Locating a Tree in A Phylogenetic Network in Quadratic Time

Philippe Gambette[a], Andreas D.M. Gunawan[b], Anthony Labarre[a],
Stéphane Vialette[a], Louxin Zhang[b]
[a]LIGM, Univ. Paris-Est Marne-la-Vallée
[b]Department of Mathematics, National Univ. of Singapore

Abstract

A fundamental problem in the study of phylogenetic networks is how to determine whether a phylogenetic network displays a phylogenetic tree or not. We develop a quadratic-time algorithm for this problem for binary nearly-stable phylogenetic networks. We also show that the number of reticulations in a reticulation visible or nearly stable phylogenetic network is bounded by a linear function of the number of taxa.

1 Introduction

Genetic materials are transferable between organisms by hybridization, recombination and horizontal gene transfer besides traditional reproduction. Recent studies in comparative genomics suggest that these "lateral" processes are a driving force in evolution that shapes the genome of a species [3, 11, 16]. Accordingly, phylogenetic networks have commonly been used to model reticulate evolutionary histories of species [3, 4, 10]. A plethora of methods for reconstructing reticulate evolutionary histories of species and related algorithmic issues have extensively been studied over the past two decades [5, 6, 12, 13, 18].

A phylogenetic network is an acyclic digraph with a set X of labeled leaves (that is, vertices of outdegree zero) and a root (having indegree zero). The leaves correspond one-to-one to a collection of taxa under study, whereas the unique root represents their least common ancestor. Vertices with indegree one represent speciation events. Vertices of indegree at least two represent an evolutionary process by which genetic material was horizontally transfered from one species to another.

A fundamental question in the study of phylogenetic networks is how to determine whether a tree is displayed by a phylogenetic network over the same taxon set (in a sense we define precisely below), called the tree containment problem [6]. Answering this question is indeed useful to validate and justify a phylogenetic network model by testing whether it displays existings phylogenies over a set of taxa under study.

The problem is *NP*-complete in general [8], and even on the more restricted class of tree-sibling time-consistent regular networks [17]. Although great effort has been made in studying it, it is shown to be polynomial-time solvable only for a couple of interesting classes of phylogenetic networks, namely normal networks

and tree-child networks [17]. It is open whether the tree containment problem is polynomial-time solvable for a class of phylogenetic networks that is superior to child-tree networks, particularly those with the so-called reticulation-visibility property [6, 17].

In the present paper, we study the tree containment problem for nearly stable phylogenetic networks (which are defined in the next section), which form a superclass including normal and child-tree networks. Recombination histories of viruses, hybridization histories of plants, and histories of horizontal gene transfers reported in literature often have such a property [7, 10]. Our key results include (i) the number of reticulations in a reticulation-visible or nearly stable phylogenetic network is linearly bounded above in terms of the number of taxa and (ii) the tree containment problem is solvable for nearly stable phylogenetic networks in quadratic time.

2 Concepts and Notions

A *phylogenetic tree* on a set of taxa X is a rooted tree in which one vertex is designated as the root, all the edges are oriented away from the root, and the species in X are one-to-one mapped to leaves.

A *phylogenetic network* (network for short) on a set of taxa X is a directed acyclic graph with a single root (with indegree 0) with properties that (i) its leaves (with outdegree 0) are one-to-one mapped to taxa in X, (ii) there are no vertices with indegree and outdegree one, and (iii) there is a path from the root to any other vertex. We identify each leaf with the taxon corresponding to it and refer to the directed edges as branches.

A network is *binary* if its root, leaves and the other vertices have degree 2, 1 and 3, respectively. A vertex in a network is called a *reticulation vertex* (reticulation for short) if it has indegree at least two and outdegree one and a *tree vertex* if it has indegree one and outdegree at least two. A branch is a *tree branch* if it ends in a tree vertex; it is called a *reticulation branch* otherwise. A phylogenetic tree is simply a network with no reticulations.

