A Faster Algorithm for the Tree Containment Problem for Binary Nearly Stable Phylogenetic Networks

Jittat Fakcharoenphol
Computer Engineering
Kasetsart University
Bangkok, Thailand
Email: jittat@gmail.com

Tanee Kumpijit
Computer Engineering
Kasetsart University
Bangkok, Thailand
mail: tanee.kumpijit@gmail.com

Attakorn Putwattana
Computer Engineering
Kasetsart University
Bangkok, Thailand
Email: atkpwn@gmail.com

***Abstract*—Phylogenetic networks and phylogenetic trees are leaf-labelled graphs used in biology to describe evolutionary histories of species whose leaves correspond to a set of taxa in the study. Given a phylogenetic network N and a phylogenetic tree T over the same set of taxa, if one can obtain T from N by edge deletions and contractions, we say that N contains T. A fundamental problem, called the *tree containment problem*, is to determine if N contains T. In general networks, this problem is NP-complete, but can be solved in polynomial time when N is a normal network, a binary tree-child network, or a level-k network. Recently, Gambette, Gunawan, Labarre, Vialette and Zhang showed that it is possible to solve the problem for a more general class of networks called binary nearly stable networks. Not only that binary nearly stable networks include normal and tree-child networks, they claim that important evolution histories also match this generalization. Their algorithm is also more efficient than previous algorithms as it runs in time $O(n^2)$ where n is the number of taxa. This paper presents a faster $O(n \log n)$ algorithm. We obtain this improvement from a simple observation that the iterative algorithm of Gambette *et al.* only performs very local modifications of the networks. Our algorithm employs elementary data structures to dynamically maintain certain internal data structures used in their algorithm instead of recomputing at every iteration.**

I. Introduction

In evolutionary biology, while the term "tree of life" is generally used to represent an evolutionary history of species, it is known that there are evolutionary events that break the tree constraint. Therefore, to describe events such as horizontal gene transfer, hybridization or recombination, phylogenetic networks are used [1], [2], [3], [4].

A *phylogenetic network* is a directed acyclic graphs (DAG) with its leaves, vertices with no outgoing edges, correspond to the set of taxa in study. Its root, a vertex with no incoming edge, is the common evolutionary ancestor of those taxa. Other vertices represent the evolutionary process (see Figure 1, for example).

A fundamental problem in this area called *tree containment problem* is to answer if a given phylogenetic network contains or *displays* (defined next section) a given phylogenetic tree [1]. When devising new phylogenetic network model, verifying that a proposed model does not violate well-known phylogenetic tree model is a useful validation step.

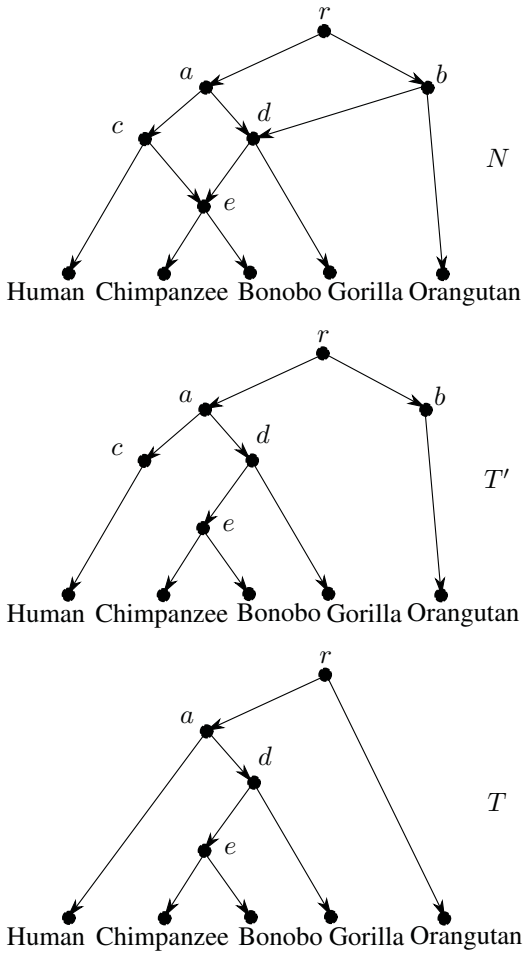

Fig. 1. This figure shows a network N, and trees T and T'. To see that N contains T, observe that one can obtain T' from T by subdividing at branches $(a, Human)$ and $(r, Orangutan)$ and clearly $T' \subseteq N$.

