Reconstructing *k*-Reticulated Phylogenetic Network from a Set of Gene Trees

Hoa Vu[1], Francis Chin[1], W.K. Hon[2], Henry Leung[1], K. Sadakane[3], Ken W.K. Sung[4], and Siu-Ming Yiu[1]

[1] Department of Computer Science, The University of Hong Kong
[2] Department of Computer Science, National Tsinghua University, Taiwan
[3] Informatics Research Division, National Institute of Informatics, Japan
[4] Department of Computer Science, National University of Singapore, Singapore

Abstract. The time complexity of existing algorithms for reconstructing a level-x phylogenetic network increases exponentially in x. In this paper, we propose a new classification of phylogenetic networks called *k-reticulated network*. A k-reticulated network can model all level-k networks and some level-x networks with $x > k$. We design algorithms for reconstructing k-reticulated network ($k = 1$ or 2) with minimum number of hybrid nodes from a set of m binary trees, each with n leaves in $O(mn^2)$ time. The implication is that some level-x networks with $x > k$ can now be reconstructed in a faster way. We implemented our algorithm (ARTNET) and compared it with CMPT. We show that ARTNET outperforms CMPT in terms of running time and accuracy. We also consider the case when there does not exist a 2-reticulated network for the input trees. We present an algorithm computing a maximum subset of the species set so that a new set of subtrees can be combined into a 2-reticulated network.

1 Introduction

The study of evolutionary history of a species plays a crucial role in biomedical research. For example, understanding the evolutionary history of a virus (e.g. SARS) may help us deduce the natural reservoirs of the virus, thus identifying the source of the virus. The details of how the virus evolves may help to uncover clues to treat or vaccinate the virus and understand how it evolves resistance to existing drugs. A traditional representation of evolutionary history is phylogenetic tree (a rooted, unordered tree with distinctly labeled leaves, each represents a species or a strain of the species). To construct a phylogenetic tree, a common practice is to select a group of genes, which are believed to be critical for evolution, to represent the species. However, selecting a different set of genes may end up with a different phylogenetic tree (called a gene tree). To deal with this issue, researchers may try to extract the subtrees which are common in all trees (known as the maximum agreement subtree problems, see [1-3] for examples) and ignore the other non-common parts. This may result in a small tree. Also, information not in the common subtree will be lost.

It is now well-known that the differences in the gene trees are not due to errors. There exist evolutionary events (known reticulation events), such as hybridization,

Z. Cai et al. (Eds.): ISBRA 2013, LNBI 7875, pp. 112–124, 2013.

horizontal gene transfer, and recombination, that may cause the genes to evolve differently and a phylogenetic tree is not powerful enough to model the resulting evolutionary history [4]. To model the evolutionary history better, phylogenetic network is proposed. Phylogenetic network is a generalization of phylogenetic tree (note that in this paper, we focus on rooted bifurcating (each node has at most 2 descends) phylogenetic tree/network). Phylogenetic network is defined as a rooted, directed acyclic graph in which (1) exactly one node has indegree 0 (the root), and all other nodes have indegree 1 or 2; (2) all nodes with indegree 2 (hybrid nodes or reticulation nodes) have outdegree 1, and all other nodes have outdegree 0 or 2; and (3) all nodes with outdegree 0 (leaves) are distinctly labeled. For a hybrid node h in a phylogenetic network, every ancestor s of h such that h can be reached using two disjoint directed paths starting from the children of s is called a split node of h (and h is called a hybrid node of s). The edges attached to a hybrid node is called hybrid edges. Figure 1 shows an example. Typically, a split node is used to represent a speciation event (two different species are evolved) while a hybrid node is used to represent the reticulation event between the two descendants of the split node.

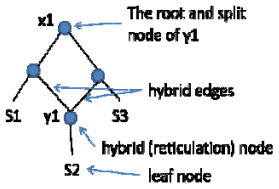

Fig. 1. An example phylogenetic network

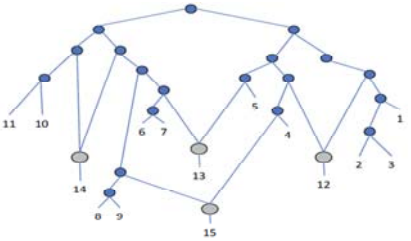

Fig. 2. A level-4 network but is an 2-reticulated network for a set of 15 HIV-1 sequences resulting from 9 gene tr

We say that a phylogenetic network *N* is compatible with (*induces* or *displays*) a set of gene trees if each tree can be obtained from *N* by deleting one of the hybrid edges of each hybrid node and contracting all nodes with outdegree and indegree equal 1 (see Figure 3 for an example). If there is no restriction, for any given set of trees, we can always have a phylogenetic network that induces the trees. However, reticulation events are hard to occur, so a more biological meaningful question is to ask for such a phylogenetic network with the minimum number of hybrid nodes.