For a binary phylogenetic network N, we shall use r_N to denote the root of N. Let x and y be vertices in N. We say that x is a *parent* of y and y is a *child* of x if (x, y) is a branch. More generally, we say that x is an *ancestor* of y and equivalently y is a *descendant* of x if there is a directed path from x to y. A vertex x in N is a *stable ancestor* of a vertex v if it belongs to all directed paths from the r_N to v. It is *stable* if there exists a leaf l such that x is a stable ancestor of l.

Proposition 2.1 *Let N be a binary network. The following facts are true.*

(1). *A vertex is stable if it has a stable tree child.*

(2). *A reticulation is stable if and only if its unique child is a stable tree vertex.*

(3). *If a tree vertex is a stable, then its children cannot both be a reticulation.*

Proof The proof can be found in Appendix A.

A network is a *tree-child* network if every vertex has a child that is a tree vertex [2]. It is clear that a network is a tree-child network if and only if every vertex is stable. It is *reticulation-visible* if all its reticulations are stable [6]. It is *nearly stable* if every vertex either is stable itself or has stable parents.

Given a binary phylogenetic tree T and a binary network N, we say that N displays T if there is a spanning subtree T' of N that is a subdivision of T, i.e. T' has the same vertex set as N and T can be obtained from T' by performing all possible contractions of branches incident with the vertices with outdegree and indegree 1, branches incident with the "dummy leaves" (leaves in T' that correspond to tree vertices in N), and branches incident with a vertex of indegree 0 and outdegree 1.

In this work, we study the problem of determining whether a phylogenetic tree is displayed by a network or not, called the *tree containment problem* (TCP). We shall present a quadratic time algorithm to solve the TCP for binary nearly stable networks.

3 How Many Reticulations in A Network?

A network over n leaves can have a very large number of reticulations. To analyze the time complexity of an algorithm designed for solving a network problem, we need to know the size of the network compared to n.

After we delete a reticulation branch from each reticulation in a binary network N, we shall obtain a spanning subtree T'. All leaves in the network are still leaves in T', but T' may contain some "dummy leaves" that correspond to tree vertices whose outgoing branches have both been removed. The following lemma says that T' does not contain any dummy leaves if we carefully select which reticulation branches to remove.

Lemma 3.1 *Let N be a binary reticulation-visible phylogenetic network. We can determine which reticulation branch to remove at each reticulation so that the tree obtained after the removal of the selected branches does not contain any dummy leaves.*

Proof Let T be a tree obtained from N after the removal of a selected reticulation branch at each reticulation. To obtain a tree without dummy leaves, we need to guarantee that the reticulation branches to be removed are incident with different tree vertices. In other words, the branches to be removed form a matching that covers every reticulation in N. Since N has the reticulation-visibility property, the parents of each reticulation are both tree vertex. Such a set of reticulation branches exists and can be found by applying Hall's Theorem to a bipartite graph with one part consisting of tree vertices and another consisting of reticulations, and the edges being reticulation branches in N. Since each reticulation is the head of two reticulation branches and each tree vertex is the tail of at most two reticulation branches, there exists a matching that covers all the reticulations (see a result of N. Alon on page 429 in [1]). □

Theorem 3.1 *There are at most $4(n-1)$ reticulations in a binary reticulation-visible phylogenetic network with n leaves.*

Proof Let N be a binary reticulation-visible phylogenetic network with n leaves. Assume there are m reticulations in N. By Lemma 3.1, we can obtain a tree T with no dummy leaves. Since N is binary, an internal vertex in T has either one or two children; equivalently, T is a subdivision of a rooted binary tree T' over the same leaves as N. Therefore, T has $n-1$ internal vertices (including

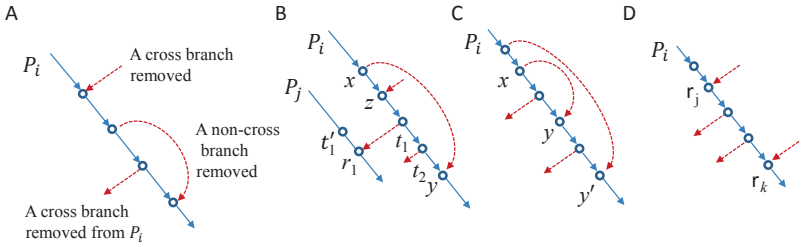


Figure 1: Illustration of the different cases in the proof of Theorem 3.1. **A**. Definition of cross and non-cross branches removed from a path. **B**. The branch (x, y) is a non-cross branch removed from a path. Assume that a cross branch (z', z) has been removed from a reticulation z inside the segment from x and y, where z' is not shown, and two cross branches have also been removed from two tree vertices t_1 and t_2 between z and y. **C**. Some cross branches must have been removed from their tails located between the heads of two non-cross branches that are removed from a path (in this case, between y and y'). **D**. If at least two cross branches have been removed from two reticulations in a path, then the upper one is not stable.

