Bioinformatics, 32, 2016, i503–i510
doi: 10.1093/bioinformatics/btw467
ECCB 2016

A program for verification of phylogenetic network models

Andreas D.M. Gunawan[1,†], Bingxin Lu[2,†] and Louxin Zhang[1,*]

[1]Department of Mathematics and [2]Department of Computer Science, National University of Singapore, Singapore 117417, Singapore

*To whom correspondence should be addressed.
†The authors wish it to be known that, in their opinion, the first two authors should be regarded as joint First Authors.

Abstract

Motivation: Genetic material is transferred in a non-reproductive manner across species more frequently than commonly thought, particularly in the bacteria kingdom. On one hand, extant genomes are thus more properly considered as a fusion product of both reproductive and non-reproductive genetic transfers. This has motivated researchers to adopt phylogenetic networks to study genome evolution. On the other hand, a gene's evolution is usually tree-like and has been studied for over half a century. Accordingly, the relationships between phylogenetic trees and networks are the basis for the reconstruction and verification of phylogenetic networks. One important problem in verifying a network model is determining whether or not certain existing phylogenetic trees are displayed in a phylogenetic network. This problem is formally called the tree containment problem. It is NP-complete even for binary phylogenetic networks.
Results: We design an exponential time but efficient method for determining whether or not a phylogenetic tree is displayed in an arbitrary phylogenetic network. It is developed on the basis of the so-called reticulation-visible property of phylogenetic networks.
Availability and Implementation: A C-program is available for download on http://www.math.nus.edu.sg/~matzlx/tcp_package.
Contact: matzlx@nus.edu.sg
Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Horizontal gene transfer and hybridization have occurred between organisms more frequently than commonly thought (Chan *et al.*, 2013; Treangen and Rocha, 2011). Accordingly, phylogenetic networks have been adopted for modeling these two genetic transfer events as well as other so-called reticulation events in evolutionary genomics (Doolittle, 1999; Moret *et al.*, 2004; Nakhleh, 2013). The computational and mathematical aspects of phylogenetic networks have been intensively studied over the past two decades (e.g. Gusfield, 2014; Huson *et al.*, 2011; Parida, 2010; Wang *et al.*, 2001).

On one hand, phylogenetic networks are very useful for dating and inferring reticulation events (Charlton *et al.*, 2008; Koblmüller *et al.*, 2007). On the other hand, it is extremely challenging to reconstruct network models correctly from sequence data or from gene trees (Huber *et al.*, 2015; Song *et al.*, 2005; Yu *et al.*, 2014). Given that phylogenetic trees of numerous gene families have been studied for over half a century, phylogenetic networks are often reconstructed and verified by examining their compatibility with existing gene trees (Cardona *et al.*, 2009; Huson *et al.*, 2011; Kanj *et al.*, 2008; van Iersel *et al.*, 2010b). This has motivated researchers to study the problem of determining whether or not a phylogenetic tree is consistent with a phylogenetic network. This problem is called the tree containment problem (TCP) (Kanj *et al.*, 2008; see also Huson *et al.*, 2011). The cluster containment problem (CCP) is another problem arising from the verification of phylogenetic models. It asks whether or not a cluster of taxa appears in a phylogenetic tree displayed in a network (Huson *et al.*, 2011).

Both the TCP and CCP are NP-complete (Kanj *et al.*, 2008), even on binary networks, in which tree nodes have an outdegree of two and reticulation nodes have an indegree of two (van Iersel *et al.*, 2010a). van Iersel *et al.* (2010a) first developed a polynomial time algorithm for solving the TCP on tree-child networks. Recently, polynomial time TCP algorithms have been published for phylogenetic networks with the reticulation-visible property (Bordewich and Semple, 2015; Gambette *et al.*, 2015; Gunawan *et al.*, 2016). The visibility property was originally introduced to capture an important feature of galled networks (Huson *et al.*, 2011). A network is reticulation-visible if every reticulation node separates the network root from at least some leaves.

The TCP algorithm of Gunawan *et al.* (2016) has been simplified into a quadratic-time algorithm by using proper data structures in the full version of the paper that is available online. One important technique used for designing the algorithm is a decomposition theorem. The technique also leads to a linear time algorithm for the CCP on reticulation-visible networks. Although our study of reticulation-visible networks is theoretically interesting, its application is limited, as a large portion of phylogenetic networks do not have the reticulation-visible property (Zhang, 2016).

In this work, a computer program for the TCP was developed from a generalization of the decomposition technique reported in Gunawan *et al.* (2016). The program solves the TCP on arbitrary networks in which nodes are not necessarily binary. Although it is of exponential time, the theoretical analysis shows that it is hundreds of times as fast as the naïve program for binary phylogenetic networks with 30 or more reticulation nodes. The evaluation tests on random networks demonstrate that it is fast enough to solve real instances arising in evolutionary genomics.

2 Algorithm

2.1 Basic concepts and notation

A phylogenetic network over a set of taxa X is an acyclic digraph in which (i) the leaves (i.e. nodes of outdegree zero) are bijectively mapped to the taxa in X and (ii) there is a unique node of indegree zero, called the root. The property (ii) implies there is a (directed) path from the root to each of the other nodes.

We also assume that a node is of outdegree one if it has indegree greater than one. These nodes are called *reticulation nodes*, representing reticulation events. Other non-leaf nodes are called the *tree nodes* of the network, which include the root and all the nodes of both indegree and outdegree one. A phylogenetic network is *reduced* if the unique child of each reticulation node is either a tree node or a leaf (Moret *et al.*, 2004). A phylogenetic network is said to be *bicombining* if each reticulation node is of indegree two. It is *binary* if all reticulation nodes are of indegree two and all tree nodes are of outdegree two. A *phylogenetic tree* is a rooted full binary tree or, equivalently, a binary phylogenetic network without reticulation nodes.

