Phylogenetic Networks: Properties and Relationship to Trees and Clusters

Luay Nakhleh[1] and Li-San Wang[2]

[1] Department of Computer Science, Rice University, Houston, TX 77005, USA
nakhleh@cs.rice.edu

[2] Department of Biology, University of Pennsylvania, Philadelphia, PA 19104, USA
lswang@mail.med.upenn.edu

Abstract. Phylogenetic networks model evolutionary histories in the presence of non-treelike events such as hybrid speciation and horizontal gene transfer. In spite of their widely acknowledged importance, very little is known about phylogenetic networks, which have so far been studied mostly for specific datasets.

Even when the evolutionary history of a set of species is non-treelike, individual genes in these species usually evolve in a treelike fashion. An important question, then, is whether a gene tree is "contained" inside a species network. This information is used to detect the presence of events such as horizontal gene transfer and hybrid speciation. Another question of interest for biologists is whether a group of taxa forms a clade based on a given phylogeny. This can be efficiently answered when the phylogeny is a tree simply by inspecting the edges of the tree, whereas no efficient solution currently exists for the problem when the phylogeny is a network. In this paper, we give polynomial-time algorithms for answering the above two questions.

1 Introduction

Phylogenies are the main tool for representing the relationships among biological entities. Their pervasiveness has led biologists, mathematicians, and computer scientists to design a variety of methods for their reconstruction. Furthermore, extensive studies have been focused on the performance of these methods under different models and settings, as well as on the combinatorial and biological properties of trees (e.g., [7, 2]). However, almost all such methods construct trees, and almost all studies have been aimed at trees. Yet, biologists have long recognized that trees oversimplify our view of evolution, since they cannot take into account such events as hybridization, lateral gene transfer, and recombination. These non-tree events give rise to edges that connect nodes on different branches of a tree, giving rise to a directed acyclic graph structure that is usually called a *phylogenetic network*.

A gene tree is a model of how a gene evolves through duplication, loss, and nucleotide substitution. Gene trees can differ from one another as well as from the species phylogeny. Such differences arise during the evolutionary process due to events such as duplication and loss, whereby each genome may end up with multiple copies of a given gene—but not necessarily the same copies that survive in another genome. Unless the genome is very well sampled, only a subset (sometimes only one copy, in fact) of the gene is used in phylogenetic analyses. As a result, the phylogeny for the gene may not agree with the species phylogeny, nor with the phylogeny for another gene. Because the gene copy has a single ancestral copy, barring recombination, the resulting history is a branching tree. Point mutations can cause some of the copies to be imperfect representations of the original, but this process does not compromise the existence of the (gene) tree. 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 tree-like pattern of descent [4]. Thus, within a species phylogeny, many tangled gene trees can be found, one for each nonrecombined locus in the genome. Incongruence among gene trees is a powerful tool for detecting recombination, hybrid speciation, and other non-treelike evolutionary events (e.g., see [6]). While testing for incongruence between two (gene) trees can be done in a straightforward manner, it is not as simple for testing the incongruence between a tree and a network, since the number of trees "inside" a network grows exponentially with the number of non-treelike events. In this paper, we give the first polynomial-time algorithm for solving this problem.

A phylogeny can be viewed as a collection of clusters of taxa (each defined as the set of leaves in a subtree). Various approaches for reconstructing phylogenies (trees and networks) have been proposed based on this view (see, e.g., [1, 3]). An interesting biological question, then, is whether a group of taxa forms a cluster in a given phylogeny. This question can be answered in a straightforward manner when the phylogeny is a tree, since each edge in a tree defines a unique cluster. However, the number of clusters in a phylogenetic network grows exponentially with the number of non-treelike events, and hence an efficient algorithm for solving the problem is not straightforward. In this paper, we present the first polynomial-time algorithm for solving this problem.

The rest of the paper is organized as follows. In Section 2 we give a background on trees, clades and clusters. In Section 3 we briefly describe evolutionary events that necessitate phylogenetic networks, and describe the graph-theoretic model of phylogenetic networks that we use in the paper, along with combinatorial properties that follow from the model. In Section 4 we introduce the concepts of network decomposition and dependency graphs. computing these structures forms the core of our algorithms. In Section 5, we define reduced inheritance profiles and present the main lemma on which the our algorithms are based. In Section 6 we describe our polynomial-time algorithms for solving the aforementioned decision problems. We conclude in Section 7 with a summary of our main results and directions for future research.

2 Background: Phylogenetic Trees

2.1 Notation

In this paper, and unless stated otherwise, all graphs are directed. Given a graph G, $E(G)$ denotes the set of (directed) edges of G and $V(G)$ denotes the set of nodes of G. We write (u, v) to denote a directed edge from node u to node v. If $e = (u, v)$ is an edge from u to v, we call u the *tail* and v the *head* of the edge and say that u is a *parent* of v. The *indegree* of a node v, denoted $indeg(v)$, is the number of edges whose head is v, while the *outdegree* of v, denoted $outdeg(v)$, is the number of edges whose tail is v. The degree of a node v is the sum of its indegree and outdegree. In an undirected graph, the degree of a node v is the number of edges incident with v. A node u is *redundant* if $indeg(u) = outdeg(u) = 1$. A directed path of length k from u to v in G is a sequence $u_0 u_1 \cdots u_k$ of nodes with $u = u_0$, $v = u_k$, and $\forall i,\ 1 \leq i \leq k,\ (u_{i-1}, u_i) \in E(G)$. Node v is *reachable* from u in G, denoted $u \rightsquigarrow v$, if there is a directed path in G from u to v.

2.2 Phylogenetic Trees, Bipartitions and Clusters

A *phylogenetic tree* is a leaf-labeled tree that models the evolution of a set of taxa (species, genes, languages, placed at the leaves) from their most recent common ancestor (placed at the root). The internal nodes of the tree correspond to the speciation events.

Mathematically, A *rooted* phylogenetic tree is a rooted tree without redundant nodes and whose leaves are labelled distinctively. An unrooted phylogenetic tree is a rooted phylogenetic tree with the root suppressed. Every edge e in an unrooted leaf-labeled tree T defines a bipartition (or, split) $\pi(e)$ on the leaves (induced by the deletion of e), so that we can define the set $\Pi(T) = \{\pi(e) : e \in E(T)\}$. Every edge e in a rooted leaf-labeled tree T defines a cluster $c(e)$ of leaves (those leaves that are reachable from the root through e), so that we can define the set $C(T) = \{c(e) : e \in E(T)\}$. A clade of a rooted tree T is the entire subtree rooted at a node of T; the set of all leaves in a clade correspond to a cluster of T.