its root) of outdegree 2 and there are $2n-2$ paths P_i ($1 \leq i \leq 2n-2$) satisfying (i) the ends of each P_i are either the root of T, a leaf, or internal vertices of out degree 2, and (ii) each vertex in the middle of P_i has both indegree and outdegree 1 if P consists of two or more branches.

For each path P_i of length ≥ 2, a vertex in the middle of P_i is either a tree vertex of N, whose outgoing branch not in P_i has been removed, or a reticulation, whose incoming branch not in P_i has been removed. For convenience of discussion, we divide the removed reticulation branches into **cross** and **non-cross** branches (with respect to T) (Figure 1A). A removed branch is called a cross branch if its tail and head are located on two different paths P_i and P_j, $i \neq j$, otherwise it's called a non-cross branch. We first have the following facts.

FACTS (1) Assume (x, y) is a non-cross branch removed from P_i. At least one cross branch has been removed from its tree vertex tail in the segment $P_i[x, y]$ from x to y of P_i. But no cross branches have been removed from their heads lying in $P_i[x, y]$.

(2) Let (x, y) and (x', y') be two non-cross branches removed from P_i, where y is an ancestor of y'. Then there exists at least one cross branch being removed from its tree vertex tail located between y and y' (Figure 1C).

(3) There are at least as many cross reticulation branches removed as non-cross reticulation branches.

Proof (1) Since multiple branches are not allowed between any two vertices in N, there exist vertices between x and y in $P_i[x, y]$. Hence, we only need to prove that there are no reticulations other than y in $P_i[x, y]$. Assume a cross branch (z', z) has been removed from a reticulation z in $P[x, y]$ (Figure 1B).

A path including (x, y) from the root of N to a leave below y does not pass z, so z is not stable on any leaf below y and hence below z in T. Moreover, since T is a subtree of N, T contains a unique path from the network root to l

that does not pass through z for every leaf l that is not below z in T. Thus, z cannot be stable on any leaf that is not below z in T. Since N and T have the same set of leaves, z is not stable in N, contradicting the reticulation-visibility property.

(2) Note that y and y' are reticulations in N. The fact follows from that y has a tree vertex child in N.

(3) By (1) and (2), we can establish an injective map from the set of non-cross reticulation branches to that of cross one. Hence, the statement in this part is also true. □

We now continue to prove the theorem. Assume $2n-1$ or more cross branches (t_i, r_i) have been removed from the $2n-2$ paths P_i. At least two heads r_j and r_k are on the same path P_i (Figure 1D). Using an argument similar to that used in the proof of part (2) of the FACTS, one of r_j and r_k which is upstream in P_i is not stable, a contradiction. Therefore, at most $2n-2$ cross branches have been removed to produce T. By (3) of the FACTS, there are also at most $2n-2$ non-cross branches removed during the process. Since we removed one incoming branch for each reticulation, we conclude that there are at most $4(n-1)$ reticulation branches in N. □

Lemma 3.2 *Let N be a binary nearly stable phylogenetic network, and let $U_{\text{ret}}(N)$ and $S_{\text{ret}}(N)$ denote the numbers of all unstable and stable reticulations in N, respectively. We can transform N into a binary reticulation-visible phylogenetic network N' with the property that N' is over the same set of leaves as N and $S_{ret}(N) \leq S_{ret}(N') \leq S_{ret}(N) + U_{ret}(N)$.*

Proof Let a be an unstable reticulation in N, whose child is denoted by b. Since N is nearly stable, b is stable. By Proposition 2.1(2), b is a stable reticulation. Let c denote a parent of a. Vertex c must be a stable tree vertex. Let d be a child of c other than a. Since c is stable, by By Proposition 2.1(3), d is a tree vertex which is stable. Let e be the parent of c. We use f to denote the parent of a other than c and g the parent of b other than a.

