BIOINFORMATICS

Vol. 00 no. 00 2010
Pages 1–9

Close Lower and Upper Bounds for the Minimum Reticulate Network of Multiple Phylogenetic Trees

Yufeng Wu

Department of Computer Science and Engineering, University of Connecticut, Storrs, CT 06269, U.S.A.

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Associate Editor: XXXXXXXX

ABSTRACT

Motivation: Reticulate network is a model for displaying and quantifying the effects of complex reticulate processes on the evolutionary history of species undergoing reticulate evolution. A central computational problem on reticulate networks is: given a set of phylogenetic trees (each for some region of the genomes), reconstruct the most parsimonious reticulate network (called the minimum reticulate network) that combines the topological information contained in the given trees. This problem is well known to be NP-hard. Thus, existing approaches for this problem either work with only two input trees or make simplifying topological assumptions.

Results: We present novel results on the minimum reticulate network problem. Unlike existing approaches, we address the fully general problem: there is no restriction on the number of trees that are input, and there is no restriction on the form of the allowed reticulate network. We present lower and upper bounds on the minimum number of reticulation events in the minimum reticulate network (and infer an approximately parsimonious reticulate network). A program called *PIRN* implements these methods, which also outputs a graphical representation of the inferred network. Empirical results on simulated and biological data show that our methods are practical for a wide range of data. More importantly, the lower and upper bounds *match* for many datasets (especially when the number of trees is small or reticulation level is low), and this allows us to solve the minimum reticulate network problem *exactly* for these datasets.

Availability: A software tool, *PIRN*, is available for download from the web page: http://www.engr.uconn.edu/~ywu.

Contact: ywu@engr.uconn.edu

1 INTRODUCTION

Reticulate evolution, a form of evolution with hybridization and genetic exchanges between two species, are common in many organisms: bacteria, plants, fish, amphibians and many others. For better understanding of reticulate evolution, several reticulate evolutionary models have been proposed and actively studied to address various reticulate processes, such as hybrid speciation, lateral gene transfer and recombination. Since most of these models are in the forms of networks, we call them reticulate networks [1]. We refer the readers to [16, 12, 11, 22, 18] for surveys of different reticulate network models.

The *key* computational problem related to these models is the *inference* of reticulate networks. Depending on the types of biological processes involved, data for network inference may be in different forms, such as phylogenetic trees for some short genomic regions (called genes in this paper) or aligned DNA sequences. In this paper, we focus on inferring reticulate networks from a set of correlated phylogenetic trees. Here is the biological motivation for our problem. Suppose multiple phylogenetic trees (called *gene* trees in this paper) are reconstructed, each from some gene for these species. Due to reticulate evolution, different genomic regions (say genes) may be inherited from different ancestral genomes and their evolutionary histories may not be the same (but are still related). Thus, these trees are *correlated* but *not* identical. *No* single phylogenetic tree can faithfully model the evolution of the species, and a more complex network model (i.e. reticulate network as studied in e.g. [2, 14, 19, 11, 22]) is needed.

Imagine we are given a set of "true" gene trees and a "true" reticulate network that models the evolutionary history of these genes. The network can be considered as a compact representation of these gene trees in the sense that one should be able to "trace" a gene tree within the network. We say such a gene tree is *displayed* in the network. This motivates a natural problem, which is called "the holy grail of reticulate evolution" in [18]: given a set of gene trees, reconstruct a reticulate network that displays *every* given gene tree. Such an inferred network reveals important correlation of evolutionary history of multiple genes. Since there exists many such networks, a common formulation is to find the one with the *fewest* reticulation events. Such a network is called the *minimum* reticulate network. The central computational problem on reticulate networks, the *minimum reticulate network problem*, is: given a set of gene trees, reconstruct the minimum reticulate network that displays these gene trees. This formulation may be reasonable when reticulation is believed to be rare.

In general, this problem is computationally challenging: even the case with only *two* gene trees is known to be NP-complete [9, 4, 5]. There are several existing approaches for reconstructing the *exact* minimum reticulate networks when there are only two gene trees [3, 17, 17, 24, 25]. Clearly restricting to just two gene trees is a *big* limitation: more gene trees will be more informative to phylogenetic inference, and DNA sequences of many genes are available. Alternatively, there are also a number of approaches making

[1] Other terms have been used in literature, such as phylogenetic networks and hybridization networks.

simplifications to the reticulate network model, e.g. by imposing additional topological constraints on reticulate networks [7, 20, 13, 15] or working with small-scale tree topological features (e.g. [14, 13, 23]). Such simplification often leads to significantly faster approaches. However, it is sometimes unclear how biologically meaningful these added topological constraints are. Even in the case where additional simplifications are reasonable, one may still want to compare with the unconstrained minimum reticulate networks.

Contributions In this paper, we present new approaches for the minimum reticulate network problem with *three or more* gene trees for unconstrained, general, reticulate networks (e.g., without needing to assume that the network has some restricted form, such as being a galled-tree or galled network). Thus our work is more general than some previous approaches (e.g. [14, 13, 15]). In particular, we develop a *lower* bound (the *RH* bound) and an *upper* bound (the *SIT* bound) for the minimum reticulate network problem with multiple gene trees. We show the correctness of the bounds. We give a closed-form formula for the *RH* bound for the case of three gene trees. We also show how to compute these bounds efficiently in practice using integer linear programming (ILP). Practical results on simulated and real biological data show that the bounds can be computed for wide range of data. Moreover, the lower and upper bounds are often *close*, especially when the number of trees is small or reticulation level is relatively low. In fact, for many simulated datasets of this type, the lower and upper bounds often match, which means our methods can reconstruct the *exact* minimum reticulate networks for these datasets. We also show the *RH* bound clearly outperforms a simple bound.

2 DEFINITIONS AND BACKGROUND

Throughout this paper, we assume trees are rooted. A phylogenetic tree is rooted and leaf-labeled by a set of species (called taxa). A leaf of a phylogenetic tree corresponds to an extant species. An internal vertex corresponds to a speciation event. In-degrees of all vertices (also called nodes), except the root, in a tree are one, while out-degrees are zero for leaves and at least two for internal nodes. A binary phylogenetic tree requires out-degrees of internal nodes to be two. A non-binary phylogenetic tree contains nodes with out-degree of three or more. Many existing phylogenetic methods assume binary phylogenetic trees, although sometimes only non-binary trees can be reconstructed in practice.

Our definition of reticulate networks is similar to that in [14, 22, 11] (also see [8, 19, 18]). A reticulate network (sometimes simply network) is a directed acyclic graph with vertex set V and edge set E, where some nodes in V are labeled by taxa. V can be partitioned into V_T (called tree nodes) and V_R (called reticulation nodes). E can be partitioned into E_T (called tree edges) and E_R (called reticulation edges). Moreover,

1. No nodes with total (in and out) degree of two is allowed. Except the root, each node must have at least one incoming edge.
2. V_R contains nodes whose in-degrees are two or more. V_T contains nodes whose in-degrees are one.
3. E_R contains edges that go into some reticulation nodes. E_T contains edges that go into some tree nodes.