There is a many-to-one relationship between rooted and unrooted phylogenetic trees: there are many ways to root an unrooted phylogenetic tree. Based on this we also see an association between clusters of a rooted tree T and bipartitions of the unrooted version of T: each cluster of a rooted tree T equals one of the two sets in the bipartition induced by an edge e in the unrooted version of T.

3 Phylogenetic Networks

3.1 Non-tree Evolutionary Events

We now describe two types of evolutionary events that give rise to network (as opposed to tree) topologies: hybridization and lateral gene transfer. In hybridization, two lineages recombine to create a new species, as symbolized in Figure 1(a). We can distinguish between *diploid hybridization*, in which the new species inherits one of the two homologs for each chromosome from each of its two parents—so that the new species has the same number of chromosomes as its parents, and *polyploid hybridization*, in which the new species inherits the two homologs of each chromosome from both parents—so that the new species has the sum of the numbers of chromosomes of its parents. Prior to hybridization, each site on each homolog has evolved in a tree-like fashion, although, due to meiotic recombination, different strings of sites may have different histories. Thus, each site in the homologs of the parents of the hybrid evolved in a tree-like fashion on one of the trees contained inside (or induced by) the network representing the hybridization event, as illustrated in Figures 1(b) and 1(c). In lateral gene transfer, genetic material is transferred from one lineage to another without resulting in the production of a new lineage, as symbolized in Figure 1(d). In an evolutionary scenario involving lateral transfer, certain sites are inherited through lateral transfer from

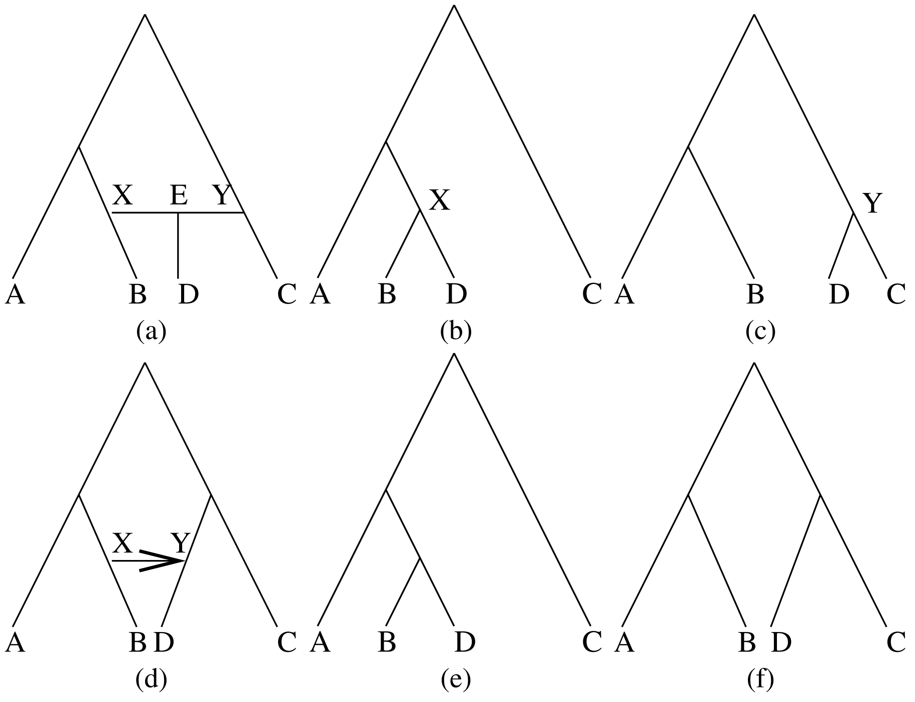


Fig. 1. Hybrid speciation: the network in (a) and its two induced trees in (b) and (c). Horizontal transfer: the network in (d) and its two induced trees in (e) and (f).

another species, as in Figure 1(e), while all others are inherited from the parent, as in Figure 1(f).

When the evolutionary history of a set of taxa involves processes such as hybridization or lateral gene transfer, trees can no longer represent the evolutionary relationship; instead, we turn to rooted directed acyclic graphs (rooted DAGs).

3.2 Phylogenetic Networks: Model and Properties

In this paper, we adopt the general model of (reduced) phylogenetic networks, as described in [5]).

Definition 1. *A* **phylogenetic network** *is a connected directed acyclic graph $N = (V, E)$, where V can be partitioned into $\{r\} \cup Tr(N) \cup Nt(N) \cup L(N)$, where:*

1. *Node r is the root; it has indegree 0.*
2. *Set $Tr(N)$ is the set of tree nodes; each node u in $Tr(N)$ has indegree 1 and outdegree > 1.*
3. *Set $Nt(N)$ is the set of network nodes; each node v in $Nt(N)$ has indegree 2 and outdegree 1.*

4. *Set $L(N)$ is the set of leaf nodes (taxa); each node x in $L(N)$ has indegree* 1 *and outdegree* 0. *Each node x in $L(N)$ is labeled uniquely by an integer i, where* $1 \leq i \leq |L(N)|$.

Figures 1(a) and 1(d) give two examples of phylogenetic networks. Given a network N, we classify its edges as *tree edges* and *network edges*. An edge $e = (u, v)$ is a tree edge if v is a tree node or a leaf; otherwise, it is a network edge. Biologically, the tree nodes correspond to regular speciation events in the evolutionary history, whereas network nodes correspond to reticulation events (e.g., hybridization, lateral gene transfer, recombination, etc.).

We say that network N is *binary* if the root and all tree nodes of N have outdegree 2. In this paper, and unless noted otherwise, all networks are binary. Further, we assume that if u is a tree node and (u, v) and (u, w) are the two edges incident from u, then at least one of the two nodes v and w is a tree node.

A *forced contraction* is an operation on a graph in which we delete a redundant node and replace the two edges incident to it by a single edge. An *augmentation* is an operation on a graph in which an edge (u, v) is replaced by two edges (u, x) and (x, v), where x is a new node. A DAG N is a *pseudo-network* if a network N' can be obtained by applying a sequence of forced contraction operations to N (alternately, if N can be obtained by applying a sequence of augmentation operations to a network N'). We generalize the *clade* concept to networks as follows. Given a network N, we say that the DAG N', rooted at node x, is a *network clade* of N, if there exists an edge $e = (u, x)$ in N whose removal disconnects N, thus creating two components, one of which is N' (rooted at x). If network clade N' does not contain network nodes, i.e., N' is a tree, we refer to N' simply as a clade. Given a network N, and a clade N', we say that N' is *maximal* if N does not contain any clade N'' such that $N' \subset N''$.