Note that $g \neq f$ as shown in Figure 2. Otherwise, f is unstable, contradicting that there are no two consecutive unstable vertices. To transfer N into a binary reticulation-visible phylogenetic network, we remove unstable vertex a by first removing the branch (c, a), and then contracting the paths f-a-b and e-c-d into branches (f, b) and (e, d). It is easy to see, after rewiring, both b and d are still stable in the resulting network. By rewiring around every unstable reticulation in N, we produce a binary reticulation-visible network N'. The inequality can be obtained from the fact that no stable reticulation is removed, and no new reticulation created during the rewiring. □

Lemma 3.3 *For a binary nearly stable phylogenetic network N, $U_{ret}(N) \leq 2S_{ret}(N)$.*

Proof It is straightforward from the fact that an unstable reticulation must have a stable reticulation as its child, and any stable reticulation can be the child of at most two unstable reticulations. □

Theorem 3.2 (i) *The number of reticulations in a binary nearly stable phylogenetic network is at most $12(n-1)$.*

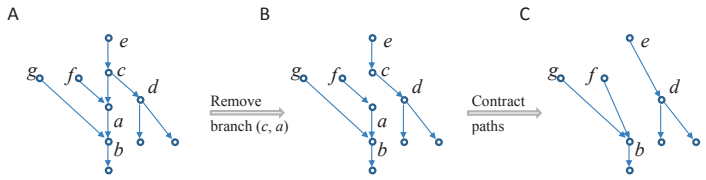


Figure 2: **A** An unstable reticulation a and its stable child b and stable parents (c and f) in the original network N. To transform N into a reticulation-visible network N_2, the incoming reticulation branch (c, a) is removed (**B**) and then paths ecd and fab are contracted (**C**). The wiring eliminates the unstable reticulation vertex a.

(ii) *The number of vertices and branches are bounded above by* $13(n-1)$ *and* $38(n-1)$.

Proof (i) By the above lemmas, we have $S_{ret}(N) + U_{ret}(N) \leq 3S_{ret}(N) \leq 3S_{ret}(N') \leq 3(4n-4) = 12(n-1)$.

(ii) We can think of the network as a flow network, with the root being the source and n leaves being sinks. Hence, the number of tree vertices is equal to $n-1$ plus the number of reticulations, i.e. at most $13(n-1)$. Since the outdegrees of each tree and reticulation vertex are 2 and 1, respectively, implying that there are at most $2(13n-13) + 12(n-1) = 38(n-1)$ branches in N. □

4 A Quadratic-Time Algorithm for the TCP

In this section, we shall present a quadratic-time algorithm for solving the TCP. If a given phylogenetic network N and a given reference phylogenetic tree contain a common subtree, then we can simplify the task of determining whether N displays T by replacing the common subtree by a new leaf. Therefore, without loss of generality, we assume that N does not contain a subtree with two or more leaves. Such a property is called the *subtree-free property.*

Lemma 4.1 *Let N be a nearly stable phylogenetic network satisfying the subtree-free property. Let $P = (r, \ldots, w, u, v, l)$ be a longest root-to-leaf path of four or more vertices in N, where r is the root of N and l a leaf end. Then the subnetwork consisting of the descendants of w has only ten possible structures given in Figure 3.*

Proof Consider the path P. Since P is a longest root-to-leaf path, the other child of v must be a leaf if v is a tree vertex and then we obtain a subtree below v, contradicting that that N has the subtree-free property. When v is a reticulation, there are two possible cases for u.

CASE 1: u is a reticulation.

Then u is unstable. If w is also reticulation, it is unstable, contradicting that N is nearly stable. Hence, w must be a tree vertex stable on a leaf which may or may not be l. Let g be the other child of w. By Proposition 2.1(3), g is either a tree vertex or a leaf. If g is a leaf, we obtain the subnetwork in Figure 3A.

