A decomposition theorem and two algorithms for reticulation-visible networks

Andreas D. M. Gunawan[a], Bhaskar DasGupta[b], Louxin Zhang[a,*]

[a]*Department of Mathematics, National University of Singapore, Singapore 119076, Singapore*
[b]*Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607, USA*

Abstract

In studies of molecular evolution, phylogenetic trees are rooted binary trees, whereas phylogenetic networks are rooted acyclic digraphs. Edges are directed away from the root and leaves are uniquely labeled with taxa in phylogenetic networks. For the purpose of validating evolutionary models, biologists check whether or not a phylogenetic tree (resp. cluster) is contained in a phylogenetic network on the same taxa. These tree and cluster containment problems are known to be NP-complete. A phylogenetic network is reticulation-visible if every reticulation node separates the root of the network from at least a leaf. We answer an open problem by proving that the problem is solvable in quadratic time for reticulation-visible networks. The key tool used in our answer is a powerful decomposition theorem. It also allows us to design a linear-time algorithm for the cluster containment problem for networks of this type and to prove that every galled network with n leaves has $2(n-1)$ reticulation nodes at most.

Keywords: phylogenetic networks, reticulation-visibility, tree containment problem, cluster containment problem, galled networks

1. Introduction

How life came to existence and evolved has been a key scientific question in the past hundreds of years. Traditionally, a phylogenetic tree has been used to model the evolutionary history of species, in which an internal node represents a *speciation event* and the leaves represent the extant species under study. These evolutionary trees are often reconstructed from the gene or protein sequences sampled from the extant species. Since genomic studies have demonstrated that genetic material is often transfered between organisms in a non-reproductive manner [4, 25], it has been commonly accepted that phylogenetic networks are more suitable than phylogenetic trees for modeling reticulation events such as horizontal gene transfer, introgression, recombination and hybridization events in genome evolution [6,

[*] Corresponding author

Email addresses: a0054645@u.nus.edu (Andreas D. M. Gunawan), bdasgup@uic.edu (Bhaskar DasGupta), matlzx@nus.edu.sg (Louxin Zhang)

7, 14, 23, 24]. Mathematically, a phylogenetic network is a rooted acyclic digraph with leaves that are uniquely labeled with extant species. The algorithmic and combinatorial aspects of networks have been intensively studied over the past two decades (*e.g.*, see [8, 14, 18, 27]).

In phylogenetics, an important task is checking the "consistency" of two evolutionary models. A somewhat simpler (but nonetheless very important) version of this task asks whether a given network is consistent with an existing tree model or not. This has motivated researchers to study the problem of determining whether a tree is displayed by a network or not, which is called the tree containment problem (TCP). The cluster containment problem (CCP) is related algorithmic problem that asks whether or not a subset of taxa is a cluster in a tree displayed by a network [18]. Both the TCP and CCP have also been investigated in the development of metrics for phylogenetic networks [3, 19].

The TCP and CCP are NP-complete [19], even on a very restricted class of networks [26]. Because of their importance, a lot of effort has been made to find network classes for which the TCP and CCP are solvable in polynomial time. For example, the TCP has been shown to be solvable in polynomial time for two rather restricted subclasses of reticulation-visible networks[12, 26]. Here, the reticulation-visibility property was originally introduced to capture an important feature of galled networks [17]. A network is reticulation-visible if every reticulation separates the network root from at least a leaf. The evolutionary network of violet species appearing in [22] is reticulation-visible, where each reticulation represents a hybridization event that usually produces a new hybrid species. Gambette *et al.* also proved that the TCP is polynomial time solvable for binary nearly stable networks and that each reticulation-visible network contains at most $4(n-1)$ reticulations [11]. Other studies related to the TCP include [5], [21] and [28].

In this paper, we make three contributions. van Iersel *et al.* has posed an open problem as to whether or not the TCP is solvable in polynomial time for *binary* reticulation-visible networks [18, 26]. In this final version of [13], we present a quadratic time TCP algorithm for *arbitrary* reticulation-visible networks. We would remark that a cubic-time TCP algorithm for *binary* reticulation-visible networks is obtained independently in [2]. In [2], it is also proved that each reticulation-visible network has at most $3(n-1)$ reticulations, which is the tight bound.

Secondly, a quadratic time CCP algorithm for reticulation-visible networks can be found in [18, page 171]. Here, we present a linear time CCP algorithms for this class of networks.

Thirdly, we present a powerful decomposition theorem (Theorem 1) to design our algorithms for the CCP and TCP. Empowered by this, we also prove that every galled network with n leaves has $2(n-1)$ reticulation nodes at most.

The rest of the paper is organized as follows. Section 2 introduces the basic concepts and notation. In Section 3, we develop the decomposition theorem mentioned above. It reveals an important structural property of reticulation-visible networks, based on which the two main theorems (Theorems 3 and 4) are proven in Sections 5 and 6, respectively. Finally, we conclude with a couple of remarks in Section 7.

2. Basic Concepts and Notation

2.1. Phylogenetic networks

In phylogenetics, *networks* are rooted acyclic digraphs that satisfies (i) edges and paths are directed away from a unique node called the *root*, (ii) there is a path from the root to *every* other node, and (iii) the nodes of indegree one and outdegree zero (the *leaves*) are *uniquely* labeled. The leaf labels represent bio-molecular sequences, extant organisms or species under study.

In a network, the root has indegree zero and outdegree greater than one and each of the other nodes has either indegree one or outdegree one exclusively. A node is called a *reticulation* node if its indegree is strictly greater than one and its outdegree is precisely one. A reticulation node is called a *bicombination* if it has indegree two. Reticulation nodes represent reticulation events occurring in evolution. Non-reticulation nodes are called *tree* nodes, including the root and leaves.

A network is called a *bicombining* network if every reticulation node is of degree three (*i.e.*, indegree two and outdegree one). A network is called *binary* if the root is of degree two, its leaves are of degree one, and all other nodes are of degree three. A *phylogenetic tree* is a binary network without reticulation nodes.

For convenience in describing the algorithms and proofs, we add an *open* incoming edge to the root (Figure 1). This open edge is not counted into the degree of the root. Let N be a network. For two nodes u, v in N, u is a *parent* of v (alternatively, v is a *child* of u) if (u, v) is an edge in N; u is an *ancestor* of v (alternatively, v is a *descendant* of u) if there is a path from u to v. When u is an ancestor of v, we simply say v is *below* u and u is *above* v sometimes.

Let N be a network. We use the following notation:

- $\rho(N)$: the root of N;
- $\mathcal{L}(N)$: the set of all leaves in N;
- $\mathcal{R}(N)$: the set of all reticulation nodes in N;
- $\mathcal{T}(N)$: the set of the root and other tree nodes of outdegree greater than one in N;
- $\mathcal{V}(N)$: the set of all nodes in N (i.e., $\mathcal{V}(N) = \mathcal{R}(N) \cup \mathcal{T}(N) \cup \mathcal{L}(N)$);
- $\mathcal{E}(N)$: the set of all edges in N;
- $p(u)$: the set of the parents of $u \in \mathcal{R}(N)$ or the unique parent of $u \in \mathcal{T}(N) \setminus \{\rho(N)\}$;
- $c(u)$: the set of the children of $u \in \mathcal{T}(N)$ or the unique child of $u \in \mathcal{R}(N)$;
- $\mathcal{D}_N(u)$: the *subnetwork* node-induced by $u \in \mathcal{V}(N)$ and all the descendants of u;
- $N - E$: the spanning *subnetwork* of N with the node set $\mathcal{V}(N)$ and the edge set $\mathcal{E}(N)\setminus E$ for a subset $E \subseteq \mathcal{E}(N)$;
- $N - V$: the *subnetwork* of N with the node set $\mathcal{V}(N)\setminus V$ and the edge set $\{(x, y) \in \mathcal{E}(N) \mid x \notin V, y \notin V\}$ for a subset $V \subseteq \mathcal{V}(N)$.

Let T be a phylogenetic tree. For $S \subseteq \mathcal{V}(T)$ and $w \in \mathcal{V}(T)$, w is called the lowest common ancestor (LCA) of the nodes in S, denoted lca(S) if it is an ancestor of every node in S and any other "common" ancestor of the nodes in S is above w in T.

2.2. The visibility property

Let N be a network and $u, v \in \mathcal{V}(N)$. We say that u is *visible* (or stable) on v if every path from the root $\rho(N)$ to v *must* contain u [17] (also see [18, p. 165]). In computer science, u is called a dominator of v if u is visible on v [10, 20].

A network node is *visible* if it is visible on some leaf. A network is *reticulation-visible* if every reticulation node is visible. It is not hard to see that each reticulation node separates the root from some leaves in a reticulation-visible network.

The network in Figure 1A is reticulation-visible. Clearly, all trees are reticulation-visible, as they do not contain any reticulation nodes. In fact, reticulation-visible networks form a rather large subclass of networks. The widely studied tree-child networks, galled trees and galled networks are all reticulation-visible [3, 14, 17, 27]. Reticulation-visible networks have two useful properties, as outlined in the following proposition.

Proposition 1. *Let N be a reticulation-visible network and $E \subseteq \mathcal{E}(N)$.*

(i) (**Reticulation separability**) *The child and the parents of a reticulation node are all tree nodes;*

(ii) (**Visibility inheritability**) *If $N - E$ is connected and $\mathcal{L}(N - E) = \mathcal{L}(N)$, $N - E$ is also reticulation-visible.*

Proof (i) Suppose on the contrary that $u, v \in \mathcal{R}(N)$ such that v is the child of u. Let w be another parent of v. Since N is acyclic, w is not below v and hence is not below u. Since w is not a descendant of u, there is a path $P(\rho(N), w)$ from $\rho(N)$ to w that does not contain u.

We now prove that u is not visible on any leaf by contradiction. Assume that u is visible on a leaf ℓ. There is a path P' from $\rho(N)$ to ℓ containing u. Since v is the only child of u, v appears after u in P'. Define $P'[v, \ell]$ to be the subpath of P' from v to ℓ. The concatenation of $P(\rho(N), w)$, (w, v), and $P'[v, \ell]$ gives a path from $\rho(N)$ to ℓ. However, this path does not contain u, which is a contradiction.