Let N be a phylogenetic network. In this paper, we use the following notation:

$\rho(N)$ is the root of N; $L(N)$ is the set of leaves in N; $T(N)$ is the set of tree nodes in N; $R(N)$ is the set of reticulation nodes in N; $V(N)$ is the set of all nodes (i.e. $L(N) \cup T(N) \cup R(N)$); $E(N)$ is the set of edges in N; N–S is the subnetwork with the node set $V(N) - V(S)$ and the edge set $\{(x, y) \in E(N) | \{x, y\} \subseteq V(N) - V(S)\}$ for a vertex subset S.

For $u, v \in V(N)$ such that $u \neq v$, u is a *parent* of v and, equivalently, v is a *child* of u if there is an edge from u to v. In general, v is a *descendant* of u if a direct path exists from u to v.

Consider a phylogenetic network M over a set of taxa X and a phylogenetic tree G over a set of taxa Y such that $Y \subseteq X$. After all but one incoming edges are removed for every reticulation node, M becomes a tree M' (Fig. 1). Note that some internal nodes of M may have become leaves in M'. After the nodes that are not in any path from $\rho(M)$ to a leaf in Y and all the edges incident to them are removed from M', M' becomes a tree M'' over Y. M'' is called a *subtree history* of the network M for Y. The tree G is *consistent* with M if G can be obtained from a subtree history of M for Y by contraction (i.e., contracting all the nodes with both indegree and outdegree one) (Fig. 1). If M is a tree without nodes of indegree and outdegree one over the same set of taxa as G, by definition, $M = M' = M''$ and so G will not be consistent with M unless they are identical. In this paper, we study the following model verification problem.

2.1.1 Tree containment

Instance: A phylogenetic network N over X and a phylogenetic tree G over Y such that $Y \subseteq X$.

Question: Is G consistent with N?

Here, we would like to point out that the TCP has been studied only for a phylogenetic network and a phylogenetic tree over the same set of taxa in literature (Huson *et al.*, 2011).

Since the TCP is NP-complete (Kanj *et al.*, 2008), we aim to develop a program that takes exponential time in the worst case but fast enough to check whether or not a phylogenetic tree is consistent with a phylogenetic network arising from the study of a gene's evolution.

2.2 Networks with dummy nodes

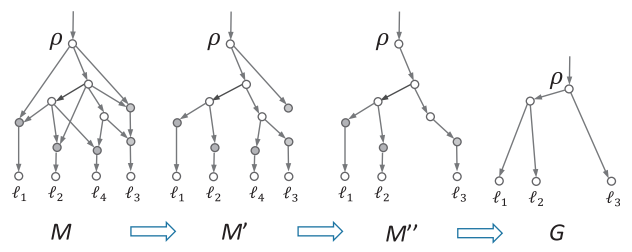
A dummy reticulation node is a node that has indegree greater than one and outdegree zero, and a dummy leaf is an unlabeled leaf. As an evolutionary model, a phylogenetic network is assumed not to have such nodes, as they are not meaningful. However, the algorithm to be developed works in a recursive manner. Occasionally, the input network will be simplified into a network with such kind of nodes when the algorithm runs. Therefore, for convenience, in this work, we allow a phylogenetic network to have dummy reticulation nodes and leaves. Such a network is called a *general phylogenetic network*. We define the consistency of a phylogenetic tree with a general phylogenetic network similarly.

For a phylogenetic network N over X and $Y \subset X$, we use N_Y to denote the general phylogenetic network obtained from N by removing all the leaves not labeled by a taxa in Y. It is not hard to see that N_Y is over Y. It is also true that a phylogenetic tree G over Y is consistent with N if and only if G is consistent with N_Y.

In the rest of this paper, because of the above fact, we shall investigate the TCP for general phylogenetic networks and phylogenetic trees over the same set of taxa.

2.3 Tree components

In this subsection, we introduce some facts about the structural decomposition of phylogenetic networks studied in Gunawan *et al.* (2016). These facts are needed for developing our TCP algorithm in next subsection.



ρ
ℓ_1 ℓ_2 ℓ_4 ℓ_3
M
ρ
ℓ_1 ℓ_2 ℓ_4 ℓ_3
M'
ρ
ℓ_1 ℓ_2 ℓ_3
M''
ρ
ℓ_1 ℓ_2 ℓ_3
G

Fig. 1. Illustration of tree containment. M is a phylogenetic network over taxa $\{\ell_1, \ell_2, \ell_3, \ell_4\}$. Reticulation nodes are represented by filled circles. An incoming edge for the root is added for visualization. G is a phylogenetic tree over $\{\ell_1, \ell_2, \ell_3\}$ that is displayed in M

Let N be a network over a set of taxa X. For $u, v \in V(N)$, v is a *vertical descendant* of u if a direct path exists from u to v that contains either a single edge from u or multiple edges, which all enter a node in $T(N) \cup L(N)$. Let $\mathrm{VD}(u)$ denote the set of all vertical descendants of u. If u is a dummy reticulation node, $\mathrm{VD}(u)$ is empty.

Let $r \in R(N)$ have a unique child $c(r)$. If $c(r)$ is also in $R(N)$, $\mathrm{VD}(r) = \{c(r)\}$. If $c(r) \notin R(N)$, then $\mathrm{VD}(r) \subseteq T(N) \cup L(N)$. Additionally, $\mathrm{VD}(r)$ induces a subtree of N with the vertex set $\mathrm{VD}(r)$ and the edges set $\{(v', v'') \in E(N) | v', v'' \in \mathrm{VD}(r)\}$. This subtree is denoted by $C(r)$ and is called a *tree component* of N. The subtree induced by $\{\rho\} \cup [\mathrm{VD}(\rho) \cap (T(N) \cup L(N))]$ is also called a tree component, denoted $C(\rho)$, where $\rho = \rho(N)$. The concept of a tree component is illustrated in Figure 2.

