BIOINFORMATICS ORIGINAL PAPER

Vol. 29 no. 18 2013, pages 2269–2276
doi:10.1093/bioinformatics/btt378

Phylogenetics

Advance Access publication June 29, 2013

LNETWORK: an efficient and effective method for constructing phylogenetic networks

Juan Wang, Maozu Guo*, Xiaoyan Liu, Yang Liu, Chunyu Wang, Linlin Xing and Kai Che

Department of Computer Science and Engineering, Harbin Institute of Technology, Harbin, Heilongjiang 150001, China

Associate Editor: David Posada

ABSTRACT

Motivation: The evolutionary history of species is traditionally represented with a rooted phylogenetic tree. Each tree comprises a set of clusters, i.e. subsets of the species that are descended from a common ancestor. When rooted phylogenetic trees are built from several different datasets (e.g. from different genes), the clusters are often conflicting. These conflicting clusters cannot be expressed as a simple phylogenetic tree; however, they can be expressed in a phylogenetic network. Phylogenetic networks are a generalization of phylogenetic trees that can account for processes such as hybridization, horizontal gene transfer and recombination, which are difficult to represent in standard tree-like models of evolutionary histories. There is currently a large body of research aimed at developing appropriate methods for constructing phylogenetic networks from cluster sets. The CASS algorithm can construct a much simpler network than other available methods, but is extremely slow for large datasets or for datasets that need lots of reticulate nodes. The networks constructed by CASS are also greatly dependent on the order of input data, i.e. it generally derives different phylogenetic networks for the same dataset when different input orders are used.

Results: In this study, we introduce an improved CASS algorithm, LNETWORK, which can construct a phylogenetic network for a given set of clusters. We show that LNETWORK is significantly faster than CASS and effectively weakens the influence of input data order. Moreover, we show that LNETWORK can construct a much simpler network than most of the other available methods.

Availability: LNETWORK has been built as a Java software package and is freely available at http://nclab.hit.edu.cn/~wangjuan/LNETWORK/.

Contact: maozuguo@hit.edu.cn

Supplementary information: Supplementary data are available at *Bioinformatics* online.

Received on February 4, 2013; revised on June 24, 2013; accepted on June 26, 2013

1 INTRODUCTION

Phylogenetic networks are a generalization of phylogenetic trees; they can represent non–tree-like evolutionary histories caused by processes such as hybridization, horizontal gene transfer and recombination (Doolittle, 1999; Rieseberg, 1997). Developing appropriate methods for inferring phylogenetic networks has been identified as an important research area (Huson, 2005; Linder *et al.*, 2004), and there have been many recent studies on this topic (Huson and Scornavacca, 2010; Huson *et al.*, 2011). Phylogenetic networks can be typologically classified into unrooted and rooted networks, and functionally classified into implicit and explicit networks (Huson and Scornavacca, 2010). Implicit networks can be used to represent conflicting patterns that may be the result of a variety of causes, such as model misspecification (Huson and Bryant, 2006). Explicit networks can capture biological processes such as hybridization (Linder and Rieseberg, 2004; Maddison, 1997), recombination (Gusfield and Bansal, 2005; Song and Hein, 2005) and horizontal gene transfer (Nakhleh, 2010). Explicit networks are usually rooted, as evolution is inherently directed. However, rooted phylogenetic networks may also be implicit networks, depending on how they are constructed and interpreted (Huson and Scornavacca, 2010).

The program Dendroscope (Huson and Scornavacca, 2012) is used to compute rooted phylogenetic networks. It combines a number of methods used to compute implicit networks, such as the cluster network method (Huson and Rupp, 2008), with methods used for computing explicit networks, such as the hybridization network method (Huson and Scornavacca, 2012). The program SplitsTree4 (Huson and Bryant, 2006) is designed to compute implicit networks. It uses a variety of methods, including the Z-closure super network method (Huson *et al.*, 2004). Woolley *et al.* (2008) used computer simulations to compare the performance of most of the unrooted implicit network approaches, i.e. they compared the simulated trees with all of the trees embedded in the constructed network.

Phylogenetic networks are an important part of biological research. Wang *et al.* (2012) used the Dendroscope program to confirm the hypothesis that indehiscent sporangia promote the establishment of a persistent population in different regions of the Qinghai-Tibetan Plateau. Schwarzer *et al.* (2011) analyzed Amplified Fragment Length Polymorphism (AFLP) loci using the Neighbor-net method and revealed three main clusters and several smaller subclusters. Kelly *et al.* (2010) inferred super networks (Whitfield *et al.*, 2008) from several incongruent gene trees, and uncovered new evidence for a complex history of reticulate evolution in Nicotiana.

Our study develops a method for constructing rooted implicit networks. In the following, unless otherwise specified, we refer to rooted phylogenetic networks as simply networks.

