SEEING THE TREES AND THEIR BRANCHES IN THE NETWORK IS HARD

I. A. KANJ

School of Computer Science, Telecommunications, and Information Systems, DePaul University, Chicago, IL 60604-2301, USA
E-mail: ikanj@cs.depaul.edu

L. NAKHLEH* and C. THAN†

Department of Computer Science, Rice University, Houston, TX 77005-1892, USA
**E-mail: nakhleh@cs.rice.edu* †*E-mail: cvthan@cs.rice.edu*

G. XIA

Department of Computer Science, Acopian Engineering Center, Lafayette College, Easton, PA 18042, USA
E-mail: gexia@cs.lafayette.edu

Phylogenetic networks are a restricted class of directed acyclic graphs that model evolutionary histories in the presence of *reticulate* evolutionary events, such as horizontal gene transfer, hybrid speciation, and recombination. Characterizing a phylogenetic network as a collection of trees and their branches has long been the basis for several methods of reconstructing and evaluating phylogenetic networks. Further, these characterizations have been used to understand molecular sequence evolution on phylogenetic networks.

In this paper, we address theoretical questions with regard to phylogenetic networks, their characterizations, and sequence evolution on them. In particular, we prove that the problem of deciding whether a given tree is contained inside a network is NP-complete. Further, we prove that the problem of deciding whether a branch of a given tree is also a branch of a given network is polynomially equivalent to that of deciding whether the evolution of a molecular character (site) on a network is governed by the *infinite site model*. Exploiting this equivalence, we establish the NP-completeness of both problems, and provide a parameterized algorithm that runs in time $O(2^{k/2}n^2)$, where n is the total number of nodes and k is the number of recombination nodes in the network, which significantly improves upon the trivial brute-force $O(2^k n)$ time algorithm for the problem. This reduction in time is significant, particularly when analyzing recombination hotspots.

Keywords: Phylogenies; Networks; Complexity; Parameterized algorithms.

1. Introduction

Phylogenies, i.e., evolutionary histories, play a major role in representing the relationships among biological entities. Their pervasiveness has led biologists, mathematicians, and computer scientists to design a variety of methods for their reconstruction. Until recently, most of these methods were designed to construct trees. Yet, biologists have long recognized that trees oversimplify our view of evolution in certain cases, since they cannot model events such as hybrid speciation, horizontal gene transfer (HGT), and recombination. These events, which are collectively referred to as *reticulation events* or *reticulate evolutionary events*, give rise to non-treelike evolutionary histories which are best modeled by *phylogenetic networks*. Reconstructing and evaluating the quality of phylogenetic networks is very important, given the emerging evidence of the ubiquity of reticulation events and the evolutionary roles they play.

Relationships between phylogenetic networks on one hand, and the trees and their branches on the other, have great significance. From the computational perspective, these relationships form the basis for the wide array of methods that have been devised for reconstructing phylogenetic networks. [8,10] From the biological perspective, these relationships shed light on how molecular sequences evolve down these networks. Events such as recombination, hybrid speciation, and lateral gene transfer break up the genomic history into many small pieces, each of which has a strictly treelike pattern of descent. [9] Identifying these trees and reconciling their discordance is the basis for several phylogenetic network reconstruction methods. [3,13] Understanding the relationship between a phylogenetic network and its branches, particularly in terms of the *clusters* (or *splits*) of taxa that they induce, has been the basis for another category of reconstruction methods. [1,4] Very recently, Nakhleh and colleagues introduced new approaches for augmenting a tree into a phylogenetic network to fit the evolution of a set of sequences based on parsimony [6] and likelihood [5] criteria.