A tree component is *trivial* if it contains only a network leaf or if it is empty. A non-trivial tree component must contain some internal nodes. If N is reduced, for each reticulation node, its child is either a tree node or a leaf. This implies that all the tree components form a disjoint partition of $T(N) \cup L(N)$ if N is reduced.

Let $u \in V(N)$. The node u is said to be *visible* if there is a leaf ℓ such that every path from $\rho(N)$ to ℓ contains u. It is not hard to see that $\rho(N)$ is visible with respect to every leaf.

A tree component C is *visible* if the root of C is visible. In the network given in Figure 2, C_0 is visible with respect to every leaf; C_1 is visible with respect to ℓ_1. However, C_2 and C_3 are not visible. N is *reticulation-visible* if every tree component is visible.

Let $r', r'' \in R(N) \cup \{\rho(N)\}$. We say that r'' and its tree component $C(r'')$ are right below $C(r')$ if r'' is the child of a node in $C(r')$. In Figure 2, C_3 is right below both C_1 and C_2.

A tree component $C(r)$ is *exposed* if it contains only a single leaf or if every tree component right below it is trivial in N. In Figure 2, only C_3 is an exposed component containing more than one node.

Lemma 2.1. *Let C be an exposed tree component. C is visible if and only if C contains a leaf or if a reticulation node r exists right below C such that the parents of r are all in C.*

Proof. Let u be the root of C. If C contains a leaf x, then a unique path P exists from u to x that contains only tree nodes. Consider a path P' from the network root to x. If u does not appear in P', P' and P intersects at a reticulation node in P, as x is in them. This is impossible. Hence P' must contain u, implying that u is visible with respect to x. Similarly, we can prove that u is visible with respect to the child of a reticulation node y if all the parents of y are in C. By definition, we have proved that C is visible.

Conversely, let C be visible with respect to a leaf x and let x be not in C. Since C is exposed, x is the child of a reticulation node r right below C. Assume a parent y of r is not in C. Since C is exposed, y is not below C, as there is a non-trivial component below C otherwise. This implies that any path from the network root to x through y does not contain the root of C, contradicting the visibility assumption on C. This completes the proof.

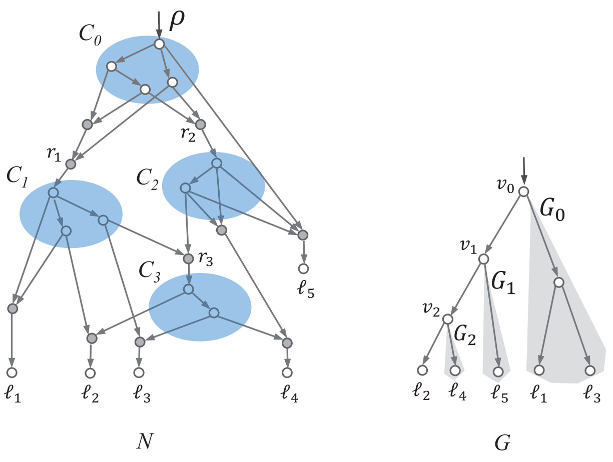


Fig. 2. A network N and a phylogenetic tree G displayed in N. *Left panel*: C_0 is the tree component containing the network root; $C_i = C(r_i)$ for $1 \le i \le 3$. For each of the other reticulation nodes, the tree component contains only its unique child. *Right panel*: G_0, G_1 and G_2 are subtrees branching off the path from the root v_0 to ℓ_2

2.4 Description of the algorithm

In this section, we shall describe how to generalize the TCP algorithm for reticulation-visible networks, appearing in Gunawan *et al.* (2016), into one that works for arbitrary networks. We first introduce notation and facts on which the algorithm is based and then present the algorithm in Figure 4.

Let N and G be a network and a phylogenetic tree over the same set of taxa X, respectively (Fig. 2). In the rest of this paper, for simplicity, we use the same symbol to denote the leaf that represents the same taxa in G and N.

It is not hard to see that N contains at least one exposed non-trivial tree component. Consider $r \in R(N)$ such that $C(r)$ is non-trivial, exposed and visible. Let L_r be the set of network leaves with respect to which $C(r)$ is visible and let $\ell \in L_r$. A unique path then exists from the tree root $\rho(G)$ to ℓ in G, say

$$P_\ell : v_0 = \rho(G), v_1, \cdots, v_t, v_{t+1} = \ell, \tag{1}$$

where $t \ge 0$. P_{ℓ_2} is shown in the tree G in Figure 2, where $t = 2$. It is not hard to see that $G - P_\ell$ is a union of $t+1$ disjoint trees G_i, each of which branches off from P_ℓ at v_i for $i = 0, \cdots, t$. For example, in Figure 2, G_0 consists of ℓ_1, ℓ_3 and their parent; G_1 and G_2 are simply ℓ_5 and ℓ_4, respectively. For convenience, we set $G_{t+1} = \{\ell\}$.

Define

$$s_G(r) = \min\{\ i | 0 \le i \le t+1, L(G_i) \cap L_r \ne \emptyset\}, \tag{2}$$

where $L(G_i)$ denotes the set of labeled leaves in G_i. Since $\ell \in L(G_i) \cap L_r$, the index $s_G(r)$ is well defined. For G in Figure 2, if $L_r = \{\ell_2, \ell_5\}$, we have $s_G(r) = 1$, as G_1 contains ℓ_5 in L_r but G_0 contains neither ℓ_2 nor ℓ_5; if $L_r = \{\ell_2\}$, then $s_G(r) = 3$.

