The *level* of a phylogenetic network is the largest number of reticulations in any biconnected component (of its underlying undirected graph).

When computing the possible vertices v_1 and v_2 in T whose subtrees are displayed by N_{γ_1} and N_{γ_2}, respectively, we have to make sure that we are not using N_γ to display subtrees in both N_{γ_1} and N_{γ_2}. In the previous algorithm this was not necessary because, if γ is stable, then N_γ cannot display a subtree of T_{v_1} as well as a subtree of T_{v_2}.

Recall that Γ is the set of roots of non-trivial tree components in N and let Γ^* be the set of roots of unstable tree components in N. Furthermore, we call a subnetwork S of N *nice* if S contains the root of N, and for each $u \in V(S)$ and each leaf ℓ that u is stable on in N, S_u contains ℓ. Note that, if S is nice and $u \in V(S)$, then S_u is also nice. For technical reasons, we use a slightly extended notion of subdivisions that allow adding an arc incoming to the root before subdividing arcs. Then, all subdivisions of T in N containing the root of N are nice. The dynamic programming table has an entry for each triple $(u, v, R) \in V(N) \times V(T) \times 2^{\Gamma^*}$ with the following semantics:

$$[u, v, R] := 1 \iff N_u \text{ contains a nice subdivision } S \text{ of } T_v \text{ with } V(S) \cap \Gamma^* = R \tag{2}$$

Note that N displays T if and only if $[\rho(N), \rho(T), R] = 1$ for some $R \subseteq \Gamma^*$. We give some special cases of $[u, v, R]$ for which (2) can be easily verified:

Case 1. If $u \in \Gamma^* \setminus R$, then we set $[u, v, R] = 0$ since all nice subdivisions of T in N_u contain u, and, thus, $u \in V(S) \cap \Gamma^*$ but $u \notin R$.

Case 2. If u is a reticulation with child w, then we set $[u, v, R] = [w, v, R]$ for all $v \in V(T)$ and $R \subseteq \Gamma^*$, since N_u cannot display any more of T than N_w.

Case 3. If u is stable on a leaf $\ell \notin \mathcal{L}(T_v)$, then we set $[u, v, R] = 0$ for all $R \subseteq \Gamma^*$.

Case 4. If u and v are leaves, then we set $[u, v, R] = 1$ if and only if u and v have the same label and $R = \varnothing$.

Case 5. If u is a leaf and v is not a leaf, then we set $[u, v, R] = 0$ for all $R \subseteq \Gamma^*$.

Case 6. If u is a tree vertex of N and $[w, v, R \setminus \{u\}] = 1$ for any child w of u in N and Case 3 does not apply, then we set $[u, v, R] = 1$.

We call an entry $[u, v, R]$ *trivial* if it corresponds to any of the above cases. Otherwise, we set $[u, v, R] = 1$ if and only if there is a size-d matching M between the children $v_1, v_2, \ldots, v_d$ of v in T to the children of u in N and pairwise disjoint sets $R_1, R_2, \ldots, R_d$ such that $\forall_i\ [M(v_i), v_i, R_i] = 1$ and $\bigcup_i R_i = R \setminus \{u\}$.

Lemma 4. *For $[u, v, R]$ computed as above, (2) holds.*

Proof. We prove the lemma by induction on the index of u in any fixed DAG-ordering of N. We suppose that $[u, v, R]$ is non-trivial, as the other cases are evident. This also implies the induction base (where u is a leaf of N).

"$\Leftarrow$": Suppose that there is a nice subdivision S of T_v in N_u with $V(S) \cap \Gamma^* = R$. First, u is not a leaf of S, since all leaves of S are labeled. Second, u does not have degree two in S since, otherwise, $[w, v, R] = 1$ for the child w of u in S, contradicting the non-triviality of $[u, v, R]$. Hence, there is a matching M between the children $v_1, v_2, \ldots, v_d$ of v in T and the children of u in S such that, for each child v_i of v, $S_i := S_{M(v_i)}$ is a subdivision of T_{v_i} and, by the above observation, niceness of S implies niceness of S_i. Also note that M has size d. Then, by induction hypothesis, $[M(v_i), v_i, R_i] = 1$ for all i, where $R_i := V(S_i) \cap \Gamma^*$. Since the S_i are pairwise disjoint, the sets R_i are pairwise disjoint and, since $\bigcup_i V(S_i) = V(S) \setminus \{u\}$, we have $\bigcup_i R_i = \bigcup_i V(S_i) \cap \Gamma^* = V(S) \cap \Gamma^* \setminus \{u\} = R \setminus \{u\}$. Thus, by construction, $[u, v, R] = 1$.

