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, USA

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 PI RN 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, PI RN, is available for download from the web page: http://www.engr.uconn.edu/~ywu.
Contact: ywu@ engr.uconn.edu

Supplementary information: Supplementary data is available at Bioinformatics online.

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 networks1. We refer the readers to (Huson, 2007; Huson and Bryant, 2006; Nakhleh, 2009; Semple, 2007) 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 article) or aligned DNA sequences. In this article, 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 article) 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. Baroni et al., 2004; Huson, 2007; Huson et al., 2005; Nakhleh et al., 2004; Semple, 2007) 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 (Nakhleh, 2009); 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 (Bordewich and Semple, 2004, 2007; Hein et al., 1996). There are several existing approaches for reconstructing the exact minimum reticulate networks when there are only two gene trees (Bordewich et al., 2007; Linz and Semple, 2009; Wu, 2009; Wu and Wang, 2010). 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 simplifications to the reticulate network model, e.g. by imposing additional topological constraints on reticulate
networks (Gusfield, 2005; Huson and Klopper, 2007; Huson et al., 2009; Nakhleh et al., 2005) or working with small-scale tree topological features (Huson et al., 2005; Huson and Klopper, 2007, van Iersel et al., 2008). 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 article, we present new approaches for the minimum reticulate network problem with more than some previous approaches (Huson and Klopper, 2007; Huson et al., 2005; Huson et al., 2009). 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 article, 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 (Huson, 2007; Huson et al., 2005; Semple, 2007) (Hallett and Lagergren, 2001; Nakhleh, 2009; Nakhleh et al., 2004). 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.

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.

(3) $E_R$ contains edges that go into some reticulation nodes. $E_T$ contains edges that go into some tree nodes.

(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 $N$, 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 $N$) 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 $N$). Now suppose we are given a phylogenetic tree $T$. We say $T$ is displayed in $N$ when we can obtain an induced tree $T'$ from $N$ 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 $T_N$. 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 Supplementary Materials for more discussion. Note that we do not require that in-degrees of reticulation nodes are precisely two as what was imposed in (Huson et al., 2005). 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 $N$, we define the reticulation number (denoted as $R_N$) as the summation of the reticulation number of each reticulation node in the network. Sometimes $R_N$ is also called the number of reticulation events in $N$. 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, $R_N = (3 - 1) + (2 - 1) = 3$. Our definition of reticulation number is similar to that of the hybridization number in (Bordewich et al., 2007; Semple, 2007).

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 $R_{\text{min}}$ for $T_1, T_2, \ldots, T_K$ is a reticulate network $N$ that displays each $T_i$ and $R_{\text{min}}$ is minimized among all possible $N$. We call $R_{\text{min}}$ the reticulation number of $T_1, \ldots, T_K$, which is denoted as $R(T_1, T_2, \ldots, T_K)$. 