A phylogenetic network $N = (V, E)$ defines a partial order on the set V of nodes, and based on this partial order, we assign times to the nodes of N; $t(u)$ denotes the time associated with node u. If there is a directed path p from node u to node v, such that p contains at least one tree edge, then $t(u) < t(v)$. If $e = (u, v)$ is a network edge, then $t(u) = t(v)$ (since reticulation events occur instantaneously). Further, if there is a directed path from node u to node v, $u \neq v$, we say that u is *above* v and that v is *below* u, both denoted by $u > v$. We say that node u in N is a *lowest network node* if (1) u is a network node, and (2) for any network node v, $v \neq u$, we have $u \not> v$.

Lemma 1. *Let N be a network, u be a lowest network node, and $e = (u, v)$ be the edge incident from u. Then, the subgraph $N' \subset N$ rooted at v is a maximal clade.*

Proof. By definition of lowest network node, all nodes below u are either tree nodes or leaves; hence, N' is a clade. Assume N'' is also a clade, and that $N' \subset N''$. Then, N'' contains node u, which is a network node – a contradiction. Therefore, N' is a maximal clade.

Given a network N and two nodes u and v in N, we say that u and v cannot co-exist in time if there is a directed path $p = \langle u_0, u_1, \ldots, u_k \rangle$ in N, where $u_0 = u$ and $u_k = v$, and p satisfies three properties: (1) p contains at least one tree edge, (2) for any tree edge e on p, we have $e = (u_i, u_{i+1})$ (may not be (u_{i+1}, u_i)), $0 \leq i \leq k - 1$, and (3) the orientation of a network edge on p is irrelevant.

Since events such as hybridization and lateral gene transfer occur between two lineages (nodes in the network) that co-exist in time, a phylogenetic network N must satisfy the *synchronization property*, which states that if two nodes x and y cannot co-exist in time, then there do not exist two edges $e = (x, v)$ and $e' = (y, v)$ in N. If a network N violates the synchronization property (which may happen due to missing taxa in the phylogenetic analysis), then N can be augmented to remedy this violation, as we show in the following theorem.

Theorem 1. *For any phylogenetic network N, there exists an augmentation of N into a pseudo-network N' that satisfies the synchronization property.*

Proof. If N satisfies the synchronization property, then $N' = N$. Assume N does not satisfy the synchronization property, and let $e_1 = (x_1, v)$ and $e_2 = (x_2, v)$ be two network edges such that $x_1 \rightsquigarrow x_2$. Let N' be the network obtained from N by replacing edge e_1 by two new edges $e_1' = (x_1, y)$ and $e_1'' = (y, v)$, where y is a new node. Now, the network node v has the two parents y and x_2. It is clear that $x_2 \not\rightsquigarrow y$, since the only way to reach y is through x_1, and $x_2 \not\rightsquigarrow x_1$ (otherwise, N would be cyclic). It is also clear that $y \not\rightsquigarrow x_2$, since if y reaches x_2, it has to be via a path that passes through v, and since x_2 reaches v, N would be cyclic. We apply the same process to every pair of edges that violates the synchronization property.

We write $N|_{L'}$, where $L' \subset L(N)$, to denote the subgraph N' obtained from N by removing all leaves not in L', and then applying forced contraction operations and removal of nodes of outdegree 0 (other than the leaves in L'). We now describe some properties of phylogenetic networks.

Proposition 1. *Let $N = (V, E)$ be a phylogenetic network.*

1. $outdeg(r) + \sum_{s \in Tr(N)} (outdeg(s) - 1) = |Nt(N)| + |L(N)|$.
2. *For every node $v \in V$, $r \rightsquigarrow v$.*
3. *For every node $v \in V$, there exists at least one leaf l below v.*
4. *(Taxon sampling) $N|_{L'}$ is a phylogenetic network, for any $L' \subset L(N)$.*

Proof.

1. By the observation $\sum_{v \in V} outdeg(v) = \sum_{v \in V} indeg(v)$.
2. Let V' be the set of all nodes that cannot be reached from r. Let x be a maximal element (in terms of the partial order induced by N on V) in V'; then $indeg(x) = 0$ (otherwise N would be cyclic). However, the only node with indegree 0 is r – a contradiction.
3. Let $R(v) = \{u \in V : v > u\}$. If $R(v) = \emptyset$, then $outdeg(v) = 0$, i.e., v is a leaf. If $R(v) \neq \emptyset$, and since N is acyclic (and finite), then there exists at least one node x in $R(v)$ with outdegree 0. It follows from Definition 1 that x is a leaf.
4. Straightforward.

In this paper, we focus on binary networks, but the results extend to general networks in a straightforward manner.

3.3 Networks and Trees

There is a fundamental connection between (species) networks and (gene) trees. A gene tree is a model of how a gene evolves through duplication, loss, and nucleotide substitution. Gene trees can differ from one another as well as from the species phylogeny. Such differences arise during the evolutionary process due to events such as duplication and loss, whereby each genome may end up with multiple copies of a given gene—but not necessarily the same copies that survive in another genome. Unless the genome is very well sampled, only a subset (sometimes only one copy, in fact) of the gene is used in phylogenetic analyses. As a result, the phylogeny for the gene may not agree with the species phylogeny, nor with the phylogeny for another gene. Because the gene copy has a single ancestral copy, barring recombination, the resulting history is a branching tree. Point mutations can cause some of the copies to be imperfect representations of the original, but this process does not compromise the existence of the (gene) tree. Events such as recombination, hybridization, and lateral gene transfer break up the genomic history into many small pieces, each of which has a strictly treelike pattern of descent [4]. Thus, within a species phylogeny, many tangled gene trees can be found, one for each nonrecombined locus in the genome. Yet, in the presence of these processes, the evolutionary history of the species fails to be modeled as a tree; in this case, networks are used to model the species phylogeny. We say that a (species) network "induces" (or, contains) a (gene) tree or alternately, a (gene) tree is *induced by* (or, contained inside) a (species) network. We formalize this concept as follows.