The index $s_G(r)$ can be computed by calling a simple dynamic programming algorithm on G and L_r, which takes linear time $O(|L(G)|)$ (see Gunawan *et al.*, 2016).

For each v_i, we use $G'(v_i)$ to denote the subtree rooted at v_i in G. Let $[r]_N$ denote the subnetwork consisting of all the descendants of r and the edges between them in N. Formally, for each $s_G(r) \le i \le t+1$, $G'(v_i)$ is said to be displayed in the subnetwork below r, if it is consistent with the following subnetwork:

$[r]_N - \{c(r'), r' | r' \in R(N)$ such that $c(r') \notin L(G'(v_i))\}$, where $c(r')$ is the unique leaf child for $r' \in R(N)$.

If G is displayed by N, the subtree $G'(v_{s_G(r)})$ must be displayed in the subnetwork of N below r. The reason for this is because r and $C(r)$ are visible with respect to a leaf $\ell' \in L_r \cap L(G_{s_G(r)})$, as well as ℓ, forcing $v_{s_G(r)}$ to correspond to a node in $C(r)$ in any display of G in N.

However, a super subtree containing $G'(v_{s_G(r)})$ may possibly be displayed below r in N. Let us define

$$d_G(r) = \min\{j | G'(v_j) \text{ is displayed below } r \text{ in } N\}. \tag{3}$$

The index $d_G(r)$ can be found in quadratic time (Gunawan *et al.*, 2016).

Consider the network N' in Figure 3, $C(r_3) = C_3$. It is visible with respect to ℓ_2 and exposed. For the tree G in Figure 2, since $L_{r_3} = \{\ell_2\}$, $s_G(r_3) = 3$. However, $d_G(r_3) = 2$, as $G'(v_2)$ is displayed in the subnetwork below r_3.

Let $N - C(r) + \ell$ denote the network obtained from N as follows:

- for each $x \in R(N)$ right below $C(r)$, remove the edges to x from its parents in $C(r)$ if the child of x is not in the subtree $G'(v_{d_G(r)})$, and
- replace the resulting subnetwork below r with ℓ so that ℓ becomes the child of r.

Similarly, we use $G - G'(v_{d_G(r)}) + \ell$ to denote the tree obtained from G by replacing $G'(v_{d_G(r)})$ with ℓ.

For $C(r_3) = C_3$ and ℓ_2 in the network N' in Figure 3 and G in Figure 2, since $d_G(r_3) = 2$, $N' - C_3 + \ell_2$ and $G - G'(v_2) + \ell_2$ are those shown in the second row in Figure 3.

The following theorem was first proved for reticulation-visible networks by Gunawan *et al.* (2016). Its correctness for arbitrary networks is proved in the Appendix.

Theorem 2.1.: *Assume $C(r)$ is non-trivial, exposed and visible with respect to ℓ in N. Let $N - C(r) + \ell$, $G - G'(v_{d_G(r)}) + \ell$, $s_G(r)$, and $d_G(r)$ be defined above.*

i. *If $d_G(r) > s_G(r)$, N does not display G.*
ii. *If $d_G(r) \le s_G(r)$, N displays G if and only if $N - C(r) + \ell$ displays $G - G'(v_{d_G(r)}) + \ell$.*

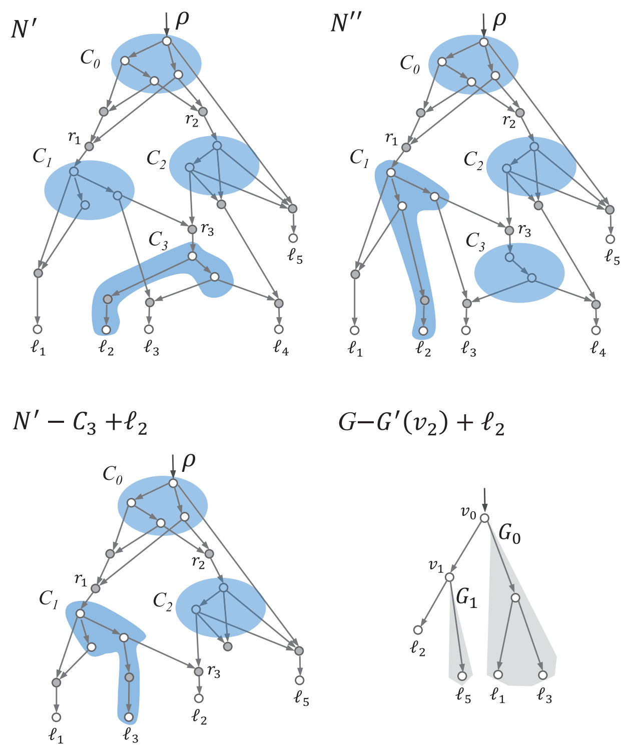


Fig. 3. Illustration of the TCP algorithm. *Top left panel*: the network N' obtained from N (Fig. 2) by deleting the edge entering the parent of ℓ_2 from a node in C_1. Note that C_3 is visible with respect to ℓ_2 in N'. *Top right panel*: the network N'' obtained from N (Fig. 2) by deleting the edge entering the parent of ℓ_2 from a node in C_3. *Bottom left panel*: the network derived from N' by replacing C_3 with ℓ_2 after the subtree $G'(v_2)$ is found to be displayed below r_3. *Bottom right panel*: the tree obtained from G (Fig. 2) by replacing $G'(v_2)$ with ℓ_2. $G'(v_2)$ is the subtree of G consisting of ℓ_2, ℓ_4 and its parent v_2 (Fig. 2)

Consider $r \in R(N)$ such that $C(r)$ is exposed and non-trivial. Assume that $C(r)$ is not visible but a reticulation node r' exists right below $C(r)$ such that $C(r')$ consists of only a leaf. Since $C(r)$ is not visible, r' has at least one parent not in $C(r)$.