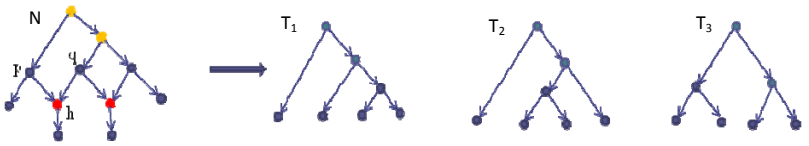

Fig. 3. Network N is compatible with T_1, T_2. T_3

A common classification of phylogenetic network is the *level-x network* [11 – 12]. A level-x network is one in which each biconnected component (also known as *blob* [5]) of the network contains at most x hybrid nodes. A level-0 network is a phylogenetic tree, a level-1 network is also known as a galled tree [5] or a galled network [6]. There are algorithms that reconstruct a level-x network, however, the time complexity increases exponentially in x even if we only consider some restricted cases. Thus, in practice, if $x > 2$, the algorithm is not fast enough. On the other hand, the evolutionary history of quite many viruses can only be modeled by high level networks (with $x > 2$). For example, to capture all known reticulation events of HIV [7], we need to use a level-4 network (Figure 2 shows the network).

In this paper, we propose to consider a new classification of networks by restricting the maximum number of hybrid nodes each split node may have, namely a k-reticulated network is one in which each split node can correspond to at most k hybrid nodes. This new classification is also supported by evidence in real life cases. Several studies of recombination in bacteria have shown that recombination rates decrease as sequence divergence increases [8-9]. These studies imply that the number of recombination events of a split node will be limited as the descendants from the same split node will diverge more as the number of generations increases. This observation is also supported by a computer simulation study [10]. Therefore, networks with limited reticulation events for each split node while no limit on the total number of reticulation events in each blob seem to be more biologically relevant and can model the recombination events in nature more appropriately.

Our Contributions. This new classification of phylogenetic networks is more powerful than level-x networks. By definition, every level-x network is also an x-reticulated network. And some level-x network can be modeled by a k-reticulated network with $k < x$ (see Figure 2 for an example of a level-4 network which is also a 2-reticulated network). So, even solving the problem for k-reticulated network with k as small as 2, some of the meaningful high level networks can be constructed efficiently. We show that given a set of binary gene trees, one can reconstruct an 1-reticulated or 2-reticulated network (if one exists) with minimum number of hybrid nodes compatible with all trees in $O(mn^2)$ time where m is the number of trees and n is the number of leaves. We also consider the problem that when a compatible 2-reticulated network does not exist, compute a subset of species with maximum size so that a 2-reticulated network exists. This problem is believed to be NP-hard and we provide an $O(2^{mm}n^{3m})$ algorithm to solve it. We implement the 2-reticulated network reconstruction algorithm (ARTNET) and compare it with the program CMPT [13] that reports a phylogenetic network with the smallest number of hybrid nodes. We only consider the case when a 2-reticulated network exists for the input set of trees. The experiments show that ARTNET is more efficient than CMPT. When the number of hybrid nodes increases, the running time of ARTNET only increases slightly while that of CMPT increases rapidly. Regarding accuracy evaluation, ARTNET also outperforms CMPT.

Related Work. Several methods of constructing phylogenetic networks have been proposed. Nakhleh *et al.* [6] have developed an algorithm for constructing a level-1 phylogenetic network from two phylogenetic trees running in polynomial time.

However, Nakhleh *et al.*'s algorithm can handle two trees only. Huynh *et al.* [12] have succeeded in providing a $O(|T|^2n^2)$ algorithm reconstructing a galled network from a set T of multiple phylogenetic trees of arbitrary degree. Huson and Klopper [14] gave an $O(n^k)$ algorithm constructing restricted level-k network from a set of trees. A rooted phylogenetic tree can be uniquely represented by the set of triplets obtained by taking all combinations of three leaves in the tree [12]. It takes $O(n^3)$ running time to construct a galled network in the algorithm designed by Jansson, Nguyen and Sung [15]. Extending to level-2 network, Van Iersel *et al.* [11] developed an $O(n^8)$ time algorithm. Habib and To [16] have solved the general problem of constructing level-k network from a dense triplet set T in exponential running-time $O(|T|^{k+1}n^{\lfloor\frac{4k}{3}\rfloor+1})$. Gambette *et al.* [17] have shown that we can decide in optimal $O(n^4)$ time whether there exists a simple unrooted level-1 network for a set of all quartets.

Notations. Let u is a node in a tree T, $\boldsymbol{T[u]}$ = the subtree of T rooted at u, and $L(T)$ = the leaf label set of T. If u is a node in network N, a subnetwork $\boldsymbol{N[u]}$ is obtained from N by only retaining all nodes and their incident edges which are reachable from u, and $\boldsymbol{L(N)}$ is the set of leaf labels of N. Given a subtree t of T, $\boldsymbol{T\backslash t}$ is a subtree obtained by removing t from T. Similarly, with a subnetwork N' of N, $\boldsymbol{N\backslash N'}$ is a network obtained by removing N' from N. Given a tree T with the leaf set L, and $L' \subseteq L$. $\boldsymbol{T|L'}$ denotes a subtree obtained by first deleting all nodes which are not on any directed path from the root to a leaf in L' along with their incident edges, and then, for every node with outdegree 1 and indegree less than 2, contracting its outgoing edge.

2 Algorithms for Reconstructing k-Reticulated Network (k = 1, 2)

Denote $\mathbf{P}(N, T_i, L)$ the procedure to reconstruct a k-reticulated network N compatible with $T_1, T_2, \ldots, T_m$, where k = 1 or 2. We employ the divide-and-conquer technique.