1141
Wang, 2010) are exponential, these methods may work reasonably well in practice. As shown in (Bordewich et al., 2007; Wu and Wang, 2010), we store these pairwise distances in a matrix $D$, where $D[i,j]=D_{ij}$ signifies computing the pairwise reticulation distances $D_{ij}$ for two trees $T_i$ and $T_j$, which are practical for many pairs of trees of moderate sizes. Note that we compute $D_{ij}$ for each pair of $T_i$ and $T_j$ using the methods (Bordewich et al., 2007; Wu and Wang, 2010). We store these pairwise distances in a matrix $D$, where $D[i,j]=D_{ij}$. 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 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 $D[q,b]$ 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 $T_q$ and $T_b$, and thus is a reticulate network for $T_q$ and $T_b$. We now show a stronger lower bound (called RH bound) based on $D$ values. Here is the high-level idea. The pairwise distance $D_{ij}$ specifies how similar trees $T_i$ and $T_j$ are: the larger $D_{ij}$ is, the more different $T_i$ and $T_j$ are. Recall that if tree $T_i$ is displayed in a network $N$, we should be able to determine $T_i$ 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 $N$, 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, we can use the pairwise reticulation distances $D_{ij}$ to determine whether the two trees are similar enough: if the display choices for $T_i$ and $T_j$ are similar, we can determine which incoming edge at each reticulation node must be kept. The choice (called display choice) of which incoming edge at each reticulation node $V_j$ in $N$ can be determined by keeping only one incoming edge at each reticulation node of $N$. The choice of $V_j$ suggests that $V_j$ in $N$ should have $r_i$ reticulation node(s) (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 make display choices for each tree. Pattern methods for either computing the reticulation number $R(T_1,\ldots,T_K)$ or reconstructing $N_{mix}$ 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 (Gusfield, 2005; Huson and Klopper, 2007; Huson et al., 2009; Nakhleh et al., 2005). 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 (Bordewich et al., 2007; Wu and Wang, 2010) exist for computing the pairwise reticulation distance $D_{ij}$ for two trees $T_i$ and $T_j$, which are practical for many pairs of trees of moderate sizes. Now suppose that we compute $D_{ij}$ for each pair of $T_i$ and $T_j$, using the methods (Bordewich et al., 2007; Wu and Wang, 2010). We store these pairwise distances in a matrix $D$, where $D[i,j]=D_{ij}$. 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 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 $D[q,b]$ 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 $T_q$ and $T_b$, and thus is a reticulate network for $T_q$ and $T_b$. We now show a stronger lower bound (called RH bound) based on $D$ values. Here is the high-level idea. The pairwise distance $D_{ij}$ specifies how similar trees $T_i$ and $T_j$ are: the larger $D_{ij}$ is, the more different $T_i$ and $T_j$ are. Recall that if tree $T_i$ is displayed in a network $N$, we should be able to determine $T_i$ 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 $N$, 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 $N'$ for trees with pairwise distances $D[i,j]=D_{ij}$ we need to make display choices for the trees different enough: if the display choices for $T_i$ and $T_j$ are similar, we can determine which incoming edge at each reticulation node must be kept. The choice (called display choice) of which incoming edge at each reticulation node $V_j$ in $N'$ can be determined by keeping only one incoming edge at each reticulation node of $N$. The choice of $V_j$ suggests that $V_j$ in $N'$ should have $r_i$ reticulation node(s) (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 make display choices for each tree. Pattern methods for either computing the reticulation number $R(T_1,\ldots,T_K)$ or reconstructing $N_{mix}$ 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 (Gusfield, 2005; Huson and Klopper, 2007; Huson et al., 2009; Nakhleh et al., 2005). 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 (Bordewich et al., 2007; Wu and Wang, 2010) exist for computing the pairwise reticulation distance $D_{ij}$ for two trees $T_i$ and $T_j$, which are practical for many pairs of trees of moderate sizes. Now suppose that we compute $D_{ij}$ for each pair of $T_i$ and $T_j$, using the methods (Bordewich et al., 2007; Wu and Wang, 2010). We store these pairwise distances in a matrix $D$, where $D[i,j]=D_{ij}$. 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 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 $D[q,b]$ 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 $T_q$ and $T_b$, and thus is a reticulate network for $T_q$ and $T_b$. We now show a stronger lower bound (called RH bound) based on $D$ values. Here is the high-level idea. The pairwise distance $D_{ij}$ specifies how similar trees $T_i$ and $T_j$ are: the larger $D_{ij}$ is, the more different $T_i$ and $T_j$ are. Recall that if tree $T_i$ is displayed in a network $N$, we should be able to determine $T_i$ 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 $N$, 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 $N'$ for trees with pairwise distances $D[i,j]=D_{ij}$ we need to make display choices for the trees different enough: if the display choices for $T_i$ and $T_j$ are similar, we can determine which incoming edge at each reticulation node must be kept. The choice (called display choice) of which incoming edge at each reticulation node $V_j$ in $N'$ can be determined by keeping only one incoming edge at each reticulation node of $N$. The choice of $V_j$ suggests that $V_j$ in $N'$ should have $r_i$ reticulation node(s) (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 $N'$, we create a binary vector $V_j$ of length $r_i$ to represent which incoming edge at each reticulation node $V_j$ in $N'$. Here, $V[j]=0$ if 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.
When \( K \) binary bits per node, we want to know whether we can pick a display vector for each node, while \( v_1 \) and \( v_2 \) have value 0, and \( v_3 \) has value 1 at the node.