(ii) Let $r \in \mathcal{R}(N - E)$. We assume it is visible on a leaf ℓ in N. Since $\mathcal{L}(N - E) = \mathcal{L}(N)$ and $N - E$ is connected, there is at least a path from $\rho(N)$ to ℓ in $N - E$. Any path from $\rho(N)$ to ℓ in $N - E$ is also a path in N and hence must contain r. This implies that r is visible on ℓ in $N - E$. □

2.3. The TCP and CCP

In a digraph, the *suppression* of a node x of indegree and outdegree one is a process in which x is removed and the two edges (y, x) and (x, z) incident to x are merged into a edge (y, z). A tree T' is called a *subdivision* of another tree T if T can be obtained from T' by a series of suppressions.

Consider a network N. The removal of all but one of the incoming edges for each reticulation node results in a subtree. However, new (dummy) leaves may or may not exist in the obtained tree. For example, the removal of four edges $e_1, e_2, (x, v)$, and (x, u) in the network given in Figure 1A results in the tree in Figure 1B, in which x is a new leaf alongside

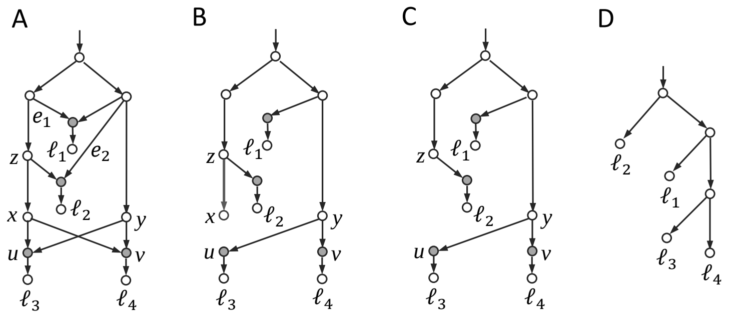

Figure 1: The network in **A** displays the tree in **D** through the removal of four edges $e_1, e_2, (x, v)$, and (x, u). The removal of the four edges results in the subtree in **B**, in which x is a dummy leaf; the further removal of x gives the subtree in **C**, a subdivision of the displayed tree. Reticulation nodes are represented by shaded circles.

the original leaves ℓ_i $(1 \leq i \leq 4)$. If the obtained tree contains dummy leaves, we will have to remove them and some of their ancestors to obtain a subtree with the *same* set of leaves as N.

Definition 1 (Tree Containment). *Let N be a network and T be a phylogenetic tree such that $\mathcal{L}(N) = \mathcal{L}(T)$. We say that N displays* (*or contains*) *T if $E \subseteq \mathcal{E}(N)$ and $V \subseteq \mathcal{V}(N)$ exist such that* (i) *E contains all but one of the incoming edges for each $u \in \mathcal{R}(N)$, and* (ii) *$(N - E) - V$ is a subdivision of T.*

Because of the existence of dummy leaves, V can be non-empty to guarantee that $(N - E) - V$ is a subdivision of T.

The TCP is to determine whether or not a network displays a phylogenetic tree. A bicombining network with k reticulation nodes can display as many as 2^k phylogenetic trees. Hence, for a network with as many as 32 reticulation nodes [14, p. 325], a naive exhaustive search is definitely infeasible.

The set of all the labeled leaves below a node is called the *cluster* of the node in a phylogenetic tree. An internal node in a network may represent different clusters in different trees displayed in the network. Given a *subset* of labeled leaves $B \subseteq \mathcal{L}(N)$, B is a *soft cluster* in N if B is the cluster of a node in some tree displayed in N. The CCP is to determine whether or not a subset B of $\mathcal{L}(N)$ is a soft cluster in a network N.

The TCP and CCP were both proven to be NP-complete even for binary networks [19]. Therefore, it is interesting to investigate whether there are polynomial time algorithms for the TCP and CCP on certain subclasses of phylogenetic networks. For instance, it is open whether or not the TCP is solvable in polynomial time. It is also interesting to explore how to develop algorithms that are fast enough to solve the TCP and CCP instances that arise from current evolutionary study.

3. A Decomposition Theorem for Reticulation-Visible Networks

In this section, we first present a decomposition theorem for reticulation-visible networks. As we shall see later, it plays a vital role in our study of algorithmic and combinatorial

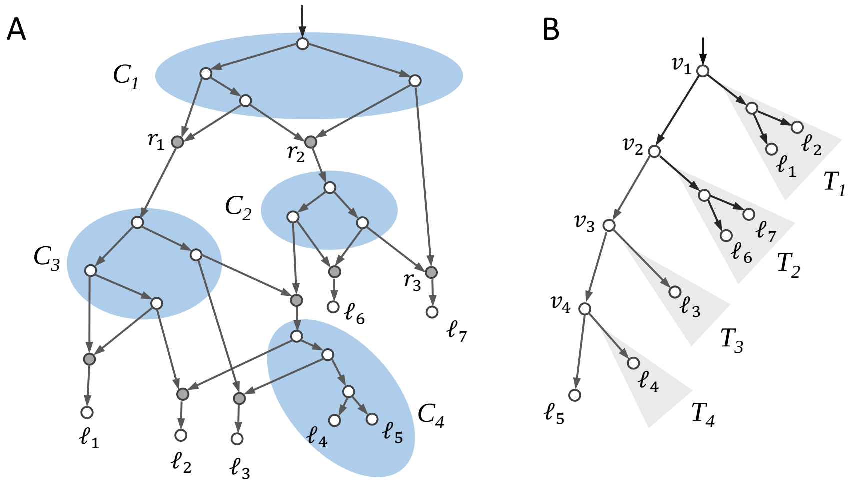

Figure 2: (**A**) A binary reticulation-visible network with nine tree-node components, of which four (C_1–C_4) are the big ones. (**B**) A tree considered for containment in the network.

aspects of reticulation-visible networks.

Consider a reticulation-visible network N. By Proposition 1, every reticulation node is only adjacent to tree nodes. Additionally, in $N - \mathcal{R}(N)$, each connected component C (ignoring edge direction) is actually a rooted subtree of N in which the edges are directed away from its root. Indeed, if C contains two nodes u and v both with indegree zero, where the indegree is defined over $N - \mathcal{R}(N)$, the path between u and v (ignoring edge direction) must contain a node x with indegree 2, contradicting the assumption that x is a tree node in N. Hence, the connected components of $N - \mathcal{R}(N)$ are called the *tree-node components* of N.

Let C be a tree-node component of N and let $\mathcal{V}(C)$ denote its node set. C is called a *single-leaf component* if $\mathcal{V}(C) = \{\ell\}$ for some $\ell \in \mathcal{L}(N)$ and a *big tree-node component* if $|\mathcal{V}(C)| \geq 2$. The binary reticulation-visible network in Figure 2A has four big tree-node components and five single-leaf components.

By definition, any two distinct tree-node components C' and C'' of N are node-disjoint. We say that C' is *below* C'' if a reticulation node r exists such that C' is rooted at the child of r, whereas a parent of r is in C'' in N.

Theorem 1. (**Decomposition Theorem**) *Let N be a reticulation-visible network with m tree-node components $C_1, C_2, \ldots, C_m$, and assume $|\mathcal{V}(N)| > 1$. The following statements are true:*

(i) $\mathcal{T}(N) \cup \mathcal{L}(N) = \uplus_{k=1}^{m} \mathcal{V}(C_k)$, *where $\uplus$ denotes the union of disjoint sets;*

(ii) *For each $r \in \mathcal{R}(N)$, each of its parents is a tree node in some C_i and $c(r)$ is the root of another tree-node component below C_i;*

(iii) *For each tree-node component C_k, $|\mathcal{V}(C_k)| = 1$ if and only if $\mathcal{V}(C_k) = \{\ell\}$ for some $\ell \in \mathcal{L}(N)$ (i.e., it is a single-leaf component). If $|\mathcal{V}(C_k)| > 1$, either C_k contains a network leaf or a reticulation node exists such that its parents are all in C_k;*

(iv) *A big tree-node component C exists such that every tree-node component below it, if any, is a single-leaf component.*

Proof (i) The set equality follows from the fact that the tree-node components are different connected components of $N - \mathcal{R}(N)$ and contain all the tree nodes of N.

(ii) Let $r \in \mathcal{R}(N)$. By the *Reticulation Separability* property (Proposition 1), $c(r)$ and the parents of r are all tree nodes. Thus by (i), each of them is in a tree-node component. Furthermore, since $c(r)$ is of indegree 0 in $N - \mathcal{R}(N)$, $c(r)$ must be the root of the tree-node component to which it belongs.

(iii) Let C be a tree-node component such that $|C| = 1$. Suppose on the contrary, $C = \{u\}$ such that $u \notin \mathcal{L}(N)$. Since u is the only non-leaf tree node in C, $p(u) \in \mathcal{R}(N)$ and $c(u) \subseteq \mathcal{R}(N)$. Let $c(u) = \{c_1, c_2, \ldots, c_k\}$. Since N is reticulation-visible, $p(u)$ is stable on a leaf ℓ below $p(u)$, and subsequently ℓ is also below some nodes in $c(u)$. Since k is finite, a child c_i exists in $c(u)$ such that (i) c_i is an ancestor of ℓ, and (ii) c_i is not below any other node in $c(u)$. Using the same argument as in the proof of Part (a) of Proposition 1, we can prove that there is path from $\rho(N)$ to ℓ through c_i that does not contain $p(u)$. This contradicts the fact that N is reticulation-visible. Therefore, $|C| = 1$ if and only if it is a single-leaf component.

Assume that C is a big tree-node component of N (i.e., $|\mathcal{V}(C)| \geq 2$). Let $\rho(C)$ be the root of C. Since N is reticulation-visible, the reticulation parent of $\rho(C)$ and hence $\rho(C)$ itself are both visible on a network leaf, if $\rho(C)$ is not the root of N. If $\rho(C)$ is the root of N, it is visible on every leaf in N. Let $\rho(C)$ be visible on $\ell \in \mathcal{L}(N)$.