In general network, Kanj, Nakhleh, Than and Xia [5] show that it is NP-complete to solve this problem. This motivates van Iersel, Semple, and Steel [6] to study a more restricted version of the problem and show that for a more restricted models of

978-1-4799-1966-6/15/$31.00 ©2015 IEEE

tree-sibling time consistent regular networks, the problem remains NP-complete. Iersel *et al.* also present polynomial time algorithms for others interesting restricted versions, namely for normal networks, binary tree-child networks and level-k networks.

Gambette, Gunawan, Labarre, Vialette and Zhang [3] consider a more general version of the problem. They study the problem in *binary nearly stable networks* (to be defined in Section II), a generalization of normal networks and binary tree-child networks, and develop a quadratic-time algorithm for this case. Their algorithm is more efficient than the previous one and also work in a more general case. It has been shown that virus recombination histories, plant hybridization histories, and horizontal gene transfer histories frequently satisfy the conditions for this restricted version (see, e.g., [7] and [4]).

Our contribution. For the problem with n taxa, in this paper, we present an $O(n \log n)$-time algorithm that improves over an $O(n^2)$-time algorithm of Gambette *et al.*. Our key observation is that the iterative algorithm of Gambette *et al.* only performs very local modification of the networks. We then employ elementary data structures to dynamically maintain certain internal data structures used in [3] instead of recomputing at every iteration.

Important definitions are described in the next section. An algorithm of Gambette *et al.* is reviewed in Section III. Section IV presents our algorithm.

II. Definitions

In this section, we give basic definitions on phylogenetic networks and trees. We also define stable networks, which is the main concept from Gambette *et al.* [3]. Our notations follow [3].

A *phylogenetic network* or *network* on a set χ of taxa is a directed acyclic graph with a single root satisfying the following three conditions: (i) all of its leaves correspond (one-to-one) to the taxa in χ, (ii) it has no vertex with both in-degree and out-degree equal to one, and (iii) every vertex is reachable from the root. We also refer to the directed edges as *branches* and directed paths as *paths*.

We say that a vertex is a *reticulation vertex* or *reticulation* if its in-degree is at least two; otherwise it is a *tree vertex*. A branch which ends at tree vertex is called *tree branch*. Similarly a branch ending at reticulation is called *reticulation branch*.

A *binary network* is a network whose degrees of its root, leaves and the other vertices are 2, 1 and 3, respectively. We call a binary network with no reticulation a *phylogenetic tree*.

Consider a binary network $N = (V, E)$. Let u and v be vertices in N. We say that u is a *parent* of v and v is a *child* of u if $(u, v) \in E$. In addition, u is said to be an *ancestor* of v and v is said to be a *descendant* of u if v is reachable from u. If all the paths from root of N, denoted by r_N, to v contain u, we will call u a *stable ancestor* of v and v is *stable descendant* of u. A vertex u is *stable* if it is a stable ancestor of some leaf ℓ.

If all reticulation vertices are stable, we call that network *reticulation-visible* [1]. A *nearly stable* network is a network that for every vertex, it is stable or its parents are.

As in [3], by *contracting* a branch (u, v), we mean replacing u and v by a single vertex w which all neighbors of u and v become its neighbors and remove edge (u, v). To contract a path $P = (u_1, u_2, \cdots, u_k)$, we can simply contract all edges (u_i, u_{i+1}) for all integer i ; $2 \leq i < k$. After contraction, path P becomes a single branch (u_1, w).

Given a phylogenetic tree T and a binary network N, we say that N *displays* T if T can be transformed to a spanning subtree T' of N by performing a sequence of *subdividing* operation to some of its branches. To subdivide on branch (x, y) is to insert new vertex t and replace (x, y) with (x, t) and (t, y). (see Figure 1.)

In this paper, we consider only the special case of the *tree containment problem* or TCP for short, for which we are given a set of taxa χ, a binary nearly stable phylogenetic network N on χ, and a phylogenetic tree T on χ, and we have to decide whether T is displayed by N.

A network $N' = (V', E')$ is *subphylogeny* or *subnetwork* of $N = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. A network N with a reference phylogenetic tree T has *subphylogeny-free property* if they have no common subphylogeny larger than one vertex [3].