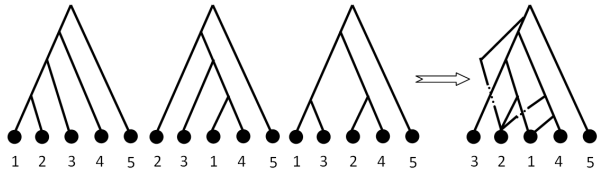


Fig. 1. An illustration of a reticulate network with three reticulation events for three trees. Each tree is displayed in the network: the tree can be obtained by keeping one incoming edge at each reticulation node.

4. A node is labeled by some taxon iff its out-degree is zero. This helps to ensure labeled nodes correspond to extant species and remove some redundancy in the network.

In addition, we have one more restriction:

R_1 For a reticulate network $\mathcal{N}$, when only *one* of the incoming edges of each reticulation node is kept and the rest are deleted, we always derive a tree T'.

We first consider the derived tree T' (that is embedded in $\mathcal{N}$) as in restriction R_1. When we recursively remove non-labeled leaves and contract edges to remove degree-two nodes of T' (called cleanup), we obtain a phylogenetic tree T (for the same set of species as in $\mathcal{N}$). Now suppose we are given a phylogenetic tree T. We say T is *displayed* in $\mathcal{N}$ when we can obtain an induced tree T' from $\mathcal{N}$ by properly choosing a single edge to keep at each reticulation node so that T' is topologically *equivalent* to T after cleanup. We denote the induced T' (if exists) as $T_{\mathcal{N}}$. See Figure 1 for an illustration.

Note restriction R_1 implies the network is *acyclic*. Biologically, reticulate networks often forbid cycles. This is because many reticulation events need to be properly time-ordered. Thus, we focus on *acyclic* reticulate networks in this paper. That is, when we refer to a reticulate network, we mean an acyclic reticulate network (unless otherwise stated).

There are subtle issues related to networks with nodes whose out-degrees are more than two (called non-binary nodes). See the supplemental materials for more discussion. Note that we do not require that in-degrees of reticulation nodes are precisely two as what was imposed in [14]. We also assume the root of each input tree T_i is attached to an outgroup species o. The root of a reticulate network for these trees is also attached to o.

We define the reticulation number of a reticulation node as its in-degree minus one. For a reticulate network $\mathcal{N}$, we define the reticulation number (denoted as $R_{\mathcal{N}}$) as the summation of the reticulation number of each reticulation node in the network. Sometimes $R_{\mathcal{N}}$ is also called the number of reticulation events in $\mathcal{N}$. For the reticulate network in Figure 1, the reticulation node 2 has three entering edges, and the other reticulation node 1 has two entering edges. Thus, $R_{\mathcal{N}} = (3-1)+(2-1) = 3$. Our definition of reticulation number is similar to that of the hybridization number in [3, 22].

Suppose we are given a set of K gene trees $T_1, T_2, \ldots, T_K$ (for the same set of species). The minimum reticulate network $\mathcal{N}_{min}$ for $T_1, T_2, \ldots, T_K$ is a reticulate network $\mathcal{N}$ that displays *each* T_i and $R_{\mathcal{N}}$ is *minimized* among all possible $\mathcal{N}$. We call

$R_{\mathcal{N}_{min}}$ the reticulation number of $T_1, \ldots, T_K$, which is denoted as $R(T_1, T_2, \ldots, T_K)$. For the special case of $K = 2$, we call $D_{T_i,T_j} = R(T_i, T_j)$ the reticulation *distance* between two trees T_i and T_j. Now we formulate the central problem in this paper.

The General Minimum Reticulate Network (GMRN) Problem. Given a set of phylogenetic trees $T = \{T_1, \ldots, T_K\}$, reconstruct the minimum reticulate network $\mathcal{N}_{min}$ for $T_1, T_2, \ldots, T_K$. This formulation is based on parsimony, and may be justified when reticulation is relatively rare in the evolutionary history.

One should note that the GMRN problem can be further specified by the type of input trees. There are two types of input phylogenetic trees: binary or non-binary. For a non-binary tree T, we say T is displayed in $\mathcal{N}$ if some refinement of T (i.e. splitting the non-binary nodes in T in some way to make T binary) is displayed in $\mathcal{N}$. When input trees are binary, network reconstruction may be easier. For simplicity, in this following, the input trees are assumed to be *binary*, unless otherwise stated. We remark that some of our results are applicable to non-binary input trees: the *RH* bound in Section 3 clearly works for non-binary trees too, and the high-level approach of the *SIT* bound may also be applicable to non-binary trees.

Previous Work on the GMRN problem. There is an exact method for the $K = 2$ case of the minimum reticulate network problem [3], although this special case is known to be NP-complete [5]. It is useful to note that when we allow cycles in the network, the minimum reticulate network problem is equivalent to the rooted subtree prune and regraft (rSPR) distance problem. The rSPR distance problem, another NP-complete problem [9, 4], is well known to be closely related to reticulate evolution. Previously, we showed that the rSPR distance can often be practically computed for many moderately sized trees [24]. We give more background to the rSPR distance problem in the supplemental material. It was shown in [1] that the reticulation number (called hybridization number in [1]) for trees T_1 and T_2 is closely related to the rSPR distance between T_1 and T_2, although the two values are not always equal. The main difference between the rSPR distance and the reticulation number is that the latter forbids *cycles* and thus can be more realistic biologically. Recently, we have extended our previous approach in [24] to allow computing the pairwise *reticulation* distance between two rooted binary trees [25]. Although the worst case running time of the practical methods in [3, 24, 25] are exponential, these methods may work reasonably well in practice. As shown in [3, 24, 25], exact reticulation number (with or without cycles) can be computed for two quite different trees with 20 or more leaves. Thus, although intractable theoretically, the two-tree minimum reticulate network problem can be solved in practice if the size of two trees is moderate or the two trees are not very different topologically.

It becomes more computationally challenging when there are three or more gene trees. There is currently *no* known practical methods for either computing the reticulation number $R(T_1, \ldots, T_K)$ or reconstructing $\mathcal{N}_{min}$ for trees $T_1, \ldots, T_K$ when $K \geq 3$. Often approximation is made. A common approach is to impose structural constraints to limit the complexity of the network (e.g. [7, 20, 13, 15]). Although these approaches are theoretically interesting and have been shown to work for some biological data, it is still very desirable to explore the reconstruction of reticulated networks displaying multiple complete gene trees *without* additional structural constraints.

3 A LOWER BOUND

We now focus on developing a lower bound on $R(T_1, \ldots, T_K)$. The lower bound helps to better quantify the *range* of $R(T_1, \ldots, T_K)$.

Recall that several exact methods [3, 25] exist for computing the *pairwise* reticulation distance D_{T_i,T_j} for two trees T_i, T_j, which are practical for many pairs of trees of moderate sizes. Now suppose that we compute D_{T_i,T_j} for each pair of trees T_i and T_j, using the methods [3, 25]. We store these pairwise distances in a matrix D, where $D[i, j] = D_{T_i,T_j}$ [2]. Admittingly, computing $D[i, j]$ for all T_i and T_j can be slow when K and/or the size of trees are large (unless T_i and T_j are very similar). One should note that the GMRN problem is much more complex, and thus, calculation of $D[i, j]$ is justifiable computationally. This leads to the following question: can we use the pairwise reticulation distances D to estimate $R(T_1, \ldots, T_K)$ when $K \geq 3$?