2.1 Reconstructing 1-Reticulated Network

Base Case: if each input tree is a single node with the same label, return a network which is that single node of the same label; otherwise consider the following cases:

Case I: *Bipartition*
$\{T_1,\ldots, T_m\}$ admit a ***leaf-set-bipartition*** (L_1, L_2) if for every tree T_i with root r_i and its children r_{i1} and r_{i2}, $L(T_i[r_{i1}]) = L_1$ and $L(T_i[r_{i2}]) = L_2$, then find $\mathbf{P}(N_1, T_i[r_{i1}], L_1)$ and $\mathbf{P}(N_2, T_i[r_{i2}], L_2)$. If N_1 and N_2 exist, network N is obtained by creating a new node r becoming the parent of the roots of N_1 and N_2. (Fig. 4)

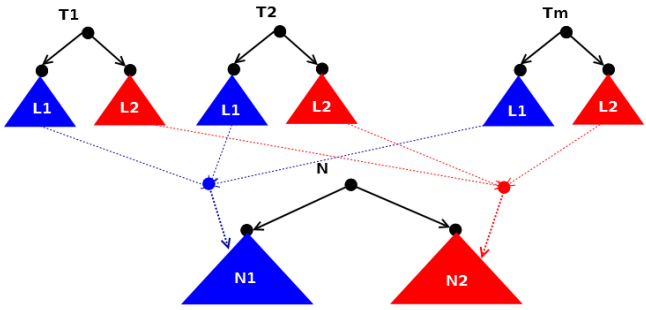

Fig. 4. The tree set admit a leaf set bipartition

Case II: *Tripartition*
$\{T_1,\ldots,T_m\}$ admit a ***leaf-set-tripartition*** (L_1, L_h, L_2) if for every tree T_i with root r_i and its children r_{i1} and r_{i2}, there exists a node $h_i \neq r_i$ such that $L(T_i[h_i]) = L_h$; and if $h_i \neq r_{i1}$ and $h_i \neq r_{i2}$ for every $i = 1\ldots m$; $\{T_i' = T_i \backslash T_i[h_i]\}$ admit a leaf-set-bipartition (L_1, L_2). Otherwise, $L_1 = L(T_i')$ and $L_2 = \emptyset$.

If $h_i \neq r_{i1}$ and $h_i \neq r_{i2}$ for $i = 1\ldots m$, the problem can be divided into 3 subproblems: $\mathbf{P}(N_1, T_i'[r_{i1}], L_1)$; $\mathbf{P}(N_2, T_i'[r_{i2}], L_2)$; and $\mathbf{P}(N_h, T_i[h_i], L_h)$. Network N can be combined from N_1, N_2 and N_h by first creating a new node r to be the parent of the roots of N_1 and N_2. Find node u_1 in N_1 and u_2 in N_2 such that for $i = 1\ldots m$, either u_1 or u_2 corresponds to h_i's sibling s_i. Let v_1 and v_2 be the parent of u_1 and u_2 respectively, create nodes p_1 and p_2 on edges (v_1, u_1) and (v_2, u_2) respectively. A new hybrid node h is created, and let h be a child of p_1 and p_2, and h be the parent of N_h's root (Fig. 5).

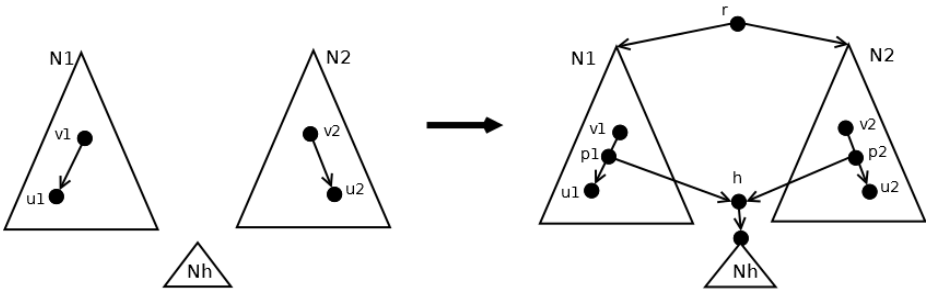

Fig. 5. Combining N_1, N_2 and N_h to get network N

Given network N compatible with tree T, ***a node u in N is said to correspond to a node s in T*** if T can be converted from N by a series of cuts in which any edge contraction related to node u will create a new node that is labeled u, then u becomes s.

If there is a tree T_i in which h_i is a child of the root r_i, the network constructed in this case is skew (i.e., there is a split node such that the path from the split node to its hybrid node is 1). The problem can be divided into 2 sub-problems: $\mathbf{P}(N', T_i', L_1\}$; and $\mathbf{P}(N_h, T_i[h_i], L_h)$. If N' and N_h can be constructed, N can be obtained by first creating a node r and making r become the parent of the root of N'. Find a node u in N' such that for every tree T_i in which h_i is not a child of the root r_i, u corresponds to s_i in T_i', which is the sibling of h_i before removing $T_i[h_i]$. Let v be the parent of u, and a new node p on edge (v, u), create a hybrid node h that is the child of p and r, and h is the parent of the root of N_h (Fig. 6).