If ℓ is in C, the proof is complete.

If ℓ is not in C, we define $\mathcal{X}(C, \ell) = \{r \in \mathcal{R}(N) \mid p(r) \cap C \neq \emptyset \text{ and } \ell \text{ is below } r\}$. For example, the tree-node component C_1 is visible on each leaf in the network given in Figure 2; $\mathcal{X}(C_1, \ell_2) = \{r_1, r_2\}$ and $\mathcal{X}(C_1, \ell_7) = \{r_2, r_3\}$. Clearly, $\mathcal{X}(C, \ell)$ is finite.

Next, we introduce a topological ordering to $\mathcal{V}(N)$, such that a path from u to v exists only if $v \prec u$ in the ordering (for instance, via breadth-first ordering). Then for any $r', r'' \in \mathcal{X}(C, \ell)$, we have $r' \prec r''$ only if there is a path from r'' to r' in N; in other words r' is below r''. Since $\mathcal{X}(C, \ell)$ is finite, a maximal element r_m with respect to $\prec$ exists in $\mathcal{X}(C, \ell)$, such that r_m is not below any other node in $\mathcal{X}(C, \ell)$.

Let $p(r_m) = \{p_1, p_2, \ldots, p_k\}$. Since $r_m \in \mathcal{X}(C, \ell)$, we may assume that $p_1 \in \mathcal{V}(C)$. Assume that $p_{k_0} \notin \mathcal{V}(C)$ for some $1 < k_0 \leq k$. Since N is acyclic and r_m is maximal under the ordering $\prec$, p_{k_0} is not below any node in C. Hence, there is a path P from $\rho(N)$ to p_{k_0} that does not contain any node in C. Since ℓ is a descendant of r_m, P can be extended into a path from $\rho(N)$ to ℓ that does not contain $\rho(C)$. This contradicts the statement that $\rho(C)$ is visible on ℓ. Therefore, the parents of r_m are all in C, and r_m is inner.

(iv) This is derived from the fact that N is acyclic and finite. □

Let N be a binary reticulation-visible network. Since N is an acyclic digraph and has, at most, $8\,|\mathcal{L}(N)|$ nodes [11], we can determine the tree-node components using the breadth-first search in $O(|\mathcal{L}(N)|)$ time. Additionally, a topological ordering of its nodes like the one given in the proof of Theorem 1(iii) can also be found in $O(|\mathcal{L}(N)|)$ time. Using such topological ordering, we can derive a topological ordering for the tree-node components by looking at the ordering induced on the set of roots of every tree-node component. In the tree-node component ordering, C' is below C'' only if there is a path from the root of C'' to the root of C', i.e. $C' \prec C''$. Using this ordering, we can identify one of the lowest tree node components described in Theorem 1(iv) in constant time.

For a non-binary network, however, the number of nodes may not be bounded from above by a function linear to the number of leaves. The above processes for finding all the big tree-node components and determining the lowest ones take $O(|\mathcal{E}(N)|)$ time.

4. Galled Networks

We shall apply the decomposition theorem to bound the number of nodes in a galled network in this section. A bicombining network is *galled* if each reticulation node r has an ancestor u such that two internal node-disjoint paths exist from u to r in which all nodes except r are tree nodes. Galled networks are reticulation-visible [18].

Let N be a reticulation-visible network. A $r \in \mathcal{R}(N)$ is *inner* if its parents are all in the same tree-node component of N; it is called a *cross-reticulation* otherwise.

Theorem 2. *Let N be a bicombining network. N is galled if and only if every reticulation node is inner.*

Proof (**Necessity**) Let N be galled. N is reticulation-visible [18, p. 165]. Assume, on the other hand, that N contains a cross-reticulation r. By definition, the parents of r are in different tree-node components. Assume that p_1 and p_2 are two parents of r in different tree-node components. Let C_{p_i} be the tree-node components containing p_i for $i = 1, 2$. We now consider the parent r_i of $\rho(C_{p_i})$, for $i = 1, 2$. Since N is acyclic, we may assume that r_1 *is not below* r_2, which also implies that p_1 is not below r_2.

First, r_2 is a reticulation node. Second, p_2 is below r_2 and hence r is also below r_2. However, we can reach r from p_1 using a single edge without passing through r_2 because p_1 is not below r_2. This contradicts the following lemma:

> **Separation Lemma** ([18, p. 163]) Let N be a galled network and r a reticulation in N. Then for every pair of nodes $u \in \mathcal{V}(\mathcal{D}_N(r))$ and $v \in \mathcal{V}(N)\backslash\mathcal{V}(\mathcal{D}_N(r))$, every path (ignoring direction) connecting u to v in N must pass through r.

This completes the proof for necessity part.

(**Sufficiency**) Let N be a bicombining such that each reticulation is inner. For each reticulation node r, by definition, its parents p'_r and p''_r are both in the same tree-node component C. Since C is a subtree of N, $\mathrm{lca}(p'_r, p''_r)$ is also a tree node in C and hence two internal node-disjoint paths from $\mathrm{lca}(p'_r, p''_r)$ to r exist in which all but r are tree nodes. Therefore, N is galled. □

Corollary 1. *If N is a galled network with n leaves, then*
(i) $|\mathcal{R}(N)| \leq 2(n-1)$;
(ii) $|\mathcal{V}(N)| \leq 6n-5$.

Proof (i) Let N be a galled network with n leaves. We will consider the decomposition of N into tree-node components. Since the root of each tree-node component is either $\rho(N)$ or the unique child of a reticulation, the following holds:

$$|\mathcal{R}(N)| = (\text{no. of tree-node components in } N) - 1. \tag{1}$$

We have proved that N contains only inner reticulation nodes, the parents of which are all in the same tree-node component. Therefore, all the tree-node components are connected in a tree structure. Precisely, if G is the graph for which the nodes are the tree-node components and in which a node X is connected to another Y by an edge if the tree-node components corresponding to them are separated by a reticulation node between them, then G is a rooted tree.

Consider a leaf x in G. Since there is no reticulation node below the tree-node component represented by x, it must contain at least one leaf.

In addition, G may contain internal nodes of indegree and outdegree one. Assume that C_y is a tree-node component represented by an internal node y of indegree and outdegree one in G. That y is of degree two implies that there is only an inner reticulation r below C_y. Since there are no multiple edges between any pair of nodes, C_y must contain a leaf in N.

Let n' be the number of network leaves contained in the tree-node components corresponding to the nodes of degree two in G. G has $n-n'$ leaves at most and hence $n-n'-1$ internal nodes of degree greater than two. Therefore, the number of nodes in G is, at most, $n'+(n-n')+(n-n'-1) \leq 2n-1$. In other words, N contains $2n-1$ tree-node components at most. By Eqn. (1), $|\mathcal{R}(N)| \leq 2(n-1)$.

(ii) For a node u in N, we use $d_i(u)$ and $d_o(u)$ to denote the indegree and outdegree of u, respectively. Since N is a bicombining network, each reticulation node is of indegree two and of outdegree one. Hence:

$$n + |\mathcal{T}(N)\backslash\{\rho(N)\}| + 2|\mathcal{R}(N)| = \sum_{v\in\mathcal{V}(N)} d_i(v) = \sum_{u\in\mathcal{V}(N)} d_o(u) = \sum_{u\in\mathcal{T}(N)} d_o(u) + |\mathcal{R}(N)|, \tag{2}$$

where $\mathcal{T}(N)$ is the set of the nodes of outdegree greater than one, including the root. Eqn. (2) implies that:

$$|\mathcal{T}(N)| \leq \sum_{u\in\mathcal{T}(N)} (d_o(u)-1) = |\mathcal{R}(N)| + n - 1 \leq 3(n-1),$$

where the first inequality follows from the fact that $d_o(u) \geq 2$ for each $u \in \mathcal{T}(N)$. Therefore, $|\mathcal{V}(N)| = n + |\mathcal{T}(N)| + |\mathcal{R}(N)| \leq 6n-5$. □

5. A Quadratic-time Algorithm for the TCP

In this section, we shall present a dynamic programming algorithm for the TCP that takes quadratic time.

5.1. The rationale behind our algorithm

The TCP has been known to be solvable in polynomial time only for tree-child networks [26] and the so-called nearly-stable networks [11]. In a tree-child network, each reticulation node is essentially connected to a leaf by a path consisting of only tree nodes. In a nearly-stable network, each child of a node is visible if the node is not visible. Because of the simple local structure around a reticulation node in such a network, one can determine whether or not the network displays a phylogenetic tree by examining the reticulation nodes one by one [11, 26]. However, any approach that works on reticulation nodes one by one is not powerful enough for solving the TCP for a reticulation-visible network with the structure shown in Figure 1, which could have many reticulation nodes above the parents of two reticulation nodes at the bottom. We have to deal with the whole set of reticulation nodes simultaneously for a reticulation-visible network of this kind.

Our algorithm for the TCP relies primarily on Theorem 1. Recall that this decomposition theorem says that in a reticulation-visible network, all the tree nodes and leaves can be partitioned into a collection of disjoint connected components, each having at least *two nodes* if it contains a non-leaf tree node (Figure 2). Most importantly, each component *contains* either a network leaf or all the parents of at least one reticulation node.

The topological property uncovered by this theorem allows us to solve the TCP by the divide-and-conquer approach. Take the TCP for example, we work on the tree-node components one by one in a bottom-up fashion. When working on a tree-node component, we construct a tree in which multiple leaves may have the same label and compare it with the input tree to decipher all the reticulation nodes immediately below it.

5.2. The algorithm

By Part (iv) of Theorem 1, there are at least one "lowest" big tree-node component below which there are only (if any) single-leaf components (Figure 3). Here, we focus on one selected lowest big tree-node component C. The notation introduced before Proposition 2 will be used in the rest of this section.

We assume that C contains k network leaves

$$\ell_1, \ell_2, \ldots, \ell_k, \tag{3}$$

and that below C, there are m inner reticulations:

$$\mathrm{IR}(C) = \{r_1, r_2, \ldots, r_m\}, \tag{4}$$

and m' cross-reticulations:

$$\mathrm{CR}(C) = \{r'_1, r'_2, \ldots, r'_{m'}\}. \tag{5}$$

In the component shown in Figure 3, $k = 2$, $m = 3$, and $m' = 4$. Since C is a big tree-node component, by Part (iii) of Theorem 1, it has two nodes at least, implying that $k + m + m' \geq 2$.