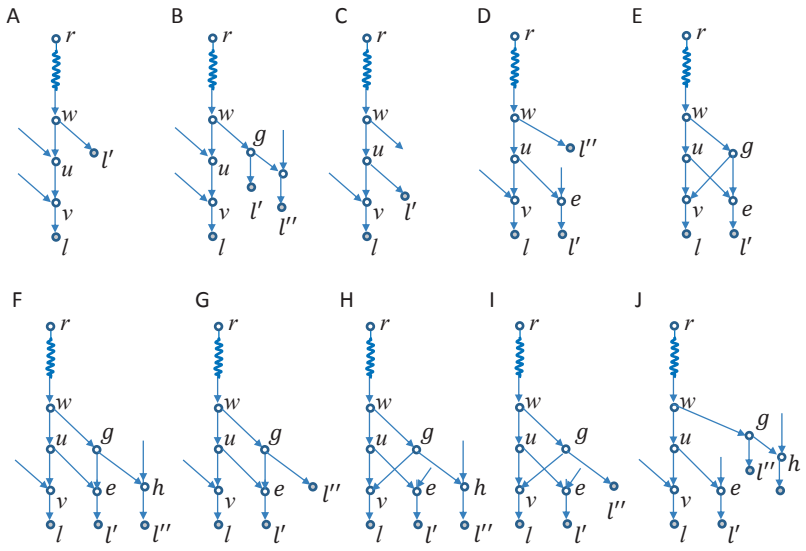


Figure 3: All possible subnetworks at the end of a longest path in a nearly stable network. Here, r is the network root and the segment between r and w is represent by a coiled path.

Assume g is a tree vertex. If g has a child that is a tree vertex, then this child must have two leaves as its children, as P is a longest path, contradicting the subtree-free property. If the children of g are both reticulations, then w is unstable. Hence, we conclude that one child of g is a leaf and the other is a reticulation with a leaf child, obtaining the subnetwork in Figure 3B.

CASE 2: u is a tree vertex.

Let e denote the other child of u, different from v. If e is a tree vertex, by the fact that P is a longest path, it must have two children that are leaves and so there is subtree below e, contradicting that N has the subtree-free property. If e is a leaf vertex, we obtain the subnetwork in Figure 3C.

If e is a reticulation, then e has a leaf child. In this case, u is unstable. Therefore, w must be a stable tree vertex. We consider the other child, g, of w in the following subcases.

CASE 2.1 Vertex g is a leaf.

If g is a leaf, we have the subnetwork given in Figure 3D.

CASE 2.2 Vertex g is a tree vertex and also a parent of e and v.

We obtain the subnetwork in Figure 3E.

CASE 2.3 Vertex g is a tree vertex. In addition, g is a parent of e, but not a parent of v.

Then w is stable on l', the unique child of e. Let h be the other child of g, where $h \neq e$. If h is a reticulation, its child must be a leaf, as P is a longest path. Thus, we have the subnetwork given in Figure 3F. Similarly, if h is a tree vertex, its children must be two leaves and then there is a subtree below h, a contradiction. If h is a leaf, we obtain the subnetwork in Figure 3G.

CASE 2.4 Vertex g is a tree vertex. In addition, g is a parent of v, but not a parent of e.

Using a discussion similar to that of CASE 2.3, we have two possible subnetworks (Figure 3H and I) in this case.

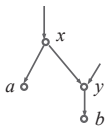
CASE 2.5. Vertex g is a tree vertex. In addition, g is neither a parent of v nor a parent of e.

Again we look at its children of g. If both are reticulations, then w is unstable, a contradiction. If neither are reticulations, then there is a subtree below g; if one of them is a reticulation and the other is a tree vertex then again there is a subtree. If one is a reticulation while the other is a leaf, we obtain the subnetwork in Figure 3J.

CASE 2.6 Vertex g is a reticulation.