Let $\mathcal{X}$ be a set of taxa. A rooted phylogenetic tree T on $\mathcal{X}$ represents a cluster C (a proper subset of $\mathcal{X}$), if there is an edge e in T such that the set of taxa below e equals C. Each rooted phylogenetic tree T is uniquely defined by the set of clusters that it represents (Huson *et al.*, 2011). Given a network N and a cluster C, when switching an incoming edge on and all of the

*To whom correspondence should be addressed.

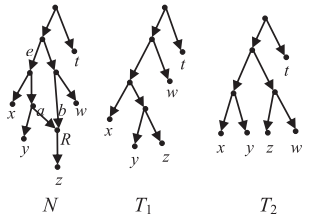

Fig. 1. The network N is on $\{x, y, z, w, t\}$. Consider the cluster $C = \{x,y\}$. When switching the edge b on and a off, the set of leaves reachable from e is equal to C, so N represents the cluster C in the soft-wired sense. However, the network does not represent the cluster C in the hard-wired sense, as there is no tree edge for which the set of taxa below the edge equals C. The tree edge e represents the cluster $\{x,y,z\}$ in the hard-wired sense. The rooted phylogenetic trees T_1 and T_2 are both trees represented by N

others off for each reticulate node in N, if there is a tree edge $e = (u, v)$ (indegree of v at most 1) in N such that the set of leaves reachable from e equals C, we say that N represents C *in the soft-wired sense*. Alternatively, if there is a tree edge e in N such that the set of taxa below e equals C, we say that N represents C *in the hard-wired sense*. Given a network N and a phylogenetic tree T, when switching an incoming edge on and all of the others off for each reticulate node in N, if N equals T, then we say that N represents T (see Fig. 1).

Phylogenetic trees are commonly constructed from different datasets, e.g. from different genes. Each constructed phylogenetic tree carries important evolutionary information, and to reconstruct complete evolutionary histories of all of the relevant taxa, we would preferably like to use all of the clusters represented in the various constructed phylogenetic trees. However, such a set of clusters cannot usually be expressed as a single phylogenetic tree. Therefore, it is necessary to construct a phylogenetic network to express the set of clusters. Several recent publications have studied this problem (Huson and Rupp, 2008; Huson *et al.*, 2009; van Iersel *et al.*, 2010). Note that individual gene trees will probably contain a mixture of correct and incorrect clusters; however, in our article, we assume that all clusters contained by these trees are correct.

The rooted phylogenetic network can, in theory, be used to explicitly describe evolution in the presence of reticulate events such as hybridization, horizontal gene transfer and recombination. However, in biology, those reticulate events are rare (Linder *et al.*, 2004; Nakhleh, 2010). Consequently, it is reasonable that the desired network should minimize the number of reticulate nodes contained in the network. The Cass algorithm (van Iersel *et al.*, 2010) can construct a network with fewer reticulate nodes than any other available method, but it is not suitable when the input consists of large datasets or datasets that need many reticulate nodes. Moreover, the networks constructed by Cass are dependent on the order of input data, i.e. it generally constructs different phylogenetic networks when the order of the input data is different.

In the following, we use the redundant clusters in a network to refer to the clusters that remain after removing the input clusters from all of the clusters represented by the network. In the context of phylogenetic analysis, the clusters represent putative monophyletic groups of related species. Biologically, the ideal situation would be that all of the clusters represented in the input trees would be represented in the constructed network, and no others. Therefore, we invoke the parsimony principle to argue that the best constructed network for representing a set of clusters in the soft-wired sense is one that minimizes the number of redundant clusters; this is based on the premise that there will be as few reticulate nodes as possible. This new criterion brings the model closer to the above ideal situation.

To achieve this aim, we introduce an improved Cass algorithm called Lnetwork, which constructs a network that represents the given set of clusters in the soft-wired sense. Lnetwork is significantly faster than Cass and effectively weakens the influence of the input data order. Moreover, Lnetwork can construct a network that represents fewer redundant clusters and has fewer reticulate nodes than most other programs. Furthermore, the trees represented by the networks constructed by Lnetwork are closer to the input trees than those represented by the networks produced by the other programs we consider.

2 PRELIMINARIES

A rooted phylogenetic network N on $\mathcal{X}$ is a directed acyclic graph with a single node of indegree 0, called the root, and the leaves bijectively labeled as $\mathcal{X}$. The indegree of a node v is denoted by $\delta^-(v)$. Any node v where $\delta^-(v) > 1$ is called a *reticulate node* or a *reticulation*, and all of the other nodes are called *tree nodes*. Any edge leading to a reticulate node is called a *reticulate edge*, and all of the other edges are called *tree edges*. The *reticulation number* of a network $N = (V, E)$ is defined as $\sum_{v \in V:\delta^-(v)>0} (\delta^-(v) - 1) = |E| - |V| + 1$.