Let $\rho(C)$ denote the root of C. We further define:

$$L_C = \{\, \ell_1, \ell_2, \ldots, \ell_k, c(r_1), c(r_2), \ldots, c(r_m)\} \,. \tag{6}$$

By Part (iii) of Theorem 1, $k + m \geq 1$. In other words, L_C is not empty.

Proposition 2. *The root of C, $\rho(C)$, is visible on each leaf $\ell \in L_C$ in N.*

Proof For each $\ell_i \in L_C$ $(1 \leq i \leq k)$, each path from $\rho(N)$ to ℓ_i must pass $\rho(C)$ before reaching ℓ_i, as there are only tree nodes in the unique path from $\rho(C)$ to ℓ_i.

For each $c(r_i) \in L_C$ $(1 \leq i \leq m)$, any path P from $\rho(N)$ to $c(r_i)$ must contain r_i. Since the parents of r_i are all in C, P must contain $\rho(C)$.

Taken together, the facts imply that $\rho(C)$ is visible on each network leaf in L_C. □

We fix $\ell \in L_C$ for the rest of this section. Assume that we determine whether or not N displays a phylogenetic tree T. Since T has the same set of labeled leaves as N, $\ell \in \mathcal{L}(T)$ and a unique path P_T exists from $\rho(T)$ to ℓ in T. Let:

$$P_T :\; v_1, v_2, \ldots, v_t, v_{t+1}, \tag{7}$$

where $v_1 = \rho(T)$ and $v_{t+1} = \ell$. Since T is binary, $T - P_T$ is the union of t disjoint subtrees $T_1, T_2, \ldots, T_t$, where T_i $(1 \leq i \leq t)$ is the subtree branching off from P_T at v_i (Figure 2B). For the sake of convenience, we also consider the single leaf ℓ as a subtree, written as T_{t+1}. Define:

$$s_C = \min\{s \mid \mathcal{L}(T_s) \cap L_C \neq \emptyset\}. \tag{8}$$

Since $\ell \in \mathcal{L}(T_{t+1}) \cap L_C$, s_C is well defined. For $C = C_4$ in the network and the tree shown in Figure 2, $s_C = 4$. If ℓ_4 and ℓ_6 are switched in the tree, then $s_C = 2$ for the resulting tree.

Proposition 3. *The index s_C can be computed in $O(|\mathcal{L}(N)|)$ time.*

Proof Since T is a binary tree with the same set of labeled leaves as N, T has $2|\mathcal{L}(N)| - 1$ nodes and $2|\mathcal{L}(N)| - 2$ edges. For each $x \in \mathcal{V}(T)$, we define a flag variable f_x to indicate whether the subtree below x contains a network leaf in L_C or not. We first traverse T in the post-order:

- For a leaf $x \in \mathcal{L}(T)$, $f_x = 1$ if $u \in L_C$ and 0 otherwise.
- For a non-leaf node x with children y and z, $f_x = \max\{f_y, f_z\}$.

By definition, $s_C = \min\{i \mid f_{\rho(T_i)} = 1\}$. Obviously, the whole process computes s_C in $O(|\mathcal{L}(T)|) = O(|\mathcal{L}(N)|)$ time. □

Proposition 4. *If N displays T, then $\mathcal{D}_T(v_{s_C})$ is displayed in $\mathcal{D}_N(\rho(C))$, where $\mathcal{D}_N(x)$ is defined in Section 2.1 for a node x in a network N.*

Proof When $s_C = t+1$, $v_{t+1} = \ell$ and $\mathcal{D}_T(v_{s_C}) = T_{t+1}$, which consists of the single leaf $\{\ell\}$. Thus, the statement is true.

When $s_C < t+1$, by the definition of s_C, a network leaf ℓ' exists in $\mathcal{L}(T_{s_C}) \cap L_C$ such that $\ell' \neq \ell$. If N displays T, a subdivision T' of T exists in N. By Proposition 2, $\rho(C)$ is visible on both ℓ and ℓ'. This implies that ℓ and ℓ' are both below $\rho(C)$ in T'. Since T' is a tree, $\text{lca}_{T'}(\ell, \ell')$, the LCA of ℓ and ℓ', is also below $\rho(C)$ in T'. Since $\text{lca}_{T'}(\ell, \ell')$ is the node in T' that corresponds to v_{s_C}, the subnetwork of T' below $\text{lca}_{T'}(\ell, \ell')$ is a subdivision of $\mathcal{D}_T(v_{s_C})$, implying that $\mathcal{D}_N(\rho(C))$ displays $\mathcal{D}_T(v_{s_C})$. □

If N displays T, then C may display a subtree of T that properly contains $\mathcal{D}_T(v_{s_C})$. In other words, it may display a subtree $\mathcal{D}_T(v_j)$ for some $j < s_C$. Therefore, we define:

$$d_C = \min\{j \mid \mathcal{D}_T(v_j) \text{ is displayed in } \mathcal{D}_N(\rho(C))\} \quad (9)$$

For example, $d_C = 3$ for $C = C_4$ and $\ell = \ell_5$ in the network and the tree given in Figure 2. In this case, we have $d_C \le s_C$. Notice that if ℓ_4 and ℓ_6 are switched in the tree, then $d_C = 5$. Therefore, for the tree obtained from the swap, $d_C = 5 > 2 = s_C$, from which we can conclude that the resulting tree is not displayed in the network by Proposition 4.

Proposition 5. *If N displays T, then T has some subdivision in N such that the node corresponding to v_{d_C} is in C.*

Proof Let N display T. Assume that T' is a subdivision of T in N in which the node corresponding to v_{d_C} is u. If u is in C, the proof is complete.

Assume that u is not in C. Since ℓ is below v_{d_C} in T, ℓ is below u in T'. By Proposition 2, $\rho(C)$ is visible on ℓ, and $\rho(C)$ is contained in the unique path P from the network root to ℓ in T'. Since u is in P but not in C, $\rho(C)$ is below u in P. Let P' be the subpath of P from u to $\rho(C)$.

By the definition of d_C, $\mathcal{D}_T(v_{d_C})$ is displayed in $\mathcal{D}_N(\rho(C))$. It has a subdivision T^* in $\mathcal{D}_N(\rho(C))$. Let u'' be the node in T^* corresponding to v_{d_C}. Let P'' be the path from $\rho(C)$ to u'' in C. Since the subtree below u'' in T^* and the subtree below u in T' are the subdivisions of the subtree below v_{d_C} in T, the tree

$$T'' = T' - \mathcal{D}_{T'}(u) + P' + P'' + \mathcal{D}_{T^*}(u'')$$

is also a subdivision of T in N, in which u'' is the node in C corresponding to v_{d_C}. Here, $G + H$ is the graph with the same node set as G and the edge set is the union of $E(G)$ and $E(H)$ for graphs G and H such that $V(H) \subseteq V(G)$. □

To compute d_C as defined in Eqn.(9) in linear-time, we create a tree T_C from C by attaching a new leaf, which has the same label as $c(r)$, below every parent node in $p(r) \cap \mathcal{V}(C)$ for each $r \in \text{IR}(C) \cup \text{CR}(C)$. In other words, T_C has the node set:

$$\begin{aligned} \mathcal{V}(T_C) &= \mathcal{V}(C) \cup \{x_r^{(i)} \mid r \in \text{IR}(C) \,\&\, 1 \le i \le |p(r)|\} \\ &\quad \cup \{z_r^{(i)} \mid r \in \text{CR}(C) \,\&\, 1 \le i \le k\}, \end{aligned} \quad (10)$$

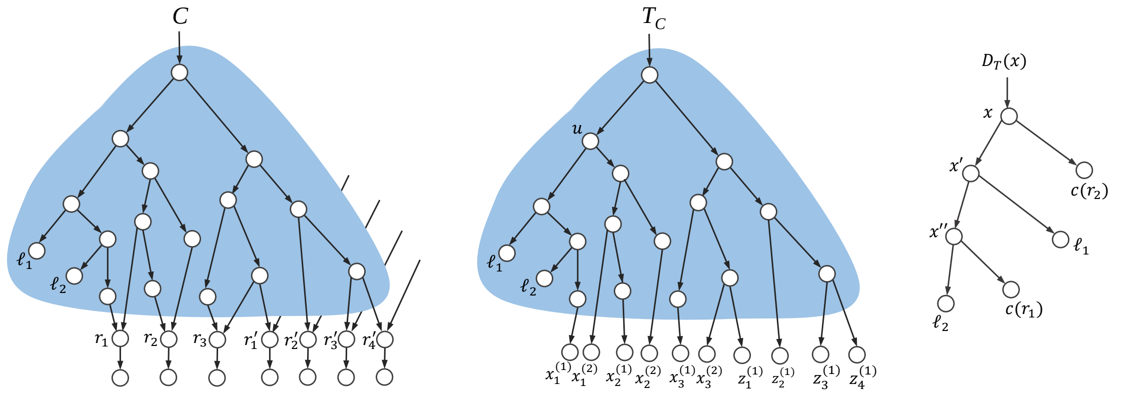

Figure 3: Illustration of the tree T_C (middle) constructed from a lowest tree-node component C (left). C contains two network leaves. Below it are three inner-reticulations (r_i) and four cross-reticulations (r'_i). Here, each reticulation node is of indegree two. The right tree is used for illustration of computing S_u for a node u in T_C.

where $k = |p(r) \cap \mathcal{V}(C)|$ and $x_r^{(i)}$ and $z_r^{(i)}$ are new leaves with the same label as $c(r)$ for each $r \in \mathrm{IR}(C) \cup \mathrm{CR}(C)$, and has the edge set

$$\begin{aligned} \mathcal{E}(T_C) \;\; &= \;\; \mathcal{E}(C) \cup \{(u_r^{(i)}, x_r^{(i)}) \mid r \in \mathrm{IR}(C) \text{ s.t. } p(r) = \{u_r^{(i)} \mid 1 \le i \le |p(r)|\}\} \\ &\quad\;\; \cup \{(v_r^{(i)}, z_r^{(i)}) \mid r \in \mathrm{CR}(C) \text{ s.t. } p(r) \cap \mathcal{V}(C) = \{v_r^{(i)} \mid 1 \le i \le k\}\}. \end{aligned} \tag{11}$$

T_C has $1 + \sum_{v \in \mathcal{V}(C)} d_o(v)$ nodes at most, where $d_o(v)$ is the outdegree of v in N. For bicombining reticulations, T_C is illustrated in Figure 3.