This case is impossible. Otherwise, w is unstable. □

The subnetwork below g of the structures shown in the panels B, G, I, J and that below u in the panel C of Figure 3 have the following pattern:



in which a leaf, a, has a reticulation brother, y, and a leaf nephew, b. Such a pattern is called an *uncle-nephew structure*. Note that if a and b are not siblings in a tree which is displayed in N, then the reticulation branch (x, y) should not be used; additionally, if a and b are siblings, either (x, y) or other dangling branch can be used. In the latter, however, any tree displayed in the network resulting from the removal of the dangling branch is also displayed in the one after (x, y) is removed. Hence, to determine whether N displays a tree T, we can simplify the network by eliminating y using the following process:

> **Uncle-Nephew Reduction** Remove the dangling branch if a and b are siblings in T and (x, y) otherwise and then contract vertices with indegree and outdegree 1.

In each of the other cases, we can also simplify a given network by using information on a given tree. To summarize how to simplify the network, we use the following notation for each vertex w in a network N:

- $R(w)$ denotes the subnetwork consisting of all the descendants of w;
- $(-, x)$ denotes the dangling branch entering x from its parent not in $R(w)$ for x in $R(w)$.
- $N' + (x, y)$ denotes the network obtained by adding (x, y) into N' for a subnetwork N' and a branch (x, y) of N.
- $N' - (x, y)$ denotes the network obtained by removing (x, y) from N' for a subnetwork N' of N.
- $p_T(x)$ denotes the parent of x in T for a vertex x in a tree T.

Theorem 4.1 *Let N be a binary nearly stable network with no uncle-nephew structure, and T a tree with the same set of labeled leaves. Let w be a tree vertex in N. Define N' as follows.*

(i) *When $R(w)$ is in Figure 3A, define $N' = N - (w, u)$ if l and l' are not sibling in T and $N' = N - \{(-, u), (-, v)\}$ otherwise.*

(ii) *When $R(w)$ is in Figure 3D, define $N' = N - (-, v)$ when l and l'' are siblings, or when l and l' are siblings and their parent is a sibling of l'' in T. $N' = N - (u, v)$ otherwise.*

(iii) *When $R(w)$ is in Figure 3E, define $N' = N - \{(u, e), (g, v)\}$.*

(iv) *When $R(w)$ is in Figure 3F, defines $N' = N - \{(g, e), (-, v)\}$ if l and l' are sibling in T and $N' = N - (u, e)$ otherwise.*

(v) *When $R(w)$ is in Figure 3H, define $N' = N - \{(g, v), (-, e)\}$ if l and l' are sibling in T and $N' = N - (u, v)$ otherwise.*

Then, N' is nearly stable and N displays T only if N' displays T.

Proof N' is nearly stable, as we don't remove any leaves and only reduce possible paths from the network root to a leaf in the transformation from N to N'.

Assume $R(w)$ is the subnetwork in Figure 3A and N displays T. Then there exists a subtree T' of N that is a subdivision of T and let $p_T(l)$ corresponds x in T'. Clearly, x is of degree 3 and hence a tree vertex in N. We consider two cases.

CASE A. Leaves l and l' are not siblings in T.

We first have that $x \neq u$, $x \neq v$ for u and v in Figure 3A. We also have $x \neq w$. Otherwise, l' must be a child of x in T' and l is a sibling of l' in T, a contradiction. Therefore, the path from x to l contains two or more vertices and v is the parent of l in this path. If u is the parent of v in the same path, neither the dangling branch $(-, v)$ nor (w, u) is in T', indicating that $N' = N - (w, v)$ also displays T.

If $p_{T'}(v) \neq u$ in the same path, then (u, v) is not in T' and hence u becomes a dummy leaf in T', as there is no leaf other than l below u in $R(w)$. If (w, u) is in T', then $(-, u)$ is not in T' and $T' + (-, u) - (w, u)$ is a subtree of N" in which only the dummy leaf u is relocated. Hence, N' also displays T.

CASE B. Leave l and l' are siblings in T.

Then x is a common ancestor of l and l' in N. If $x = w$, the path from x to l in T' must be w, u, v, as this is only path from w to l in N. Hence, $(-, u)$ and $(-, v)$ are not in T'. Therefore, T' is a subtree of N' and N' also displays T.

If $x \neq w$, then x is an ancestor of w and hence w is the parent of l' in the path from x to l' in T'. Note that $p_{T'}(l) = v$. If $p_{T'}(v) = u$, then $(-, u)$ is in T', but both $(-, v)$ and (w, u) are not. $T'' = T' + (w, u) - (-, u)$ is a subtree of N'. Noting that T'' is also a subdivision of T, N' displays T.