The *number of reticulate nodes* in a network $N = (V, E)$, denoted by r_N, is $r_N = |\{v : v \in V, \delta^-(v) > 1\}|$.

The reticulation number of a network is no less than the number of reticulate nodes in the network. The number of reticulate nodes is used as a variable in the following Theorem 2, whereas the reticulation number is used to describe the complexity of a network.

A graph is called *connected* if every pair of nodes is connected by some (undirected) path. A *cut node* or *cut edge* is a node or edge (except for leaves or edges leading to leaves), respectively, whose removal will disconnect the graph. A graph is *biconnected* if it contains no cut nodes. A *biconnected component* of a graph is a maximal biconnected subgraph.

A network is said to be a *level k network* if each biconnected component has a reticulation number of at most k. A level k network is called a *simple level* $\leq k$ *network* if it has no cut nodes. A network is *binary* if each reticulate node has indegree 2 and outdegree 1, and each tree node that is not a leaf has outdegree 2. A network has *tree-child* property if every internal node has at least one child that is a tree node. A network $N = (V, E)$ is *time consistent* if it has a time assignment, i.e. a mapping $\tau : V \to \mathrm{IN}$, such that $\tau(u) < \tau(v)$ for a tree edge (u, v) and $\tau(u) = \tau(v)$ for a reticulate edge (u, v).

Let $\mathcal{X}$ be a set of taxa. A *cluster* C on $\mathcal{X}$ is any subset of $\mathcal{X}$, excluding both the empty set $\emptyset$ and the full set $\mathcal{X}$. Two different clusters, C_1 and C_2, on $\mathcal{X}$ are called *compatible* if $C_1 \cap C_2 = \emptyset$ or $C_1 \subset C_2$ or $C_2 \subset C_1$; otherwise they are called *incompatible*. A cluster set $\mathcal{C}$ on $\mathcal{X}$ is called *compatible* if and only if $\mathcal{C}$ is pairwise compatible; otherwise it is called *incompatible*. The *incompatibility graph* $IG(\mathcal{C}) = (V, E)$ for $\mathcal{C}$ is an undirected graph with node

set $V = \mathcal{C}$ and edge set E, such that any two clusters $C_1, C_2 \in \mathcal{C}$ are connected by an edge if and only if they are incompatible. For each pair of incompatible clusters, $C_1, C_2 \in \mathcal{C}$, an *incompatibility statement* is defined by the three terms $C_1 \backslash C_2$, $C_1 \cap C_2$ and $C_2 \backslash C_1$.

Let $\mathcal{C}$ be a set of clusters on $\mathcal{X}$. Given a subset S of $\mathcal{X}$, the restriction of $\mathcal{C}$ to S, denoted by $\mathcal{C}|_S$, is the result of removing all of the elements in $\mathcal{X} \backslash S$ from each cluster in $\mathcal{C}$. S with $|S| > 1$ is called an *ST-set* (strict tree set) with respect to $\mathcal{C}$ if S and any one cluster $C \in \mathcal{C}$ are compatible, and $\mathcal{C}|_S$ is also compatible. An ST-set S is *maximal* if there are no other ST-sets containing S except itself. There is a subtree with respect to a maximal ST-set S, which is constructed for the cluster set $\{C | C \in \mathcal{C}, C \subset S\} \cup S$.

An ancestor u of a node v in a network $N = (V, E)$ is called a *stable ancestor* if each path from the root to v passes through u; otherwise it is called an *unstable ancestor*. Conversely, v is called a *stable descendant* of u if u is a stable ancestor of v, and v is called an *unstable descendant* of u if u is an unstable ancestor of v. The *tripartition* associated with a tree edge $e = (u, v)$ in N (N is on $\mathcal{X}$) is $\theta(e) = (A(e), B(e), C(e))$, where $A(e) = \{x \in \mathcal{X} | x$ is a stable descendant of $v\}$, $B(e) = \{x \in \mathcal{X} | x$ is an unstable descendant of $v\}$ and $C(e) = \{x \in \mathcal{X} | x$ is not a descendant of $v\}$. Here we use $\Theta(N)$ to denote all tripartitions in N. The *tripartition distance* between two networks N_1 and N_2 is $|\Theta(N_1) \Delta \Theta(N_2)|/2$, where Δ denotes the symmetric difference. The tripartition distance is a metric on the set of all of the rooted phylogenetic networks that have the tree-child property and are time consistent (Cardona, 2009).