Almost all of the aforementioned methods are based on understanding relationships among networks, trees, and clusters of taxa. Further, some of them rely on analysis of the evolution of sequences on networks. In this paper, we provide a theoretical treatment of the computational complexity of establishing some of these relationships. Nakhleh and Wang [14] devised efficient algorithms for restricted cases of some of these problems, while leaving the computational complexity of the general cases as open questions. In this paper, we prove that the problem of deciding whether a given tree is contained inside a network is NP-complete. Further, we prove that the problem of deciding whether a branch of a given tree is also a branch of a given network is polynomially equivalent to that of deciding whether the evolution of a molecular character (site) on a network is

governed by the *infinite site model*. Exploiting this equivalence, we establish the NP-completeness of both problems, and provide a parameterized algorithm that runs in time $O(2^{k/2}n^2)$, where n is the total number of nodes and k is the number of recombination nodes in the network, which significantly improves upon the trivial brute-force $O(2^k n)$ time algorithm for the problem. This improvement is very significant in practice. [12] Kanj *et al.* [7] considered the problem of character compatibility on a different model of phylogenetic networks that is used in historical linguistics. Whereas the NP-hardness result from that work is modified and used here, that is not the case, however, for the new parameterized algorithm that we present here. The algorithmic techniques do not carry over to the biologically-motivated model of phylogenetic networks that we consider here. Due to the lack of space, many proofs have been omitted.

2. Phylogenetic Networks, Trees, and the Infinite Site Model

Let $T = (V, E)$ be a tree, where V and E are the *tree nodes* and *tree edges*, respectively, and let $L(T)$ denote its leaf set. Further, let $\mathcal{X}$ be a set of taxa (species). Then, T is a phylogenetic tree over $\mathcal{X}$ if there is a bijection between $\mathcal{X}$ and $L(T)$. A tree T is said to be *rooted* if the set of edges E is directed and there is a single distinguished internal node r with in-degree 0.

A character c labeling the leaves of T is a function $c : L(T) \to \{0, 1\}$. Biologically, such character corresponds to a single SNP, and the two states it takes are the two possible *alleles* that the SNP may exhibit.[a] The commonly assumed model of evolution of SNPs is the *infinite site model*, which states that when a character (site) mutates, it changes its state to a new one that is not observed anywhere else in the tree. We denote by $c(v)$ the state of character c for node v. A *haplotype* of length k is a sequence of such characters $c_1 \cdots c_k$. A *full labeling*, or labeling for short, for character c on the tree is an extension, $\hat{c}$, of character c to label all the nodes of T; i.e., $\hat{c} : V(T) \to \{0, 1\}$ and $\hat{c}(v) = c(v)$ for every $v \in L(T)$. In this paper, we focus on characters that exhibit *exactly* two states.

Definition 2.1. A character c is *compatible* on tree T if there is a labeling $\hat{c}$ which extends c such that there exists exactly one edge $e = (u, v) \in E(T)$ where $\hat{c}(u) \neq \hat{c}(v)$, and for all other edges $e' = (u', v') \neq e$, $\hat{c}(u') = \hat{c}(v')$.

Notice that if SNP c evolves under the infinite site model, then there is a tree on which it is compatible. Hence, the compatibility criterion reflects this model of evolution. A sequence of characters $c_1 \cdots c_k$ is compatible on tree T if every

[a] Even though SNPs may exhibit all four states (A, C, T, and G), bi-allelic SNPs, i.e., SNPs that exhibit two states, are the most common.

character c_i, $1 \leq i \leq k$, is compatible on T. By this definition of compatibility, it suffices to establish the computational complexity of and develop algorithms for testing the compatibility of single characters. Therefore, from this point on, we focus on the case of a single character. Testing whether a character is compatible on a tree T with n leaves can be done in $O(n)$ time. [11]

As explained in Section 1, when reticulation events occur, the evolutionary history of a set of sequences is best modeled by a phylogenetic network. A phylogenetic network $N = (V, E)$ is a rooted directed acyclic graph, with set $L(N)$ of leaves, such that there is a bijection between a set of taxa $\mathcal{X}$ and $L(N)$. A network N has three types of nodes: (1) one node r with in-degree 0, which corresponds to the root; (2) nodes with in-degree 1, which correspond to *coalescence* events; and (3) nodes with in-degree 2, which correspond to recombination. Fig. 1 shows an example of a phylogenetic network on four taxa A, B, C, and D. A