For each $r \in \mathrm{IR}(C)$, T_C contains multiple leaves with the same label as $c(r)$. To detect whether or not $\mathcal{D}_T(v_j)$ is displayed in $\mathcal{D}_N(\rho(C))$, we have to consider which of these leaves will be removed. Such leaves will be referred to as the *ambiguous* leaves. We define:

$$A_r = \{x_r^{(i)} \mid 1 \le i \le |p(r)|\} \tag{12}$$

for $r \in \mathrm{IR}(C)$ and

$$A(T_C) = \cup_{r \in \mathrm{IR}(C)} A_r. \tag{13}$$

$A(T_C)$ is the set of ambiguous leaves in T_C.

For each $r \in \mathrm{CR}(C)$, T_C may also contain multiple leaves with the same label as $c(r)$. Unlike ambiguous leaves, all these leaves can be removed to obtain a subtree of N. Such leaves are called *optional* leaves. We use $O(T_C)$ to denote the set of optional leaves in T_C.

Determining whether or not a subtree of T is displayed in C is very similar to the problem studied in [29]. Here, we shall develop a dynamic programming algorithm for the task. More precisely, we shall compute the following set S_u of nodes in T:

$$S_u = \{x \in \mathcal{V}(T) \mid \mathcal{D}_N(u) \text{ displays the subtree } \mathcal{D}_T(x) \text{ where } u \text{ corresponds to } x\}$$

for each u in T_C. Here, $x \in S_u$ if and only if every leaf ℓ below x in T can be injectively mapped to a leaf ℓ', which has the same label as ℓ, below u in T_C such that for $I = \{\ell' \in \mathcal{L}(\mathcal{D}_{T_C}(u)) \mid \ell \in \mathcal{L}(\mathcal{D}_T(x))\}$, the following facts are true:

(i) the subtree of T_C over I is a subdivision of $\mathcal{D}_T(x)$,

(ii) $I \cap \{\ell_1, \ell_2, \ldots, \ell_k\} = \mathcal{L}(\mathcal{D}_{T_C}(u)) \cap \{\ell_1, \ell_2, \ldots, \ell_k\}$, and

(iii) For each $r \in \mathrm{IR}(C)$ such that $A_r \subseteq \mathcal{L}(\mathcal{D}_{T_C}(u))$, we have $c(r) \in \mathcal{L}(\mathcal{D}_T(x))$.

The reason for (ii) is that we cannot eliminate a network leaf in C when we remove the edges entering a reticulation node to form a subtree of N. The reason for (iii) is that we must keep exactly one incoming edge of each $r \in \mathrm{IR}(C)$ to form a subtree of N. If $A_r \subseteq \mathcal{L}(\mathcal{D}_{T_C}(u))$, then u is always an ancestor of $c(r)$ in any subtree of T_C, hence $c(r) \in \mathcal{L}(\mathcal{D}_T(x))$ because we require (i).

For the node u in T_C and the given subtree $\mathcal{D}_T(x)$ of T in Figure 3, $x \in S_u$. However, neither x' nor x'' is in S_u. For x', r_2 violates the condition (iii). For x'', the condition (ii) does not hold, as ℓ_1 in the right-hand side but not in the left-hand side of the equation.

We introduce a Boolean variable f_{ux} to indicate whether or not $x \in S_u$. That is,

$$f_{ux} = \begin{cases} 1 & \text{if } \mathcal{D}_{T_C}(u) \text{ displays } \mathcal{D}_T(x), \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

We recursively compute f_{ux} for each u and x by traversing both T_C and T in the post-order. To compute f_{ux} in linear time for a fixed x, we have to identify which u does not satisfy the conditions (ii) and (iii) listed in the last paragraph.

For a subset $X \subseteq \mathrm{IR}(C)$, we define

$$V_{\text{lca}}(X) = \{\text{lca}_{T_C}(A_r) \mid r \in X\}.$$

Proposition 6. *All the nodes in $V_{\text{lca}}(IR(C))$ can be determined in $O(|\mathcal{E}(T_C)|)$ time.*

Proof We first pre-process T_C in $O(|\mathcal{E}(T_C)|)$ time so that for any two nodes u and v in T_C, $\text{lca}_{T_C}(u, v)$ can be found in $O(1)$ time (see [1, 15] for example).

Initially, each LCA node is undefined. We visit all leaves in T_C in the post-order. When visiting an ambiguous leaf ℓ in A_r for $r \in \mathrm{IR}(C)$, we set $\text{lca}_{T_C}(r) = \ell$ if $\text{lca}_{T_C}(r)$ is undefined, and $\text{lca}_{T_C}(r) = \text{lca}_{T_C}(\text{lca}_{T_C}(r), \ell)$ otherwise. Since each LCA operation takes $O(1)$ time, the whole process takes $O(|\mathcal{E}(T_C)|)$ time. □

Proposition 7. (i) *Let ℓ be a leaf in T_C that is neither ambiguous nor optional. If $\ell \notin \mathcal{L}(\mathcal{D}_T(x))$, $f_{ux} = 0$ for every u in the unique path from $\rho(C)$ to ℓ.*

(ii) *For each $r \in IR(C)$ such that $c(r) \notin \mathcal{L}(\mathcal{D}_T(x))$, $f_{ux} = 0$ for every node u in the path from $\rho(C)$ to $\text{lca}_{T_C}(A_r)$, including the two endnodes.*

Proof (i) Since ℓ is neither ambiguous nor optional, it is a leaf of N. If a node contained in the unique path P from $\rho(C)$ to ℓ corresponds to x when $\mathcal{D}_T(x)$ is displayed, ℓ must be a leaf in $\mathcal{D}_T(x)$. Therefore, if $\ell \notin \mathcal{L}(\mathcal{D}_T(x))$, $f_{ux} = 0$ for any u in P.

$x \in \mathcal{L}(T)$?	f_{ux}, where u is an unmarked node in T_C
Yes	Case 1 (Base case): u is a leaf in T_C $f_{ux} = 1$ if u has the same label as x $f_{ux} = 0$ otherwise; Case 2: u is not a leaf, and $f_{vx} = 0$ for each $v \in c(u)$ $f_{ux} = 0$; Case 3: u is not a leaf, and $f_{vx} = 1$ for some $v \in c(u)$ $f_{ux} = 1$;
No. $c(x) = \{y, z\}$	Case 4: $f_{vx} = 0$ for each $v \in c(u)$, and no distinct v' and v'' in $c(u)$ exist such that $f_{v'y} = 1$ and $f_{v''z} = 1$ $f_{ux} = 0$; Case 5: $f_{vx} = 1$ for some $v \in c(u)$, or $f_{v'y} = 1$ and $f_{v''z} = 1$ for two distinct v' and v'' in $c(u)$ $f_{ux} = 1$;

Table 1: The update rules for computing f_{ux} for an internal node u in T_C.

(ii) Let $r \in \mathrm{IR}(C)$ and let u be a node in the unique path from $\rho(C)$ to $\mathrm{lca}_{T_C}(A_r)$ inclusively in C. Since A_r contains at least two ambiguous leaves, $\mathrm{lca}_{T_C}(r)$ is an internal node in C. Any subtree of N contains exactly one incoming edge of r below $\mathrm{lca}_{T_C}(r)$. This implies that $c(r)$ must appear in any subtree displayed below u. Since $c(r) \notin \mathcal{L}(\mathcal{D}_T(x))$, $\mathcal{D}_T(x)$ cannot be displayed at u. □

Let $\bar{T}_x$ be the minimum subtree of T_C containing:

$$\left[\mathcal{L}(T_C)\backslash(A(T_C) \cup O(T_C) \cup \mathcal{D}_T(x))\right] \cup V_{\mathrm{lca}}(\mathrm{IR}(C)\backslash\mathcal{L}(\mathcal{D}_T(x))) \cup \{\rho(C)\}.$$

Notice that $\bar{T}_x$ is a subtree on the top of T_C with the root $\rho(C)$. By Propositions 6 and 7, a node in T_C satisfies the conditions (ii) and (iii) if and only if it is not in $\bar{T}_x$. Therefore, we have the following fact.

Proposition 8. *For $u \in \mathcal{V}(T_C)$, $f_{ux} = 1$ if and only if $u \notin \mathcal{V}(\bar{T}_x)$ and one of the following two conditions is true:*

(a) *a child $v \in c(u)$ exists such that $f_{vx} = 1$;*

(b) *x has the children y and z and distinct $v' \in c(u)$ and $v'' \in c(u)$ exist such that $f_{v'y} = 1$ and $f_{v''z} = 1$.*

Proposition 9. *There is an algorithm that takes T_C and T as input and computes d_C in Eqn. (9) in $O(|\mathcal{E}(T_C)| \cdot |\mathcal{V}(T)|)$ time.*

Proof Our dynamic programing algorithm traverses T in the post-order: for each x, compute f_{ux} for all u in T_C in four steps:

Step 1. Pre-process T_C so that the LCA of any two nodes can be found in $O(1)$ time in T_C.

Step 2. Traverse the leaves in T_C to compute the nodes in $V_{\text{lca}}(\text{IR}(C))$.

Step 3. Mark the nodes in the subtree $\bar{T}_x$ of T_C. For each leaf $\ell \not\in A(T_C) \cup O(T_C) \cup \mathcal{L}(\mathcal{D}_T(x))$, mark the nodes in the path from $\rho(T_C)$ to ℓ. For each $r \in \text{IR}(C)$ such that $c(r) \not\in \mathcal{L}(\mathcal{D}_T(x))$, mark the nodes in the path from $\rho(T_C)$ to $\text{lca}_{T_C}(A_r)$ inclusively.