Clearly, the *largest* value $D[i_0, j_0]$ in D is necessarily a *lower* bound of $R(T_1, \ldots, T_K)$ when $K \geq 3$: a reticulate network displaying all trees certainly also displays trees T_{i_0} and T_{j_0} and thus is a reticulate network for T_{i_0} and T_{j_0}. We now show a *stronger* lower bound (called *RH* bound) based on D values.

Here is the high-level idea. The pairwise distance D_{T_i,T_j} specifies how similar trees T_i and T_j are: the larger D_{T_i,T_j} is, the more different T_i and T_j are. Recall that if tree T_i is displayed in a network $\mathcal{N}$, we should be able to derive T_i by keeping only one incoming edge at each reticulation node and performing cleanup. The choice (called display choice for T_i) of keeping which incoming edge at each reticulation node for a tree T_i may not be unique. However, clearly if one makes the same display choices for T_i and T_j when displaying T_i and T_j in $\mathcal{N}$, then T_i and T_j will be identical. More generally, the more similar the display choices for trees T_i and T_j, the closer T_i and T_j will be. Thus, to allow an $\mathcal{N}$ for trees with pairwise distances $D[i, j] = D_{T_i,T_j}$, we need to make display choices for the trees different enough: if the display choices for T_i and T_j are too similar, it will lead to contradiction when $D[i, j]$ suggests T_i and T_j are more different. In the following, a rigorous analysis based on this idea allows us to decide whether an $\mathcal{N}$ with a specific number (say r) of reticulation events is feasible.

To fix ideas, we first consider the situation where each reticulation node in $\mathcal{N}$ has in-degree of two. We will remove this assumption in a moment. Suppose that $\mathcal{N}$ has r reticulation nodes (each with two incoming edges). For each reticulation node, we arbitrarily call one incoming edge the left edge and the right edge for the other. We encode the left edge as 0 and the right edge as 1, and call these two edges 0-edge and 1-edge. Recall that to display a tree, we need to keep exactly one of these two edges. Since a tree T_i is displayed in $\mathcal{N}$, we create a binary vector $v_i[1 \ldots r]$ to represent which incoming edge T_i is kept at each reticulation node V_j in $\mathcal{N}$. Here, $v_i[j]$ is 0 if T_i keeps the 0-edge at reticulation node V_j, and 1 if T_i keeps the 1-edge at V_j. We call v_i the *display vector* for T_i. For example, in Figure 1, consider the reticulation node labeled as 1, and we assign the left/right edges as shown. T_1 and T_3 keep the left edge at this node, while T_2 keeps the right edge. Thus, v_1 and v_3 have value 0, and v_2 has value 1 at the node.

[2] It is *not* known whether pairwise reticulation distance satisfies the triangle inequality, although it clearly does when cycles are allowed. In practice, pairwise reticulation distances often obey the triangle inequality: we have not found a counter-example from many simulation datasets.

For a given T_i and a network $\mathcal{N}$, v_i can always be constructed (at least conceptually) based on how T_i is displayed in $\mathcal{N}$. Note that if there are multiple choices to display T_i, we simply pick an arbitrary one and this does not affect our solution. We define $D_h[v_i, v_j]$ as the Hamming distance between two display vectors v_i and v_j. Here, v_i and v_j (and thus $D_h[v_i, v_j]$) depend on $\mathcal{N}$. To simplify notations, we do not explicitly include $\mathcal{N}$ in their definitions. Lemma 3.1 is crucial to our lower bound.

LEMMA 3.1. *For any two trees T_i and T_j displayed in a reticulate network $\mathcal{N}$, $D_h[v_i, v_j] \geq D[i, j]$.*

PROOF. For contradiction, assume $D_h[v_i, v_j] < D[i, j]$. Thus T_i and T_j make different choices at less than $D[i, j]$ reticulation nodes of $\mathcal{N}$. Imagine we remove from $\mathcal{N}$ those incoming edges at reticulation nodes that are *not* kept by both T_i and T_j. This produces a network with less than $D[i, j]$ reticulation nodes. This is because all reticulation nodes where v_i and v_j match (and thus T_i and T_j keep the *same* incoming edges) have only one incoming edge and are no longer reticulation nodes in the reduced network. This contradicts the fact that $D[i, j]$ is the reticulation distance between T_i and T_j. □

Lemma 3.1 implies that if a network $\mathcal{N}$ with r reticulation events exists, then we should be able to find binary vectors v_i (of length r) for each tree T_i, and $D_h[v_i, v_j] \geq D[i, j]$ for any two such vectors v_i and v_j. On the other hand, if such vectors do *not* exist, we know that at least $r + 1$ reticulation events are needed (and the value $r + 1$ is a lower bound on $R(T_1, \ldots, T_K)$). We can illustrate this formulation more intuitively using a binary *hypercube*. On a hypercube with r binary bits per node, we want to know whether we can pick K points $v_1 \ldots v_K$ that are far apart enough such that the Hamming distance between v_i and v_j is at least $D[i, j]$ for each i and j. One should note this is not always feasible due to the limited size of the hypercube. Formally,

The Binary Hypercube Point Placement Problem. Can we choose K nodes $v_1, \ldots, v_K$ from a r-dimensional binary hypercube so that $D_h[v_i, v_j] \geq D[i, j]$ for each pair of v_i and v_j?

A lower bound on $R(T_1, \ldots, T_K)$ based on the Hypercube Point Placement problem is to find (possibly in a binary search style) the *smallest* integer r such that the Hypercube Point Placement problem *has* a solution. Such r is necessarily a lower bound on $R(T_1, \ldots, T_K)$. We call this lower bound Reticulation on Hypercube bound (or **RH** bound).

We do not know a polynomial-time algorithm for the Binary Hypercube Point Placement problem with more than three trees. When $K = 3$, however, the *RH* bound has a simple analytical form (see Section 3.2). To develop a practical method for the general case, we use integer linear programming (ILP) to solve this problem. We create a binary variable $V_{i,k}$ to represent the coordinates for point v_i. That is, the coordinates of v_i on the hypercube are specified by $V_{i,1} \ldots V_{i,r}$. Without loss of generality, we set $V_{1,k} = 0$ for all $1 \leq k \leq r$. We create a binary variable $M_{i,j,k}$ for each v_i, v_j and position k ($1 \leq k \leq r$) to indicate whether two vectors v_i and v_j match at position k. $M_{i,j,k} = 1$ if $V_{i,k} = V_{j,k}$, and 0 otherwise. Now, we have the following formulation.

Optimization goal: none (since this is a feasibility problem)

Subject to

1 $M_{i,j,k} + V_{i,k} + V_{j,k} \geq 1$, for each v_i, v_j, where $i < j$ and $1 \leq k \leq r$.

2 $M_{i,j,k} - V_{i,k} - V_{j,k} \geq -1$, for each v_i, v_j, where $i < j$ and $1 \leq k \leq r$.

3 $\sum_{k=1}^{r} M_{i,j,k} \leq r - D[i, j]$, for each v_i, v_j, where $i < j$.

For each $1 \leq i \leq K$ and $1 \leq k \leq r$, there is a binary variable $V_{i,k}$.

For each $1 \leq i < j \leq K$, and $1 \leq k \leq r$, there is a binary variable $M_{i,j,k}$.

Constraint 1 says if both $V_{i,k}$ and $V_{j,k}$ are 0, $M_{i,j,k}$ is 1 (i.e. matched). Similarly, constraint 2 says if both $V_{i,k}$ and $V_{j,k}$ are 1, $M_{i,j,k}$ is 1 (i.e. matched). Constraint 3 imposes the pairwise Hamming distance requirement. Our experience shows that the ILP is practical to solve for all datasets we simulated (see Section 5).