"$\Rightarrow$": Suppose $[u, v, R] = 1$. Then, by construction, there are M and R_i such that R_i are pairwise disjoint, $\forall_i\ [M(v_i), v_i, R_i] = 1$, and $\bigcup_i R_i = R \setminus \{u\}$. By induction hypothesis, Lemma 4 holds for each $[M(v_i), v_i, R_i]$, implying that, for each i, there is a nice subdivision S_i of T_{v_i} in $N_{M(v_i)}$ and $V(S_i) \cap \Gamma^* = R_i$. To show that these subdivisions are pairwise vertex-disjoint, assume that S_i and

S_j intersect for some $i \neq j$. Let w be the minimum with respect to $\leq_N$ among the vertices in $V(S_i) \cap V(S_j)$. Then, w is neither a reticulation (otherwise its child is smaller wrt. $\leq_N$) nor a leaf (since T_{v_i} and T_{v_j} cannot share leaves). Hence, w is in a tree-component of N and it has a root r. Then, both S_i and S_j contain r as well, as otherwise, $M(v_i) = M(v_j)$ contradicting M being a matching. If r is stable on some leaf ℓ then, by niceness of S_i and S_j, both contain ℓ, contradicting again that T_{v_i} and T_{v_j} are leaf-disjoint. If r is not stable, then $r \in V(S_i) \cap R_i$ and $r \in V(S_j) \cap R_j$, contradicting that R_i and R_j are disjoint. □

To compute all $[u, v, R]$ for fixed u and v where d is the out-degree of v, we enumerate all partitions of Γ^* into $d+1$ cells, one for each child v_i of v, corresponding to the R_i, plus one cell corresponding to "$\notin R$". Then, we construct the bipartite graph B whose vertices are the children of u in N and v in T, respectively, and the edge set is $\{u_j v_i \mid [u_j, v_i, R_i] = 1\}$. Finally, we set $[u, v, R] = 1$ if B has a size-d matching for any of the partitions of Γ^*. Since one cell of the bipartition has size d, such a matching can be computed in $O(d^{2.5})$ time [6]. Thus, the implied bottom-up dynamic programming runs in $O((\vec{\Delta}_T + 1)^{|\Gamma^*|} \cdot (\vec{\Delta}_N + \vec{\Delta}_T^{2.5}) \cdot |V(N)| \cdot |V(T)|)$ time. If N and T are forward-binary, this simplifies to $O(3^{|\Gamma^*|} \cdot |V(N)| \cdot |V(T)|)$. We can, however, further refine the algorithm by splitting off biconnected components of N. To this end, we use the following lemma.

Lemma 5 (See also [14])**.** *Let $u \in \Gamma$ such that $N - u$ is disconnected, let $v := \mathrm{LCA}_T(\mathcal{L}(N_u))$, and let (N', T') be the result of contracting N_u and T_v, respectively, into a single vertex and giving a new label λ to both of them. Then, N displays T if and only if N_u displays T_v and N' displays T'.*

Proof. First, note that u is stable on all leaves of $\mathcal{L}(N_u)$.

"⇒": Let S be a subdivision of T in N. Since u is stable on all leaves of $\mathcal{L}(N_u)$, we know that N_u displays T_v and cannot display T_w for any $w >_T v$. Thus, the result S' of contracting S_u into a single vertex and labeling it λ displays T' and it is clearly a subdivision of N'.

"⇐": Since $V(N') \cap V(N_u) = \{u\}$, the result of gluing a subdivision of T' in N' (which has to contain u as leaf) and a subdivision of T_v in N_u together at u is contained in N and it is clearly a subdivision of T. □

With Lemma 5, we can check tree containment in all biconnected components of N independently.

Theorem 3. *Let T be a tree, let N be a network, and let $\vec{\Delta}_N$ and $\vec{\Delta}_T$ be their respective maximum out-degrees. Let t^* be the maximum number of unstable tree components of any biconnected component of N (see Definition 1). Then, we can decide whether N displays T in $O((\vec{\Delta}_T + 1)^{t^*} \cdot (\vec{\Delta}_N + \vec{\Delta}_T^{2.5}) \cdot |V(N)| \cdot |V(T)|)$ time. If N and T are forward-binary, this is $O(3^{t^*} \cdot |V(N)| \cdot |V(T)|)$ time.*

We finish this section with a note on polynomial-time preprocessing concerning the number t^* of unstable tree components in any biconnected component. Indeed, to show that Tree Containment does not admit a polynomial-size kernel (see [8, 9] for more details on "kernelization") it suffices to show that instances (N_i, T_i) of Tree Containment can be combined to a single instance (N, T) such that 1. the number t^* of unstable tree components in any biconnected component of N is in $O(\max_i |N_i|)$ and 2. N displays T if and only if N_i displays T_i for each i (see [2, 10] or [8, Section 15.1.3] for details on "AND compositions").