Step 4. Traverse T_C in a bottom-up manner. For each node u, if it is *unmarked* in Step 3, compute f_{ux} using the formulae given in Table 1; if it is marked in Step 3, $f_{ux} = 0$.

The correctness of the above computation process follows from Propositions 7 and 8. Step 1 takes constant time [1, 15]. By Proposition 6, Step 2 can be done in $O(|\mathcal{E}(T_C)|)$ time.

Two paths from $\rho(T_C)$ to nodes in $V_{\text{lca}}(\text{IR}(C))$ may have a common part starting at $\rho(T_C)$. We mark the nodes in each of these paths in a bottom-up manner: whenever we reach a marked node, we stop the marking process in the current path. In this way, each marked node of outdegree k is visited k times at most. Hence, Step 3 can be executed in $O(|\mathcal{E}(T_C)|)$ time.

In Step 4, at a node u, we simply need to check whether or not $f_{vx} = 1$ for each child v of u and whether or not $f_{vy} = 1$ for each child y of x and each child v of u. Since x has only two children, Step 4 takes $\sum_{u \in \mathcal{V}(T_C)} O(|c(u)|) = O(|\mathcal{E}(T_C)|)$ time.

Taken together, these facts imply that our algorithm uses
$\sum_{x \in \mathcal{V}(T)} O(|\mathcal{E}(T_C)|) = O(|\mathcal{N}(T_C)| \cdot |\mathcal{V}(T)|)$ time.

After we know the values of f_{ux} for every u in T_C and every x in T, we can compute d_C such that $\mathcal{D}_T(v_j)$ is displayed in $\mathcal{D}_N(\rho(C))$ as $d_C = \min_{1 \le j \le t+1}\{j \mid f_{uv_j} = 1 \text{ for some } u \in \mathcal{V}(T_C)\}$. □

Based on the facts presented above, we obtain the following TCP algorithm (Table 2).

We now analyze the time complexity of the TCP Algorithm. Step 1 can be done in $O(|\mathcal{E}(N)|)$ time if a breadth-first search is used.

Step 3 is a while-loop. During each execution of this step, the current network is obtained from the previous network by replacing the big tree-node component examined in the last execution with a new leaf node. Because of this, the modification done in the last two lines in Step 3.6 makes the tree decomposition of the current network available before the current execution. Step 3.1 takes constant time. The time spent in Step 3.2 for each execution is $O(|\mathcal{E}(T_C)|)$. Step 3.3 takes $O(|\mathcal{V}(T)|)$ time.

By Proposition 3, the total time spent in Step 3.4 is $O(|\mathcal{L}(N)|^2)$. By Proposition 9, the total time spent in Step 3.5 is $\sum_i O(|\mathcal{V}(T_{C_i})| \cdot |\mathcal{L}(T)|)$, which is $O(|\mathcal{E}(N)| \cdot |\mathcal{L}(T)|)$. The time spent in Step 3.6 for each execution is $O(\sum_{u \in \mathcal{V}(C)} |c(u)|)$. Hence, the total time spent in Step 3.6 is $O(|\mathcal{E}(N)|)$.

Taken altogether, these facts imply that the TCP Algorithm takes quadratic time. That is,

```
Input: A reticulation-visible network $N$ and a binary phylogenetic tree $T$.
1. Compute the tree-node components of $N$: $C_t \prec C_{t-1} \prec \ldots \prec C_1$,
   where $\prec$ is a topological order such that $C_i$ is below $C_j$ only if $i < j$.
2. $N' \leftarrow N$ and $T' \leftarrow T$;
3. Repeat unless ($N'$ becomes a single node) {
   3.1. Select the lowest big tree-node component $C$;
   3.2. Compute $L_C$ in Eqn. (6) and select $\ell \in L_C$;
   3.3. Compute the path $P_T$ from the root to $\ell$ in Eqn. (7);
   3.4. Determine the smallest index $s_C$ defined by Eqn. (8);
   3.5. Determine the smallest index $d_C$ defined by Eqn. (9);
   3.6. If ($s_C < d_C$), output "$N$ does not display $T$";
        else {
          For each $r \in \mathrm{CR}(C)$ {
            if ($c(r) \notin \mathcal{D}_T(v_{d_C})$), delete all edges $(z, r)$ for $z \in p(r) \cap \mathcal{V}(C)$;
            if ($c(r) \in \mathcal{D}_T(v_{d_C})$), delete all edges $(z, r)$ for $z \in p(r) \backslash \mathcal{V}(C)$;
          }
          Replace $C$ (resp. $\mathcal{D}_T(v_{d_C})$) by a leaf $\ell_C$ in $N'$ (resp. $T'$);
          Remove $C$ from the list of tree-node components;
          Update $\mathrm{CR}(C')$ for the affected big tree-node components $C'$;
        } /* end if */
   } /* end repeat */
```

Table 2: The TCP algorithm.

Theorem 3. *Given a reticulation-visible network N and a phylogenetic tree T with the same set of labeled leaves as N, the TCP for N and T can be solved in $O(|\mathcal{E}(N)| \cdot |\mathcal{L}(T)|)$ time.*

6. A Linear-time Algorithm for the CCP

As another application of the decomposition theorem, we shall present a linear-time algorithm for the CCP in this section. Given a reticulation-visible network N and a subset $B \subseteq \mathcal{L}(N)$, we would like to determine whether or not B is a cluster of some node in a tree displayed by N. The edges entering a reticulation node are called *reticulation edges* in this section.

Assume that N has t big tree-node components $C_1, C_2, \ldots, C_t$. Consider the lowest big tree-node component C. We use the same notation as in Section 5: L_C is defined in Eqn. (6); $\rho(C)$ denotes the root of C; $\mathrm{IR}(C)$ and $\mathrm{CR}(C)$ denote the set of inner and cross-reticulations below C, respectively. Finally, we set $\bar{B} = \mathcal{L}(N) \backslash B$.

When $L_C \cap B \neq \emptyset$ and $L_C \cap \bar{B} \neq \emptyset$, L_C contains two leaves ℓ_1 and ℓ_2 such that $\ell_1 \in B$, but $\ell_2 \notin B$. If B is the cluster of a node z in a subtree T of N, z is in the unique path P from $\rho(T')$ $(= \rho(N))$ to ℓ_1 in T.

Assume that z is between $\rho(N)$ and $\rho(C)$ in P. Since $\rho(C)$ is visible on ℓ_2, ℓ_2 is below $\rho(C)$ no matter which incoming edge is contained in T for each $r \in \mathrm{IR}(C)$. This implies that ℓ_2 is below z and thus in B, which is a contradiction. Therefore, if B is a soft cluster, it must be a soft cluster of a node in C.

When $L_C \cap \bar{B} = \emptyset$ (i.e., $L_C \subseteq B$), we construct a subnetwork N' of N by deleting:

- all but one of the incoming edges for each $r \in \mathrm{IR}(C)$,
- all incoming edges whose tails are in C for each $r \in \mathrm{CR}(C)$ such that $c(r) \notin B$, and
- all incoming edges but one with a tail in C for each $r \in \mathrm{CR}(C)$ such that $c(r) \in B$.

Note that $T' = \mathcal{D}_{N'}(\rho(C))$ is a subtree with the following set of network leaves:

$$\hat{B} = L_C \cup \{c(r) \mid r \in \mathrm{CR}(C) \text{ s.t. } c(r) \in B\}. \tag{15}$$

Hence, if $B = \hat{B}$, B is then the cluster of $\rho(C)$ in N'. If $\hat{B} \subsetneq B$, we replace T' by a new leaf ℓ_C and set $B' = \{\ell_C\} \cup \left(B \backslash \hat{B}\right)$.

Proposition 10. *Assume that $\hat{B} \subsetneq B$. B is a soft cluster in N if and only if B' is a soft cluster in N'.*

Proof Assume that B' is the cluster of a node z in a tree T'' displayed in N'. When N' is constructed, ℓ_C replaces the subtree T' rooted at $\rho(C)$ for which the leaves are $\hat{B}$; so if we re-expand ℓ_C into T' to get a subtree T of N, the cluster of z in T becomes $\left(B \backslash \hat{B}\right) \cup \hat{B} = B$ and thus B is a soft cluster in N.

Assume that B is the cluster of a node z in a subtree T of N. Let E be the set of reticulation edges removed to obtain T from N. Since $\hat{B} = B \cap \mathcal{L}(\mathcal{D}_N(\rho(C))) \neq B$, B contains a leaf $\bar{\ell}$ that is not below $\rho(C)$. Since $\bar{\ell}$ is below z in T, z must be above $\rho(C)$ in T.

Consider a reticulation node $r \in \mathrm{CR}(C)$ such that $c(r) \in B$. Since $c(r)$ is in B, it is a leaf below z in T. By our definition of cross-reticulation, r has at least one parent in C. Let (p_r, r) be an edge such that $p_r \in C$. Note that all but one of the incoming edges of r are in E. Define:

$$\begin{aligned} E' &= [E \cup \{(p, r) \in \mathcal{E}(N) \mid r \in \mathrm{CR}(C) \text{ s.t. } c(r) \in B \text{ and } p \notin \mathcal{V}(C)\}] \\ &\quad \backslash \{(p_r, r) \mid r \in \mathrm{CR}(C) \text{ s.t. } c(r) \in B\}. \end{aligned}$$

It is not hard to see that (p_r, r) is the unique incoming edge of r that is not in E' for each $r \in \mathrm{CR}(C)$ such that $c(r) \in B$.