2.1 CASS algorithm

Consider a set of taxa $\mathcal{X}$ and a set of clusters $\mathcal{C}$ on $\mathcal{X}$. The CASS algorithm includes several steps to construct a network for $\mathcal{C}$ as follows. First, it finds all of the non-trivial biconnected components $\mathcal{C}_1, \cdots, \mathcal{C}_p$ of the incompatibility graph $IG(\mathcal{C})$. For each $\mathcal{C}_i$, $i \in \{1, \cdots, p\}$, let $\mathcal{C}'_i$ be the result of collapsing all of the maximal ST-sets for $\mathcal{C}_i$. As it does not know the level number of the network for $\mathcal{C}'_i$, it first sets $k = 0$. Then CASS(k) seeks to construct a simple level $\leq k$ network, i.e. N_i, for $\mathcal{C}'_i$. If there exists such a network, it outputs the resulting network and halts; otherwise it lets $k = k + 1$ and continues. This is pivotal for constructing networks. Second, CASS constructs a unique phylogenetic tree T for the cluster set $\mathcal{C}''$, which contains the clusters that remain after removing all of the clusters in $\mathcal{C}_1, \cdots, \mathcal{C}_p$ from $\mathcal{C}$, and all of the clusters that consist of all of the taxa of each $\mathcal{C}_i$, $i \in \{1, \cdots, p\}$, and all of the maximal ST-sets of $\mathcal{C}_1, \cdots, \mathcal{C}_p$. Third, it integrates T with all of the simple level $\leq k$ networks, $N_1, \cdots, N_p$, to obtain the resulting level k network, using ancestor nodes displacement.

CASS(k) constructs the simple level $\leq k$ network for the non-trivial biconnected components $\mathcal{C}'_i$ as follows. It loops over all taxa x. For each choice of x, CASS(k) tries x as the label of the leaf below the reticulate node, and thus CASS(k) removes it from each cluster. It subsequently collapses all of the maximal ST-sets of the resulting cluster set. CASS(k) repeats this step k times. If, after this process, the resulting clusters $\mathcal{C}_0$ are compatible, then CASS(k) creates a network consisting of a phylogenetic tree for $\mathcal{C}_0$; otherwise CASS(k) returns an empty network. Further, CASS(k) 'expands', i.e. it replaces each leaf labeled with a maximal ST-set S with the subtree of S. Then, CASS(k) adds a new leaf below the new reticulate node and labels it as the latest removed taxon. As it does not know where to add the new reticulate node, CASS(k) tries to add the reticulate node below each pair of edges. These steps are repeated k times. For the constructed simple level $\leq k$ network, CASS(k) checks whether it represents all of the input clusters. If it does, then the algorithm returns the resulting network; otherwise it returns an empty network.

As we have seen, when constructing the simple level $\leq k$ network for the non-trivial biconnected components $\mathcal{C}'_i$, CASS first sets $k = 0$, then runs CASS(k). If CASS(k) returns a network that is not an empty network, it halts; otherwise it lets $k = k + 1$ and continues. Assuming that there is a set of clusters represented by a level l network, we can imagine that CASS must run CASS(0), CASS(1), $\cdots$, CASS(l) until it has constructed the level l network. This results in an excessive consumption time for a high level l. When running CASS(k), the label of the node below the reticulate node is uncertain, so each taxon, $x \in \mathcal{X}$, must be tried repeatedly until the network N represents the complete set of clusters. Such uncertainty also results in constructed networks that are greatly dependent on the order of input data. However, the following Theorem 2 shows that there is a network for a given set of clusters, and the network has certain labels below its reticulate nodes. We use Theorem 2 to develop the LNETWORK algorithm. Our experiments show LNETWORK is faster than CASS and weakens the influence of input data order.

2.2 Seed-growing algorithm

The seed-growing algorithm (Huson *et al.*, 2009) has been used to solve the Restricted Maximum Compatible Subset (RMCS) problem for an incompatible cluster set $\mathcal{C}$, i.e. it finds a minimum set of taxa $R \subset \mathcal{X}$, such that $C|_{\mathcal{X} \backslash R}$ is compatible.

Suppose that L is the list of all of the incompatibility statements for $\mathcal{C}$. The algorithm maintains a set of candidate solutions S, called seeds. A seed s^* resolves an incompatibility statement X, Y, Z, if $X \subseteq s^*$ or $Y \subseteq s^*$ or $Z \subseteq s^*$. Each seed $s \in S$ is labeled by the number rank(s) of incompatibility statements that it has been shown to resolve in succession, starting from the beginning of the list L. Initially, the three parts of the first incompatibility statement are chosen as three seeds, and we set rank(s)= 1 for each seed s. The algorithm chooses a seed s^* that has a minimum size among all of the seeds. If rank(s^*) = $|L|$, then s^* is an optimal solution and the algorithm halts. Otherwise, if s^* resolves the (rank(s^*)+1)-th incompatibility statement X, Y, Z, we increment rank(s^*) by 1; otherwise, we define three new seeds $s_1 = s^* \cup X$, $s_2 = s^* \cup Y$ and $s_3 = s^* \cup Z$, with rand(s_1) = rand(s_2) = rand(s_3) = rand(s^*) + 1, and then we add these to S and remove s^* from S. The above steps repeat until the algorithm halts.