Let C_k be a caterpillar tree with k leaves labeled with $\{1, 2, \ldots, k\}$. Given k instances (N_i, T_i) of Tree Containment with disjoint label-sets, let N denote the result of, for each i, replacing the leaf labeled i in C_k by N_i. Likewise, let T be the result of, for each i, replacing the leaf labeled i in C_k by T_i. It is then straightforward to verify that N displays T if and only if, for each i, N_i displays T_i. Note that the argument above is independent of the actual parameter that we take per block. For example, it holds as well for the "level" of N.

Observation 5. *Let φ^* map networks to integers such that, for all networks N and all cut-vertices u in N, we have $\varphi^*(N) = \max\{\varphi^*(N_u), \varphi^*(N^u)\}$ where N^u results from N by contracting N_u into a single vertex (with new label). Then,* Tree Containment *does not admit a polynomial-size kernel with respect to φ^*, unless $NP \subseteq coNP/\text{poly}$.*

5 Conclusion

We developed efficient algorithms for the Tree Containment problem in various settings, continuing existing efforts to speed up the process of solving the problem in special types of networks, as well as developing first parameterized algorithms and preliminary results concerning efficient and effective preprocessing. We showed that, if each label occurs at most k times in N, the problem can be solved in $O(|N| \cdot \vec{\Delta}_T \cdot k^2)$ time (where $\vec{\Delta}_T$ is the maximum out-degree in T). Together with the powerful network decomposition of Gunawan et al. [14], this implies an $O(|N|)$-time algorithm for binary reticulation visible or nearly stable networks. We further developed an algorithm that solves the general case in $O((\vec{\Delta}_T + 1)^{t^*} \cdot (\vec{\Delta}_N + \vec{\Delta}_T^{2.5}) \cdot |N| \cdot |T|)$ time where t^* is the maximum number of unstable tree components in any biconnected component of N. For binary N and T, this simplifies to $O(3^{t^*} \cdot |N| \cdot |T|)$. The discovery of the parameter t (and t^*) is interesting in its own regard, as previous algorithms used to study phylogenetic networks focus on the "number r of reticulations" or the "maximum number of reticulations in a biconnected component" (the "level"), but the parameter t^* can be arbitrarily small when compared to these parameters. As there is an implementation of an $O(1.618^r \cdot |N| \cdot |T|)$-time algorithm for Tree Containment [13], I am eager to compare our algorithm to it on practical data sets. Preliminary comparisons show its potential on data-sets generated from simulating evolutionary processes (see Figure 3). Finally, I am highly motivated to research more parameters of phylogenetic networks as we presume that practical networks are likely to be highly structured (since evolution is not a totally random process). The distance of the input network to being reticulation visible or nearly stable seems to be the canonical starting point.

Acknoledgement. Big thanks go to Celine Scornavacca for her thorough proof-reading.

References

[1] M. A. Bender and M. Farach-Colton. The LCA problem revisited. In *Proc. 4th LATIN*, volume 1776 of *LNCS*, pages 88–94. Springer, 2000.

[2] H. L. Bodlaender, B. M. P. Jansen, and S. Kratsch. Kernelization lower bounds by cross-composition. *SIAM J. Discrete Math.*, 28(1):277–305, 2014.

[3] M. Bordewich and C. Semple. Reticulation-visible networks. *Advances in Applied Mathematics*, (78):114–141, 2016.

[4] P. Briggs and L. Torczon. An efficient representation for sparse sets. *ACM Letters on Programming Languages and Systems (LOPLAS)*, 2(1-4):59–69, 1993.

[5] J. M. Chan, G. Carlsson, and R. Rabadan. Topology of viral evolution. *Proceedings of the National Academy of Sciences*, 110(46):18566–18571, 2013.

[6] B. G. Chandran and D. S. Hochbaum. Practical and theoretical improvements for bipartite matching using the pseudoflow algorithm. *CoRR*, abs/1105.1569, 2011. URL http://arxiv.org/abs/1105.1569.