For a given \( T_i \) and a network \( N \), \( v_1 \) can always be constructed (at least conceptually) based on how \( T_i \) is displayed in \( N \). 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 \( D_b[v_i, v_j] \) as the Hamming distance between two display vectors \( v_i \) and \( v_j \). Here, \( v_i \) and \( v_j \) are vectors (and thus \( D_b[v_i, v_j] \)) depend on \( N \). To simplify notations, we do not explicitly include \( N \) 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 \( N \), \( D_b[v_i, v_j] \geq D(i,j) \).

**Proof.** For contradiction, assume \( D_b[v_i, v_j] < D(i,j) \). Thus \( T_i \) and \( T_j \) make different choices at less than \( D(i,j) \) reticulation nodes of \( N \). Imagine we remove from \( N \) 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 \( N \) with \( r \) reticulation events exists, then we should be able to find binary vectors \( v_r \) (of length \( r \)) for each tree \( T_1 \) and \( D_b[v_r, v_r] \geq D(i,j) \) 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 \( R(T_1, ..., T_k) \)). 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 \( v_1, ..., v_K \) 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 \( v_1, ..., v_K \) from a \( r \)-dimensional binary hypercube so that \( D_b(v_i, v_j) \geq D(i,j) \) for each pair of \( v_i \) and \( v_j \)?

A lower bound on \( R(T_1, ..., T_k) \) 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 \( R(T_1, ..., T_k) \).

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 \( V_{j,k} \) to represent the coordinates for point \( v_j \). That is, the coordinates of \( v_j \) on the hypercube are specified by \( V_{j,k} \). Without loss of generality, we set \( V_{j,k} = 0 \) for all \( 1 \leq k \leq r \). We create a binary variable \( M_{j,k} \) for each \( v_j \) and \( v_k \) at position \( k \) (\( 1 \leq k \leq r \)) to indicate whether two vectors \( v_j \) and \( v_k \) match at position \( k \). \( M_{j,k} = 1 \) if \( V_{j,k} = V_{k,k} \) and 0 otherwise. Now, we have the following formulation.

**Optimization goal:** None (since this is a feasibility problem)

**Subject to**

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

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

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

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

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

Constraint 1 says if both \( V_{j,k} \) and \( V_{k,k} \) are 0, \( M_{j,k} \) is 1 (i.e. matched). Similarly, constraint 2 says if both \( V_{j,k} \) and \( V_{k,k} \) are 1, \( M_{j,k} \) 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 \( v_r \) to allow \( v_r \) to be non-binary. That is, if there are \( d \) incoming edges at a reticulation node, we allow \( v_r \) 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 \( D_b[v_i, v_j] \) be the Hamming distance between vectors \( v_i \) and \( v_j \). In this general case, Lemma 3.1 still holds for non-binary vectors \( v_r \) 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 \( d \) incoming edges (since we only have two trees). Thus, there are \( D_b[v_r, v_j] \) 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 \( N \) with unrefined reticulation nodes satisfies pairwise distances \( D \), then there exists another network \( N' \) that has only refined reticulation nodes and gives the binary vectors \( v_r \) satisfying the pairwise distance constraints of \( D \).

To see this property, we consider a network \( N \) with one reticulation node \( q \) with \( d \geq 3 \) incoming edges. Then we transform \( N \) to \( N' \) by replacing \( q \) with \( q_1, ..., q_{d-1} \), where each \( q_i \) is a reticulation node with in-degree of two. Note that we do not have to ensure \( N' \) and \( N'' \) are equivalent: we only need to show \( N'' \) gives a solution to the Binary Hypercube Point Placement problem. Clearly, \( N' \) and \( N'' \) have the same reticulation number (although vectors for \( N'' \) are
The special case of keep the same edges at MAF of multiple trees is a lower bound on (Semple, 2007) for more details. It is easy to see that the size of a network with MAF is given in the Supplementary Material. Also see e.g. in (Semple, 2007) 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 $\lfloor \frac{d_1 + d_2 + d_3}{2} \rfloor$ 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 $d_1 = D_{T_1}$. Let $d_2 = D_{T_2}$, 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 $D_3(v_2, v_3) \geq d_3$. We claim that $D_3(v_2, v_3) \geq d_1 + 2e - d_2$.