3.1 Networks with In-degree of Three or More

We now resolve the remaining issue where some reticulation nodes have in-degree of three or more. In this section, we call a reticulation node "refined" if its in-degree is two, and "unrefined" if its in-degree is at least three.

Here we can no longer represent a reticulation node as binary value, as done previously. So we extend our definitions of display vectors v_i to allow v_i to be non-binary. That is, if there are d incoming edges at a reticulation node, we allow v_i to be from 0 to $d - 1$, where the value indicates which one of the d branches T_i is kept at this node. The incoming edges are numbered starting from zero on the left and to the right with increment of one. We still let $D_h[v_i, v_j]$ be the Hamming distance between vectors v_i and v_j. In this general case, Lemma 3.1 still holds for non-binary vectors v_i and v_j. To see this, we prune any incoming edge at reticulation nodes if it is not chosen by T_i and T_j. Then each remaining reticulation node has only *two* incoming edges (since we only have two trees). Thus, there are $D_h[v_i, v_j]$ reticulation events in this reduced network, and the rest of proof for Lemma 3.1 follows.

We now show that it is not necessary to consider unrefined reticulation nodes in the sense that if a network $\mathcal{N}$ with unrefined reticulation nodes satisfies pairwise distances D, then there exists another network $\mathcal{N}'$ that has only refined reticulation nodes and gives the binary vectors v_i satisfying the pairwise distance constraints of D. That is, if we can not find a network with only refined reticulation nodes, we also can not find a network with unrefined reticulation nodes and the same reticulation number.

To see this property, we consider a network $\mathcal{N}$ with one reticulation node q with $d \geq 3$ incoming edges. Then we transform $\mathcal{N}$ to $\mathcal{N}'$ by replacing q with $q_1, \ldots, q_{d-1}$, where each q_i is a reticulation node with in-degree of two. Note that we do not have to ensure $\mathcal{N}$ and $\mathcal{N}'$ are equivalent: we only need to show $\mathcal{N}'$ gives a solution to the Binary Hypercube Point Placement problem. Clearly, $\mathcal{N}$ and $\mathcal{N}'$ have the same reticulation number (although vectors for $\mathcal{N}'$ are longer). Now, suppose tree T_i keeps edge j at q (where $0 \leq j \leq d - 1$), we then keep edge 1 at q_j in $\mathcal{N}'$ if $j \geq 1$ (and 0 if $j = 0$), and keep edge 0 for all other $q_{j'}$ (where $j' \neq j$). In other words, we create a mapping of the display vectors v_i from $\mathcal{N}$ to $\mathcal{N}'$ for each T_i. Note that such mapping ensures that if two trees keep the same edge at q, they will keep the same edges at

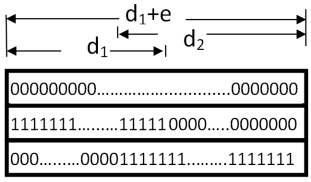


Fig. 2. Vectors v_1, v_2 and v_3 (listed from top to bottom) that maximize the Hamming distance between v_2 and v_3. v_1 is all-0, while the suffix of v_2 and prefix of v_3 are zeros.

$q_1, \ldots, q_{d-1}$ in $\mathcal{N}'$; otherwise, they will keep at least one different incoming edge at $q_1, \ldots, q_{d-1}$ in $\mathcal{N}'$. In either case, if the pairwise distance constraints are satisfied in $\mathcal{N}$, they are also satisfied in $\mathcal{N}'$. So, if we can not find display vectors $\mathcal{N}'$ for networks with refined reticulation nodes only, we also can not find display vectors for networks allowing unrefined reticulation nodes. In other words, the *RH* bound holds for networks with unrefined nodes.

Remark. The *RH* lower bound is still applicable when the input trees are non-binary, as long as the pairwise reticulation distances are obtained for the non-binary trees. These are easy to verify and we omit the details due to the lack of space.

Remark. A commonly used concept in reticulate networks is the so-called maximum agreement forest (MAF). A brief description on MAF is given in the supplemental material. Also see e.g. [22] for more details. It is easy to see that the size of a MAF of multiple trees is a lower bound on $R(T_1, \ldots, T_K)$. However, experience show that the *RH* bound is often higher than the MAF bound (see Section 5).

3.2 Special Case of Three Trees

The special case of $K = 3$ allows us to study the *RH* bound in an analytical way. We let d_1, d_2 and d_3 be the pairwise reticulation distances of the three trees, where $d_1 \geq d_2 \geq d_3$. Proposition 3.2 shows the *RH* bound for three trees in an analytical form.

PROPOSITION 3.2. *The RH lower bound for three trees T_1, T_2 and T_3 is equal to $\lceil \frac{d_1+d_2+d_3}{2} \rceil$ if $d_2 + d_3 > d_1$, and equal to d_1 if $d_2 + d_3 \leq d_1$.*

PROOF. We first consider the case $d_2 + d_3 > d_1$. Clearly, the *RH* bound is at least d_1, which is the minimum size of the hypercube. Now we investigate whether there exists a reticulate network with $d_1 + e$ reticulation nodes for these three trees. Without loss of generality, let T_1 be the input tree where $d_1 = D_{T_1,T_2}$ and $d_2 = D_{T_1,T_3}$, and the display vector v_1 (for T_1) is fixed to be all-0. Then, the display vector v_2 for T_2 must have at least d_1 positions with value 1 (and thus v_2 has no more than e positions with value 0). Similarly, the display vector v_3 must have at least d_2 positions with value 1 (and thus v_3 has no more than $d_1 + e - d_2$ positions with value 0). Note that $D_h[v_2, v_3] \geq d_3$. We claim that $D_h[v_2, v_3] \leq d_1 + 2e - d_2$. This is because the Hamming distance between v_2 and v_3 counts the positions where v_2 has value 0 and v_3 has value 1 (or vice versa). Since the number of 0s in v_2 is no more than e, there are at most e positions where v_2 has 0 and v_3 has 1. Similarly, there are at most $d_1 + e - d_2$ positions where v_2 has 1 and v_3 has 0. Thus, $D_h[v_2, v_3] \leq e + d_1 + e - d_2 = d_1 + 2e - d_2$. Also note that we can always construct v_2 and v_3 so that $D_h[v_2, v_3] = d_1 + 2e - d_2$. See Figure 2 for an illustration.

Therefore, if $d_1 + 2e - d_2 < d_3$, we can *not* find three vectors v_1, v_2 and v_3 satisfying the pairwise distances D and thus $R(T_1, T_2, T_3) \geq d_1 + e + 1$ in this case. The largest such e is equal to $\lfloor \frac{d_2+d3-d_1-1}{2} \rfloor$ (which is non-negative since $d_2 + d_3 > d_1$). The *RH* bound is then $d_1 + \lfloor \frac{d_2+d3-d_1-1}{2} \rfloor + 1 = \lceil \frac{d_1+d_2+d_3}{2} \rceil$.

The case when $d_2 + d_3 \leq d_1$ is simple. We create three vectors of d_1 bits: v_1 is an all-0 vector, v_2 is an all-1 vector and v_3 contains d_2 1s. It is easy to verify these three vectors satisfy all three pairwise distance constraints. □

In practice, it is very likely $d_2 + d_3 > d_1$. In this case, $\lceil \frac{d_1+d_2+d_3}{2} \rceil > d_1$, where d_1 is a trivial lower bound.