Let A be a solution found by the seed-growing algorithm. We then add A to $\mathcal{A}$, where $\mathcal{A}$ is a solution set of the RMCS problem for $\mathcal{C}$. Further, for each seed $s \in S$ with $|s| = |A|$, if s is also a solution of the RMCS problem for $\mathcal{C}$, we add s to $\mathcal{A}$. $\mathcal{A}$ will be applied in the context of LNETWORK.

3 DEFINITIONS, LEMMAS AND THE MAIN THEOREM

The *incompatibility degree of a cluster set* $\mathcal{C}$, denoted by $d(\mathcal{C})$, is equal to the number of edges in the incompatibility graph $IG(\mathcal{C})$. The *incompatibility degree of a taxon* $x \in \mathcal{X}$ with respect to $\mathcal{C}$,

denoted by $d(x)$, is $d(x) = d(\mathcal{C}) - d(\mathcal{C}|_{\mathcal{X}\setminus\{x\}})$. The *frequency of a taxon* $x \in \mathcal{X}$ with respect to $\mathcal{C}$, denoted by $f(x)$, is $f(x) = |\{C \in \mathcal{C} | x \in C\}|$. A cluster $C \in \mathcal{C}$ is called a *minimal cluster* if there is no cluster $C_0 \in \mathcal{C}$ such that $C_0 \subsetneq C$. For two clusters, C_1 and C_2, we say $C_1 < C_2$ if $C_1 \subsetneq C_2$. Let C_0 be a cluster on $\mathcal{X}$. We say $\mathcal{C} < C_0$ if $C < C_0$ for any cluster $C \in \mathcal{C}$.

Given a set of seeds S, for any two seeds, $s_1, s_2 \in S$, with $|s_1| = |s_2|$, we define the *order* of s_1 and s_2 as follows. Let $s_1 = \{a_1, a_2, \cdots, a_n\}$ and $s_2 = \{b_1, b_2, \cdots, b_n\}$. We say $s_1 > s_2$ if $d(a_i) > d(b_i)$, where a_i and b_i are the first pair of taxa with different incompatibility degrees, or if for all of $i \in \{1, 2, \cdots, n\}$, $d(a_i) = d(b_i)$ and $f(a_j) > f(b_j)$, where a_j and b_j are the first pair of taxa with different frequencies; otherwise $s_1 = s_2$.

Suppose that the seed-growing algorithm finds that $A = \{a_1, a_2, \cdots, a_n\}$ is a solution of the RMCS problem for $\mathcal{C}$. Then, for each element a_i, $i \in \{1, 2, \cdots, n\}$, let $\mathcal{C}_i = \{C \in \mathcal{C} | a_i \in C\}$. Let $\overline{\mathcal{C}} = \mathcal{C}_1 \cap \mathcal{C}_2 \cap \cdots \cap \mathcal{C}_t$ and $M = (\mathcal{C}_1 \setminus \overline{\mathcal{C}}) \cup (\mathcal{C}_2 \setminus \overline{\mathcal{C}}) \cup \cdots \cup (\mathcal{C}_t \setminus \overline{\mathcal{C}})$, where $t \leq n$. We can say that $\{a_1, a_2, \cdots, a_t\}$ can be merged, if $M < A$, where A can be seen as a cluster.

LEMMA 1. *Given three cluster sets $\mathcal{C}_1, \mathcal{C}_2$ and $\mathcal{C}_3$ on $\mathcal{X}$ and a cluster C_0 on $\mathcal{X}$, let $M_1 = (\mathcal{C}_1 \setminus (\mathcal{C}_1 \cap \mathcal{C}_2)) \cup (\mathcal{C}_2 \setminus (\mathcal{C}_1 \cap \mathcal{C}_2))$, $M_2 = (\mathcal{C}_1 \setminus (\mathcal{C}_1 \cap \mathcal{C}_3)) \cup (\mathcal{C}_3 \setminus (\mathcal{C}_1 \cap \mathcal{C}_3))$, $M_3 = (\mathcal{C}_2 \setminus (\mathcal{C}_2 \cap \mathcal{C}_3)) \cup (\mathcal{C}_3 \setminus (\mathcal{C}_2 \cap \mathcal{C}_3))$, $\overline{\mathcal{C}} = \mathcal{C}_1 \cap \mathcal{C}_2 \cap \mathcal{C}_3$ and $M = (\mathcal{C}_1 \setminus \overline{\mathcal{C}}) \cup (\mathcal{C}_2 \setminus \overline{\mathcal{C}}) \cup (\mathcal{C}_3 \setminus \overline{\mathcal{C}})$. Then $M < C_0$ if and only if $M_1 < C_0$, $M_2 < C_0$ and $M_3 < C_0$.*

PROOF. See the Supplementary Material.