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 $v_3$ has 1 and $v_1$ has 0. Thus, $D_3(v_2, v_1) \leq d_1 + e - d_2 = d_1 + 2e - d_2$. Also note that we can always construct $v_2$ and $v_3$ so that $D_3(v_2, v_3) = d_1 + 2e - d_2$. 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 $\lfloor \frac{d_1 + d_2 + d_3}{2} \rfloor$ (which is non-negative since $d_2 + d_3 \geq d_1$). The RH bound is then $d_1 + \lfloor \frac{d_1 + d_2 + d_3}{2} \rfloor + 1 = \lfloor \frac{d_1 + d_2 + d_3}{2} \rfloor + 1$.

![Fig. 2. Vectors $v_1, v_2$ and $v_3$ (listed from top to bottom) that maximize the Hamming distance between $v_2$ and $v_3$.](image)

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 $d_1$ 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, if $v_2'$ is the all-0 vector, $v_2$ is an all-1 vector and $v_3$ contains $d_1$ 1s, then $v_2'$ 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 $N_{\text{match}}$). 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 $N'$ in a step-by-step way: ‘insert’ the given gene trees one by one into $N'$ in some fixed order. When we say a tree $T$ is inserted into $N'$, we mean adding reticulation edges into $N'$ such that $T$ is displayed in the updated network $N'$. Note that addition of new reticulation edges increases $R_N$. 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 $N'$. 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 be $T_1, T_2, \ldots, T_K$.

The general procedure of the SIT bound (for a fixed order) is as follows.

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

Note that we only add reticulation edges in $N'$ and do not delete any existing edges. Thus, any tree already displayed in $N'$ is still displayed in the updated $N'$ by choosing the original reticulation edges when the tree is first inserted for their display vectors in $N'$.

This ensures that each of the input trees is displayed in the final $N'$. Obviously, step 3 is most critical, which we will discuss next.

4.1 Inserting tree $T$ into $N'$

We consider the ‘min-cost tree insertion problem’, where we want to update $N'$ by adding the fewest reticulation edges to $N'$ so that a given
simply enumerate all trees some tree $T$

display vector of $T$

node of input trees

new reticulation edges in fewest subtrees is closely related to the maximum agreement forest added reticulation edges is nodes that are not in subtrees in $N'$ follows the original edge and $r$. To see this, we note that the display choices made by $N'$ may contain a number of new reticulation events, we need to minimize the number of trees in $N'$. A useful observation is that the problem of finding $F(T(N'))$ with the fewest subtrees in the agreement forest are connected. In practice, however, 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 $N'$. Suppose we remove all the new reticulation edges in $N'$. The edge removals break tree $T(N')$ (the tree created by keeping edges in $N'$ according to a display vector of $T$) into a forest $F(T(N'))$. Thus, the number of newly added reticulation edges is exactly the number of trees in $F(T(N'))$ minus one. To minimize the number of new reticulation events, we need to minimize the number of trees in $F(T(N'))$. A useful observation is that the problem of finding $F(T(N'))$ with the fewest subtrees is closely related to the maximum agreement forest problem (see the Supplementary Material) as follows.

Imagine that we choose a tree $T'$ that is displayed in $N$ so that the display vector of $T'$ agrees with that of $T$ for $N'$ at each reticulation node of $N$. Recall that $N'$ may contain a number of new reticulation nodes that are not in $N$. Also note $T'$ is not necessarily one of the input trees $T_i$. We claim that $F(T(N'))$ 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 $F(T(N'))$ must also be subtrees of $T'$. So, we have:

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

Lemma 4.1 implies that to find the best tree insertion, we can find some tree $T'$ displayed in $N$ such that the number of new reticulation edges in $T$ 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 $N$ is small, we may simply enumerate all trees $T'$ displayed in $N$ and then find which $T'$ gives the smallest agreement forest with $T$. This quickly becomes infeasible as the number of reticulation nodes in $N$ grows: when there are $r$ reticulation nodes in $N'$, there may exist $2^r$ trees $T'$ displayed in $N'$. 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 Supplementary Material for detailed description of the formulation.

Updating $N'$: after tree $T'$ and the associated agreement forest are found, we update $N$ as follows. We add new reticulation edges in $N'$ to connect subtrees of $T'$ in the found agreement forest to make $T$ displayed in the updated network $N''$. First, we determine the order of subtree connection with an approach similar to the algorithm building two-tree hybridization networks in (Semple, 2007). The subtree with the special outgroup taxon $a$ 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 $N''$) 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 $N''$ 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 $N'$. There are two sources of cycles. First, the found agreement forest may induce cycles (see (Baroni et al., 2005)). 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 (Wu and Wang, 2010).