Let N be a network with p network nodes $h_1, h_2, \ldots, h_p$. Further, assume that the two edges incident into h_i are e_{i_1} and e_{i_2}. An *inheritance profile*, IP, for N is a set of size p and which contains exactly one of the two edges e_{i_1} and e_{i_2} for each network nodes h_i. A rooted tree T is *induced* by (or, contained in) a network N if there exists an inheritance profile IP such that T can be obtained from N as follows: for network node h_i, if $e_{i_1} \in IP$, remove edge e_{i_2}; otherwise, remove edge e_{i_1}. (and then apply forced contraction operations to the resultant graph). Biologically, the evolutionary history of a gene within the species network corresponds to a tree T induced by N. Associated with this tree is an inheritance profile IP that decides how to obtain T from N; in this case, we say that IP is a *valid* inheritance profile that induces T. A network N induces (or, contains) a cluster C, $C \subseteq L(N)$, if there exists a tree T such that N induces T and C is a cluster of T.

Proposition 2. *Let N be a nonempty network. Then N induces at least one phylogenetic tree.*

Proof. We show the proposition by induction on the number of leaves in N. The base case (one leaf) is trivial. Assume the hypothesis is true for $|L(N)| = n$, and consider the case where $|L(N)| = n + 1$. Let N_n be the DAG obtained by restricting N to the first n leaves. By the induction hypothesis, there exists a tree T_n that is induced by N_n. By Proposition 1, there exists a path P_{n+1} connecting the root and leaf $n + 1$, and there exists a node v that is the lowest node in both P_{n+1} and the embedding of T_n in N_{n+1}. T is obtained by joining the edges and nodes below v in P_{n+1} and T_n. Since T is connected by construction, if T is not a tree, then there exists a (not necessarily oriented) cycle in T. This contradicts the choice of v as the lowest node in P_{n+1}.

As mentioned before, deciding whether a cluster or a tree are induced by a given network plays a significant role in solving major problems such as network reconstruction, gene tree and species network relationships, exploring the network space in hill-climbing heuristics for solving hard optimization network reconstruction problems, measuring distances and error rates between networks in simulation studies, and many other tasks. We now formalize the two decision problems.

Problem 1. (THE NETWORK-TREE CONTAINMENT PROBLEM)

Input: A phylogenetic network N and a tree T.
Question: Does N contain T?

Problem 2. (THE NETWORK-CLUSTER CONTAINMENT PROBLEM)

Input: A phylogenetic network N and a cluster C.
Question: Does N contain C?

A trivial approach for solving the Network-Cluster Containment Problem is to find "the" lowest common ancestor, x, of C in the network N, and test whether the cluster is contained in the network clade rooted at x. This approach may fail for at least two reasons: (1) x may not be unique in a network, and (2) the network clade rooted at x may contain many of the network nodes of N, in which case the search for a solution would take time that is exponential in the number of network nodes, and hence, probably the network size.

In Section 6 we show that these two problems can be decided in polynomial time. In order to obtain these results, we first introduce the concept of *network decomposition* which forms the basis for our algorithms.

4 Network Decomposition

Before we give the technical details of our algorithms, we describe the network representation we use, which is vital for achieving the running times of the algorithms in the next sections. We assume that a network N is represented using an $n \times n$ adjacency matrix M^N, where n is the number of nodes in the network. We have $M^N[u, v] = 1$ if there is an edge $(u, v) \in E(N)$, and $M^N[u, v] = 0$ otherwise. Using this representation, a forced contraction operation takes $O(1)$ time, and an edge deletion takes $O(1)$ time, as well.

4.1 Preprocessing Networks

An *SH-loop* (speciation-hybridization) is a cycle that contains only network edges, and that consists of two paths p_1 and p_2, such that p_1 and p_2 starting from the same tree node v_0, pass through two sets of network nodes, and end at the same network node v_1. Let $e_1 = (v_0, x)$ and $e_2 = (v_0, y)$ be the two network edges incident from v_0. We break the SH-loop by removing either e_1 or e_2, and applying forced contraction operations to all redundant nodes. We repeat the same process until N is SH-loop-free, i.e., N does not contain any SH-loops.

Preprocessing of a network N can be achieved in polynomial time. To preprocess a network, cycles of network edges in the network need to be detected. A depth-first search achieves this goal. There are at most $\min\{|Tr(N)|, |Nt(N)|\}$ such cycles. Breaking each cycle by removing an edge, followed by a forced contraction operation, takes $O(1)$. Therefore, the overall running time of the preprocessing is $O(|V(N)|$ $(|V(N)| + |E(N)|)) = O(|V(N)|^2) = O(|E(N)|^2)$ (since in binary networks $|V(N)| = \Theta(|E(N)|)$). The following result shows that preprocessing a network N does not change the set of trees induced by N.

Proposition 3. *Let N be a phylogenetic network, and let N' be the network obtained after the preprocessing. Then, $\mathcal{T}(N) = \mathcal{T}(N')$.*

Proof. Since N' is a subgraph of N, we only need to show that each tree T that is induced by N is also induced by N'. As each step in the preprocessing removes one edge from N, it suffices to show this is true if N' and N differ by one edge. Consider an inheritance profile IP that induces T; if each edge in IP is in N', we are done. Otherwise the edge is in an SH-loop of N, pointing from a tree node v_0 to a network node x (x may be suppressed in N'). Let y be the other node immediately below v_0 in the SH-loop, and let v_3 be the lowest network node where the two paths of the SH-loop meet. Let p_1 and p_2 be the two paths in the SH-loop containing edges (v_0, x) and (v_0, y), respectively. Let z and w be the vertices immediately above v_3 in p_1 and p_2, respectively. Notice that since p_1 and p_2 consist of network nodes only (except for v_0), the leaf sets below x, y, and v_3 are identical; call it L. Let L' be the subset of leaves such that the path to root from each leaf in L' passes through (v_0, x) (it also must pass through v_3); such path necessarily passes through (x, y); we only need to consider the case when L' is nonempty. Then IP contains every edge in p_1, and no leaf in L' reaches the root through nodes only in p_2 in T. Hence we can add all edges in p_2 and remove conflicting edges in IP, as they do not lead to leaves from the root. The result is an inheritance profile for N' that also induces T.

4.2 Maximal Clades and Connections

Unless noted otherwise, all networks are SH-loop free. Given a phylogenetic network N, we seek to decompose N into maximal-size clades and disjoint subgraphs of N that connect those clades. To formalize this, we first define some concepts.

Given a node x in network N, we say that a network node y ($y \neq x$) in N is x-*convergent* if any directed path from y to a leaf of N passes through x. Given a maximal clade A of N, and the root a of A, we say that subgraph J of N is the *connection* of A if J is the subgraph obtained by restricting N to all a-convergent nodes and their incident edges.