4 AN UPPER BOUND

We now present an upper bound on $R(T_1, \ldots, T_K)$. The combination of the *RH* lower bound and the upper bound quantifies the *range* of $R(T_1, \ldots, T_K)$. In the best scenario, if the upper bound *matches* the *RH* bound, these bounds would actually determine the *exact* value of $R(T_1, \ldots, T_K)$ (and also reconstruct $\mathcal{N}_{min}$). On the high level, the upper bound performs stepwise insertion of trees into a reticulate network (and thus is called the *SIT* bound). The *SIT* bound is very accurate and also computable in practice for many datasets.

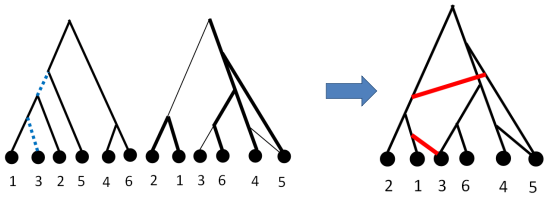
The basic idea of the *SIT* bound is to reconstruct a reticulate network $\mathcal{N}$ in a step-by-step way: "insert" the given gene trees one by one into $\mathcal{N}$ in some fixed order. When we say a tree T is inserted into $\mathcal{N}$, we mean adding reticulation edges into $\mathcal{N}$ such that T is displayed in the updated network $\mathcal{N}'$. Note that addition of new reticulation edges increases $R_{\mathcal{N}}$. Thus, every time we insert a new tree, we seek to add as *few* new reticulation edges as possible by *reusing* existing reticulation edges. At the same time, we also ensure no cycles exists in $\mathcal{N}'$. Often, it is unclear which order of inserting trees gives the best result. For now, we assume that K is relatively small so that we can enumerate all possible orders of insertion to find the best result. See Section 4.2 for ways to handle larger K. Thus, we can assume the order of tree insertion is fixed to $T_1, T_2, \ldots, T_K$. The general procedure of the *SIT* bound (for a fixed order) is as follows.

1. Initialize $\mathcal{N}$ to be T_1.
2. for $i = 2$ to K
3. Insert T_i into $\mathcal{N}$ by adding the smallest number of new reticulation edges.

Note that we only add reticulation edges in $\mathcal{N}$ and do not delete any existing edges. Thus, any tree already displayed in $\mathcal{N}$ is still displayed in the updated $\mathcal{N}'$ by choosing the original reticulation edges when the tree is first inserted for their display vectors in $\mathcal{N}'$. This ensures that each of the input trees is displayed in the final $\mathcal{N}$. Obviously, step 3 is most critical, which we will discuss next.

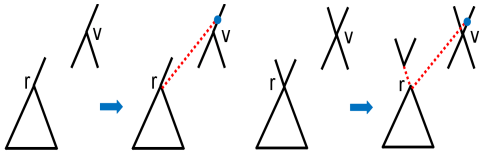
4.1 Inserting tree T into $\mathcal{N}$

We consider the **min-cost tree insertion problem**, where we want to update $\mathcal{N}$ by adding the *fewest* reticulation edges to $\mathcal{N}$ so that a given tree T is displayed in the updated $\mathcal{N}'$, and $\mathcal{N}'$ remains acyclic. Note that the min-cost tree insertion problem is NP-complete because it contains the two-tree minimum reticulate



(a) Insert a tree (left) into $\mathcal{N}$ (b) Updated $\mathcal{N}$

Fig. 3. Inserting a tree (left) to a network (middle). After adding new reticulation edges (thick lines), the resulting network (right) displays the tree.



(a) Between tree nodes (b) Between reticulation nodes

Fig. 4. Attaching a subtree in $\mathcal{N}$. Left: insert a reticulation edge between two tree nodes r and v. Right: insert a reticulation edge between two reticulation nodes r and v. The dashed lines are the newly added edges.

network problem (an NP-complete problem) as a sub-problem. That is, constructing the minimum reticulate network for trees T_1 and T_2 can be solved by inserting T_2 into T_1 with the minimum cost. In the following, we develop a practical method to solve the min-cost tree insertion problem. Each node of the reconstructed network here has one or two incoming edges (except the root), and one or two outgoing edges (except the leaves).

After inserting T (and some new reticulation edges are added), T is displayed in the updated network $\mathcal{N}'$. Suppose we *remove* all the new reticulation edges in $\mathcal{N}'$. The edge removals break tree $T(\mathcal{N}')$ (the tree created by keeping edges in $\mathcal{N}'$ according to a display vector of T) into a *forest* $F(T(\mathcal{N}'))$. Thus, the number of newly added reticulation edges is *exactly* the number of trees in $F(T(\mathcal{N}'))$ minus one. To minimize the number of needed new reticulation events, we need to minimize the number of trees in $F(T(\mathcal{N}'))$. A useful observation is that the problem of finding $F(T(\mathcal{N}'))$ with the fewest subtrees is closely related to the maximum agreement forest problem (see the supplemental material) as follows.

Imagine that we choose a tree T' that is displayed in $\mathcal{N}$ so that the display vector of T' agrees with that of T for $\mathcal{N}'$ at each reticulation node of $\mathcal{N}$. Recall that $\mathcal{N}'$ may contain a number of new reticulation nodes that are not in $\mathcal{N}$. Also note T' is not necessarily one of the input trees T_i. We claim that $F(T(\mathcal{N}'))$ is an agreement forest for T and T'. To see this, we note that the display choices made by T' are identical to T except those at the new reticulation nodes (where T' follows the original edge and $T(N')$ follows the new edge). So the subtrees in $F(T(\mathcal{N}'))$ must also be subtrees of T'. So, we have:

LEMMA 4.1. *The forest induced by removing newly added reticulation edges of $T(\mathcal{N}')$ is an agreement forest between T and some tree T' that is displayed in the original $\mathcal{N}$.*

Lemma 4.1 implies that to find the best tree insertion, we can find some tree T' displayed in $\mathcal{N}$ s.t. the number of trees in the maximum agreement forest between T and T' is minimized. Figure 3 shows an example of tree insertion. The dashed lines in the tree (left) divide the tree into a forest, which also appears in the existing network (middle, thick lines). Inserting the tree into the network is to add new reticulation edges (right, thick lines) into the networks so that the subtrees in the forests are properly connected to match the given tree.

When the number of reticulation nodes in $\mathcal{N}$ is small, we may simply enumerate all trees T' displayed in $\mathcal{N}$ and then find which T' gives the smallest agreement forest with T. This quickly becomes infeasible as the number of reticulation nodes in $\mathcal{N}$ grows: when there are r reticulation nodes in $\mathcal{N}$, there may exist 2^r trees T' displayed in $\mathcal{N}$. To develop a practical method, we develop an integer linear programming (ILP) formulation to solve the tree insertion problem in an optimal way (without explicit enumeration). The output of the ILP formulation includes the display choices of T' as well as the associated agreement forest formed by cutting edges in T. See the supplemental material for detailed description of the formulation.