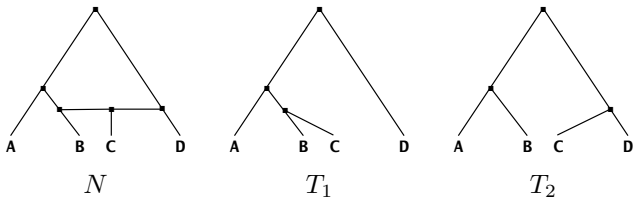

Fig. 1. A phylogenetic network N on four taxa A, B, C, and D, and the two trees T_1 and T_2 it contains. Character c_1, where $c_1(A) = c_1(D) = 0$ and $c_1(B) = c_1(C) = 1$ is compatible on tree T_1, but not compatible on tree T_2. Character c_2, where $c_2(A) = c_2(B) = 0$ and $c_2(C) = c_2(D) = 1$ is compatible on T_2 but not on T_1. It can be easily checked that there does not exist any tree T on which both characters are compatible. However, both characters are compatible on network N. The two horizontal edges in N are directed towards the parent of C.

phylogenetic network N induces, or contains, a set of trees; these trees model the evolutionary histories of sets of non-recombining segments (or, genes) in the genomic sequences. We denote by $\mathcal{T}(N)$ the set of all trees contained inside network N. Each such tree is obtained by the following two steps: (1) for each node of in-degree 2, remove one of the incoming edges, and then (2) for every node x of in-degree and out-degree 1, whose parent is u and child is v, remove node x and its two adjacent edges, and add a new edge from u to v. If node x is the root and its out-degree is 1, remove x and make its only child the new root for the tree. Figure 1 shows the two trees contained inside network N. The membership problem of trees and networks, which is heavily used in network reconstruction methods, is formulated as follows.

Problem 2.1. *Tree Containment (*TC*)*

Input: *A phylogenetic network N and tree T over the same set $\mathcal{X}$ of taxa.*
Question: *Is $T \in \mathcal{T}(N)$?*

In the next section, we prove that the TC problem is NP-complete. The notion of character compatibility is extended to phylogenetic networks so as to reflect the biological fact that the evolutionary history of a character is modeled by one of the trees inside the network.

Definition 2.2. A character c is *compatible* on network N if c is compatible on at least one tree $T \in \mathcal{T}(N)$.

The problem of testing the infinite site model on a phylogenetic network can be defined as follows.

Problem 2.2. *Infinite Site on Phylogenetic Networks (*ISPN*)*
Input: *Phylogenetic network N and a binary character c labeling the leaves of N.*
Question: *Is c compatible on N?*

Given a network N with k nodes of in-degree 2, the size of $\mathcal{T}(N)$ is $O(2^k)$. Therefore, The ISPN problem is solvable in $O(2^k n)$ time, given the algorithm for solving the problem when N is a tree. [11] In the next sections, we prove that ISPN is NP-complete and introduce a more efficient algorithm for solving it.

Further, we establish equivalence between the ISPN problem and another problem from phylogenetics, namely the Cluster Containment problem. [14] Let T be a phylogenetic tree on a set $\mathcal{X}$ of taxa. We say that edge e induces, or defines, cluster $X \subseteq \mathcal{X}$, where X is the set of all leaves reachable from the root of T through edge e. We denote by $\mathcal{C}(T)$ the set of all clusters defined by tree T. This notion is extended to networks by $\mathcal{C}(N) = \cup_{T \in \mathcal{T}(N)} \mathcal{C}(T)$. The Cluster Containment problem is defined as follows.

Problem 2.3. *Cluster Containment (*CC*)*
Input: *A phylogenetic Network N and set X of taxa.*
Question: *Is $X \in \mathcal{C}(N)$?*

Nakhleh and Wang [14] devised a polynomial time algorithm for a restricted version of the CC problem, yet its complexity in the general case was left open. We show that CC and ISPN are polynomially equivalent, thus establishing NP-completeness of the former problem as well.

3. Computational Complexity of the TC Problem

A straightforward way to answer this question is to generate all possible trees from the network and compare them with T. Checking if two trees are identical can be

done in polynomial time. However, for a network N with k recombination nodes, the number of trees induced by the network is $O(2^k)$, and therefore checking if a tree is contained in a network using this brute-force approach takes exponential time.