III. Previous Results

In this section, we review an algorithm of Gambette *et al.* [3] which answers tree containment problem for a binary nearly stable phylogenetic network N and a phylogenetic tree T on the same set of leaf of size n in $O(n^2)$ time.

The key observation of Gambette *et al.* is a simple reduction procedure for transforming a network N to a "smaller" network N' while ensuring that if N contains T, N' also contains T. The procedure works by considering the leaf taxon on the longest path in N.

Gambette *et al.* have characterized the tree structures based on nodes on the longest path from the root with 4 or more vertices. Let $P = (r, \ldots, w, u, v, \ell)$ be the longest path. Gambette *et al.* shows that the subnetwork rooted at w belongs to one of the 10 possible cases shown in Figure 2.

More precisely, Gambette *et al.* show the following lemma.

Lemma 1 (Lemma 4, [3])**.** *Let N be a binary nearly stable network and $P = (r_N, \cdots, w, u, v, \ell)$ be longest path, that consists of four or more vertices, from r_N to some leaf. Let $R(N, w)$ be the subnetwork of N rooted at w. The structure of $R(N, w)$ must be in one of the cases shown in Figure 2.*

For each of the cases, Gambette *et al.* provide a rule that removes some edge from N while maintaining the containment property. (See discussion on Uncle-Nephew reduction and Theorem 3 in [3].)

The algorithm of Gambette *et al.* is shown as Algorithm 1, $TCP(N, T)$. The algorithm repeatedly finds the longest path P from the root $r_{N'}$ (using procedure $LongestPath$) and performs the reduction steps as discussed above in Procedure $Simplify(\cdot)$ working with the subnetwork of N' rooted at w.

Procedure $Simplify$ ensures that some edge gets deleted from N'. Then to keep N' simple, degenerated reticulations

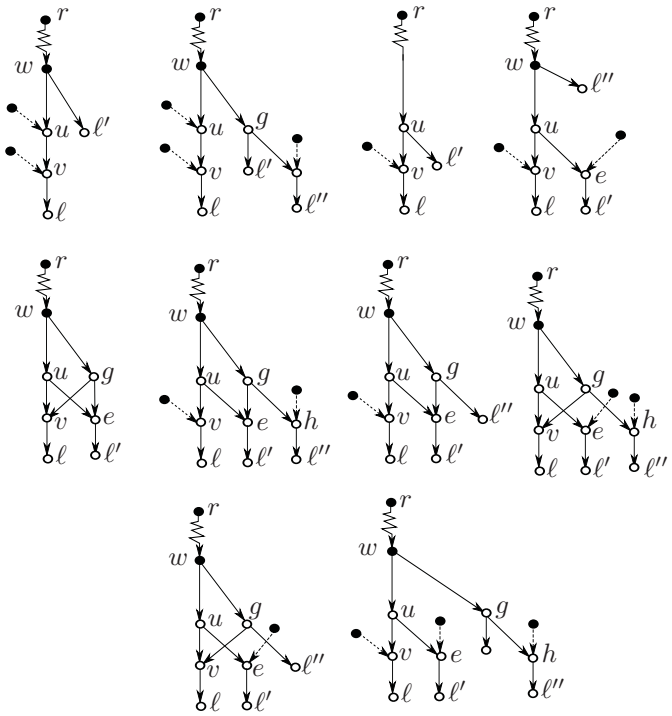

Fig. 2. All possible cases of $R(N, w)$ described in [3].

(i.e., vertices whose in and out degree is 1) are contracted (in procedure $ContractDegenerated()$). Finally, to preserve the subphylogeny-free property, if clear equivalence between parts of N' and T' is observed, i.e., there exists a pair of leaves ℓ and ℓ' sharing the same parent in *both* N' and T', both leaves are replaced with a new combined leaf (in procedure $ConbineLeaves()$).

Algorithm 1 TCP(N, T)

```
Preprocessing, N has subphylogeny-free property.
N' ← N
T' ← T
while N' has reticulation vertex do
    P ← LongestPath(r_{N'})
    Let ℓ be last vertex of P.
    Let v ← prev_P(ℓ).
    Let u ← prev_P(v).
    Let w ← prev_P(u).
    Simplify(N', R(N', w), T').
    ContractDegenerated().
    while ∃ ℓ, ℓ' ∈ L(N'), L(T') with same parent in both N' and T' do
        ConbineLeaves_{N'}(ℓ, ℓ').
        ConbineLeaves_{T'}(ℓ, ℓ').
    end while
end while
if N' is identical to T' then
    return True
end if
return False
```

The correctness of TCP follows from the correctness of the reduction steps. They show the following result.