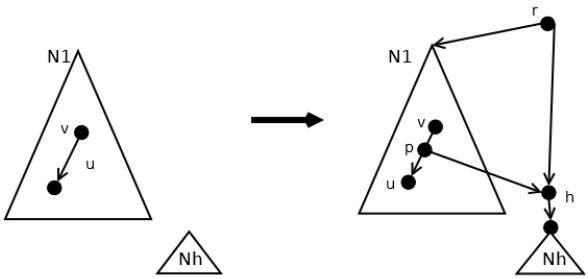

Fig. 6. Combining N_1 and N_h to get a skew network N

2.2 Reconstructing 2-Reticulated Network

To solve this problem, we also consider the base case, Case I and Case II as in the above. In addition, we need to consider Case III – Quadripartition as follows.

Case III: *Quadripartition*
The tree set $\{T_1, T_2, \ldots, T_m\}$ is said to admit a ***leaf-set-quadripartition*** $(L_1, L_{h1}, L_{h2} L_2)$ if for every tree T_i with root r_i and its children r_{i1} and r_{i2}, there exists a node $h_{i2} \notin \{r_i, r_{i1}, r_{i2}\}$ such that $L(T_i[h_{i2}]) = L_{h2}$; and $\{T_i' = T_i \backslash T_i[h_{i2}], i = 1, 2, \ldots, m\}$ admit a leaf-set-tripartition (L_1, L_{h1}, L_2). If there exist a 2-reticulated network N' compatible with $\{T_1', \ldots, T_m\}$; and a 2-reticulated network N_{h2} compatible with $\{T_i[h_{i2}], i = 1, \ldots, m\}$.

If N' is a non-skew network, N' is created by combining three 2-reticulated networks N_1, N_2 and N_{h1} (as case II). Find two nodes a and b in two distinct networks out of three networks N_1, N_2 and N_{h1} such that either a or b corresponds to node s_i in T_i', which is the sibling of h_{i2} in T_i, for $i = 1, 2, \ldots, m$. Attaching N_{h2} to N' is done similarly to case II by creating a new hybrid node h_2 (Fig. 7).

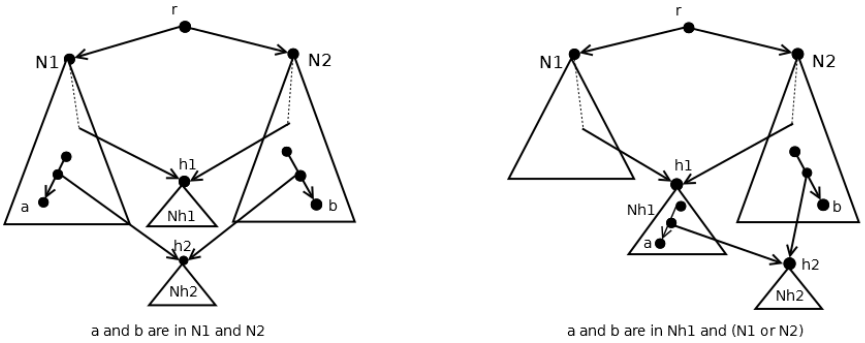

Fig. 7. Combining non-skew network *N'* and N_{h2} to get a 2-reticulated network *N*

If *N'* is a skew-network, *N'* is created by combining two 2-reticulated networks N_1 and N_{h1}. Find nodes *a* and *b* in N_1 and N_{h1} respectively such that for $i = 1 \ldots m$, either *a* or *b* corresponds to node s_i in T_i', which is the sibling of h_{i2} in T_i. Attaching N_{h2} to *N'* by creating a new hybrid node h_2. (Fig.8)

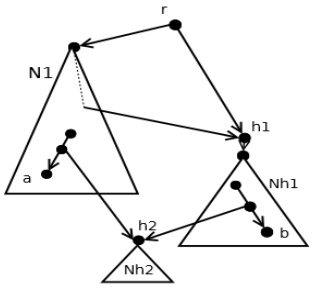

Fig. 8. Combining skew network N' and N_{h2} to get a 2-reticulated network N

2.3 Algorithm Correctness

Lemma 1. Given a network *N* compatible with tree *T* and a node *v* in *N*, if all nodes in a subnetwork *N*[*v*] cannot be reached from any other nodes outside *N*[*v*] without passing through node *v*, then there exists a node *u* in *T* such that its subtree *T*[*u*] and *N*[*v*] have exactly the same leaf label set, and *N*[*v*] is compatible with *T*[*u*].

Theorem 1. The algorithm described in section 3.1 and 3.2 can construct a 2-reticulated network compatible *N* with a given set of trees if and only if *N* exists.

Proof. Assume there is a 2-articulated compatible network *N* for $\{T_1, T_2, \ldots, T_m\}$. Consider the root *r* of N:

Case 1: *r* is the only node in *N*. The theorem is obviously correct.
Case 2: *r* does not correspond to any hybrid node

Let r_1 and r_2 be two children of r, and L_1 and L_2 be the leaf set of $N[r_1]$ and $N[r_2]$ respectively. As r does not correspond to any hybrid node, any node outside $N[r_1]$ (resp. $N[r_2]$) has to pass through r_1 before reaching any node inside $N[r_1]$ (resp. $N[r_2]$). From Lemma 1, there are nodes u_1 and u_2 in every tree T_i such that $L(T[u_1]) = L_1$, and $L(T[u_2]) = L_2$, and $N[r_1]$ and $N[r_2]$ are compatible with $T[u_1]$ and $T[u_2]$ respectively. We have $L_1 \cap L_2 = \emptyset$ and $L_1 \cup L_2 = L$, so in N and every tree T_i, their root is the only common ancestor of any node in L_1 and any node in L_2. This means the input tree set admit a leaf set bipartition, corresponding to case I in the algorithm.