From Lemma 1, it follows that $B = \{a_1, a_2, \cdots, a_t\}$ can be merged if and only if any subsets of B can be merged.

We merge elements that can be merged for A such that $A = \{A_1, A_2, \cdots, A_k\}$, where $A_i = \{a_{i1}, a_{i2}, \cdots, a_{im_i}\}$, $A_i \cap A_j = \emptyset (i \neq j)$, $i, j \in \{1, 2, \cdots, k\}$ and $\sum_{l=1}^{k} m_l = n$. Each A_i is called a *removed taxon* with respect to $\mathcal{C}$. A removed taxon A_i is called *trivial* if it contains only one element; otherwise it is called *non-trivial*. Let $\overline{\mathcal{C}}_i = \{C \in \mathcal{C} | C \subseteq A, C \cap A_i \neq \emptyset\}$ and $i \in \{1, 2, \cdots, k\}$. A_i and A_j $(i \neq j)$ are called *conflicting* if there exists a cluster $C \in \overline{\mathcal{C}}_i \cup \overline{\mathcal{C}}_j$, such that C, A_i are incompatible, and C, A_j are incompatible. A_i is called *conflicting* if the cluster set $\mathcal{C}|_{A_i}$ is incompatible. Then, $\mathcal{C}'_{ij} = \{C \in \overline{\mathcal{C}}_i \cup \overline{\mathcal{C}}_j | C, A_i$ are incompatible, and C, A_j are incompatible$\}$ is called the conflicting cluster set of A_i and A_j, whereas $\mathcal{C}'_i = \{C_1, C_2 \in \mathcal{C}|_{A_i} | C_1$ and C_2 are incompatible$\}$ is called the conflicting cluster set of A_i. Obviously, if A_i and A_j are conflicting then A_i or A_j is non-trivial, and if A_i is conflicting then A_i is non-trivial. A is called *decomposable* if there is a conflicting removed taxon $A_i \in A$, or if there are two removed taxa A_i, $A_j \in A$, such that A_i and A_j are conflicting; otherwise it is called *indecomposable*. Let $A_c = \{i \in \{1, 2, \cdots, k\} | A_i$ is conflicting, or there exists a removed taxon $A_j \in A$, such that A_i and A_j are conflicting$\}$. Let

$$\mathcal{C}' = \bigcup_{i,j \in A_c} \mathcal{C}'_{ij} \bigcup_{i \in A_c} \mathcal{C}'_i \bigcup_{i \in A_c} \{A_i\}. \tag{3.1}$$

$\mathcal{C}'$ is obviously incompatible when A is decomposable; hence, the seed-growing algorithm can find a solution $A' = \{A'_1, A'_2, \cdots, A'_m\}$ for $\mathcal{C}'$ after merging the elements that can be merged. We subsequently decompose A by means of A' as follows.

Decomposing: Find all of the removed taxa from A' that are subsets of A_1. Without loss of generality, let A'_1 and $A'_2 \in A'$ be subsets of A_1. Then we decompose A_1 into A'_1, A'_2, $A_1 \setminus (A'_1 \cup A'_2)$. If no such removed taxon exists, we decompose A_1 into itself. We then decompose $A_2, \cdots, A_k$ in the same way.

After applying the decomposing process to A, we get $A = \{A''_{11}, \cdots, A''_{1s_1}, A''_{21}, \cdots, A''_{2s_2}, \cdots, A''_{k1}, \cdots, A''_{ks_k}\}$; for the sake of brevity, we denote this as $A = \{A_1, A_2, \cdots, A_k\}$. Then, if A is still decomposable, we compute $\mathcal{C}'$ using Formula (3.1). Further, we obtain a solution A'' for the RMCS problem for $\mathcal{C}'$ using the seed-growing algorithm, and merge the elements for A'' that can be merged. Subsequently, we decompose A by means of A'' using the decomposing process. We repeat the above steps until A is indecomposable, at which point we say that A has undergone decomposing.

Suppose that $A = \{A''_1, A''_2, \cdots, A''_k\}$ is indecomposable. For each A''_i, $i \in \{2, \cdots, k-1\}$, we say that the removed order of A''_i is after $A''_1, A''_2, \cdots, A''_{i-1}$ and before $A''_{i+1}, \cdots, A''_k$. Specifically, the removed orders of A''_1 and A''_k are the first and the last, respectively. For each A''_i with $|A''_i| > 1$, we need to decide its removal order using the following sorting process.