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. (i) 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. (ii) Solving the min-cost tree insertion problem optimally becomes more difficult when data grows. So instead of considering all possible $T'$ displayed in $N$ when inserting $T$, we randomly choose a fixed number (say 10) of trees $T'$ displayed in $N$ (in addition to all the inserted gene trees) and find the best way of inserting $T$ based on one of the $T'$. 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 (Hudson, 2002). 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: (i) lineage merging, which occurs at rate $r$; (ii) lineage splitting, which occurs at rate $s$. 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 Fig. 5a. 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)
Table 1. Compare the RH and MAF bounds for K trees

| K | n=10 | n=20 |
|---|------|------|
| Qt | 46/54 | 65/64 |
| RH | 46/54 | 65/64 |
| MAF | 46/54 | 65/64 |
| RH-MAF | 58/41 | 61/38 |
| n=10 | n=20 |
| Qt | 46/54 | 65/64 |
| RH | 46/54 | 65/64 |
| MAF | 46/54 | 65/64 |
| RH-MAF | 58/41 | 61/38 |

Table 2. Results for the grass data

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

As shown in Figure 5a, PI RN 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 5b shows gaps between the RH and SIT bounds increase. Figure 5c 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 5d shows that PI RN 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 from three to nine) for 10 taxa and reticulation level 5.0. Both percentages of optimal solutions found and running time (in minutes for the left figure and seconds for the right one) are shown.

As shown in Figure 5a, PI RN 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 5b shows gaps between the RH and SIT bounds increase. Figure 5c 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 5d shows that PI RN 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 from three to nine) for 10 taxa and reticulation level 5.0. Both percentages of optimal solutions found and running time (in minutes for the left figure and seconds for the right one) are shown.

As shown in Figure 5a, PI RN 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 5b shows gaps between the RH and SIT bounds increase. Figure 5c 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 5d shows that PI RN 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 from three to nine) for 10 taxa and reticulation level 5.0. Both percentages of optimal solutions found and running time (in minutes for the left figure and seconds for the right one) are shown.

As shown in Figure 5a, PI RN 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 5b shows gaps between the RH and SIT bounds increase. Figure 5c 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 5d shows that PI RN 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 from three to nine) for 10 taxa and reticulation level 5.0. Both percentages of optimal solutions found and running time (in minutes for the left figure and seconds for the right one) are shown.

As shown in Figure 5a, PI RN 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 5b shows gaps between the RH and SIT bounds increase. Figure 5c 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 5d shows that PI RN 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 from three to nine) for 10 taxa and reticulation level 5.0. Both percentages of optimal solutions found and running time (in minutes for the left figure and seconds for the right one) are shown.
Remark: the following lists several aspects on the performance of \textit{PIRN}. (i) Simulation shows that \textit{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, \textit{PIRN} can compute the \textit{RH} and \textit{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 \textit{RH} bound is often much faster and more scalable than the \textit{SIT} bound. Experience shows that the ILP formulation for computing the \textit{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 \textit{RH} bound computation will also benefit from future improvements in computing the pairwise reticulation distances. The simulation results in this section are based on an earlier version of the method in (Wu and Wang, 2010) and speedup may be possible with the latest methods. (iii) The number of trees $K$ and the similarity of tree topologies have impact on \textit{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 approaches can be applied to larger data by using efficient machines may also help for more difficult cases. (iv) Finally, our.

\textbf{AKNOWLEDGEMENT}

I thank Simone Linz for useful discussions and for sharing the grass tree dataset.

\textbf{Funding:} U.S. National Science Foundation (IIS-0803440) and the Research Foundation of University of Connecticut.

\textbf{Conflict of Interest:} none declared.

\section*{REFERENCES}

Baeza,M. et al. (2005) Bounding the number of hybridisation events for a consistent evolutionary history. J. Math. Biol., 51, 171–182.

Baeza,M. et al. (2004) A framework for representing reticulate evolution. Ann. Comb., 8, 391–418.