Lemma 2. *Let A and J be a clade and its connection, respectively, in a network N. Then, when reversing the orientation of its edges, J has a rooted tree topology, where each leaf is a tree node in N and each internal node is a network node in N. Further, the root of J is a lowest network node.*

Proof. Let a be the root of clade A. Assume J has a node v that is a tree node in N. By Proposition 1 and the definitions of J and a, there does not exist a tree node v' that is reachable from v but not from a; hence, there exists a directed path from v to a and that consists of network nodes only. Moreover, the path is unique. If there exist two such paths from v to a then the two paths form an SH-loop. Consider the union of all such paths to a from speciation nodes in J; since those paths are unique, it follows that, when the orientation of the edges in J is reversed, J forms a rooted tree, and the leaves of J are the set of nodes in J that are tree nodes in N. Other properties follow directly.

4.3 Computing the Decomposition

We now define the concept of *T-decomposition* (tree decomposition) of a network.

Definition 2. *A* T-decomposition *of a network N is an ordered set of pairs $D = \{(A_i, J_i)\}_{1 \leq i \leq m}$, where A_i and J_i are a maximal clade and its connection, respectively, in N_i (N_i is obtained by removing the subgraphs A_{i-1} and J_{i-1} from N_{i-1}, except for the leaves of J_{i-1}, i.e., the tree nodes, and applying forced contraction operations to the resultant graph; for the base case, $N_1 = N$); m is the cardinality of the decomposition.*

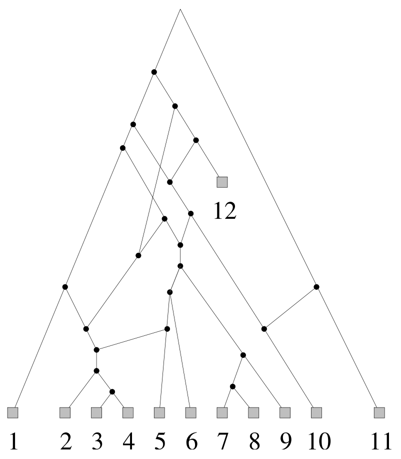
Figure 2(b) shows a T-decomposition of the network in Figure 2(a). Before computing a T-decomposition of a network, the network has to be preprocessed as described in Section 4.1.

Definition 2 leads to an algorithm naturally. To compute A_i in N_i, we find a maximal clade, which is rooted at the tree node immediately below a lowest network node (based on Lemma 1). To compute J_i, we use Lemma 2: reverse the orientation of all edges in N_i, do a depth-first search starting from the root of A_i until tree nodes are encountered. J_i is the search tree together with the tree nodes immediately above (and edges connecting them to J_i). This algorithm can be achieved in $O(|Nt(N)||V(N)|)$ time. To find (A_i, J_i) in N_i, we first find a lowest network node v. To find v, we rank the network nodes (a node has a lower rank if it is closer to the root) using topological sort ($O(|V(N)| + |E(N)|)$ running time). We keep a doubly-linked list to allow constant running time update whenever a network node is deleted from the network, so finding a lowest network node can be achieved at no extra cost. The maximal clade A_i is the clade rooted at the node immediately below v. To find the connection J_i, we start from v and do a depth-first search with all edges in N_i reversed, and stop whenever a tree node is encountered. We then remove A_i and J_i from N_i, apply forced contraction to all redundant nodes encountered in the DFS step for finding J_i. Notice that in the two steps for finding A_i and J_i, we visit each edge at most once in component A_i and J_i. These edges are removed from N_i, and the tree nodes in J_i are suppressed in N_i (which takes constant time per node). The overall running time is thus $O(m(|V(N)| + |E(N)|)) = O(|Nt(N)||V(N)|)$.

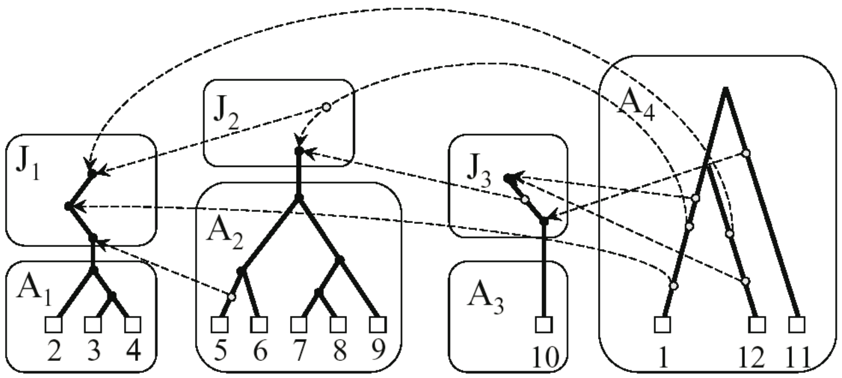
We now show some properties of the T-decomposition.

Proposition 4. *Let $D = \{(A_i, J_i)\}_{1 \leq i \leq m}$ be a T-decomposition of a network N.*

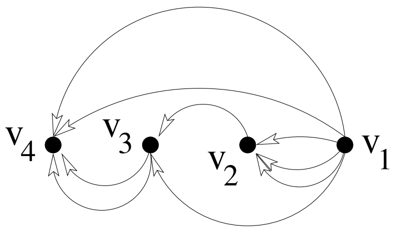
1. *N_m is a phylogenetic tree.*
2. *Each edge in N belongs to exactly one component in the decomposition.*
3. *Each network node belongs to exactly one connection in the decomposition.*



(a)



(b)



(c)

Fig. 2. (a) A phylogenetic network N. (b) A T-decomposition D of N. (c) The dependency digraph $K^{N,D}$.

4. *$\{L(A_i)\}_{1 \le i \le m}$, where $L(A_i)$ denotes the leaf set of A_i, forms a partition of $L(N)$.*
5. *If N is binary, then N_i is binary for all $1 \le i \le m$.*

Proof.

1. By definition N_m does not have network nodes.
2. Observe that by the algorithm, $E(N_{i+1}), E(A_i), E(J_i)$ and the edge connecting A_i and J_i, form a partition of $E(N_i)$, $1 \le i \le m-1$.
3. At each step of the decomposition algorithm, we remove all nodes in the computed connection; N_m does not have network nodes.
4. First notice that $L(A_i)$ is a subset of $L(N_i)$. Assume $L(N_i) - L(N) \neq \emptyset$. Then there exists a node v with outdegree 0 in $L(N_i)$ but not in $L(N)$, which means either $v \in Tr(N)$ or $v \in Nt(N)$. If $v \in Nt(N)$ then all nodes immediately above v except one are in A_i or J_i, which is not possible since v cannot be lower than the lowest network node determining A_i. If $v \in Tr(N)$, then $v \in L(J_i)$, and at least two nodes in J_i are immediately below v, contradicting the fact that N is SH-loop free. Therefore, $L(A_i) \subseteq L(N_i) \subseteq L(N)$. Since A_i is nonempty, it has a lowest node, which must be a leaf in $L(N_i)$. Finally, notice that $L(N_{i+1}) = L(N_i) - L(A_i)$, $1 \le i \le m-1$.
5. Straightforward.