We define

$$N' = N - \{(x, r') \in E(N) | x \notin C(r)\} \tag{4}$$

and

$$N'' = N - \{(x, r') \in E(N) | x \in C(r)\}. \tag{5}$$

If r' has only one parent in $C(r)$ in N, r' is of indegree and outdegree one in N'. In this case, r' becomes a tree node and is merged into $C(r)$ in N' (Figure 3). If r' has multiple parents in $C(r)$, $C(r')$ is only right below $C(r)$ in N'. Hence, $C(r)$ has become visible with respect to the unique leaf child of r' in N'.

In the network N in Figure 2, C_3 is exposed, but not visible. If the parent of ℓ_2 is chosen, N' and N'' are shown in the top row in Figure 3.

By the definition of tree containment, a tree is in N if and only if it is in either N' or N''.

By combining the two possible cases discussed above, we obtain an algorithm for solving the TCP summarized in Figure 4.

On the input network N and tree G in Figure 2, the algorithm runs as follows:

1. Since C_3 is exposed and non-trivial, C_3 is selected in Step 1.
2. Since C_3 is not visible, Step 3 is executed. If we assume the leaf ℓ_2 is chosen, the resulting networks N' and N'' are given in Figure 3. Hence, the algorithm runs recursively on N' and then on N''. In this example, the algorithm outputs TRUE and exits before running on N''.

TCP ALGORITHM

```
Input: An arbitrary network $N$ and a phylogenetic tree $G$.
Output: TRUE if $N$ displays $G$ and FALSE otherwise.

0. If $|V(N)| \le 1$, output TRUE if $N = G$ and FALSE otherwise;
1. Select an exposed and non-trivial tree component $C(r)$,
   where $r$ is a reticulation node;
2. If $C(r)$ is visible with respect to a leaf $\ell$
    2.1. Compute $s_G(r)$ and $d_G(r)$ as defined in Eqn.(2) and (3);
    2.2. If $d_G(r) > s_G(r)$, output FALSE;
    2.3. If $d_G(r) = 0$, output TRUE;
    2.4. If $0 < d_G(r) \le s_G(r)$ do {
        (i) Compute $N_r = N - C(r) + \ell$ and
            $G_r = G - G'(v_{d(r)}) + \ell$ as defined above;
        (ii) Output TRUE if $N_r$ displays $G_r$ and FALSE if not;
    }
3. If $C(r)$ is not visible but a leaf exists below $C(r)$, do {
    3.1. Compute $N'$ and $N''$ as defined in Eqn. (4) and (5);
    3.2. If $N'$ displays $G$, output TRUE;
    3.3. If $N''$ displays $G$, output TRUE;
    3.4. Output FALSE;
    }
4. If no leaf exists below $C(r)$, do {
    4.1. If $N - r - C(r)$ displays $G$, output TRUE;
    4.2. Output FALSE;
    }
```

Fig. 4. A TCP algorithm for arbitrary networks

3. In N', C_3 is visible with respect to ℓ_2, Hence, Step 2 is executed. The indices $s_G(\ell_2)$ and $d_G(\ell_2)$ are first computed. Their values are 3 and 2. Since $0 < d_G(\ell_2) < s_G(\ell_2)$, N' is simplified into $N' - C_3 + \ell_2$ and G into $G - G'(v_2) + \ell_2$, shown in the bottom row in Figure 3. The algorithm is then called on the simplified network and tree.
4. Since C_1 is exposed, non-trivial and visible with respect to ℓ_1, Step 2 is executed again on $N' - C_3 + \ell_2$, eliminating C_1 from the network and ℓ_1, ℓ_3 from the tree $G - G'(v_2) + \ell_2$.
5. Note that C_0, C_2 and the tree component $\{r_1\}$ remain in $N''' = (N' - C_3 + \ell_2) - C_1 + \ell_1$ and the tree has become a tree with three leaves: ℓ_1, ℓ_2 and ℓ_5. Here, the tree component $\{r_1\}$ is the one associated with the reticulation parent of r_1. In N''', C_2 is visible and exposed, Step 2 will be executed to dissolve C_2. Step 3 will then be executed when the tree component $\{r_1\}$ is examined, followed by the execution of Step 2 for dissolving C_0 before outputting TRUE.

2.5 Implementation of the algorithm

A couple issues arising during implementation of our algorithm are worth mentioning here. First, since the algorithm decodes tree components one by one, the input network and any additional networks created in Step 3 of the algorithm are each represented as a sorted list of non-trivial tree components. The non-trivial tree components are sorted in post order so that after the tree components listed before a tree component is dissolved, the latter becomes exposed. In this way, the non-trivial components are dissolved from the first to the last. When a tree component is dissolved, each tree component listed behind it will be updated if a reticulation node below the former is also below the latter.

Our evaluation tests showed that the running time of our algorithm was very sensitive to the order in which the tree components were listed. Ideally, visible tree components with more nodes should be listed first whenever possible.

We used an array to save the leaf below each reticulation node for which the tree component is a single leaf. The list is updated at the end of Step 2.4(i) when an exposed and visible tree component is replaced by a network leaf.

Second, we used a two-dimensional table to implement a dynamic programming method for computing $d_G(r)$, defined in Equation (3). The details of the dynamic programming method can be found in Gunawan *et al.* (2016).

3 Analyses of the algorithm's performance

In this section, we first analyze the worst-case time complexity of the algorithm for bi-combining networks. We then report its performance on one of the largest networks in the literature and on random networks.

3.1 Measuring the efficiency of the algorithm

Our algorithm is a recursive algorithm by nature. Here are some useful observations on its efficiency.