Bordewich,M. et al. (2007) A reduction algorithm for computing the hybridization number of two trees. Evol. Bioinform., 3, 86–98.

Bordewich,M. and Semple,C. (2004) On the computational complexity of the rooted subtree prune and regraft distance. Ann. Comb., 8, 409–423.

Bordewich,M. and Semple,C. (2007) Computing the minimum number of hybridization events for a consistent evolutionary history. Dis. Appl. Math., 155, 944–958.

Grass Phylogeny Working Group (2001) Phylogeny and supertree classification of the grasses (poaceae). Ann. Mo. Bot. Gard., 88, 873–887.

Gusfield,D. (2005) Optimal, efficient reconstruction of Root-Unknown phylogenetic networks with constrained and structured recombination. J. Comput. Syst, Sci., 70, 383–399.

Hallett,M. and Lagergren,J. (2001) Efficient algorithms for lateral gene transfer problems. In Proceeding of Fifth Annual Conference on Research in Computational Molecular Biology (RECOMB-2001), ACM, New York, NY, USA, pp. 143–156.

Hein,J. et al. (1996) On the complexity of computing evolutionary trees. Dis. Appl. Math., 71, 153–169.

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

Huson,D. (2007) Split networks and reticulate networks. In Gusfield,D. and Steel,M. (eds), Reconstructing Evolution: New Mathematical and Computational Advances. Oxford University Press, Oxford, UK, pp. 247–276.

Huson,D. and Bryant,D. (2006) Application of phylogenetic networks in evolutionary studies. Mol. Biol. Evol., 23, 254–267.

Huson,D. and Klöpper,T. (2007) Beyond galled trees - decomposition and computation of hybridisation number. In Speed,T. and Huang,H. (eds), Proceeding of RECOMB 2007: The 11th Annual International Conference Research in Computational Molecular Biology. Vol. 4453 of LNBI, Springer, Berlin / Heidelberg, pp. 211–225.

Huson,D. et al. (2005) Reconstruction of reticulate networks from gene trees. In Proceeding of RECOMB 2005: The 9th Annual International Conference Research in Computational Molecular Biology. Vol. 3500 of LNBI, Springer, Berlin / Heidelberg, pp. 233–249.

Huson,D. et al. (2009) Computing galled networks from real data. Bioinformatics, 25, 485–493.

Linz,S. and Semple,C. (2009) Hybridization in nonbinary trees. IEEEACM Trans. Comput. Biol. Bioinform., 6, 30–45.

Nadkarni,L. (2009) Evolutionary phylogenetic networks: models and issues. In Heath,L. and Ramakrishnan,N. (eds), The Problem Solving Handbook for Computational Biology and Bioinformatics. Springer, New York Inc, New York, USA.

Nadkarni,L. et al. (2004) Reconstructing reticulate evolution in species - theory and practice. In Proceeding of 6th Annual International Conference Research on Computational Molecular Biology, Springer / Heidelberg, pp. 337–346.

Nadkarni,L. et al. (2005) Reconstructing reticulate evolution in species - theory and practice. J. Comp. Biol., 12, 796–811.

Schum,H. (2003) Phylogenetic trees from large datasets. PhD Thesis, Heinrich-Heine- Universität, Dusseldorf.

Semple,C. (2007) Hybridization networks. In Gusfield,D. and Steel,M. (eds), Reconstructing Evolution: New Mathematical and Computational Advances. Oxford University Press, Oxford, pp. 277–309.

van Iersel,L. et al. (2008) Constructing level-2 phylogenetic networks from triplets. In Vingron,M. and Wong,L. (eds), Proceeding of RECOMB 2008: The 12th Annual International Conference Research in Computational Molecular Biology. Vol. 4955 of LNBI, Springer, Berlin / Heidelberg, pp. 450–462.

Wu,Y. (2009) A practical method for exact computation of subtree prune and regraft distance. Bioinformatics, 25, 190–196.

Wu,Y. and Wang,J. (2010) Fast Computation of the exact hybridization number of two phylogenetic trees. In Brodowski,M. et al. (eds), Proceeding of DIMA 2010: The 6th International Symposium on Bioinformatics Research and Applications, Vol. 6053 of Lecture Notes in Bioinformatics, Springer, Berlin / Heidelberg, pp. 203–214.