If $p_{T'}(v) \neq u$, then $(-, v)$ is in the path from x to l in T'. This implies that (u, v) is not in T' and u is a dead-end in T'. If (w, u) is in T', the subtree $T'' = T' + (u, v) - (-, v)$ of N' is a subdivision of T. If (w, u) is not in T', the subtree $T'' = T' + (w, u) - (-, u) - (-, v)$ of N' is a subtree of N'. Hence, N' displays T'.

Similarly, we can prove that N displays T only if N' displays T when $R(w)$ is the subnetwork in the panels D, F, and H in Figure 3. Note also that the subnetworks in the panels F and H are essentially identical (if the positions of v and e are switched). Due to the limited space, the details are omitted here. The case when $R(w)$ is the subnetwork in Figure 3E is trivial, as deletion of which two reticulation branches from v and e does not affect outcome.

By Theorem 4.1, we are able to determine whether a nearly stable phylogenetic network N displays a binary tree T or not by repeatedly executing the following tasks in turn until the resulting network N' becomes a tree:

- Compute a longest path P in $N' = N$;
- Simplify N' by considering the subnetwork at the end of P according to the cases in Lemma 4.1;
- Contract degenerated reticulations in N' and replace the parent of a pair of leaves appearing in both N' and T with a new leaf.

and then check if N' is identical to T. The pseudo code of the algorithm can be found in Appendix B.

Finally, we analyze the time complexity. Let N and T have n leaves. By Theorem 3.2, there are $O(n)$ vertices and $O(n)$ branches in N. Since we eliminate at least a reticulation in each loop step, the algorithm stops after $O(n)$ loop steps. In each loop step, a longest path can be computed in $O(n)$ time ([14], page 661), as N is acyclic; both the second and third tasks can be done in constant time. In summary, our algorithm has quadratic time complexity.

5 Conclusion

We have developed a quadratic-time algorithm for the TCP for binary nearly stable phylogenetic networks. Our algorithm not only is applicable to a superclass of tree-child networks, but also has a lower time complexity than the algorithm reported in [17]. Although phylogenetic network models built in the study of viral and plant evolution are often nearly stable, it is interesting to know whether the TCP is polynomial time solvable or not for networks with other weak properties.

In particular, the problem remains open for binary networks with the visibility property, but the upper bound we give on the number of reticulation vertices of such networks, as well as our algorithm for nearly stable phylogenetic networks, provide definitely valuable ideas to solve the problem, exactly or heuristically, on phylogenetic networks with the visibility property.

6 Acknowledgments

The project was financially supported by Merlion Programme 2013.

References

[1] Bondy, J. A., Murty, U. S. R. (2008). *Graph Theory*, Springer, Germany.

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

[3] Chan, J. M., Carlsson, G., Rabadan, R. (2013) Topology of viral evolution. *PNAS*, 110, 18566-18571.

[4] Dagan, T., Artzy-Randrup, Y., Martin, W. (2008) Modular networks and cumulative impact of lateral transfer in prokaryote genome evolution. *PNAS*, 105, 10039-10044.

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

[6] Huson, D. H., Rupp, R., Scornavacca, C. (2010) *Phylogenetic Networks*. Cambridge University Press, Cambridge, UK.

[7] Jenkins P.A., Song Y.S., Brem R.B. (2012) Genealogy-based methods for inference of historical recombination and gene flow and their application in *Saccharomyces cerevisiae*. *PLoS ONE* 7: e46947.

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

[9] Kunin, V., *et al.* (2005) The net of life: reconstructing the microbial phylogenetic network. *Genome Res.*, 15, 954-959.

[10] Marcussen, T., *et al.* (2012) Inferring species networks from gene trees in high-polyploid North American and Hawaiian violets (*Viola*, Violaceae). *Syst. Biol.*, 61, 107-126.

[11] McBreen, K., Lockhart, P. J. (2006) Reconstructing reticulate evolutionary histories of plants. *Trends Plant Sci.*, 11, 398-404.