Case 3: r corresponds to one hybrid node h
All nodes in $N[h]$ cannot be reached by any other node not in $N[h]$ without passing through node h, otherwise, there must exist another hybrid node of root r in $N[h]$, contradicting to the fact that r corresponds to exact one hybrid node. By ***Lemma 1***, there exists a node h_i in every tree T_i such that $N[h]$ is compatible with $T_i[h_i]$, and $L(N[h]) = L(T_i[h_i])$; hence, $N \backslash N[h]$ is compatible with $T_i \backslash T_i[h_i]$. As the root of $N \backslash N[h]$ does not correspond to any hybrid node, the argument can be turn back to *Case* 2. This implies that the tree set admit a leaf set tripartition, corresponding to case 2 of the algorithm.

Case 4: r corresponds to two hybrid nodes h_1 and h_2
Let p_1 and q_1 be the parents of h_1, and p_2 and q_2 be the parents of h_2, then either h_1 lies on one of the merge paths from the root r to h_2, or none of the merge paths from r to h_2 (resp. h_1) go through h_1 (resp. h_2) (figure 7).

In both cases, all nodes in the subnetwork $N[h_2]$ cannot be reached by any other node outside $N[h_2]$ without passing through node h_2; otherwise, r would correspond to another hybrid node in $N[h_2]$.

From Lemma 1, there exists a node h_{i2} in every tree T_i such that $N[h_2]$ is compatible with $T_i[h_{i2}]$, and $L(N[h_2]) = L(T_i[h_{i2}])$. Plus, $N \backslash N(h_2)$ is a compatible 2-articulated network of $T_i \backslash T_i[h_{i2}]$. As the root of $N \backslash N(h_2)$ corresponds to exact 1 hybrid node h_1, the argument can turn back to Case 3 above. This implies the tree set admit a leaf set quadripartition, corresponding to Case 3 of the algorithm. □

2.4 Time Complexity

Lemma 2. Determining whether $\{T_1, T_2, \ldots, T_m\}$ admit a leaf set bipartition or tripartition or quadripartition and partitioning every tree can be done in $O(mn)$.

Proof. Denote $LCA(X)$ the lowest common ancestor of all nodes in set X.
For every tree T_i, $i = 1, 2, \ldots, m$, denote r_i the root of T_i, and r_{i1} and r_{i2} are two children of r_i. Define a subset $L^* \subset L$:

$L^* = \emptyset$ if the tree set admit the leaf set bipartition.
$L^* = L_h$ if the tree set admit a leaf set tripartition (L_1, L_h, L_2).
$L^* = L_{h1} \cup L_{h2}$ if the tree set admit a leaf set quadripartition (L_1, L_{h1}, L_{h2}, L_2).

Determine L^*: Let $L_c = L(T_1[r_{11}])$; $L_d = L(T_1[r_{12}])$. It takes $O(mn)$ to divide L_c into two disjoint subsets L_{c1} and L_{c2}, and L_d into two disjoint subsets L_{d1} and L_{d2} (Fig. 9). Pick one leaf node v in L_c, then

For every leaf node $u \in L_c$: If $LCA(v, u)$ is not the root of every tree T_i; u is put in L_{c1}; else, u is put in L_{c2}.

For every leaf node $w \in L_d$: If $LCA(v, w)$ is the root of every tree T_i; w is put in L_{d1}; else w is put in L_{d2}.

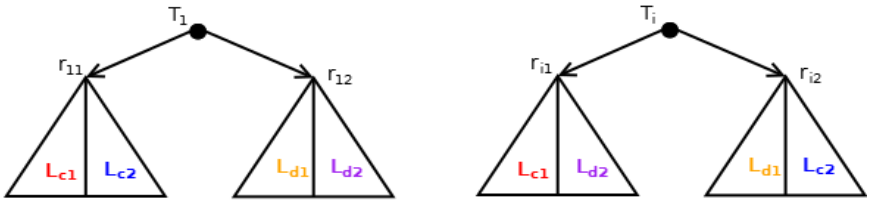

Fig. 9. Partition the leaf set

Claim 1: The tree set admits a leaf set bipartition iff $L_{c2} = L_{d2} = \emptyset$; otherwise check tripartition property.

Claim 2: The tree set admits a leaf set tripartition iff either $L^* = L_{c2} \cup L_{d2}$ or $L^* = L_{c1} \cup L_{d1}$, and there exists a node h_i in every tree T_i such that $L(T_i[h_i]) = L^*$, and $T_i \backslash T_i[h_i]$ admit a leaf set bipartition if h_i is not a child of r_i for every $i = 1 \ldots m$, taking $O(mn)$ time; otherwise, check quadripartition property.