Theorem 3.1. *The problem TC is NP-complete.*

4. The ISPN Problem: Complexity and A Parameterized Algorithm

4.1. *NP-completeness of* ISPN

Kanj *et al.* [7] proved the NP-completeness for the problem of character compatibility on phylogenetic networks when the network edges are bi-directional. We modify their proof to make it work for the ISPN problem and present the theorem.

Theorem 4.1. ISPN *is NP-complete.*

4.2. *A Parameterized Algorithm for* ISPN

A Prelude to the Algorithm

An instance of a parameterized problem is a pair consisting of an input instance x of size n and a parameter k. A parameterized problem is *fixed-parameter tractable* if it can be solved in time $f(k)n^{O(1)}$, where f is a computable function of the parameter k. [2]

Naturally, the ISPN problem can be parameterized by the number of *recombination nodes* (nodes of in-degree 2) k in the phylogenetic network, which is usually much smaller than the total number of nodes in the network. [12] Every recombination node in N has two incoming edges, and hence two possible parents. Deciding the parent of each recombination node in N *induces* a tree from N, and N is compatible if and only if there exists an induced tree from N that is compatible. Since there are $O(2^k)$ such induced trees, the ISPN problem can be solved in $O(2^k n)$ time, where n is the number of nodes in N, by enumerating all possible induced trees then checking whether any of them is compatible using the linear time algorithm. [11] We shall improve on this trivial upper bound next by presenting a simple branch-and-search algorithm that runs in $O(2^{k/2}n^2)$ time.

For two nodes u and v in N, we denote by the ordered pair (u, v) the directed edge from u to v (in case the edge exists in N). A node u in N is an *internal* node if u is not a leaf in N, that is, if the out-degree of u is greater than 0.

Definition 4.1. For a node $u \in N$ we define the *weight* of u, denoted $wt(u)$, to be the in-degree of u minus 1 if u is not the root of N, and to be 0 if u is the root of N.

Definition 4.2. A node u in N is a said to be a *recombination node* if it is weight is greater or equal to 1, otherwise, u is said to be a *non-recombination node*.

Definition 4.3. Let N be a network. A node p in N is said to be a *partition node* if there exists an induced compatible tree T from N such that there is a valid labeling for the nodes in T with all the nodes in the subtree rooted at p in T labeled with the same label, and all the other nodes in T labeled with the other label.

While applying the branch-and-search process, the algorithm will label some of the internal nodes in the network. Therefore, the network will get partially labeled as the algorithm progresses. In many cases the (resulting) network can be simplified, or even, its compatibility can be inferred easily. We describe next some of the scenarios in which the compatibility of the network can be directly decided. We also describe some operations that simplify the network. The algorithm will make use of these operations and simplifications.

Proposition 4.1. *If there is at most one leaf of label 0 (similarly 1) in N then N is compatible.*

Proposition 4.2. *Let $u \in N$ be a node. Suppose that u has two children that are non-recombination nodes. Suppose further that these two children have different labels and none of them is a partition node. Then N is not compatible.*

Proposition 4.3. *If a labeled node $u \in N$ has a non-recombination child v such that $label(v) \neq label(u)$ and v is not a partition node, then N is not compatible.*

Proposition 4.4. *Let u be a recombination node in N and let (u', u) be an incoming edge to u. Suppose that $label(u) \neq label(u')$. Let N' be the network resulting from N by removing the edge (u', u). Then N is compatible if and only if N' is. Moreover, if p is a partition node in N then p is also a partition node in N'.*

Proposition 4.5. *Let u be a labeled non-recombination node in N with the incoming edge (u', u). Suppose that u' is unlabeled. Let N' be the network obtained from N by setting $label(u') = label(u)$ if u is not the partition node, and $label(u') = 1 - label(u)$ if u is the partition node. Then N is compatible and p is a partition node in N if and only if N' is compatible and p is a partition node in N'.*

Proposition 4.6. *Let u be a node in N and let (u', u) be an incoming edge to u. Suppose that $label(u) = label(u')$. Suppose further that u is a recombination node and let (u'', u) be another incoming edge to u. Let N' be the network resulting from N by removing the edge (u'', u). Then N is compatible with a partition node p if and only if N' is compatible with a partition node p.*