[7] R. Cole, M. Farach-Colton, R. Hariharan, T. Przytycka, and M. Thorup. An $o(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees. *SIAM Journal on Computing*, 30(5):1385–1404, 2000.

[8] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

[9] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

[10] A. Drucker. New limits to classical and quantum instance compression. *SIAM J. Comput.*, 44(5):1443–1479, 2015.

[11] J. Fakcharoenphol, T. Kumpijit, and A. Putwattana. A faster algorithm for the tree containment problem for binary nearly stable phylogenetic networks. In *Proc. 12th JCSSE*, pages 337–342. IEEE, 2015.

[12] P. Gambette, A. D. M. Gunawan, A. Labarre, S. Vialette, and L. Zhang. Locating a tree in a phylogenetic network in quadratic time. In *Proc. 19th RECOMB*, volume 9029 of *LNCS*, pages 96–107. Springer, 2015.

[13] A. D. Gunawan, B. Lu, and L. Zhang. A program for verification of phylogenetic network models. *Bioinformatics*, 32(17):i503–i510, 2016.

[14] A. D. Gunawan, B. DasGupta, and L. Zhang. A decomposition theorem and two algorithms for reticulation-visible networks. *Information and Computation*, (252):161–175, 2017.

[15] A. D. M. Gunawan. Solving tree containment problem for reticulation-visible networks with optimal running time. *CoRR*, abs/1702.04088, 2017. URL https://arxiv.org/abs/1702.04088.

[16] D. Gusfield. *ReCombinatorics: the algorithmics of ancestral recombination graphs and explicit phylogenetic networks*. MIT Press, 2014.

[17] D. H. Huson, R. Rupp, and C. Scornavacca. *Phylogenetic networks: concepts, algorithms and applications*. Cambridge University Press, 2010.

[18] I. A. Kanj, L. Nakhleh, C. Than, and G. Xia. Seeing the trees and their branches in the network is hard. *Theoretical Computer Science*, 401(1-3):153–164, 2008.

[19] T. Lengauer and R. E. Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Trans. Program. Lang. Syst.*, 1(1):121–141, Jan. 1979. ISSN 0164-0925.

[20] A. M., V. G., and P. D. Characterization of reticulate networks based on the coalescent with recombination. *Molecular Biology and Evolution*, 25(12):2517–2520, 2008.

[21] T. J. Treangen and E. P. Rocha. Horizontal transfer, not duplication, drives the expansion of protein families in prokaryotes. *PLoS Genet*, 7(1):e1001284, 2011.

[22] L. Van Iersel, C. Semple, and M. Steel. Locating a tree in a phylogenetic network. *Information Processing Letters*, 110(23):1037–1043, 2010.

Appendix

Proof of correctness of Rule 2. Let S^v be a subdivision of T_v in P and let (N', T') be the result of applying Rule 2 to (N, T).

"$\Leftarrow$": Let N' contain a subdivision S' of T'. It suffices to show that the result S of replacing $\rho(P)$ with S^v in S' is contained in N since S is clearly a subdivision of T. Since S^v is contained in P, it suffices to show that S' and S^v are vertex disjoint (except for $\rho(P)$). Towards a contradiction, assume that S' and S^v both contain a vertex $u \neq \rho(P)$ of P. Since $\mathcal{L}(S')$ and $\mathcal{L}(S^v)$ are disjoint, u is ancestor to at least two different leaves in N. Thus, u is in the tip of P, contradicting that u is in N'.

"$\Rightarrow$": Let N contain a subdivision S of T and let $u := \mathrm{LCA}_S(\mathcal{L}(T_v))$. Since $\rho(P)$ is stable on c and $c \in \mathcal{L}(T_v)$, we have $u \leq_N \rho(P)$, implying $\mathcal{L}(S_{\rho(P)}) \supseteq \mathcal{L}(T_v)$. Further, maximality of v implies $\mathcal{L}(S_{\rho(P)}) \subseteq \mathcal{L}(T_v)$. Let S' result from S by contracting $S_{\rho(P)}$ into a single vertex and labeling this vertex λ. Since $\mathcal{L}(S_{\rho(P)}) = \mathcal{L}(T_v)$, we know that S' is a subdivision of T' and it suffices to show that N' contains S'. To do this, we show that all vertices of S' are in N'. Assume towards a contradiction that S' contains a vertex w that is not in N'. Then, w is in the tip of P, implying $\mathcal{L}(S_w) \subseteq \mathcal{L}(S_{\rho(P)})$. Thus, w is a vertex of $S_{\rho(P)}$ contradicting w being in S'. □