Let $T' = N - E'$. T' may contain some dummy leaves that are internal nodes in N. However, it is easy to see that the cluster of z is equal to B and $\hat{B}$ is the cluster of $\rho(C)$ in T'. If we contract the subtree below $\rho(C)$ into a single leaf ℓ_C, the cluster of z becomes $B \cup \{\ell_C\} \backslash \hat{B}$, which is B'. Therefore, B' is a soft cluster in N'. □

When $B \cap L_C = \emptyset$, B may or may not be a soft cluster in $\mathcal{D}_N(\rho(C))$. Assuming that B is not a soft cluster in $\mathcal{D}_N(\rho(C))$, we reconstruct N' from N by:

- removing all the edges in $\{(u,r) \in \mathcal{E}(N) \mid r \in \mathrm{CR}(C) \text{ s.t. } c(r) \notin B, u \notin \mathcal{V}(C)\}$,
- removing all the edges in $\{(u,r) \in \mathcal{E}(N) \mid r \in \mathrm{CR}(C) \text{ s.t. } c(r) \in B, u \in \mathcal{V}(C)\}$, and
- replacing $\mathcal{D}_N(\rho(C))$ by a new leaf ℓ_C.

Similar to the last case, we present the following proposition.

Proposition 11. *Assume that B is not in $\mathcal{D}_N(\rho(C))$ and $L_C \cap B = \emptyset$. B is a soft cluster in N if and only if B is a soft cluster in N'.*

Proof Assume that B is the cluster of a node z in a tree T'' displayed in N'. If z is an ancestor of ℓ_C, then every leaf in L_C is in the cluster of z in any subtree of N, contradicting $B \cap L_C = \emptyset$. Let T' be a subtree of $\mathcal{D}_N(\rho(C))$ such that $\mathcal{L}(T') = \mathcal{L}(N) \backslash \mathcal{L}(T'')$. If we re-expand the node ℓ_C into the tree T', we obtain a subtree T of N. The node z is not an ancestor of ℓ_C in T'', so B is the cluster of z in T too, and hence B is a soft cluster of N.

Conversely, assume that B is the cluster of a node z in a subtree T of N. Let E be the set of reticulation edges that are removed to obtain T from N. By our assumption, B is not a soft cluster in $\mathcal{D}_N(\rho(C))$. Since $\rho(C)$ is visible on all leaves in L_C, $\rho(C)$ is not below z in T. Therefore, neither $\rho(C)$ nor z is an ancestor of the other in T.

Consider a reticulation $r \in \mathrm{CR}(C)$. Since r is a cross-reticulation, it has at least one parent in C. We select a parent p_r in C. If $c(r) \in B$, $c(r)$ is a leaf below z in T and thus the unique incoming edge of r contained in T is from a node that is not in C to r. If $c(r) \notin B$, the unique incoming edge contained in T may or may not have a tail in C. However, if this edge is not between a node in C and r, we can obtain a tree in which z still has B as its cluster by replacing the edge with the edge (p_r, r), where p_r is the selected parent of r in C. Therefore, we define:

$$\begin{aligned} E' &= [E \cup \{(p,r) \in \mathcal{E}(N) \mid r \in \mathrm{CR}(C) \text{ s.t. } c(r) \notin B, p \in p(r)\}] \\ &\quad \backslash \{(p_r, r) \mid r \in \mathrm{CR}(C) \text{ s.t. } c(r) \notin B\}. \end{aligned}$$

Note that (p_r, r) is the unique incoming edge of r that is not in E' for each $r \in \mathrm{CR}(C)$ such that $c(r) \notin B$.

Let $T' = N - E'$. T' may contain some dummy leaves that are internal nodes in N. However, it is easy to see that the cluster of z in T' remains the same as the cluster of z in T, which is equal to B. If we contract $\mathcal{D}_{T'}(\rho(C))$ into a single leaf ℓ_C, T' is a subtree of N', implying that B is a soft cluster in N'. □

We next show how to determine whether or not B is in C in linear time. Let T_C be the tree defined in Eqn. (10) and (11). For each $r \in \mathrm{IR}(C)$, A_r denotes the set of ambiguous leaves defined in Eqn. (12) and $\mathrm{lca}_{T_C}(r)$ denotes the LCA of the leaves in A_r.

Proposition 12. (i) *Let ℓ be a leaf in T_C that is neither ambiguous nor optional. If $\ell \notin B$, B is not a soft cluster of any node u in the unique path from $\rho(C)$ to ℓ in C.*

(ii) *For each $r \in IR(C)$ such that $c(r) \notin B$, B is not a soft cluster of any u in the unique path from $\rho(C)$ to $\mathrm{lca}_{T_C}(r)$ inclusively.*

Proof This can be proven in the same way as Proposition 7. □

Let $\bar{T}_B$ be the minimum subtree of T_C that connects $\big[\mathcal{L}(T_C)\backslash\,(A(T_C)\cup O(T_C)\cup B)\,\big]\cup V_{\text{lca}}(\text{IR}(C)\backslash B)\cup\{\rho(C)\}$, where $A(T_C)$ and $O(T_C)$ are the sets of ambiguous and optional leaves in T_C, respectively, and $V_{\text{lca}}(\cdot)$ is defined in Proposition 6. We further define $V_{\max}=\{v\in\mathcal{V}(T_C)\mid v\notin\mathcal{V}(\bar{T}_B)\text{ but }p(v)\in\mathcal{V}(\bar{T}_B)\}$.

Proposition 13. *B is a soft cluster in $\mathcal{D}_N(\rho(C))$ if and only if a node $v\in V_{\max}$ exists such that for each $\ell\in B$, there is a leaf below v with the same label as ℓ.*

Proof Assume that B is a soft cluster of a node u in $\mathcal{D}_N(\rho(C))$. By Proposition 12, u is not in $\bar{T}_B$ and thus it is below some $v\in V_{\max}$. For any $\ell\in B$, u and hence v have a common leaf descendant with the same label as ℓ.

Let $v\in V_{\max}$ satisfy the property that for each $\ell\in B$, a leaf $\ell'\in\mathcal{L}(T_C)$ exists that has the same label as ℓ. For each $x\in\text{IR}(C)$ such that $c(x)\notin B$, by the definition of $V_{\max}$, A_x contains an ambiguous leaf that is not below v. We select a parent p'_x of r that is not below v in C.

For each $y\in\text{IR}(C)$ such that $c(y)\in B$, we select a parent p''_y below v.

For each $r\in\text{CR}(C)$ such that $c(r)\in B$, we select a parent p_r below v.

```
            ALGORITHM 1
Input: $T_C$ and a subset $B$ of leaves in $\mathcal{D}_N(\rho(C))$.
1. If $|B| == 1$, output "Yes" and **exit**;
2. Construct $T_C$ as defined in Eqn. (10) and (11);
3. Pre-process $T_C$ so that the LCA of any two nodes can be found in $O(1)$ time;
4. Traverse the leaves in $T_C$ to compute the nodes in $V_{\text{lca}}(\text{IR}(C))$;
5. For each leaf $\ell\in\mathcal{L}(T_C)\backslash\,(A(T_C)\cup O(T_C)\cup B)$,
       mark the nodes in the path from $\rho(T_C)$ to it;
   For each $r\in\text{IR}(C)$ such that $c(r)\notin B$,
       mark the nodes in the path from $\rho(T_C)$ to $\text{lca}_{T_C}(r)$ inclusively;
6. Traverse the nodes $u$ in $T_C$ to compute the nodes in $V_{\max}$:
       check if $u$ is unmarked and its parent is marked in Step 5 when visiting $u$;
7. For each node $u\in V_{\max}$ {
       7.1 Check whether or not all leaves in $B$ are below $u$;
       7.2 Output "Yes" and **exit** if so; } /* for /*
8. Output "No" and **exit**;
```

Table 3: An algorithm to decide whether B is a soft cluster in C.

Set:

$$\begin{aligned}
&E = \{(p,r) \in \mathcal{E}(N) \mid p \in \mathcal{V}(C), r \in A(T_C) \cup O(T_C)\} \backslash (E_1 \cup E_2 \cup E_3),\\
&E_1 = \{(p'_x, x) \mid x \in \mathrm{IR}(C) \text{ such that } c(x) \not\in B\},\\
&E_2 = \{(p''_y, y) \mid y \in \mathrm{IR}(C) \text{ such that } c(r) \in B\},\\
&E_3 = \{(p_r, r) \mid r \in \mathrm{CR}(C) \text{ such that } c(r) \in B\}.
\end{aligned}$$

Therefore, $\mathcal{D}_N(\rho(C)) - E$ is a subtree in which B is the cluster of v. It is not hard to see that $\mathcal{D}_N(\rho(C)) - E$ can be extended into a subtree of N. □

Taken together, the above facts imply that we can use ALGORITHM 1 (Table 3) for determining whether a leaf subset is a soft cluster in the lowest big tree-node component or not. The correctness of ALGORITHM 1 follows from Propositions 12 and 13.

Step 1 takes constant time. Step 2 can be done in $O(\sum_{u \in \mathcal{V}(C)} |c(u)|)$ time. Step 3 takes $O(|\mathcal{E}(T_C)|)$ time (see [15]). By Proposition 6, Step 4 can be done in $O(|\mathcal{E}(T_C)|)$ time. In the proof of Proposition 9, Step 5 can be executed in $O(|\mathcal{E}(T_C)|)$ time. Obviously, Step 6 takes $O(|\mathcal{E}(T_C)|)$ time. For each node u, Step 7.1 takes $O(|\mathcal{E}(\mathcal{D}_{T_C}(u))|)$ time. Since all the examined subtrees are disjoint, the total time taken by Step 7.1 is $O(|\mathcal{E}(T_C)|)$ time.

Taking all the above facts together, we are able to give a linear-time algorithm for the CCP summarized in Table 4. It runs in linear time. Step 1 takes $O(|\mathcal{E}(N)|)$ time. Step 2 is a for-loop that runs t times. Since the total number of network leaves in C_k and the reticulation nodes below C_k is $|\mathcal{E}(C_k)|$ at most, Step 2.1 takes $O(|\mathcal{E}(C_k)|)$ time for each execution. In Step 2.2, the linear-time ALGORITHM 1 is called to compute Y in $O(|\mathcal{E}(C_k)|)$ time. Obviously, Step 2.3 takes constant time. To implement Step 2.4 in linear time, we need to use an array A to indicate whether a network leaf is in B or not. A can be constructed in $O(|\mathcal{L}(N)|)$ time. With A, each conditional clause in Step 2.4 can be determined in $|L|$ time, which is $O(|\mathcal{E}(C_k)|)$ at most. Since the total number of inner and cross-reticulations is $|\mathcal{E}(C_k)|$ at most, each line of Step 2.4 takes $O(|\mathcal{E}(C_k)|)$ time at most. Hence, Step 2.4 still takes $O(|\mathcal{E}(C_k)|)$ time. Taking all these together, the total time taken by Step 2 is $\sum_{1 \le k \le t} O(|\mathcal{E}(C_k)|) = O(|\mathcal{E}(N)|)$.