Sorting: Let $\overline{\mathcal{C}}_i = \{C \in \mathcal{C} | C \subseteq A$ and $C \cap A''_i \neq \emptyset\}$, $i \in \{1, 2, \cdots, k\}$. If there exists a cluster $C \in \overline{\mathcal{C}}_i$, such that C and A''_i are incompatible, then $C_1 = C \setminus A''_i$. A''_j should be before A''_i if $A''_j \subseteq C_1 (i \neq j)$. Then we sort A.

After applying the sorting process to A''_i with $|A''_i| > 1$, $i \in \{1, \cdots, k\}$, we get $A = \{A_1, A_2, \cdots, A_k\}$, which is a permutation of $A = \{A''_1, A''_2, \cdots, A''_k\}$. Then we say that A has undergone sorting.

THEOREM 2. *Let $\mathcal{C}$ be a cluster set on $\mathcal{X}$ with $IG(\mathcal{C})$ connected and assume that there are no ST-sets with respect to $\mathcal{C}$. $A = \{A_1, A_2, \cdots, A_k\}$ is a solution of the RMCS problem for $\mathcal{C}$, which has undergone decomposing and sorting. Then there will be a network N, with $r_N = k$ representing $\mathcal{C}$, and the leaves below the reticulate nodes of N will be $A_1, A_2, \cdots, A_k$, respectively.*

PROOF. See the Supplementary Material.

Theorem 2 tells us that there is a network representing the given cluster set and the leaves below the reticulate nodes are the removed taxa. However, because of the method of inserting the new nodes that have a child that is a reticulate node, the network N constructed by the proof of Theorem 2 represents too many redundant clusters. Therefore, when inserting those nodes, the LNETWORK(k) algorithm follows the CASS(k) algorithm, i.e. it tries to add the nodes below each pair of edges.

4 LNETWORK

LNETWORK first decomposes the incompatibility graph $IG(\mathcal{C})$ into biconnected components, as does CASS, and then constructs a simple level $\leq k$ network by applying LNETWORK(k) to each non-trivial biconnected component separately. Then it integrates those simple level $\leq k$ networks into a final network. The process with which LNETWORK(k) constructs a simple level $\leq k$ network is as follows.

Let $\mathcal{A}$ be the solution set of the RMCS problem for $\mathcal{C}$ found by the seed-growing algorithm. Then, LNETWORK(k) chooses a solution A from $\mathcal{A}$ such that $A \geq B$ for any element $B \in \mathcal{A}$. This step aims at weakening the influence of input data order by fixing the solution of the RMCS problem for $\mathcal{C}$ when the input

order of $\mathcal{C}$ is changed. The decomposing process is then applied repeatedly until A is indecomposable. Subsequently, the sorting process is applied. Then $A = \{A_1, A_2, \cdots, A_k\}$.

LNETWORK(k) first removes A_1 from each cluster of $\mathcal{C}$. It subsequently collapses all of the maximal ST-sets of $\mathcal{C}|_{\mathcal{X}\setminus A_1}$. These steps repeat k times until the resulting cluster set is compatible and the second phase of the algorithm starts. LNETWORK(k) creates a network consisting of a unique phylogenetic tree for the resulting cluster set. Then the algorithm 'expands'. Subsequently, LNETWORK(k) adds a new leaf below the new reticulate node and labels it as the removed taxon of this step. Then LNETWORK(k) tries to add the reticulate node below each pair of edges. The algorithm continues with a new 'expand' step and then hangs the next leaf below a reticulate node. LNETWORK(k) will save this network if it represents the cluster set of this step. LNETWORK(k) finds all of the networks representing the cluster set of this step and sorts them in descending order of the number of clusters represented; this reduces the number of redundant clusters in the resulting network and weakens the influence of input data order. These steps are repeated until all of the removed taxa are appended to the networks. LNETWORK outputs the resulting network, which now has the minimal number of redundant clusters. The resulting network is a simple level $\leq k$ network, where k is no less than the number of removed taxa. In particular, when the resulting network is binary, k equals the number of removed taxa. For ease of description, sometimes the construction technique mentioned in this paragraph is called as the remove-collapse-expand-attach technique of LNETWORK.

LNETWORK(k) adds a new dummy taxon to the network when it constructs the reticulate nodes with indegree >2. The dummy taxon is removed before it outputs the resulting network. We give the pseudo-code of the LNETWORK(k) algorithm in the Supplementary Material.

LEMMA 3. *LNETWORK(k) runs in time $O(m^2 d 3^m + (k^2 + |\mathcal{X}|^2)2^k)$, where d is the incompatibility degree of the cluster set $\mathcal{C}$ and m is the size of the solution of the RMCS problem for $\mathcal{C}$.*

PROOF. See the Supplementary Material.

In the Supplementary Material, there is an example illustrating all of the steps of LNETWORK(k) as it constructs a simple level $\leq k$ network.