Claim 3: If the tree set admits a leaf set quadripartition (L_1, L_{h1}, L_{h2}, L_2), one of two sets $L_{c1} \cup L_{d1}$ or $L_{c2} \cup L_{d2}$ can be either (1) L_{h1}, or (2) L_{h2}, or (3) $L_{h1} \cup L_{h2}$.
Pick any tree, say T_1, to find the $p_1 = LCA(L_{c1} \cup L_{d1})$ and $p_2 = LCA(L_{c2} \cup L_{d2})$.

1. One of two nodes p_1 or p_2 is the root r_1 and the other is not. Assume $p_1 = r_1$, and p_2 is a proper descendant of r_1, then $L_{c2} \cup L_{d2} = L_{h1}$ or $L_{c2} \cup L_{d2} = L_{h2}$.
2. If both p_1 and p_2 are r_1, find $j = 1$ or 2 such that $LCA(L_{cj})$ and $LCA(L_{dj})$ are the children of the root r_1. If j does not exist, return *"null"*; otherwise, assume $j = 1$, then $L_{h1} \cup L_{h2} = L_{c2} \cup L_{d2}$.

It takes $O(n)$ time to determine L' which is either L_{h1}, or L_{h2} or $L_{h1} \cup L_{h2}$.

- If L' is L_{h1} or $L_{h2} \rightarrow L^* = L'$
  Check if there is a node h_i in every tree T_i such that $L(T_i[h_i]) = L'$ in $O(mn)$. If yes, check if $\{T_i \backslash T_i[h_i], i = 1, \ldots, m\}$ admit a leaf set tripartition (L_1, L_h, L_2). If yes, $L_{h2} = L'$ and $L_h = L_{h1}$; else return "*null*".
  If there is a tree T_j in which there does not exist any node w such that $L(T_j[w]) = L'$ (*Fig.*10). If h_j is $LCA(L')$, then $L' = L_{h1} \cup L_{h2}$, which is examined as case 2 below.

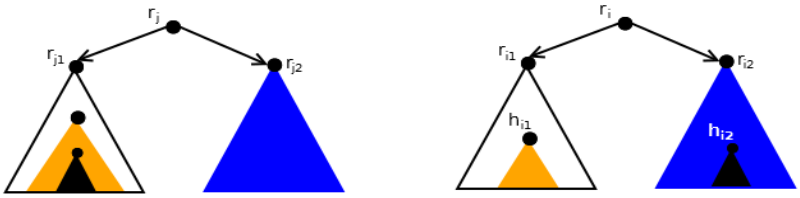

Fig. 10. With a leaf set L_{h1}, find a node w in T_j such that $L(T_j[w]) = L_{h1} \cup L_{h2}$

- If $L' = L_{h1} \cup L_{h2}$.
  Let T_k be a tree in which there is no node satisfying $L' = L(T_k[w])$. If there are exact two nodes h_{k1} and h_{k2} in T_k such that $L(T_k[h_{k1}]) \cup L(T_k[h_{k2}]) = L'$, then either $L(T_k[h_{k1}])$ or $L(T_k[h_{k2}])$ is L_{h2}; otherwise, return *"null"*. Finding h_{k1} and h_{k2} takes $O(n)$. It then takes $O(mn)$ to determine which one, $L(T_k[h_{k1}])$ or $L(T_k[h_{k2}])$, is L_{h2}, and partition every tree T_i into T_i' and $T_i[h_{i2}]$, for $i = 1, 2, \ldots, m$.

In total, checking whether the input trees admit a leaf set bipartition or tripartition or quadripartition, and partition every tree into proper subtrees takes $O(mn)$. □

Lemma 3. Given a network N compatible with m trees $\{T_1, T_2, \ldots, T_m\}$ with the same leaf label set L of size n, and a node s_i in T_i, for $i = 1, 2, \ldots, m$, then finding whether there is a node u in N corresponding to $s_1, s_2, \ldots, s_m$ can be done in $O(n)$.

Proof. For $i = 1, \ldots, m$, let u_i be a node in N having the lowest height in N such that $L(T_i[s_i]) \subseteq L(N[u_i])$.

Claim: Node u exits iff u_i is u or a descendant of u such that all nodes on the path from u to u_i are either hybrid node of a skew split node or non-hybrid nodes whose siblings are hybrid nodes. It takes $O(mn)$ to find the set $\{u_1, u_2, \ldots, u_m\}$ from N (note that u_x can be u_y), and $O(n)$ time to check (*i*) all nodes $\{u_1, u_2, \ldots, u_m\}$ lie on the same directed path; and (*ii*) The siblings of u_i, $i = 1, 2, \ldots, m$, are all hybrid nodes. If two conditions are satisfied, the node u will be the starting node x of the path created by $\{u_1, u_2, \ldots, u_m\}$ or x's sibling if x's sibling is the hybrid node of a skew split node ; otherwise, return "*null*". □

Theorem 2. Constructing a *k*-reticulated network ($k = 1$ or 2) from a set of m binary trees with the same leaf label set L of size n can be done in $O(mn^2)$.

Proof. From *Lemma 2* and *Lemma 3*, dividing and conquering take $O(mn)$ time complexity. There are $O(n)$ nodes in a tree with n leaf nodes. Hence the time complexity of our algorithm is $O(mn^2)$.