Proposition 4.7. *Let w be a node in N such that all its children are leaves labeled with the same label. Let N' be the network obtained from N by: (1) replacing w and its children with a leaf w' labeled with the same label as the children of w, (2) making every incoming edge to w an incoming edge to w', and (3) making every incoming edge to a child of w an incoming edge to w'. Then N is compatible if and only if N' is. Moreover, if p is a partition node of N then p is also a partition node of N', unless p is either w or one of its children and in which case w' is a partition node in N'.*

We describe below a procedure that simplifies the network according to the operations and simplifications described in the previous propositions.

Simplify(N)
1. **while** there is a node $u \in N$ of out-degree 0 **do** remove u from N;
2. **if** there is only one leaf in N with label 0 (or 1) **then return** (TRUE);
3. **if** the partition node p is a leaf **then return (FALSE)**;
4. **if** there exists a node $u \in N$ that has two children with different labels that are non-recombination
   nodes and such that none of them is a partition node **then return (FALSE)**;
5. **if** there exists a labeled node $u \in N$ that has a non-recombination child v such that $label(v) \neq label(u)$
   and v is not a partition node **then return (FALSE)**;
6. **if** there exists a recombination node $u \in N$ and an edge $(u', u) \in N$ such that $label(u') \neq label(u)$
   then remove (u', u) and decrease $wt(u)$ by 1;
7. **if** u' is unlabeled and has a labeled child u such that u is a non-recombination node **then**
      if u is designated as the partition node **then** set $label(u') = 1 - label(u)$;
      else set $label(u') = label(u)$;
8. **if** there exists a recombination node $u \in N$ and an edge $(u', u) \in N$ such that $label(u') = label(u)$
   then for every edge $(u'', u) \in N$ where $u'' \neq u$ **do** remove (u'', u) and decrease $wt(u)$ by 1;
9. **if** there exists a node $w \in N$ such that all the children of w are leaves labeled with the same label
   then {$*$ note that w at this point must be unlabeled or labeled as its children $*$}
      9.1. remove w and its children and replace them with a leaf w';
      9.2. label w' with the same label as the children of w;
      9.3. make all incoming edges to w and its children incoming edges to w' and set $wt(w')$ to be
         $wt(w)$ plus the sum of the weights of the children of w;
      9.4. **if** w is designated as the partition node **then** designate w' as the partition node in the
         resulting network;

Fig. 2. The procedure **Simplify**.

Proposition 4.8. *Let N be a network with a given partition node p. If the procedure **Simplify** decides the instance N, then its decision is correct, and if it applies an operation to N to obtain a network N' with a partition node p', then N is compatible with p as a partition node if and only if N' is compatible with p' as a partition node.*

Lemma 4.1. *Let N be a network with a partition node p and suppose that the procedure **Simplify** if applied to N does not decide N nor does it perform any operation to N. Then there exists a node $w \in N$ satisfying the following properties: (1) w has at least two children and all the children of w are leaves; (2) there are at least two children of w with different labels; (3) w is unlabeled; and (4) every child of w is a recombination node.*

Proof. *Let ℓ be a leaf in N such that the root-leaf path P to ℓ has maximum length. Note that ℓ must exist by step 1 of **Simplify** (every path starting at the root of N must lead to a leaf). Let w be the parent of ℓ on the path P. By the maximality of P, all the children of w must be leaves. If all the children of w are labeled with the same label, then step 9 of **Simplify** would apply to w. This shows that w has at least two children labeled with different labels, and properties (1) and (2) about w have been established.*

*Suppose, to get a contradiction, that w is labeled. Let u be a child of w such that $label(u) \neq label(w)$. By step 6 of **Simplify**, u must be a non-recombination node otherwise the edge (w, u) would be removed. By step 5 of **Simplify**, v must be a partition node. But then by step 3 of **Simplify** the procedure would have rejected the instance, contradicting the statement of the lemma. It follows that w is unlabeled establishing property (3) about w.*