Theorem 1 ([3])**.** *Let N be a binary nearly stable network and T be a phylogenetic tree. Let N' and T' be output of procedure $Simplify$. N displays T if and only if N' displays T'.*

To prove the running time, Gambette *et al.* prove the following properties regarding the numbers of various types of vertices.

Lemma 2 ([3])**.** *Let N be a binary nearly stable network with n leaves. The number of reticulation vertices is $O(n)$. The number of reticulation branches is $O(n)$. The number of tree vertices is $O(n)$.*

The number of iterations of Algorithm 1 is $O(n)$ because each iteration removes at least one reticulation branch. We need to bound the running time of each iteration. Since finding longest path in a DAG can be done in linear time using standard procedure (see, e.g., [8]), it takes $O(n)$ time for procedure $LongestPath$. The other steps clearly take $O(n)$ time. The overall running time of the algorithm is thus $O(n^2)$.

While Gambette *et al.* fully characterize the possible 10 cases in Lemma 1, they do not explicitly use the fact that the size of $R(N, w)$ is small. We state this fact explicitly as their corollary, which is crucial for our improvement in next section.

Corollary 1. *Given a binary nearly stable network N, let w be vertex defined in Algorithm 1. $R(N, w)$ has constant size. Also, the length and the number of affected paths by $Simplify(\cdot)$ is constant.*

Proof: The first part follows from Lemma 1 (also see Figure 2) that $R(N, w)$ has at most 9 vertices, and the number and the length of paths from w to any leaf is at most four. Also, $Simplify(N, R(N, w), T)$ removes only branches in $R(N, w)$, so it has no effect on vertices in $N \setminus R(N, w)$. ■

IV. Our Result

This section describes our simple improvement to the algorithm by Gambette *et al.* to obtain a nearly linear time algorithm. Our key observation is that in any iteration of the algorithm the set of vertices $R(N', w)$ which is affected by the reduction and other modification steps has constant size. Moreover, vertices in $R(N', w)$ are at the "end" of longest paths from the root. This ensure that changes in $R(N', w)$ are very "local." Thus, to find the longest path in the graph, we can retain most of the information from previous iterations.

In this section, we first focus on how to maintain the longest path distance. We then state the operations for this dynamic problem and describe how to use these operations to implement the steps in Gambette *et al.*'s algorithm. We finally show that standard priority queue data structure can be used to maintain the longest paths in the graph over the iterations of the algorithm.

The other non-trivial steps in the Gambette *et al.*'s algorithm are those that contract degenerated reticulation vertices and combine leaf taxa to ensure the subphylogeny-free property. We show a simple running time analysis in Subsection IV-B that these steps can be done in total time $O(n)$. Finally, the last subsection shows the running time of our improved algorithm.

A. Dynamically maintaining longest paths

Given a binary nearly stable phylogenetic network N with a root r_N on the set of leaves $L(N)$ of size n. We show how to use standard data structures for finding the longest path of N under the modification operations during the execution of algorithm TCP.

While our data structures work for general edge deletions, contractions, and vertex replacements, its fast running time depends crucially on how TCP updates the network.

Formally, these operations are supported by our data structures.

- Answering query: $query(N)$ returns quadruple (w, u, v, ℓ) such that path $P = (r_N, \cdots, w, u, v, \ell)$ is the longest path in N.
- Branch deletion: $delete(N, (u, v))$ removes reticulation branch (u, v) from N.
- Path contraction: $contract(N, P)$ contracts path $P = (p_1, p_2, \cdots, p_k)$ such that p_i has both in-degree and out-degree equal to one for all integer i such that $2 \leq i \leq k-1$.
- Leaf replacement: $replace(N, \ell, \ell')$ replaces parent of sibling ℓ and ℓ' in N with new leaf.

Using the operations to implement Algorithm 1. These operations can be used to implement those non-trivial steps in Algorithm 1 as follows.