3 Maximum 2-Reticulated Network Compatibility Problem

Given a set of binary trees {T_1, T_2, … T_m}, compute the maximum leaf set L such that there exists a 2-reticulated network N compatible with {$T_1|L^*$, $T_2|L^*$, …, $T_m|L^*$}.*

Using brute-force approach by considering all possible subsets of the leaf set, the problem can be done in $O(2^n mn^2)$. However, when $m \ll n$, the following algorithm produces better time complexity.

MCN(T_i) denotes the Maximum compatible 2-reticulated network for T_1, T_2, …, T_m.
MCLS(T_i) denotes the **MC**N(T_i)'s **L**eaf **S**et

Let v be a node in a tree T, ***child*$_1$(v)** and ***child*$_2$(v)** denote two children of node v.

Let ***sib_hyb*(l, h)** be an array of pointers in which *sib_hyb*(l, h)[i] is pointing to a node w_i in tree T_i; where l is a positive integer number, and h receives a value of 1 or 2. The following rules are applied in the process of removing nodes and edges when computing MCLS(T_i):

- If w_i is one of the end node of the edge that is contracted, *sib_hyb*(l, h)[i] will point to a new node created after doing contraction.
- If a whole subtree rooted at w_i is deleted from T_i, *sib_hyb*(l, h)[i] points to the sibling of w_i.
- If a whole subtree rooted at p_i, which is an ancestor of w_i, is deleted, *sib_hyb*(l, h)[i] points to the sibling of p_i.

Theorem 3. Given m trees T_1, …, T_m rooted at r_1,…, r_m respectively. $l_1$and l_2 are global variables initialized = 0; *Base case*: there is a tree that is a single node, $MCLS(T_i, i = 1, \dots, m) = \bigcap_{i=1}^{m} L(T_i)$.

$MCLS(T_i)$ is the set having the maximum size of the following terms:

1. $\max\{MCLS(T_{ia}[child_1(r_{ia})], T_{ib}), MCLS(T_{ia}[child_2(r_{ia})], T_{ib})$; with $i_1 \geq 1$ and $i_2 \geq 1$ $\{T_{ia}, a = 1 \dots i_1\} \cup \{T_{ib}, b = 1 \dots i_1\} = \{T_1, \dots, T_m\}$ *and* $\{T_{ia}, a = 1 \dots i_1\} \cap \{T_{ib}, b = 1 \dots i_1\} = \emptyset\}$;
2. $\boldsymbol{MCLS1}(T_i) = \max\{ MCLS(T_i[child_{c_i}(r_i)]) + MCLS(T_i[child_{d_i}(r_i)]) ;$ where $(c_i, d_i) \in \{(1,2),(2,1)\}$, for $i = 1, \dots, m\}$;
3. $\boldsymbol{MCLS2}(T_i) = \max\{MCLS1(T_i \backslash T_i[v_i]) + MCLS(T_i[v_i]) ;$ where v_i is some node in T_i, The sibling of v_i is pointed by *sib_hyb*(++l_1, 1)[i], for $i = 1 \dots m\}$;
4. $\max\{MCLS2(T_i \backslash T_i[w_i]) + MCLS(T_i[w_i])$, where w_1 is a node in T_i. The sibling of w_i is pointed by *sib_hyb*(++l_2, 2)[i], for $i = 1 \dots m\}$;

Before computing $MCLS(t_i)$, if there is a tree t_p whose root is pointed by any $sib\_hyb(l_1, 1)[p]$ (resp. $sib\_hyb(l_2, 2)[i]$) with a specific value l_1 (resp. l_2), then for every other tree t_j containing a node s_j that is pointed by $sib\_hyb(l_1, 1)[j]$ (resp. $sib\_hyb(l_2, 2)[j]$), replace $t_j = t_j[s_j]$ in computing $MCLS(t_i)$.

***Time Complexity*:** By applying dynamic programming, and backtracking on the recursive equations, the problem can be computed in $O(2^m mn^{3m})$. □

4 Experiments

We evaluate and compare the performance of our method, namely ARTNET, with the program CMPT [13] which constructs a network with the smallest number in reticulation from a set of binary trees. NETGET [10] is used to generate random networks. For every 2-reticulated network simulated, we produce a certain number of induced binary trees which are the input of both programs ARTNET and CMPT. We use n (number of leaf node) = 40. Figure 11 shows that we run faster than CMPT. Following [11], we use split-based false negative (FN) and false positive (FP) rates to measure the error rates of the methods. Figure 12 shows that ARTNET produces fewer false positives than CMPT. On the other hand, CMPT and ARTNET have similar performance in false negative rates.