First, the running time of the algorithm is proportional to the number of non-trivial tree components. If the input network N has only one non-trivial tree component, this tree component is rooted at the network root and hence must be visible with respect to each leaf, implying that the network is reticulation-visible. In this case, both Steps 3 and 4 will not be executed; Step 2 of the algorithm is executed only once, taking $O(|E(N)||L(G)|)$ time (Gunawan *et al.*, 2016), where $|\cdot|$ denotes the cardinality of a finite set.

Second, when Step 3 is executed, the algorithm will run on two simplified copies of the current network in sequential order. According to Gunawan *et al.* (2016), the time spent on each exposed and visible tree component C is $O(|E_C||L(G)|)$, where $E_C = \{(u, v) \in E(N) | u \in C\}$. Let m denote the number of times Step 3 is executed on the input N and G. The total running time of the algorithm is bounded above by $O((m+1)|E(N)||L(G)|)$. Since the TCP is NP-complete, m is likely an exponential function of the number of reticulation nodes in N, $|R(N)|$, unless NP = P.

Third, the input network N and tree G satisfy $L(G) \subseteq L(N)$. Hence, we simply use $|L(N)|$ to replace $|L(G)|$ in the analysis of the time complexity of the algorithm. In this way, the time complexity of our algorithm is written as $O((m+1)|E(N)||L(N)|)$.

Recall that a network is reduced if there are no consecutive reticulation nodes along a path. Given G and reduced N as an input, the naïve algorithm will consider all the reticulation nodes one by one. For each reticulation node r, there will be k_r possibilities if the indegree of r is k_r. Therefore, the naïve algorithm will create $\prod_{r \in R(N)} k_r$ trees and will then take linear time to determine whether or not G is consistent with each of them.

If N is bi-combining, each reticulation node has indegree 2, and hence $\prod_{r \in R(N)} k_r = 2^{|R(N)|}$. Combining this fact and the third point made above, we propose to measure the efficiency of our algorithm by comparing $\log_2(m)$ against $|R(N)|$. The former is denoted by $b(N, G)$ and is called the *effective reticulation number* of the algorithm for N and G. Additionally, $b(N) = \max_G b(N, G)$.

3.2 A theoretic bound for bi-combining networks

The analysis of our algorithm is difficult in general. However, we are able to establish the following results for bi-combining reduced networks.

Theorem 3.1. *For any bi-combining reduced network* N, $b(N) \le \log_2\left(\frac{1+\sqrt{5}}{2}\right) \times |R(N)| \approx 0.694|R(N)|$.

Proof. Let $m(N)$ be the largest possible number of times Step 3 is executed on N and any phylogenetic tree. When Step 3 is first executed on N, the two networks N' and N'' are created. For these three networks

$$m(N) \le 1 + m(N') + m(N''). \tag{6}$$

Let us assume that N' and N'' are created by examining an exposed non-trivial tree component C and a reticulation node r right below C. Since the tree component C is exposed, the child of r is a network leaf, say ℓ. Because C is neither trivial nor visible, another reticulation node r' exists right below C. By the construction of N', C is visible with respect to ℓ in N'. Hence, when the algorithm is called on N', Step 2 should first be executed, eliminating all the reticulations below C including r' simultaneously. Additionally, r has become a tree node in N' and N''.

Let $M(k)$ denote the largest possible number of times Step 3 is executed on any bi-combining reduced network with k reticulation nodes and any phylogenetic tree, namely

$$M(k) = \max_{N \in S}\{m(N) | |R(N)| = k\},$$

where S is the set of bi-combining reduced networks. Here, clearly, $M(1) \le 0$ and $M(2) \le 1$. By Equation (6), this discussion implies that

$$M(k) + 1 \le [M(k-2) + 1] + [M(k-1) + 1].$$

Since the Fibonacci numbers F_k satisfy $F_0 = 1$ and $F_1 = 1$ and $F_k = F_{k-1} + F_{k-2}$ $(k \geq 2)$ (Graham *et al.*, 2002), we have the following inequality:

$$M(k) + 1 \leq F_k \leq \frac{1}{\sqrt{5}}\left(\frac{1}{2}(1+\sqrt{5})\right)^k + \frac{1}{2},$$

implying that

$$b(N) \leq \lceil \log_2 M(|R(N)|) \rceil \leq \log_2\left(\frac{1}{2}(1+\sqrt{5})\right) \times |R(N)|.$$

The above theorem indicates that Step 3 of our algorithm is executed $2^{0.694|R(N)|}$ times at most on an input network N and a phylogenetic tree. Hence, the algorithm has time complexity $O\left(2^{0.694|R(N)|}|E(N)||L(N)|\right)$.

3.3 Performance for a network in the literature

Charlton *et al.* (2008) reconstructed an ancestral recombination graph with as many as 32 reticulation nodes over seven taxa to study the evolution of double-stranded RNA in fungi (Gusfield, 2014, p. 325). We redrew this network (Supplementary Fig. S1) and call it A. In this subsection, we report $b(A, G)$ for each of the 10 395 possible phylogenetic trees over the seven taxa.

The distribution of $b(A, G)$ is presented in Table 1. There are 4255 trees consistent with the model A. The effective reticulation numbers $b(A, G)$ for these trees vary from 4 to 14. However, $b(A, G)$ for the 6140 trees inconsistent with A ranges narrowly from 10 to 14.

A is bi-combining and has 77 nodes (32 reticulation nodes, 38 tree nodes and 7 leaves). Table 1 shows that for each tree, the effective reticulation number is less than 32/2. Hence, based on our theoretical analysis, the program is roughly 851 ($= 2^{16}/77$) times as fast as the naïve approach.