$Simplify(\cdot)$ can be implemented by invoking $delete(N, e)$ for every branch e that need to be removed by rules in developed in [3].

We implement $ContractDegenerated(\cdot)$ using $contract(N, P)$ for every path P that must be contracted. These paths are results from the call to $Simplify(\cdot)$ that follows rules in [3]. They can be found by considering cases in Figure 2.

$ConbineLeaves_N(\ell, \ell')$ can be implemented with $replace(N, \ell, \ell')$. The inner while loop in Algorithm 1 can be implemented to run efficiently by remembering all affected leaves $\ell \in L(N')$ and $\ell \in L(T')$ after the call to $Simplify$. Note that there are only a constant number of affected leaves; therefore the time needed to check is $O(1)$.

Implementing the operations. In order to support those operations, we maintain data structures as follow.

- H: a max-heap of every leaf of N, whose key is the length of its longest path from the root of N.
- $D[v]$, $\forall v \in N$: length of longest path from r_N to v.

Initialization. Algorithm 2 initializes the data structure. We compute a single-source longest path from r_N to every vertex, store the length in D and insert all leaves ℓ to the heap with key equal to $D[\ell]$.

Recall that $R(N, d)$ denotes the subphylogeny of N rooted at vertex d. We now discuss how to implement the supported operations.

Algorithm 2 $init(N)$

```
Compute longest path from r_N to every vertex.
for v ∈ N do
    D[v] ← longest distance from r_N to v
    if v ∈ L(N) then
        H.insert(v, D(v)) // insert v with key D(v) to H.
    end if
end for
```

Answering query. For $query(N)$, we simply use max-heap operation $findMax$ to find leaf ℓ such that path from r_N to ℓ is longest in N. To implement the TCP algorithm, we need other vertices w, u, and v before ℓ on the path as well. To obtain these vertices, we can consider its ancestors on the path. The procedure is shown in Algorithm 3.

Algorithm 3 $query(N)$

```
ℓ ← H.findMax()
k ← D[l]
v ← parent of ℓ which D[v] = k − 1
u ← parent of v which D[u] = k − 2
w ← parent of u which D[w] = k − 3
return (w, u, v, ℓ)
```

Main updating procedure. Operations $delete(\cdot)$ and $contract(\cdot)$ may modify the longest path in the network. The fundamental update procedure (shown in Algorithm 4) is our main subroutine for updating the longest path distance. It is a standard algorithm for finding longest paths in a DAG restricted to $R(N, w)$. The subroutine uses procedure $TopologicalOrder(N)$ that returns ordered set of vertices in N such that if there is a branch $(u, v) \in N$, u comes before v.

By updating topologically, it is guaranteed that when we update v, all v's parents are updated. For each vertex v, we update $D[v]$ by finding its parent p with maximum $D[p]$ and set $D[v]$ to $D[p] + 1$. If v is a leaf we also update v's entry in H.

Algorithm 4 $update(N, v)$

```
Let parent(v) denote set of parents of v.
Q ← TopologicalOrder(R(N, v))
for v ∈ Q do
    D[v] ← max_{p∈parent(v)}(D[p] + 1).
    if v ∈ L(N) then
        H.delete(v)   // Delete v from H
        H.insert(v, D[v])   // Insert v with key D[v] to H.
    end if
end for
```

Branch deletion. For operation $delete(N, (u, v))$ we remove (u, v) and that may affect longest path of vertices in $R(N, v)$, so we call $update(N, v)$ (see Algorithm 5).

Algorithm 5 $delete(N, (u, v))$

```
Remove edges (u, v) from N.
update(N, v)
```

Path contraction. For $contract(N, P)$, let $P = (p1, p2, \cdots, p_k)$. We remove vertices with both in-degree and out-degree equal to one, add branch from p_1 to p_k.

If p_k is leaf, this means only longest path from r_N to p_k is affected. Also p_k has only one parent by the definition of binary network, we set $D[p_k]$ to $D[p_1] + 1$ and insert it to the heap.

If p_k is not leaf, those vertices that its longest path from root may be affected is in $R(N, p_k)$. We recompute longest path of affected vertices by calling $update(\cdot)$ (see Algorithm 6).