Theorem 4. *Given a reticulation-visible network N and a subset B of labeled leaves in N, the CCP for N and B can be solved in $O(|\mathcal{E}(N)|)$ time.*

7. Concluding Remarks

We have presented polynomial-time TCP and CCP algorithms for arbitrary reticulation-visible networks. They rely on the powerful decomposition theorem proven in Section 3. The algorithms have been implemented in C and the computer program is available upon request.

Using the decomposition theorem, we have proved that a bicombining network is galled if and only if every reticulation node is inner. This gives another insightful characterization

```
Input: A reticulation-visible network $N$ and a subset $B \subseteq \mathcal{L}(N)$.
1. Compute the tree-node components of $N$: $C_t \prec C_{t-1} \prec \ldots \prec C_1$,
where $\prec$ is a topological order such that $C_i$ is below $C_j$ only if $i < j$.
2. for $k = 1$ to $t$ do {
   2.1. Set $C = C_k$; compute $L := L_{C_k}$ as defined in Eqn. (6);
   2.2. $Y :=$ ( Is $B$ a soft cluster in $\mathcal{D}_N(\rho(C))$? );
   2.3. if ($Y == 1$) output "Yes" and exit;
   2.4. if ($Y == 0$) {
           $\bar{B} := \mathcal{L}(N) \backslash B$;
           if ($L \cap \bar{B} \neq \emptyset$ & $B \cap L \neq \emptyset$) output "No" and exit;
           if ($B \cap L == \emptyset$) {
              Remove edges in $\{(u, r) \mid r \in \mathrm{CR}(C)$ s.t. $c(r) \notin B, u \notin C\}$;
              Remove edges in $\{(u, r) \mid r \in \mathrm{CR}(C)$ s.t. $c(r) \in B, u \in C\}$;
           }
           if ($\bar{B} \cap L == \emptyset$) {
              Remove edges in $\{(u, r) \mid r \in \mathrm{CR}(C)$ s.t. $c(r) \notin B, u \in C\}$;
              Remove edges in $\{(u, r) \mid r \in \mathrm{CR}(C)$ s.t. $c(r) \in B, u \notin C\}$;
              $B := (B \cup \{\ell_C\}) \setminus (L \cup \{c(r) \mid r \in \mathrm{CR}(C)$ s.t. $c(r) \in B\})$;
           }
           Replace $\mathcal{D}_N(\rho(C))$ by a leaf $\ell_C$;
           Remove $C$ from the list of big tree-node components;
           Update $\mathrm{CR}(C')$ for the affected big tree-node components $C'$;
        }
} /* for */
```

Table 4: The CCP algorithm.

of galled networks. A network is *tree-child* if every non-leaf node has a tree node child. Analogously, a network is tree-child if and only if each tree-node component contains a network leaf.

In [11], we proved for the first time that the number of reticulation nodes in a binary reticulation-visible network is bounded from above by a linear function in the number of the leaves. The bound was established using the fact that reticulation-visible networks are tree-based [9], that is, they can be obtained from a tree with the same set of labeled leaves by the addition of some edges between the tree edges. In the present paper, we use the decomposition theorem to show that an arbitrary galled network has at most $2(n-1)$ reticulations, in which tree nodes are not necessarily binary. Therefore, we also deliver a new technique for establishing the size of a network with the reticulation-visibility property.

One interesting problem for future research is how to extend our study into fast heuristic TCP and CCP algorithms for arbitrary networks. Other problems include (a) testing whether two reticulation-visible networks display the same set of binary trees in polynomial time and (b) applying the decomposition theorem in reconstructing reticulation-visible networks from gene trees or sequences. It would also be interesting to see the application of the theory to construct a real evolutionary model in biology. Note that two non-isomorphic phylogenetic networks could display the same set of binary trees over the same set of labeled leaves as the networks [16]. Solutions for these questions are definitely valuable in the study of phylogenetics.

Acknowledgments

The authors are grateful to Philippe Gambette, Anthony Labarre, and Stéphane Vialette for discussions on the problems studied here and Bingxin Liu for debugging our C-program. We also thank two reviewers for useful comments on the first submission of this work. The work was supported by a Singapore MOE ARF Tier-1 Grant R-146-000-177-112 and the Merlion Programme 2013. DasGupta was supported by NSF Grant IIS-1160995. This work was also presented at the RECOMB'2016.

References

[1] M.A. Bender, M. Farach-Colton, G. Pemmasani, S. Skiena, P. Sumazin, Lowest common ancestors in trees and directed acyclic graphs, J. Algorithm 57 (2005) 75–94.

[2] M. Bordewich, C. Semple, Reticulation-visible networks, Adv. in Applied Math 76 (2016) 114–141. arXiv:1508.05424, 2015.

[3] G. Cardona, F. Rosselló, G. Valiente, Comparison of tree-child phylogenetic networks, IEEE-ACM Trans. Comput. Biol. Bioinform. 6 (2009) 552–569.

[4] J.M. Chan, G. Carlsson, R. Rabadan, Topology of viral evolution, Proc. Natl. Acad. Sci. U.S.A. 110 (2013) 18566–18571.

[5] P. Cordue, S. Linz, C. Semple, Phylogenetic networks that display a tree twice, Bull. Math. Biol. 76 (2014) 2664–2679.

[6] T. Dagan, Y. Artzy-Randrup, W. Martin, Modular networks and cumulative impact of lateral transfer in prokaryote genome evolution. Proc. Natl. Acad. Sci. U.S.A. 105 (2008) 10039–10044.

[7] W.F. Doolittle, Phylogenetic classification and the universal tree, Science 248 (1999) 2124–2128.

[8] D. Fernández-Baca, S. Guillemot, B. Shutters, S. Vakati, Fixed-parameter algorithms for finding agreement supertrees, SIAM J. Comput. 44 (2015) 384–410.

[9] A.R. Francis, M. Steel, Which phylogenetic networks are merely trees with additional arcs? Syst. Biol. 64 (2015) 768–777.

[10] L. Georgiadis, N. Parotsidis, Dominators in directed graphs: a survey of recent results, applications, and open problems, in: Proc. of ISCIM'13, 2013, pp. 15–20.

[11] P. Gambette, A.D.M. Gunawan, A. Labarre, S. Vialette, L.X. Zhang, Locating a tree in a phylogenetic network in quadratic time, in: Proc. of RECOMB'15, in: LNCS, vol. 9029, Springer, 2015, pp. 96–107.

[12] P. Gambette, A.D.M. Gunawan, A. Labarre, S. Vialette, L.X. Zhang, Solving the tree containment problem for genetically stable networks in quadratic time, in: Proc. of IWOCA'15, in: LNCS, vol. 9538, Springer, 2015, pp. 197–208. Springer, 2015.

[13] A.D.M. Gunawan, L.X. Zhang, Locating a tree in a reticulation-visible network in cubic time, in: Proc. of RECOMB'16, in: LNBI, vol. 9649, Springer, 2019, pp. 266–266. arXiv:1507.02119, 2015.

[14] D. Gusfield, ReCombinatorics: The Algorithmics of Ancestral Recombination Graphs and Explicit Phylogenetic Networks, MIT Press, Cambridge, USA, 2014.

[15] D. Harel, R.E. Tarjan, Fast algorithms for finding nearest common ancestors, SIAM J. Comput. 13 (1984) 338–355.

[16] K.T. Huber, L. van Iersel, V. Moulton, T. Wu, How much information is needed to infer reticulate evolutionary histories?, Syst. Biol. 64 (2015) 102–111.

[17] D.H. Huson, T.H. Klöpper, Beyond galled trees: decomposition and computation of galled networks, in: Proc. of RECOMB'07, in: LNCS, vol. 4453, Springer, 2007, pp. 211–225.

[18] D.H. Huson, R. Rupp, C. Scornavacca, Phylogenetic Networks: Concepts, Algorithms and Applications, Cambridge University Press, Cambridge, UK, 2011.

[19] I.A. Kanj, L. Nakhleh, C. Than, G. Xia, Seeing the trees and their branches in the network is hard, Theor. Comput. Sci. 401 (2008) 153–164.

[20] T. Lengauer, R.E. Tarjan, A fast algorithm for finding dominators in a flowgraph, ACM Trans. Prog. Lang. Sys. 1 (1979) 121–141.

[21] S. Linz, K.S. John, C. Semple, Counting trees in a phylogenetic network is #P-complete, SIAM J. Comput. 42 (2013) 1768–1776.

[22] T. Marcussen, S.R. Sandve, L. Heier, *et al.*: Ancient hybridizations among the ancestral genomes of bread wheat, Science 345 (2014) 288–291.

[23] B.M.E. Moret, L. Nakhleh, T. Warnow, C.R. Linder, A. Tholse, A. Padolina, J. Sun, R. Timme, Phylogenetic networks: Modeling, reconstructibility, and accuracy, IEEE-ACM Trans. Comput. Biol. Bioinform. 1 (2004) 13–23.

[24] L. Nakhleh, Computational approaches to species phylogeny inference and gene tree reconciliation, Trends Ecol. Evol. 28 (2013) 719–728.

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

[26] L. van Iersel, C. Semple, M. Steel, Locating a tree in a phylogenetic network, Inform. Process. Lett. 110 (2010) 1037–1043.

[27] L. Wang, K. Zhang, L.X. Zhang, Perfect phylogenetic networks with recombination, J. Comp. Biol. 8 (2001) 69–78.

[28] S.J. Willson, Regular networks can be uniquely constructed from their trees, IEEE-ACM Trans. Comput. Biol. Bioinform. 8 (2011) 785–796.

[29] L.X. Zhang, Y. Cui, An efficient method for DNA-based species assignment via gene tree and species tree reconciliation, in: Proc. of WABI'10, Springer, 2010, pp. 300–311.