Updating $\mathcal{N}$. After tree T' and the associated agreement forest are found, we update $\mathcal{N}$ as follows. We add new reticulation edges in $\mathcal{N}$ to connect subtrees of T' in the found agreement forest to make T displayed in the updated network $\mathcal{N}'$. First, we determine the order of subtree connection with an approach similar to the algorithm building two-tree hybridization networks in [22]. The subtree with the special outgroup taxon o acts as the base. Then we repeatedly pick the subtree not intersecting any already connected subtree as the next to connect. Now, for each tree connection:

1. Find the root r of the next subtree (in $\mathcal{N}$) to attach.
2. Find the node v in the existing network as the attaching point to accept this subtree.
3. Create a new reticulation node in $\mathcal{N}$ to connect the subtree.

This operation depends on the types of r and v. Two cases are shown in Figure 4. The other cases are similar. In all the cases, only a single new reticulation edge is created to connect a subtree.

Cycles. A remaining issue is that cycles can be introduced when connecting subtrees in $\mathcal{N}$. There are two sources of cycles. First, the found agreement forest may induce cycles (see [1]). Enhancing the ILP formulation to avoid cycles may significantly complicate the formulation and slow the ILP solving. A practical observation is that cycles in an agreement forest are often caused by two pairs of leaves a, b and c, d so that the a/b pair is ancestral to c/d pair in T and the c/d pair is ancestral to a/b pair in T'. Here, we say a pair of leaves a and b is ancestral to a pair of leaves c and d if the the MRCA of a and b is ancestral to the MRCA of c and d. MRCA stands for the most recent common ancestor, and node a is ancestral to node b in tree T if a is on the path from b to the root of T. To forbid this type of simple cycles, we enhance the ILP formulation: for such pairs a/b and c/d, we require either a and b are not in the same subtree, or c and d are not in the same subtree of the resulting forest. Although this does not guarantee to remove all cycles, we found that cycles in the agreement forest are rare after this change. This observation is also useful for the method of computing pairwise reticulation distances in [25].

Second, cycles can appear in other parts of the network when subtrees in the agreement forest are connected. In practice, however, we find this happens relatively rare. When this type of cycles does occur, we simply start over and try another order of tree insertion. This works well in practice: in Section 5, we build acyclic networks successfully for all (thousands of) simulated datasets.

4.2 Handling larger datasets

When the size and the number of trees grow, the running time increases. To handle larger datasets, we make several simplifications. (a) Instead of enumerating all possible orders of tree insertion, we start with an arbitrary tree. At each step, we pick a tree with the smallest reticulation distance to one of the already inserted trees. (b) Solving the min-cost tree insertion problem optimally becomes more difficult when data grows. So instead of considering all possible T' displayed in $\mathcal{N}$ when inserting T, we randomly choose a fixed number (say 10) of trees T' displayed in $\mathcal{N}$ (in addition to all the inserted gene trees) and find the best way of inserting T based on one of the chosen T'. This heuristic is called the *coarse* mode (and the original approach is called the *full* mode). Our experience shows that the coarse mode works reasonably well in practice (see Section 5).

5 EXPERIMENTAL RESULTS

We have implemented a software tool called **PIRN** (which stands for Parsimonious Inference of Reticulate Network) to compute the *RH* and *SIT* bounds. Program *PIRN* is available for download from: http://www.engr.uconn.edu/~ywu/. The tool is written in C++ and uses either CPLEX (a commercial ILP solver) or GNU GLPK ILP solver (mainly a demo of the functionalities for users without a CPLEX license). In computing the *SIT* bound, *PIRN* can run full mode (slower but can give better results) or coarse mode (faster but less accurate). We test our methods for both simulated and biological data on a 3192 MHz Intel Xeon workstation.

5.1 Simulation Data

We generate simulation data using a two-stage approach: first simulate reticulate networks, and then generate a fixed number of trees displayed in the networks according to randomly generated display vectors. We simulate reticulate networks using a scheme similar to the coalescent simulation implemented in program ms [10]. For a given number of taxa (denoted as n), we start with n isolated lineages and simulate reticulation *backwards* in time. At each step, there are two possible events: (a) lineage merging, which occurs at rate 1; (b) lineage splitting, which occurs at rate r. We choose the next event according to relative probabilities of all feasible events. Lineage merging generates speciation events, while lineage splitting generates reticulation events. To speedup the simulation, lineage splitting is disabled when the number of current lineages is no more than three. The parameter r dictates the level of reticulation in the simulated network: larger r will lead to more reticulation events in simulation.

Full mode of the *SIT* bound. To test the performance of the bounds, we generate data with varying number of trees K, number of taxa n and level of reticulation r. For each settings of these three parameters, we simulate 100 datasets. We report the percentage of datasets where *optimal* solution is found (i.e. lower bound matches upper bound) in Figure 5(a). To show how close the lower and upper bounds are, we report the average gap (the difference between the upper and the lower bounds, divided by the lower bound) in Figure 5(b). We also report the average lower bound in Figure 5(c), which somewhat reflects how complex the simulated networks are. We also give the average running time for each setting in Figure 5(d). For five larger datasets, there are a small number of test cases that are too slow to run the full mode, and are excluded. The percentage of unfinished computation is usually one or two out of 100 datasets, except the cases with $n = 30/r = 3.0/K = 5$ (18% unfinished) and $n = 30/r = 5.0/K = 4$ (6% unfinished). This suggests the current practical range of the full mode of the *SIT* bound.

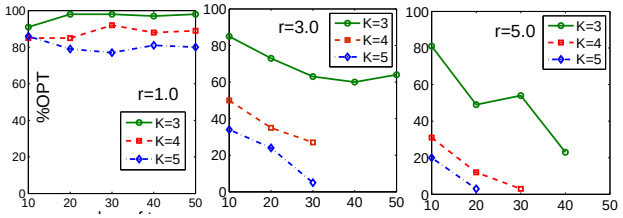
As shown in Figure 5(a), *PIRN* performs very well when the number of trees $K = 3$ or reticulation level r is small: *optimal* solution can be found for at least 80% of simulated datasets when $r = 1.0$ and $K = 5$. Even with higher reticulation level (r=3.0) and larger number of taxa (say 50), still about 60% of datasets can be solved exactly when $K = 3$. As expected, as the number of taxa, reticulation level and the number of trees grow, fewer datasets can be solved to exact, and correspondingly, Figure 5(b) shows gaps between the *RH* and *SIT* bounds increase. Figure 5(c) shows the complexity of networks increases too. Nevertheless, the gaps are still relatively small in these cases. For the more difficult settings simulated (i.e. 30 taxa, high reticulation level and five trees as input), the gap is about 25%. Running time depends on the complexity of the networks. Figure 5(d) shows that *PIRN* is practical for data of medium size.

Coarse mode of the *SIT* bound. We also test the coarse mode of our methods for larger data, as described in Section 4.2. The results are shown in Figure 6. The figure on the left shows the effects (on accuracy and running time) of increasing the number of taxa n. The figure on the right shows the effects of having more trees (i.e. increasing K from three to nine) for 10 taxa and reticulation level 5.0. There is clear trade-off between the accuracy of solutions and efficiency. The coarse mode under-performs in terms of the quality of solutions, but is more scalable, especially when K increases. When the number of taxa increases, the coarse mode is likely to run faster than the full mode, but the difference is less significant.