Algorithm 6 $contract(N, P)$

Let $parent(p_k)$ denote set of parents of p_k.
$P = (p1, p2, \cdots, p_k)$
Remove $p_2, p_3, \cdots, p_{k-1}$ from N.
Add edge (p_1, p_k) to N.
if $p_k \in L(N)$ **then**
  $D[p_k] \leftarrow D[p] + 1$.
  $H.delete(p_k)$ *// Delete p_k from H.*
  $H.insert(p_k, D[p_k])$ *// Insert p_k with key $D[p_k]$ to H.*
else
  $update(N, p_k)$
end if

Leaf replacement. For operation $replace(N, \ell, \ell')$. Let parent of ℓ and ℓ' denoted by p_ℓ. We assume $p_\ell \neq r_N$ or else the resulting network will be single vertex and not binary. Observe that p_ℓ must be tree vertex by definition of binary network. That means it only have one parent x. We remove ℓ, ℓ' and p_ℓ from N, delete ℓ and ℓ' from the heap. Then add new leaf p to N with $D[p] = D[x] + 1$ and insert it to the heap (see Algorithm 7).

Algorithm 7 $replace(N, \ell, \ell')$

Let p_ℓ be parent of ℓ and ℓ'.
Let x be parent of p_ℓ.
Add new leaf p to N with branch (x, p).
Remove ℓ, ℓ', p_ℓ from N.
$D[p] \leftarrow D[x]$ + 1
$H.delete(\ell)$ *// Delete ℓ from H.*
$H.delete(\ell')$ *// Delete ℓ' from H.*
$H.insert(p, D[p])$ *// Insert p with key $D[p]$ to H.*

Correctness. The following lemma show that operation $query(N)$ return correct longest path.

Lemma 3. *Operation $query(N)$ returns the correct answer, i.e., if $(w, u, v, \ell) = query(N)$, the path from r_N to ℓ is longest path in N.*

Proof: We prove this lemma by induction on the number of operations performed. Note that initially right after $init$ is called, the statement is true because H returns the leaf with the largest distance.

We shall prove that after we perform any supported operation, the statement remains true.

The most important operation to consider is $update(N, y)$. The correctness in this case follows from the correctness of the update order that we use (i.e., a topological order). (See full argument in standard algorithm texts such as [8].)

For $replace(N, \ell, \ell')$, we delete leaves ℓ, ℓ' that no longer exist from H and correctly insert the newly added leaf with its new longest length.

Let us consider $delete(N, (x, y))$ and $contract(N, P)$, where $P = (p1, p2, \cdots, p_k)$. Observe that affected vertices are those in $R(N, y)$ and $R(N, p_k)$ respectively. After the modification, we call $update(\cdot)$. The correctness in this case follows from the correctness of $update$. ■

Running time. We now analyze the running time for each operation. The discussion here shall be used in Subsection IV-C to bound the new algorithm's running time.

First, consider Algorithm $init(N)$. The major work in this part is to compute the longest distance. From [8], this can be done with a dynamic programming algorithm that runs in time $O(m' + n')$ in a DAG with n' vertices and m' edges. Lemma 2 shows that in this case, $m' = O(n)$ and $n' = O(n)$; thus, the running time is $O(n)$. Since all heap operations take $O(\log n)$ time, the other operations take $O(n \log n)$ time. Therefore, $init$ runs in $O(n \log n)$ time.

Given a vertex v, procedure $update(N, v)$ can be performed in $O(|R(N, v)| \log n)$ time. To see this, note that computing topological ordering of $R(N, v)$ takes $O(|R(N, v)|)$ time. Each heap operation takes $O(\log n)$ time, and we insert at most $O(|R(N, v)|)$ elements, a total of $O(|R(N, v)| \log n)$ time.

Operation $query(N)$ takes $O(1)$ time because $findMax$ in max-heap can be done in constant time. Since the running time of $delete(\cdot)$ is bounded by time for $update(N, v)$, operation $delete(N, (u, v))$ takes $O(|R(N, v)| \log n)$ time.

Given a path P ends with p_k, operation $contract(N, P)$ takes $O(|P| + |R(N, p_k)| \log n)$ time. This is because $contract(\cdot)$ removes vertices in P in $O(|P|)$ time. The rest of the procedure calls $update(N, p_k)$, which runs in time $O(|R(N, p_k)| \log n)$.

Finally, operation $replace(N, \ell, \ell')$ uses a constant number of calls to heap operations; thus, it runs in $O(\log n)$ time.