3.4 Performance for random networks

To evaluate our program, we ran the program over thousands of phylogenetic trees and networks with 5–30 leaves on a cluster with 32 GB RAM and 8 cores. On one hand, since the spaces of trees and networks with 10 or more leaves are both huge, it was very hard to have unbiased evaluation. For instance, for many random networks over 20–30 leaves generated by a computer, it is impossible to run our program on every tree against each of them. On the other hand, since there were only a small fraction of random trees displayed in each generated random network, our program finished very quickly for most trees. Additionally, it is impossible to run the naïve method on a network with 30 reticulation nodes on any computer currently. These facts prevented us to have a clear picture of our program's performance on the entire spaces of trees and networks.

Table 1. The distribution of $b(A, G)$ in the space of phylogenetic trees over the same set of taxa as A

	No. of trees consistent with A	No. of inconsistent trees
4	63	
5	561	
6	278	
7	544	
8	411	
9	478	
10	659	
11	766	40
12	433	416
13	62	5352
14		322

Each entry is the number of trees with $b(A, G)$ being equal to the corresponding number in the first column.

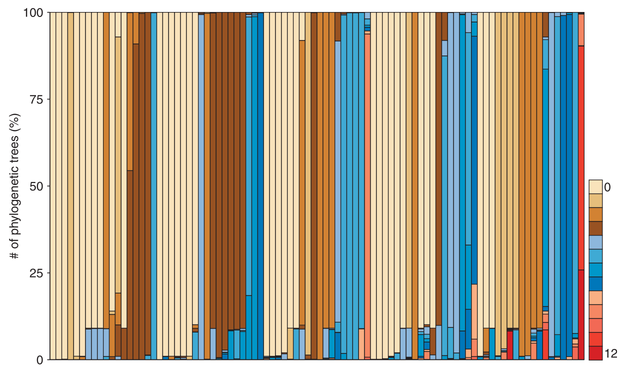


Fig. 5. Summary of the performance of the program. The data were collected from our test on 90 random networks with seven leaves, which were divided into five groups. Each group contained 18 networks with the same number of reticulation nodes, arranged roughly in increasing order of the smallest effective reticulation number in a row along the *X*-axis. The five groups were arranged from left to right in increasing order of the number of reticulation nodes. Each of the stacked bars in a column represents the percentage of trees that had the same effective reticulation number when the program ran on them against the corresponding network

Here, we report the performance of our program on five groups of random networks with seven leaves. Each group contained 18 random networks generated by a computer program designed by the last author's group. The networks in the kth group had $5(1+k)$ reticulation nodes, for each k from one to five. The percentages of the trees in the entire tree space with the same effective reticulation numbers were calculated and summarized in Figure 5. There are 10 395 phylogenetic trees with seven leaves.

We obtained several facts about our test. For all but four networks, over 90% of phylogenetic trees had an effective reticulation number of seven or less. There was a network with 20 reticulation nodes for which about 95% of phylogenetic trees had the effective reticulation number of 9 (represented by the orange bar in the middle). For the rightmost network with 30 reticulation nodes, almost every tree had an effective reticulation number of 9 or more. These facts strongly suggest that the effective reticulation number is at most half the number of reticulation nodes in the network for each tree and each network.

In terms of CPU time, the whole test on $5 \times 18 \times 10,395$ network-tree pairs took 18 h and 38 min, implying the program took 7.2 cs on an average for each network-tree pair.

4 Conclusion

The TCP is one of the important problems in the verification of phylogenetic models. This problem was surprisingly proved to be solvable in quadratic time for reticulation-visible networks recently (Gunawan *et al.*, 2016). In this work, we have implemented a fast program for solving the TCP for arbitrary networks by taking advantage of the techniques developed in that work.

The program can be straightforwardly extended to measure the difference of two networks by identifying phylogenetic trees that are in one but not in the other (Huson *et al.*, 2011). The program and its extensions will be integrated into the freely available

Dendroscope software (Huson *et al.*, 2007). Doubtless, they are valuable for application of networks in evolutionary genomics.

Acknowledgements

We thank Daniel Huson for helpful discussion on different issues of the verification of evolutionary network models and anonymous reviewers for helpful comments on the first submission. We also thank Philippe Gambette for creating an on-line tool for drawing networks. The tool was extremely useful for debugging our program.

Funding

The Singapore Ministry of Education Academic Research Fund MOE2014-T2-1-155.

Conflict of Interest: none declared.

References

Bordewich,M. and Semple,C. (2015) Reticulation-visible networks. *Adv. Appl. Math.*, **78**, 114–141 arXiv:1508.05424

Cardona,G. *et al.* (2009) Comparison of tree-child phylogenetic networks. *IEEE-ACM Trans. Comput. Biol. Bioinform.*, **6**, 552–569.

Chan,J.M. *et al.* (2013) Topology of viral evolution. *Proc. Natl. Acad. Sci. U.S.A.*, **110**, 18566–18571.

Charlton,N.D. *et al.* (2008) Phylogenetic relatedness of the M2 double-stranded RNA in *Rhizoctonia* fungi. *Mycologia*, **100**, 555–564.

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

Gambette,P. *et al.* (2015) Solving the tree containment problem for genetically stable networks in quadratic time. In: *Proceedings of 19th Annual International Conference on Research in Computational Molecular Biology*, Springer, Berlin, Germany, pp. 96–107.

Graham,R.L. *et al.* (2002) *Concrete Mathematics*, Pearson Education, Reading, USA.

Gunawan,A.D.M. *et al.* (2016) Locating a tree in a reticulation-visible network in cubic time. In *Proceedings of the 20th Annual International Conference on Research in Computational Molecular Biology*, Los Angeles, USA, pp. 266. The journal version is available online: arXiv:1603.08655.

Gusfield,D. (2014) *ReCombinatorics: The Algorithmics of Ancestral Recombination Graphs and Explicit Phylogenetic Networks*. MIT Press, Cambridge, USA.

Huber,K.T. *et al.* (2015) How much information is needed to infer reticulate evolutionary histories? *Syst. Biol.*, **64**, 102–111.