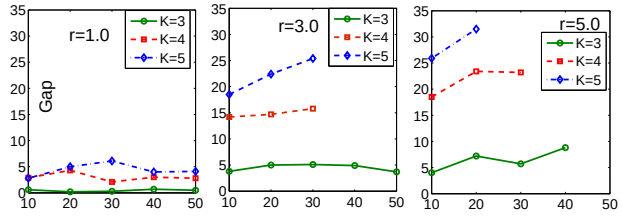
GLPK. The CPLEX solver is used in the simulation. The GLPK version in general is less robust and can handle smaller data than the CPLEX version. Our experience shows that the GLPK solver can often solve for five trees with 30 taxa when reticulation level is low and fewer number of taxa when reticulation level is higher.

The *RH* bound. We now compare the performance of the *RH* bound with the MAF bound. We note that the MAF bound is not easy to compute: finding the MAF of only two trees is known to be NP-hard. When data is small, the MAF bound can be computed (e.g. using ILP similar to that in [24]). In Table 1, we compare the *RH* bound and the MAF bound for 100 datasets. Each data has 10 or 20 taxa and contain three to seven correlated trees. The trees are selected from the local trees for recombining sequences generated by a coalescent simulator, program ms [10].

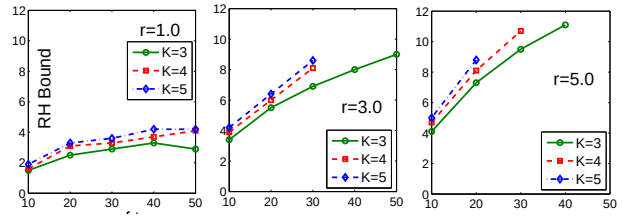
Table 1 shows that the *RH* bound outperforms the MAF bound in a majority of the simulated datasets and only very rarely the *RH* bound is lower than the MAF bound. In general, as the number of trees increases, the *RH* bound tends to outperform the MAF bound in both accuracy and running time. For example, for a dataset with 20 sequences and 6 input trees, CPLEX runs for over 11 hours without



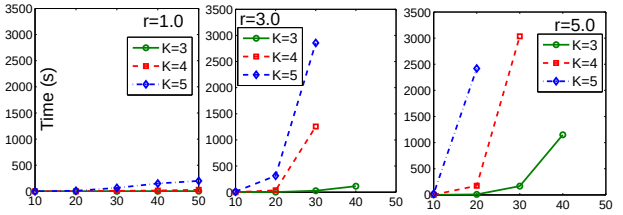
(a) % Optimal solutions found among 100 datasets



(b) Normalized gap between the two bounds (in %)



(c) Average *RH* bounds



(d) Average running time (in seconds)

Fig. 5. Performance of the *RH* bound and *SIT* bound (full mode), for 10, 20, 30, 40 and 50 taxa, number of trees K from 3 to 5, and reticulation level r at 1.0, 3,0 and 5.0. Figure 5(a) shows the percentage of exact reticulation number found among 100 simulated datasets. Figure 5(b) shows average gaps (in percentage) between the *SIT* bound and *RH* bound (normalized by the *RH* bound), and the average *RH* bound is shown in Figure 5(c). Figure 5(d) shows the average running time (in seconds). The reticulation level r for left, middle and right figures is 1.0, 3.0 and 5.0 respectively. Horizontal axis is the number of taxa, and each curve in a figure is for a value of K.

reporting a solution (it found a solution of 11, but did not validate its optimality). In contrast, it only takes less than 1 minute to compute the *RH* bound of value 12 (higher than the MAF result).

5.2 Biological Data

To evaluate how well our bounds work for real biological data, we test our methods on a Poaceae dataset. The dataset was originally from the Grass Phylogeny Working Group [6]. The dataset contains

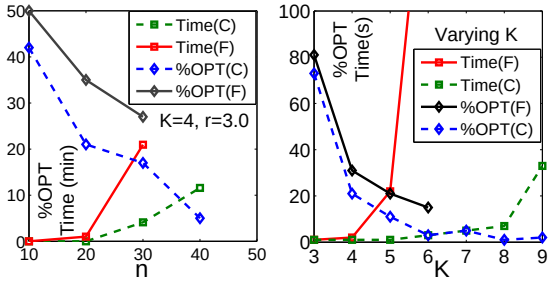


Fig. 6. Comparing the coarse (C) mode and the full (F) mode of the *SIT* bound. Left: fix r=3.0 and K=4, while varying n. Right: fix r=5.0 and n=10 and vary K. Both percentages of optimal solutions found and running time (in minutes for the left figure and seconds for the right one) are shown.

Table 1. Compare the *RH* and MAF bounds for K trees. *RH* $>$ MAF: number of datasets where the *RH* bound is larger than the MAF bound among 100 datasets. The average gaps (in percentage) between the *RH* and MAF bounds are also shown.

		K=3	K=4	K=5	K=6	K=7
n=10	$RH >$ MAF	46	56	61	58	72
	$RH =$ MAF	54	44	38	41	27
	$RH <$ MAF	0	0	1	1	1
	$100\frac{RH-MAF}{MAF}$	16.5	18.4	17.2	18.0	20.9
n=20	$RH >$ MAF	65	66	68	70	-
	$RH =$ MAF	34	34	30	30	-
	$RH <$ MAF	1	0	2	0	-
	$100\frac{RH-MAF}{MAF}$	12.6	11.9	12.1	12.8	-

Table 2. Results for the grass data. Both *RH* and *SIT* bounds are shown, as well as the running time (s: second, m: minutes). n: the number of taxa. D: pairwise distances.

Trees	n	D	*RH*	*SIT*	Time
rpoC2, waxy, ITS	10	1, 6, 6	*7*	*7*	1 s
ndhF, phyB, rbcL	21	4, 5, 6	*8*	*8*	1 s
ndhF, phyB, rbcL,rpoC2, ITS	14		*11*	*13*	26 m 38 s

sequences for six loci: internal transcribed spacer of ribosomal DNA (ITS); NADH dehydrogenase, subunit F (ndhF); phytochrome B (phyB); ribulose 1,5-biphosphate carboxylase/oxygenase, large subunit (rbcL); RNA polymerase II, subunit β'' (rpoC2); and granule bound starch synthase I (waxy). The Poaceae dataset was previously analyzed and rooted binary trees were inferred for these loci [21]. Pairwise comparison were performed in [3, 25]. Here, we provide in Table 2 our results on estimating the reticulation number using multiple (three to five) trees. Only *shared* taxa of a set of trees are kept. Thus, the pairwise distances reported here are different from those in [3, 24]. As shown in Table 2, *PIRN* finds *optimal* solutions for both datasets with three trees, and computes lower and upper bounds that are *close* for the dataset with five trees. Figure 7 shows the reconstructed reticulate network for these five trees. See supplemental material for a graphical display of the five grass trees. The network contains 13 reticulation events, and the lower bound is 11. Although the network may not be optimal, the gap between the lower and upper bounds is relatively small.