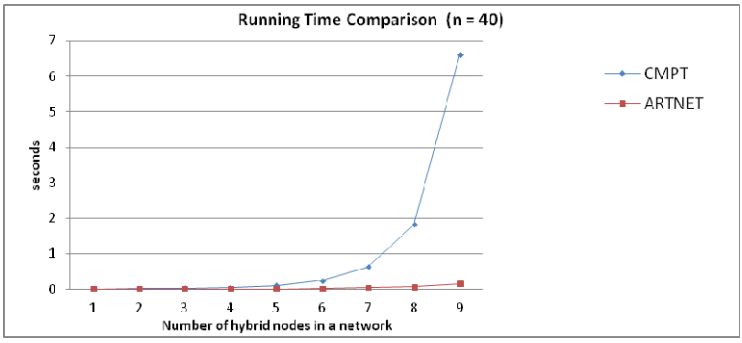

Fig. 11. Time comparison between ARTNET and CMPT

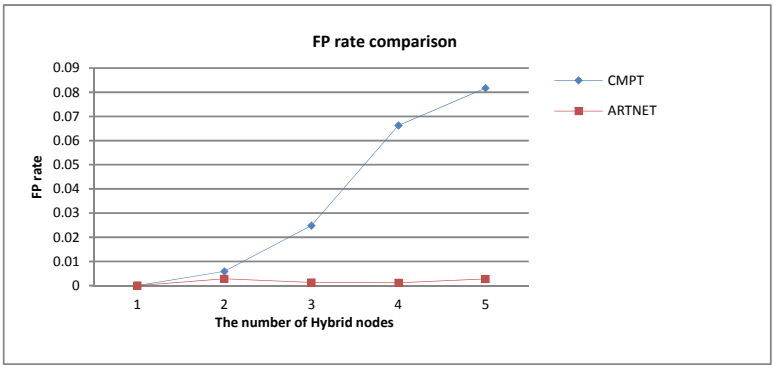

Fig. 12. Comparing the false positive rate between ARTNET and CMPT

Acknowledgement. The project is partially supported by in part by the General Research Fund (GRF) of the Hong Kong Government (HKU 719709E) and Kunihiko Sadakane is supported in part by KAKENHI 23240002.

References

[1] Lam, T.W., Sung, W.K., Ting, H.F.: Computing the unrooted maximum agreement subtree in sub-quadratic time. Nordic Journal of Computing 3(4), 295–322 (1996)
[2] Farach, M., Thorup, M.: Sparse dynamic programming for evolutionary-tree comparison. SIAM Journal on Computing 26(1), 210–230 (1997)
[3] Steel, M., Warnow, T.J.: Kaikoura tree theorems: Computing the maximum agreement subtree. Information Processing Letters 48, 77–82 (1993)
[4] Ford Doolittle, W.: Phylogenetic classification and the universal tree. Science 284(5423), 2124–2128 (1999)
[5] Gusfield, D., Bansal, V.: A fundamental decomposition theory for phylogenetic networks and incompatible characters. In: Miyano, S., Mesirov, J., Kasif, S., Istrail, S., Pevzner, P.A., Waterman, M. (eds.) RECOMB 2005. LNCS (LNBI), vol. 3500, pp. 217–232. Springer, Heidelberg (2005)
[6] Nakhleh, L., Warnow, T., Linder, C.R.: Reconstructing reticulate evolution in species – theory and practice. In: Proceedings of the 8th Annual International Conference on Research in Computational Molecular Biology (RECOMB 2004), pp. 337–346 (2004)
[7] Lee, W.-H., Sung, W.-K.: RB-finder: An improved distance-based sliding window method to detect recombination breakpoints. In: Speed, T., Huang, H. (eds.) RECOMB 2007. LNCS (LNBI), vol. 4453, pp. 518–532. Springer, Heidelberg (2007)
[8] Falush, D., Torpdahl, M., Didelot, X., Conrad, D.F., Wilson, D.J., Achtman, M.: Mismatch induced speciation in salmonella: model and data. Philos. Trans. R Soc. Lond. B Biol. Sci. 361(1475), 2045–2053 (2006)
[9] Majewski, J.: Sexual isolation in bacteria. FEMS Microbiol. Lett. 199(2), 161–169 (2001)
[10] Fraser, C., Hanage, W.P., Spratt, B.G.: Recombination and the nature of bacterial speciation. Science 315(5811), 476–480 (2007)
[11] van Iersel, L., Keijsper, J., Kelk, S., Stougie, L., Hagen, F., Boekhout, T.: Constructing level-2 phylogenetic networks from triplets. In: Vingron, M., Wong, L. (eds.) RECOMB 2008. LNCS (LNBI), vol. 4955, pp. 450–462. Springer, Heidelberg (2008)
[12] Huynh, T.N.D., Jansson, J., Nguyen, N.B., Sung, W.-K.: Constructing a smallest refining galled phylogenetic network. In: Miyano, S., Mesirov, J., Kasif, S., Istrail, S., Pevzner, P.A., Waterman, M. (eds.) RECOMB 2005. LNCS (LNBI), vol. 3500, pp. 265–280. Springer, Heidelberg (2005)
[13] Zhi-Zhong, C., Lusheng, W.: Algorithms for Reticulate Networks of Multiple Phylogenetic Trees. IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB) 9(2), 372–384 (2012)
[14] Huson, D.H., Klöpper, T.H.: Beyond galled trees - decomposition and computation of galled networks. In: Speed, T., Huang, H. (eds.) RECOMB 2007. LNCS (LNBI), vol. 4453, pp. 211–225. Springer, Heidelberg (2007)
[15] Jansson, J., Nguyen, N.B., Sung, W.-K.: Algorithms for combining rooted triplets into a galled phylogenetic network. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 349–358 (2005)
[16] Habib, M., To, T.-H.: Constructing a minimum phylogenetic network from a dense triplet set. J. Bioinformatics and Computational Biology 10(05) (2012)
[17] Gambette, P., Berry, V., Paul, C.: Quartets and Unrooted Phylogenetic Networks. Journal of Bioinformatics and Computational Biology (2011)