B. Ensure subphylogeny-free property

To ensure subphylogeny-free property, some iterations may insert up to $O(n)$ new leaves. This may seem to cause the quadratic running time. Fortunately, we can show that, throughout the algorithm, we insert only $O(n)$ new leaves.

Lemma 4. *Number of new leaves created for ensuring the subphylogeny-free property is $O(n)$*

Proof: Every time we add a new leaf to N and T, we remove 2 old leaves. That means number of leaves in the network will only decrease and original number of leaves is n. Hence, we can add at most $O(n)$ new leaves. ■

C. The running time

From lemmas above, we now analyze running time required for solving TCP for a binary nearly stable network N

and phylogenetic tree T on the same set of leaves of size n. This is our main result.

Theorem 2. *For a binary nearly stable phylogenetic network N and a phylogenetic tree T on the same set of leaves of size n, the tree containment problem can be solved in $O(n \log n)$ time.*

Proof: Preprocessing take $O(n \log n)$ time. From Lemma 2, Gambette *et al*'s algorithm terminates in $O(n)$ iterations. We can use our data structures to reduce time required in each iteration.

Each iteration, by Corollary 1, Gambette *et al.*'s algorithm modify N and T in a way that affected subphylogeny $R(N, w)$ is only constant in size. So the length and the number of paths that need to be contracted is constant.

We get that operations $update(\cdot)$, $delete(\cdot)$, $replace(\cdot)$ and $contract(\cdot)$ run in $O(\log n)$ time.

Each operation of Gambette *et al.* calls our procedure only $O(1)$ times. $Simplify(\cdot)$ calls $delete(\cdot)$ only $O(1)$ times, $ContractDegenerated()$ calls $contract(\cdot)$ only $O(1)$ times and $ConbineLeaves(\cdot)$ calls $replace(\cdot)$ only $O(1)$ times.

Therefore, time required for each iteration exclude $replace(\cdot)$ is bounded by $O(\log n)$.

For $replace(\cdot)$, some iteration may call it many times. From Lemma 4, throughout the algorithm, it will be called only $O(n)$ times. That means $replace(\cdot)$ takes $O(n \log n)$ time.

Hence this algorithm takes $O(n \log n)$ time. ■

V. Conclusion

We can answer tree containment problem for binary nearly stable phylogenetic network and phylogenetic tree on the same set of leaf of size n in $O(n \log n)$ time. An open problem remaining is whether it can be solved in linear-time or in more general case.

References

[1] D. H. Huson, R. Rupp, and C. Scornavacca, *Phylogenetic Networks: Concepts, Algorithms and Applications*. New York, NY, USA: Cambridge University Press, 2011.

[2] J. M. Chan, G. Carlsson, and R. Rabadan, "Topology of viral evolution," *Proceedings of the National Academy of Sciences*, vol. 110, no. 46, pp. 18 566–18 571, 2013.

[3] P. Gambette, A. Gunawan, A. Labarre, S. Vialette, and L. Zhang, "Locating a tree in a phylogenetic network in quadratic time," in *Research in Computational Molecular Biology*, ser. Lecture Notes in Computer Science, T. M. Przytycka, Ed. Springer International Publishing, 2015, vol. 9029, pp. 96–107.

[4] T. Marcussen, K. Jakobsen, J. Danihelka, H. Ballard, K. Blaxland, A. Brysting, and B. Oxelman, "Data from: Inferring species networks from gene trees in high-polyploid north american and hawaiian violets (viola, violaceae)," 2011.

[5] I. A. Kanj, L. Nakhleh, C. Than, and G. Xia, "Seeing the trees and their branches in the network is hard," *Theoretical Computer Science*, vol. 401, no. 13, pp. 153 – 164, 2008.

[6] L. van Iersel, C. Semple, and M. Steel, "Locating a tree in a phylogenetic network," *Information Processing Letters*, vol. 110, no. 23, pp. 1037 – 1043, 2010.

[7] P. A. Jenkins, Y. S. Song, and R. B. Brem, "Genealogy-based methods for inference of historical recombination and gene flow and their application in saccharomyces cerevisiae," *PLoS ONE*, vol. 7, no. 11, p. e46947, 11 2012.

[8] R. Sedgewick and K. Wayne, *Algorithms, 4th Edition.* Addison-Wesley, 2011.