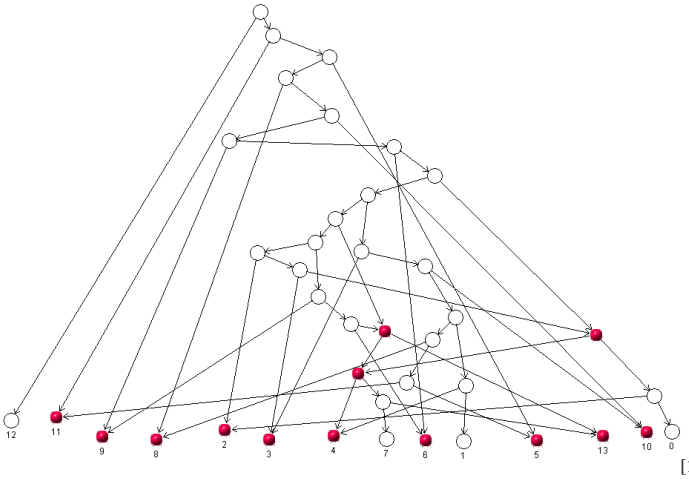


Fig. 7. A reticulate network found by program *PIRN* for five trees of a grass dataset. Red (shaded) balls represent reticulation nodes.

Remark. The following lists several aspects on the performance of *PIRN*. (i) Simulation shows that *PIRN* is often able to find the exact reticulation number when K or r is small, even when the number of taxa increases to medium size (say 50). Moreover, *PIRN* can compute the *RH* and *SIT* bounds for a wide range of data, despite the fact that we do not currently have polynomial-time algorithms for computing the bounds. We achieve this with the help of integer linear programming. (ii) Computing the *RH* bound is often much faster and more scalable than the *SIT* bound. Experience shows that the ILP formulation for computing the *RH* bound is often very fast to solve and computing the pairwise reticulation distances usually takes less time than finding a good upper bound for all trees. The *RH* bound computation will also benefit from future improvements in computing the pairwise reticulate distances. The simulation results in this section are based on an earlier version of the method in [25] and speedup may be possible with the latest methods. (iii) The number of trees K and the similarity of tree topologies have impact on *PIRN* 's optimality and running time. Using more powerful ILP solver (e.g. CPLEX) and/or more powerful machines may also help for more difficult cases. (iv) Finally, our general approaches can be applied to larger data by using efficient computable lower bounds of pairwise rSPR distances in computing the *RH* bound, and faster but less accurate heuristics to insert trees into a network.

FUNDING AND ACKNOWLEDGMENT

This work is supported by grants from U.S. National Science Foundation [IIS-0803440] and the Research Foundation of U. of Connecticut. I thank Simone Linz for useful discussions and for sharing the grass tree dataset.

REFERENCES

[1]M. Baroni, S. Grunewald, V. Moulton, and C. Semple. Bounding the number of hybridisation events for a consistent evolutionary history. *J. Math. Biol.*, 51:171–182, 2005.

[2]M. Baroni, C. Semple, and M. Steel. A framework for representing reticulate evolution. *Annals of Combinatorics*, 8:391–408, 2004.

[3]M. Bordewich, S. Linz, K. S. John, and C. Semple. A reduction algorithm for computing the hybridization number of two trees. *Evolutionary Bioinformatics*, 3:86-98, 2007.

[4]M. Bordewich and C. Semple. On the computational complexity of the rooted subtree prune and regraft distance. *Annals of Combinatorics*, 8:409–423, 2004.

[5]M. Bordewich and C. Semple. Computing the minimum number of hybridization events for a consistent evolutionary history. *Discrete Applied Mathematics*, 155:914–928, 2007.

[6]Grass Phylogeny Working Group. Phylogeny and subfamilial classification of the grasses (poaceae). *Ann. Mo. Bot. Gard.*, 88:373–457, 2001.

[7]D. Gusfield. Optimal, efficient reconstruction of Root-Unknown phylogenetic networks with constrained and structured recombination. *JCSS*, 70:381–398, 2005.

[8]M. Hallett and J. Lagergren. Efficient algorithms for lateral gene transfer problems. In *Proceedings of fifth annual conference on Research in computational molecular biology (RECOMB 2001)*, pages 149–156, 2001.

[9]J. Hein, T. Jiang, L. Wang, and K. Zhang. On the complexity of comparing evolutionary trees. *Discrete Appl. Math*, 71:153–169, 1996.

[10]R. Hudson. Generating samples under the Wright-Fisher neutral model of genetic variation. *Bioinformatics*, 18(2):337–338, 2002.

[11]D. Huson. Split networks and reticulate networks. In O. Gascuel and M. Steel, editors, *Reconstructing Evolution: New Mathematical and Computational Advances*, pages 247–276. Oxford, 2007.

[12]D. Huson and D. Bryant. Application of phylogenetic networks in evolutionary studies. *Molecular Biology and Evolution*, 23:254–267, 2006.

[13]D. Huson and T. Klopper. Beyond galled trees - decomposition and computation of galled networks. In T. Speed and H. Huang, editors, *Proc. of RECOMB 2007: The 11th Ann. International Conference Research in Computational Molecular Biology*, pages 211–225. Springer, LNBI 4453, 2007.

[14]D. Huson, T. Klopper, P. Lockhart, and M. Steel. Reconstruction of reticulate networks from gene trees. In *Proc. of RECOMB 2005: The 9th Ann. International Conference Research in Computational Molecular Biology*, pages 233–249. Springer, LNBI 3500, 2005.

[15]D. Huson, R. Rupp, P. Gambette, and C. Paul. Computing galled networks from real data. *Bioinformatics*, 25:i85–i93, 2009. Bioinformatics Suppl., Proceedings of ISMB 2009.

[16]C. R. Linder, B. M. E. Moret, L. Nakhleh, and T. Warnow. Network (reticulate) evolution: biology, models, and algorithms, A tutorial presented at the Ninth Pacific Symposium of Biocomputing (PSB 2004). 2004.

[17]S. Linz and C. Semple. Hybridization in nonbinary trees. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 6:30–45, 2009.

[18]L. Nakhleh. Evolutionary phylogenetic networks: models and issues. In L. Heath and N. Ramakrishnan, editors, *The Problem Solving Handbook for Computational Biology and Bioinformatics*. Springer, 2009. In press.

[19]L. Nakhleh, T. Warnow, and C. Linder. Reconstructing reticulate evolution in species - theory and practice. In *Proc. of 8'th Annual International Conference on Computational Molecular Biology*, pages 337–346, 2004.

[20]L. Nakhleh, T. Warnow, C. Linder, and K. S. John. Reconstructing reticulate evolution in species - theory and practice. *J. of Comp. Biology*, 12:796–811, 2005.

[21]H. Schmidt. *Phylogenetic trees from large datasets*. PhD thesis, Heinrich-Heine-Universitat, Dusseldorf, 2003.

[22]C. Semple. Hybridization networks. In O. Gascuel and M. Steel, editors, *Reconstructing Evolution: New Mathematical and Computational Advances*, pages 277–309. Oxford, 2007.

[23]L. van Iersel, J. Keijsper, S. Kelk, L. Stougie, F. Hagen, and T. Boekhout. Constructing level-2 phylogenetic networks from triplets. In M. Vingron and L. Wong, editors, *Proc. of RECOMB 2008: The 12th Ann. International Conference Research in Computational Molecular Biology*, pages 450–462. Springer, LNBI 4955, 2008.

[24]Y. Wu. A practical method for exact computation of subtree prune and regraft distance. *Bioinformatics*, 25:190–196, 2009.

[25]Y. Wu and J. Wang. Fast Computation of the Exact Hybridization Number of Two Phylogenetic Trees. In *Proc. of ISBRA 2010: The 6th International Symposium on Bioinformatics Research and Applications*, to appear, 2010.