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

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

[14] Sedgewick, R., Wayne, K. (2011) *Algorithms*, 4th Edition. Addison-Wesley Professional, USA.

[15] Stolzer, M., *et al.* (2012) Inferring duplications, losses, transfers and incomplete lineage sorting with nonbinary species trees. *Bioinformatics*, 28, i409-i415.

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

[17] Van Iersel, L., Semple, C., Steel, M. (2010) Locating a tree in a phylogenetic network. *Inform. Proces. Letts*, 110, 1037-1043.

[18] Wang, L., Zhang, K., Zhang, L. (2001) Perfect phylogenetic networks with recombination. *J. Comput. Biol.*, 8, 69-78.

Appendix A

Proposition 2.1 Let N be a binary network. The following facts are true.

(1). A vertex is stable if it has a stable tree child.

(2). A reticulation is stable if and only if its unique child is a stable tree vertex.

(3). If a tree vertex is a stable, then its children cannot both be a reticulation.

Proof (1). Consider a vertex u with a tree vertex child v. Assume v is stable on a leaf l. Then all the paths from r_N to l contain v and hence contain u. Therefore, u is also stable on l.

(2). The sufficiency follows from (1). The necessity is proved as follows.

Assume u is a reticulation stable on a leaf l and the unique child of u is x. If x is a reticulation, it has another parent v different to u. For a path from r_N to l passing u, it also passes x and so joining a path from r_N to v, the branch (v, x) and a path from x to l gives a path from r_N to l avoiding u, contradicting the fact that u is stable on l. Therefore, x is a tree vertex. It is easy to see every path from r_N to l must first pass x and then u. Hence, x is also stable on l.

(3) Similar to (2), we can prove the statement.

This concludes the proof.

Appendix B

In this section, we provide the pseudo code of the proposed algorithm. We first introduce three sub-functions:

1. Function parent$(v, flag, P)$ returns the parent of v in path P if flag is "on" and it returns the parent not in P if flag is "off".
2. Function child$(v, flag, S)$ returns the child of v in vertex set S if flag is "on" and it returns a child not in S if flag is "off");
3. Function thechild(v) returns the child if v is reticulation.

ALGORITHM FOR TCP

Input: A phylogenetic tree T and a nearly stable phylogenetic network N.

1. Simplify N and T by replacing a common subtree with a distinct leaf;
2. Set R be the number of reticulation nodes in N; $N' \leftarrow N$;
3. While $(R > 0)$ do {
   3.1. Compute a longest root-to-leaf path P ending with a leaf l.
   3.2. $v \leftarrow \text{parent}(l, \text{on}, P)$; $u \leftarrow \text{parent}(v, \text{on}, P)$; $w \leftarrow \text{parent}(u, \text{on}, P)$;
   3.3. $S \leftarrow$ (the network below w); $g = \text{child}(w, \text{off}, P)$; $e \leftarrow \text{child}(u, \text{off}, P)$;
   3.4. switch (S) {
      case Figure 3A:
        if (l and g are not siblings) {Delete (w, u);}
        else {Delete (parent$(u, \text{off}, P), u)$; Delete (parent$(v, \text{off}, P), v)$;}
      case Figure 3B or Figure 3G or Figure 3I or Figure 3J:
        Apply Uncle-Nephew-Reduction at g;
      case Figure 3C:
        Apply Uncle-Nephew-Reduction at u;
      case Figure 3D:
        if ((l and thechild(e) are siblings
         && the parent of l and thechild(e) is the sibling of g in T)
         or (l and g are siblings in T)) {
         Delete (parent$(v, \text{off}, P), v)$; } else {Delete (u, v); }
      case Figure 3E:
        {Delete (u, e); Delete (g, v); }
      case Figure 3F:
        if (l and thechild(e) are siblings in T) {Delete (g, e);}
        else {Delete (u, e);}
      case Figure 3H:
        if (l and thechild(e) are siblings in T) { Delete (g, v);}
        else {Delete (u, v);}
   3.5. update R; contract vertices with 1 outdegree and indegree in N';
   }
5. Report "YES" if N' is identical to T and "NO" otherwise.