*Finally, if w had a child that is a non-recombination node, then by step 7 of **Simplify**, w would have been labeled contradicting property (3) shown above. This establishes property (4) about w and completes the proof.* □

The Algorithm

The algorithm **ISPN-Solver** is given in Figure 3. The algorithm implicitly assumes that the partition node p is given. This assumption can be removed by trying every node in N as the partition node, then calling the algorithm with that node as the partition node. This will increase the running time of the algorithm by an $O(n)$ factor. If the algorithm **ISPN-Solver** returns TRUE on any of these calls then N must be compatible. To keep the presentation of the algorithm concise, we will not enumerate the partition nodes, but we will compensate for that by multiplying the running time of the algorithm by a linear factor at the end.

The algorithm **ISPN-Solver** is a branch-and-search process. Each stage of the algorithm starts with an instance (N, k) of the problem, where k is the total weight of all the nodes in N, and then tries to reduce k either by branching or by simplifying the network. Then the algorithm recursively works on the reduced instances. We implicitly assume that after each step, the network N and the parameter k are updated accordingly.

ISPN-Solver (N, k)
$\{*$ k is the total weight of all the nodes in N $*\}$
1. **if** $k = 0$ and N is not compatible **then** reject;
2. **while** the procedure **Simplify** is applicable to N **do** apply it;
3. let w be a node satisfying the statement of Lemma 4.1; branch as follows:
first side of the branch: set $label(w) = 1$;
second side of the branch: set $label(w) = 0$;

Fig. 3. The algorithm **ISPN-Solver**.

Theorem 4.2. *The algorithm **ISPN-Solver** correctly decides in time $O(2^{k/2}n)$ whether a phylogenetic network with a given partition node is compatible or not.*

Proof.

*The correctness of the algorithm can be easily checked. To analyze the running time of the algorithm **ISPN-Solver**, notice that the algorithm is a branch-and-bound process and its execution can be depicted by a search tree. The running time of the algorithm is proportional to the number of root-to-leaf paths, or equivalently the number of leaves in the search tree, multiplied by the time spent along each such path. Therefore, the main step in the analysis of the algorithm is deriving an upper bound on the number of leaves in the search tree. Let $\mathcal{T}$ be the search tree for the algorithm **ISPN-Solver** on an input instance (N, k), and let $T(k)$ be the number of leaves in $\mathcal{T}$. Let w be a node that the algorithm **ISPN-Solver** branches on in step 3.*

*Since all the children of w are leaves, the children of w are all labeled. Since all the children of w are recombination nodes by property (3) of Lemma 4.1, when the algorithm labels w in each of the two branches, at least one incoming edge to each child of w having the same label as w will be removed by step 8 of **Simplify** when applied next to the network. On the other hand, an incoming edge to every child of w whose label is different from w will be removed by step 6 of **Simplify**. Therefore, for every child of w, the weight of the child will be decreased by at least 1 in the next call to **Simplify**. Since w has at least two children by property (2) of Lemma 4.1, the total weight k of all the nodes in N is reduced by at least 2 in*

every side of the branch. It follows that the number of leaves $T(k)$ of the search tree $\mathcal{T}$ satisfies the recurrence relation $T(k) \leq 2T(k-2)$, and $T(k) = O(2^{k/2})$.

*Now consider a root-leaf path in the search tree $\mathcal{T}$. On every node of this path the algorithm might need to call the procedure **Simplify**, which could take $O(n)$ time since the size of N is $O(n)$. However this need not be the case with a careful implementation of this procedure. Instead of calling this procedure at each node of N, we only call it on the nodes on which the operation is applicable. The time spent by the procedure in each such call is proportional to the number of nodes/edges removed plus the number of nodes labeled in the call. Since we can only have $O(n)$ nodes/edges, the total time spent by the procedure on a root-leaf path of $\mathcal{T}$ is proportional to the size of the network, which is $O(n)$. It follows that the running time of the algorithm is $O(2^{k/2}n)$.* □

Corollary 4.1. *The* ISPN *problem can be solved in time $O(2^{k/2}n^2)$, where n is the number of nodes and k is the number of recombination nodes, respectively, in the phylogenetic network.*

5. The Cluster Containment Problem

Let T be phylogenetic tree on set $\mathcal{X}$ of taxa and rooted at node r. Each edge $e = (u, v)$ induces a cluster c_e of taxa, which is the set of leaves reachable from root r only through v. It is easy to see that the leaves in c_e are exactly the leaves of the subtree rooted at v. A cluster c_e is contained in a network N if it is a cluster in a tree induced from N.