Huson,D. *et al.* (2007) Dendroscope: an interactive viewer for large phylogenetic trees. *BMC Bioinformatics*, **22**, 460.

Huson,D.H. *et al.* (2011) *Phylogenetic Networks: Concepts, Algorithms and Applications*. Cambridge University Press, Cambridge, UK.

Kanj,I.A. *et al.* (2008) Seeing the trees and their branches in the network is hard. *Theor. Comput. Sci.*, **401**, 153–164.

Koblmüller,S. *et al.* (2007) Reticulate phylogeny of gastropod-shell-breeding cichlids from Lake Tanganyika: the result of repeated introgressive hybridization. *BMC Evol. Biol.*, **7**, 7.

Moret,B.M.E. *et al.* (2004) Phylogenetic networks: modeling, reconstructibility, and accuracy. *IEEE-ACM Trans. Comput. Biol. Bioinform.*, **1**, 13–23.

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

Parida,L. (2010) Ancestral recombinations graph: a reconstructability perspective using random-graphs framework. *J. Comput. Biol.*, **17**, 1345–1370.

Song,Y.S. *et al.* (2005) Efficient computation of close lower and upper bounds on the minimum number of recombinations in biological sequence evolution. *Bioinformatics*, **21**, i413–i422.

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

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

van Iersel,L. *et al.* (2010b) Phylogenetic networks do not need to be complex: using fewer reticulations to represent conflicting clusters. *Bioinformatics*, **24**, i124–i131.

Wang,L. *et al.* (2001) Perfect phylogenetic networks with recombination. *J. Comp. Biol.*, **8**, 69–78.

Yu,Y. *et al.* (2014) Maximum likelihood inference of reticulate evolutionary histories. *Proc. Natl. Acad. Sci. USA.*, **111**, 16448–16453.

Zhang,L.X. (2016) On tree-based phylogenetic networks. *J. Comput. Biol.*, **23**, 553–565.

Appendix

A Proof of Theorem 2.1

For convenience, we simply set $s = s_G(r)$ and $d = d_G(r)$.

(i) The fact is equivalent to that if N displays G, then $d \leq s$.

Assume that N displays G. By definition, a subtree history M of N exists such that G is obtained from M by contracting nodes of indegree and outdegree one. Without loss of generality, we assume the root of M is the network root, i.e., $\rho(M) = \rho(N)$.

Since r is visible with respect to ℓ, the unique path $\rho(M)$ to ℓ in M must contain r. $G'(v_{t+1})$ is displayed below r. The proof is complete if $s = t + 1$.

If $s < t + 1$, by the definition of s, $G(s)$ contains a leaf $\bar{\ell} \neq \ell$ such that r is also visible with respect to $\bar{\ell}$. Thus, r is also in the unique path from $\rho(M)$ to $\bar{\ell}$. As the least common ancestor of ℓ and $\bar{\ell}$, v_s must be mapped to a node u below r in M. Hence, $G'(v_s)$ can be obtained from the substree of M rooted at u by contraction. Hence, $s \leq d$. The proof of the fact (i) is completed.

(ii) Assume $d \leq s$. Note that $G'(v_s)$ is a subtree of $G'(v_d)$.

(*Sufficiency*) Let $N_r = N - C(r) + \ell$ and $G_r = G - G'(v_d) + \ell$.

Assume that N_r displays G_r. On one hand, a subtree history M_r of N_r exists such that G_r can be obtained from M_r by contraction.

On the other hand, since $G'(v_d)$ is displayed below r, a subtree history M'' exists in $[r]_N - \{c(r'), r' | r' \in R(N) \text{ s.t. } c'(r) \notin L(G'(v_d))\}$ from which $G'(v_d)$ can be obtained by contraction. Let M' be the tree obtained from M_r by replacing the leaf ℓ with M''.

Clearly, M' is a subtree history of N and G can be obtained from M' by contraction. Hence, N displays G.

(*Necessity*) Assume that a subtree history M' of N exists from which G can be obtained by contraction. If $\rho(M') \neq \rho(N)$, there is a path from $\rho(N)$ to $\rho(M')$ consisting of nodes of indegree and outdegree one in M'. Without loss of generality, we assume that $\rho(M') = \rho(N)$.

Let v_d correspond to a node $u \in V(M')$. Since v_d is an ancestor of ℓ, u is in the path P from $\rho(N)$ to ℓ. Since r is visible with respect to ℓ, r and its child $c(r)$ are both in the path P. Hence u is either above r or below $c(r)$, the root of the tree component $C(r)$.

If u is below $c(r)$, by the definition of d, v_{d+1} must correspond to node above $c(r)$ in M'. Let M'_r be the subtree obtained from M' by replacing $M'(c(r))$ with ℓ, where $M'(c(r))$ is the subtree rooted at the unique child $c(r)$ of r. Clearly, M'_r is a subtree history of N_r. In addition, G_r can be obtained from M'_r by contraction. The proof is completed for the case that u is below $c(r)$.

If u is above r and hence strictly above $c(r)$ in M'. Let M'' be the subtree history of the following subnetwork below r

$$[r]_N - \{c(r'), r' | r' \in R(N) \text{ such that } c'(r) \notin L(G'(v_d))\},$$

from which $G'(v_d)$ can be obtained by contraction. Let $P(u, c(r))$ be the path from u to $c(r)$ in M'. We consider

$$M = M' - M'(u) + P(u, c(r)) + M'',$$

where $M'(u)$ is the subtree of M' rooted at u. Clearly G can be obtained from M by contraction. Since M'' corresponds to $G'(v_d)$, the subtree obtained from $M' - M(u) + P(u, c(r))$ by replacing $c(r)$ with a leaf labeled with ℓ is a subtree history of N_r from which G_r can be obtained by contraction. This finishes the proof.