Let (u, v) be a terminal edge (i.e., and edge incident with a leaf) that belongs to connection J_i; v is a tree node in N. If N is binary, then for the three edges incident to v, two belong to the same component, because v is suppressed in the i'th step in the decomposition algorithm. We define $\iota(u, v)$ to be the index of this component. It is straightforward to show that $\iota(u, v) > i$.

Finally, we show that exactly one terminal edge from each component in a T-decomposition is used to induce a tree T.

Lemma 3. *If T is a tree induced by a network N, and D is a T-decomposition of N, then exactly one terminal edge from each connection in D is used to induce that tree.*

Proof. Assume the two terminal edges $e_1 = (x_1, y_1)$ and $e_2 = (x_2, y_2)$ from connection J_i are needed to induce tree T. Further, assume v_i is the root of A_i, and S_i is the leaf set of A_i. Assume, as well, that u_i is the network node such that (u_i, v_i) is an edge in N. Notice that each of the two edges e_1 and e_2 were either a single edge or a path of edges in N.

Exactly one of the two edges, say e_1, reaches S_i in T, whereas the other edge, e_2, reaches a set S' of leaves, where $S_i \cap S' = \emptyset$; otherwise, the underlying undirected graph of T contains a cycle – a contradiction.

It follows that the path p from x_2 to u_i contains a node z, dividing p into two paths p_1 (from x_2 to z) and p_2 (from z to u_i), and such that there is a terminal edge (z, w) in some connection J_j, $i \neq j$, where the set S' of leaves is under w. The node w must be a network node.

Now consider all possible nodes on the path p_2 (between node z and node u_i, exclusive). If there were no such nodes, and since u_i is a network node, it follows that node z has two network node children (w and u_i) – a contradiction to the assumption that N

does not have a tree node whose two children network nodes (notice that node z cannot be a network node, since by definition, a network node has outdegree 1).

Now, assume there were nodes on the path p_2 from z to u_i, and let s be such a node. If s were a tree node, then there exists at least one leaf t in N that is reachable from s through paths that do not contain any network nodes (otherwise, the subnetwork rooted at s contains a tree node whose two children are also network nodes, which is a contradiction). In this case, the edges on the path from x_2 to s cannot be in the connection J_i (those edges would be in maximal clade A_m) – a contradiction to the assumption that edge e_2 is a terminal edge in connection J_i. Therefore, all nodes on the path p_2 from z to u_i are network nodes, and hence tree node z has two children that are network nodes – a contradiction.

Therefore, exactly one terminal edge from each connection is used to induce a tree T in a network N.

4.4 Dependency Graphs

Given a network N and its T-decomposition D, we define the *dependency digraph* $K^{N,D}$ to facilitate our algorithm design.

Definition 3. *Given a network N and its T-decomposition $D = \{(A_i, J_i)\}_{1 \leq i \leq m}$, the dependency graph is a directed multigraph $K^{N,D}$, where node v_i in $K^{N,D}$ corresponds to the pair (A_i, J_i) in D, and edge (v_i, v_j) $(i > j)$ in $K^{N,D}$ corresponds to a terminal edge connecting J_j and J_i in N.*

Figure 2(c) shows a dependency graph of the network and T-decomposition given in Figures 2(a) and 2(b). In other words, $K^{N,D}$ is the graph resulting from replacing each component (A_i, J_i) in D by a single node v_i, and hence, $K^{N,D}$ is necessarily connected. If $K^{N,D}$ had a cycle, then N would be cyclic. Therefore, we have the following result.

Proposition 5. *The dependency graph $K^{N,D}$ is connected and acyclic. Moreover, (v_i, v_j) is an edge in $K^{N,D}$ only if $i > j$.*

At the end of the decomposition process, we keep a matrix that shows which component each edge belongs to. So querying which component an edge (u, v) belongs to, as well as computing the value of $\iota(u, v)$, take $O(1)$ time. Thus, computing the dependency graph $K^{N,D}$ takes $O(m|E(N)|) = O(|Nt(N)||V(N)|)$ time. Further, in $K^{N,D}$, we keep track of the correspondence between edges of N and edges of $K^{N,D}$.

5 Reduced Inheritance Profiles and the Cluster Lemma

Given a T-decomposition D of cardinality m, a *reduced inheritance profile* is a set of size m that contains exactly one terminal edge per connection in the decomposition. We only keep the terminal edges because all inheritance profiles having the same set of terminal edges necessarily induce the same tree. A reduced inheritance profile extends into an inheritance profile in a straightforward manner, as no edges in the reduced inheritance profile are incident with the same network node. We say that a reduced inheritance profile is *valid* if it induces a tree. The following results show the correspondence between inheritance profiles and reduced inheritance profiles.

Proposition 6. *Let D be a T-decomposition of a network N. Then,*

1. *For each valid reduced inheritance profile IP there exists a valid inheritance profile IP' that contains IP and induces the same tree.*
2. *For each valid inheritance profile IP' there exists a unique valid reduced inheritance profile IP that induces the same tree.*

Proof.

1. To compute the inheritance profile IP', for each connection J_i we take the unique path from the terminal edge e_i to the root of J_i; for each network node v_i in J_i, we choose the edge on the path as the value of x_i in the inheritance profile. For other nodes we make choices arbitrarily. Since no leaf can be reached from the root through nodes not on the path, choosing different edges incident with these nodes in the profile do not affect the tree topology.
2. Given a valid inheritance profile IP', for each connection J in the D there is exactly one path connecting a terminal edge in J and the root of J. We retain this edge and drop all other terminal edges in $J \cap IP'$. We obtain IP by repeating the same process for all connections.

The dependency graph can be seen as a compact representation, mainly for reduced inheritance profiles.

Lemma 4. *Let D be a T-decomposition of a network N, $K^{N,D}$ be the dependency graph, and IP be a valid reduced inheritance profile. Then, $K^{N,D}$, restricted to the edges in IP, forms a tree.*

We are now in position to show the correlation between clusters and a T-decomposition of a network – a result that forms the basis for our algorithms.