We can easily determine if a cluster c is in a tree by finding the least common ancestor $lca(c)$ of leaves in c, and then comparing the leaf set under $lca(c)$ and c. The CC problem is hard because there are many different trees that can be induced from the network N. The NP-hardness of CC is a byproduct of the following theorem.

Theorem 5.1. *The problems* CC *and* ISPN *are polynomially equivalent.*

Corollary 5.1. *The problem* CC *is NP-hard.*

Corollary 5.2. *The* CC *problem when parameterized by the number of recombination nodes k in the network is solvable in time $O(2^{k/2}n^2)$, where n is the number of nodes in the network.*

References

1. D. Bryant and V. Moulton. NeighborNet: An agglomerative method for the construction of planar phylogenetic networks. In R. Guigo and D. Gusfield, editors, *Proc. 2nd*

Workshop Algorithms in Bioinformatics (WABI'02), volume 2452 of *Lecture Notes in Computer Science*, pages 375–391. Springer Verlag, 2002.

2. R. Downey and M. Fellows. *Parameterized Complexity*. Springer, New York, 1999.
3. M.T. Hallett and J. Lagergren. Efficient algorithms for lateral gene transfer problems. In *Proc. 5th Ann. Int'l Conf. Comput. Mol. Biol. (RECOMB01)*, pages 149–156, New York, 2001. ACM Press.
4. D.H. Huson. SplitsTree: A program for analyzing and visualizing evolutionary data. *Bioinformatics*, 14(1):68–73, 1998.
5. G. Jin, L. Nakhleh, S. Snir, and T. Tuller. Maximum likelihood of phylogenetic networks. *Bioinformatics*, 22(21):2604–2611, 2006.
6. G. Jin, L. Nakhleh, S. Snir, and T. Tuller. Inferring phylogenetic networks by the maximum parsimony criterion: a case study. *Molecular Biology and Evolution*, 24(1):324–337, 2007.
7. I. Kanj, L. Nakhleh, and G. Xia. Reconstructing evolution of natural languages: Complexity and parameterized algorithms. In *12th Annual International Computing and Combinatorics Conference*, volume 4112 of *Lecture Notes in Computer Science*, pages 299–308. Springer, 2006.
8. C.R. Linder, B.M.E. Moret, L. Nakhleh, and T. Warnow. Network (reticulate) evolution: biology, models, and algorithms. In *The Ninth Pacific Symposium on Biocomputing (PSB)*, 2004. A tutorial.
9. W.P. Maddison. Gene trees in species trees. *Systematic Biology*, 46(3):523–536, 1997.
10. V. Makarenkov, D. Kevorkov, and P. Legendre. Phylogenetic network reconstruction approaches. *Genes, Genomics, and Bioinformatics*, 6, 2005.
11. L. Nakhleh. *Phylogenetic Networks*. PhD thesis, The University of Texas at Austin, 2004.
12. L. Nakhleh, D. Ringe, and T. Warnow. Perfect phylogenetic networks: A new methodology for reconstructing the evolutionary history of natural languages. *LANGUAGE*, 2005. In press.
13. L. Nakhleh, D. Ruths, and L.S. Wang. RIATA-HGT: A fast and accurate heuristic for reconstrucing horizontal gene transfer. In L. Wang, editor, *Proceedings of the Eleventh International Computing and Combinatorics Conference (COCOON 05)*, pages 84–93, 2005. LNCS #3595.
14. L. Nakhleh and L.S. Wang. Phylogenetic networks, trees, and clusters. In *Proceedings of the 2005 International Workshop on Bioinformatics Research and Applications (IWBRA 05)*, pages 919–926, 2005. LNCS #3515.