Lemma 5. (Cluster Lemma) *Let $D = \{(A_i, J_i)\}_{1 \leq i \leq m}$ be a T-decomposition of a network N. Each cluster C induced by N can be written as $C = \cup_j C_j$, where each C_j is an element of $\{L(A_i) : 1 \leq i \leq m\}$, except for at most one of the C_j's, which may be a proper subset of an element of $\{L(A_i) : 1 \leq i \leq m\}$.*

Proof. The lemma is trivially true for $|C| = 1$. Assume $|C| > 1$, and let T be a phylogenetic tree that contains C. Let IP be a reduced inheritance profile that induces T in N. Construct the tree T' in the dependency graph according to Lemma 4. Let v in N be a lowest common ancestor of all leaves in C. Node v must be a tree node (if it were a network node, then the node below v would also be a common ancestor, contradicting the fact that v is a lowest common ancestor). Let $A_{i_1}, A_{i_2}, \ldots, A_{i_k}$ be k maximal clades in D and such that each of them has nonempty intersection with C. Let e_{i_j} be the terminal edge in J_{i_j} that is an element of IP. There are two cases: (1) v is in a maximal clade A_l but not in any connection in D. Let $L(v)$ be the cluster in A_l below v; or (2) v is in a connection J_q in D. Then there is a terminal edge $(u, v) \in J_q$. Let $l = \iota(u, v)$, and let $L(v)$ be the cluster in A_l below v. In both cases, $L(v)$ is nonempty, and if $L(v) \neq L(A_l)$, then $L(v)$ is the C_j that is a proper subset of an element of $\{L(A_i) : 1 \leq i \leq m\}$. Furthermore, for any A_{i_j}, $i_j \neq l$, $1 \leq j \leq k$, $i_j \neq m$, any path from v to a leaf in $L(A_{i_j})$ passes through e_{i_j} by Lemma 6; thus, $L(A_{i_j}) \subseteq C$. Any leaf in $L(A_{i_j}) \setminus L(A_x)$ can be reached from v through at least one terminal edge in IP, so by Lemma 5, $l = \max\{i_1, \ldots, i_k\}$.

Corollary 1. *Let $D = \{(A_i, J_i)\}_{1 \le i \le m}$ be a T-decomposition of a network N, IP be a reduced inheritance profile, and C be a cluster. Then, when restricted to the nodes whose corresponding maximal clades have nonempty intersection with C and to the edges in IP, the dependency graph $K^{N,D}$ forms a tree. Further, the root of that tree has the highest index.*

Proof. Using the collapsing argument in the proof of Lemma 4, and Proposition 5.

Corollary 1 gives an algorithm for computing the component that contains the node which "determines" a cluster C: compute the corresponding subtree in the dependency graph and return the component corresponding to the root. If one of the C_j's in the Cluster Lemma is a proper subset of an element in $\{L(A_i) : 1 \le i \le m\}$, the component that contains that C_j must be the root, based on the proof of the Cluster Lemma.

6 Polynomial-Time Algorithms for the Decision Problems

6.1 Deciding the Network-Cluster Containment Problem

We are finally in a position to describe a polynomial-time algorithm for deciding the Network-Cluster Containment Problem. The algorithm is given in Figure 3. Let $D = \{A_i, J_i\}_{1 \le i \le m}$ be a T-decomposition of a network N, and let $C \subset L(N)$ be a cluster. We define the set $\psi(C) = \{i : 1 \le i \le m \text{ and } L(A_i) \subseteq C\}$. The basic idea is to compute a set E_C of edges that are *incompatible* with C, i.e., edges that cannot co-exist with C in the same tree induced by N.

```
Algorithm TestCinN($N$,$C$)

1. Compute a T-decomposition $D = ((A_1, J_1), \ldots, (A_m, J_m = \emptyset))$.
2. Test if $C$ can be decomposed into the following form: $\bigcup_{i \in \psi(C)} L(A_i) \cup L'$, where $L' = \emptyset$
   or $L' \subset L(A_l)$ for some $l$. If not, return NO. If $L' = \emptyset$ then let $l = \max_{i \in \psi(C)} i$.
3. Partition $V = V(K^{N,D})$ into two sets: $V_C = \{v_i | i \in \psi(C)\}$ and $\overline{V_C} = V - V_C$.
   Compute the set $E_C = \{e_{ij} = (v_i, v_j) | e_{ij} \in E(K^{N,D}), v_i \in \overline{V_C}, v_j \in V_C, j \ne l\}$.
4. If $L' \ne \emptyset$, test if $L'$ is a cluster of $A_l$. If not, return NO; otherwise:
   (a) Let $v'$ be the root of the clade whose leaf set is $L'$.
   (b) For each terminal edge $(u, v)$ in $A_i$, for some $i \in \psi(C)$ and $\iota(u, v) = l$, (edge
       $(u, v)$ connects the $i$'th component to the $l$'th component), add $(u, v)$ to $E_C$ if $u$ is
       not a descendant of $v'$ in $N_i$.
5. Remove all terminal edges in $N$ and that correspond to edges in $E_C$ (and apply forced
   contraction operations); let the result be $N_C$. If $N_C$ is connected, return YES. Other-
   wise, return NO.
```

Fig. 3. Algorithm **TestCinN** for deciding the NETWORK-CLUSTER CONTAINMENT PROBLEM

Theorem 2. *Algorithm TestCinN(N,C) decides the Network-Cluster Containment Problem in $O(|V(N)|^2)$ time.*

Proof. We first show the correctness of the algorithm. Step 3 in the algorithm is well defined: if $(v_i, v_j) \in E(N)$ then $i > j$ by Proposition 5.

Assume N_C is connected. It is easy to show N_C is still a network (note that we only remove terminal edges). By Proposition 2, N_C induces a tree T'; note that N also induces T'. There exists a reduced inheritance profile IP that induces T' in N_C. Let e be the terminal edge in $IP \cap E(J_l)$. We now show T' contains C. For any terminal edge e', assume it is from component i. Assume $i \in \psi(C)$. If $e' \in IP$, then e' is below e in T', otherwise e' is in E_C. So the cluster below e in T contains each $L(A_i)$, $i \in \psi(C)$. Now assume $i \notin \psi(C)$ and $i \neq l$. If $e' \in IP$, then $\iota(e') \notin \psi(C) \cup \{l\}$. Therefore $\bigcup_{i \in \psi(C)} L(A_i) \subseteq C$, and for each $i \notin \psi(C) \cup \{l\}$, $L(A_i) \cap C = \emptyset$. Finally, if L' is not empty, notice that for any terminal edge e' from component i, $i \in \psi(C)$, if $e' \in IP$, then e' is lower than v' in T'. Therefore the cluster determined by v' in T' is exactly $\bigcup_{i \in \psi(C)} L(A_i) \cup L'$.

Now, assume N induces tree T that contains cluster C; we show that N_C is connected. Let $G = K^{N,D}$ be the dependency graph, and let G_C be the graph obtained by removing edges in E_C from $K^{N,D}$. Let IP be a reduced inheritance profile that induces T. If none of the edges in T is in E_C then G_C (and hence N_C) is connected. To see this is true, assume otherwise; then there exists $(u, v) \in IP \cap E_C$. Either (u, v) is an edge in step 3 or step 4(b). The first case contradicts Corollary 1, since the edge connects some vertex v_i, $i \in \psi$ in G_C to a vertex v_j above v_l. In the second case, the cluster determined by u in A_l properly contains L'. If $(u, v) \in J_i$ (in which case $L(A_i) \subseteq C$), then for any vertex below u in T', its corresponding cluster does not contain $L(A_i)$.

We now analyze the running time of the algorithm. (Recall that N is binary.) Step 1 takes $O(|Nt(N)||V(N)|)$ time. Steps 2 and 3 take $O(|L(N)|)$ time if we keep track of which component each leaf in $L(N)$ belongs to, when we compute D. In step 4, first notice the number of terminal edges is bounded by $2|Nt(N)|$; for each terminal edge, testing its membership in E_C takes constant time. In Step 5, testing if L' is a cluster in A_i takes $O(|L(A_i)|) = O(|L(N)|)$ time by doing a depth-first search; we can also find v in 5(a) at the same time. Testing for each (u, v) if it should be added to E_C takes constant time, and there are $O(|Nt(N)|)$ of them. Finally, in Step 6, removing E_C from N takes $O(|E_C|) = O(|Nt(N)|)$ time; testing the connectedness of N_C can be achieved by a depth-first search. The overall running time is $O(|Nt(N)||V(N)|)$=$O(|V(N)|^2)$. If the T-decomposition is given, the running time is $O(|V(N)| + |E(N)|) = O(|V(N)|)$.

Based on the proof of Theorem 2, we have the following result.

Corollary 2. *Given any network N, a phylogenetic tree T, and a cluster C in T, N induces T if and only if N_C induces T.*

Proof. Let IP be an inheritance profile that induces T. Notice that no edge of IP is in E_C. Since IP induces T in N, it also induces T in N_C.

6.2 Deciding the Network-Tree Containment Problem

Using algorithm TestCinN(N,C), Figure 4 describes our polynomial-time algorithm for deciding the Network-Tree Containment Problem.

```
Algorithm TestTinN($N$,$T$)

1. Compute a T-decomposition $D = ((A_1, J_1), \ldots, (A_m, J_m = \emptyset))$.
2. For each nontrivial cluster $C$ in $T$ ($C \neq L(N)$ and $|C| > 1$), call TestCinN($N$,$C$);
   update $N$ by removing $E_C$ from $N$.
3. If $N$ is connected, return YES; otherwise, return NO.
```

Fig. 4. Algorithm **TestTinN** for deciding the NETWORK-TREE CONTAINMENT PROBLEM

Theorem 3. *Algorithm TestTinN(N,T) decides the Network-Tree Containment Problem in $O(|V(N)||L(N)|)$ time.*

Proof. We denote by N' the network obtained at the end of Step 2. We want to show that N induces T if and only if N' is connected. By Corollary 2, after each iteration in Step 2 of the algorithm, the new N still induces T; so N' is connected.

Assume N' is connected. Then N' induces a tree T'. Let IP be an inheritance profile in N' that induces T'. It suffices to show $T' = T$ since N' is a subnetwork of N. Now consider any nontrivial cluster C in T. C can be decomposed using the Cluster Lemma. Let $l = \max \psi(C)$, and let $e = P \cap E(J_l)$. Since we call TestCinN() with C as the input cluster, every leaf in C in T' is below e. If L' in the TestCinN algorithm is empty, the cluster below e in T' is C. If L' is not empty, L' is a cluster in A_l; in this case let v' be the lowest common ancestor in L'. The cluster in T' determined by v' is C.

We now analyze the running time of the algorithm. (Recall that N is binary.) We only need to compute T-decomposition once, which takes $O(|Nt(N)||V(N)|)$ time. Each iteration in Step 2 takes $O(|V(N)|)$ time, and the number of clusters in T is $O(|L(N)|)$. The final step takes $O(|V(N)| + |E(N)|) = O(|V(N)|)$ time. The overall time is therefore $O(|V(N)|^2)$.

7 Conclusion and Future work

Phylogenetic networks are the appropriate model for evolutionary histories in the presence of reticulation events. Very little is known about their combinatorial properties, and many problems are still open in this domain. In this paper, we presented polynomial-time algorithms for two major problems, namely (1) deciding whether a tree is induced by a network, and (2) deciding whether a cluster is induced by a network. Those two algorithms are based on a novel network decomposition that we introduced. Directions for future research include enumerating the numbers of trees and clusters induced by a network, efficient techniques for network space traversal, and accurate reconstruction of networks from sets of clusters and trees.

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] J. Felsenstein. *Inferring Phylogenies*. Sinauer Associates, Inc., Sunderland, MA, 2003.

[3] D.H. Huson. SplitsTree: A program for analyzing and visualizing evolutionary data. *Bioinformatics*, 14(1):68–73, 1998.

[4] W.P. Maddison. Gene trees in species trees. *Systematic Biology*, 46(3):523–536, 1997.

[5] B.M.E. Moret, L. Nakhleh, T. Warnow, C.R. Linder, A. Tholse, A. Padolina, J. Sun, and R. Timme. Phylogenetic networks: modeling, reconstructibility, and accuracy. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1(1):13–23, 2004.

[6] L. Nakhleh, T. Warnow, and C.R. Linder. Reconstructing reticulate evolution in species – theory and practice. In *Proceedings of the Eighth Annual International Conference on Research in Computational Molecular Biology (RECOMB 2004)*, pages 337–346, 2004.

[7] D.L. Swofford, G.J. Olsen, P.J. Waddell, and D.M. Hillis. Phylogenetic inference. In D.M. Hillis, B.K. Mable, and C. Moritz, editors, *Molecular Systematics*, pages 407–514. Sinauer Assoc., Sunderland, Mass., 1996.