5 RESULTS

All of the experiments were performed on a computer with an Intel Xeon E5504 2.0 GHz CPU, 8GB RAM and 147GB HDD. The operating system was Debian 4.1 32 bit with Java 1.6 installed. LNETWORK was written in Java.

We tested LNETWORK with both the practical (i.e. real biological data) and artificial data (`https://sites.google.com/site/cassalgorithm/data-sets`) used by CASS and compared the results of LNETWORK with those of other programs. The results are summarized in Tables 1, 2 and 3. The experiments test two main aspects of the LNETWORK algorithm. First, they test the influence of input data order; the results are shown in Table 1. Second, they test the complexities of the network, i.e. the level, the reticulation number and the redundant cluster number; the results are shown in Tables 2 and 3.

Table 1. The influence of input data order on the LNETWORK and CASS algorithms

Data		LNETWORK				CASS			
$\lvert\mathcal{C}\rvert$	$\lvert\mathcal{X}\rvert$	n	mean	min	max	n	mean	min	max
35	22	1	0	0	0	2	6.5	6.5	6.5
25	15	1	0	0	0	2	3	3	3
22	13	2	1	1	1	2	0.5	0.5	0.5
27	15	2	1	1	1	3	3	3	3
25	13	3	1.2	0.5	1.5	4	6.3	2	7.5
22	11	1	0	0	0	3	3	2.5	3.5
17	10	3	1.3	1	1.5	3	2	1.5	2.5
13	8	2	1	1	1	4	3.6	1.5	4
23	11	2	1	1	1	4	5.6	3	7.5
18	10	3	2.5	1.5	3.5	4	1.5	0.5	3
22	11	1	0	0	0	3	3.2	1.5	5
12	10	1	0	0	0	2	3	3	3
21	10	2	1.5	1.5	1.5	4	3.9	1.5	5.5
13	7	2	1	1	1	4	3.8	1.5	4
22	10	1	0	0	0	2	1.5	1.5	1.5
21.3	11.7	1.8	1.2	1.1	1.4	3.1	3.4	2.2	4

Note: n represents the number of constructed networks, and mean, min, max represent, respectively, the mean, the minimum, the maximum of tripartition distances of those networks.
Each dataset has $|\mathcal{C}|$ clusters and $|\mathcal{X}|$ taxa.

Table 2. Results of LNETWORK and CASS for artificial datasets with $|\mathcal{C}|$ clusters and $|\mathcal{X}|$ taxa

Data		LNETWORK				CASS			
$\lvert\mathcal{C}\rvert$	$\lvert\mathcal{X}\rvert$	t (s)	k	r	c	t	k	r	c
14	4	0	3	3	0	1 s	3	3	0
30	5	2	4	4	0	2 s	4	4	0
62	6	18	5	5	0	11 s	5	5	0
42	10	1	4	4	14	10 s	4	4	34
39	11	38	6	6	18	21 s	5	5	7
61	11	15	6	6	43	1 m 26 s	5	5	48
75	30	1	2	2	19	4 s	2	2	19
180	51	4	2	2	0	40 s	2	2	0
70	56	1	1	4	0	1 s	1	4	0
270	76	12	2	2	0	6 m 22 s	2	2	0
404	122	44	2	2	0	1 h 14 m	2	2	0
113.4	34.7	13	3.4	3.6	8.5	7 m 34 s	3.2	3.5	10

Note: t, k, r and c represent, respectively, the running time, the level, the reticulation number and the redundant cluster number.

To test the influence of the input data order, for each dataset the program constructs the networks for every permutation of input data order. The running time for this process is factorial; thus, in the experiment the scale of data considered is small. To compare the differences between the constructed networks, we use the tripartition distances of networks. Table 1 shows, for each dataset, the number (n) of the different networks constructed by each program and the mean, the minimum (min) and the

Table 3. Results of LNETWORK, CASS, the cluster network and the galled network for practical datasets with $|\mathcal{C}|$ clusters and $|\mathcal{X}|$ taxa

Data		LNETWORK		CASS		Cluster network		Galled network	
$\|\mathcal{C}\|$	$\|\mathcal{X}\|$	r	c	r	c	r	c	r	c
86	37	8	11	8	27	12	17	14	27
38	20	6	15	6	25	8	48	6	48
43	22	5	3	4	12	9	9	6	15
72	27	7	19	7	43	13	21	8	21
52	22	8	15	7	33	14	21	9	26
79	27	8	44	8	89	20	355	11	19
38	16	9	36	7	50	13	98	9	46
41	16	5	4	5	29	10	12	6	7
12	8	2	0	2	2	3	5	2	5
45	20	7	28	7	66	17	161	8	101
22	11	3	1	3	5	4	5	3	4
17	10	3	4	3	8	4	9	3	13
46	16	8	15	7	34	19	41	8	6
22	11	4	13	4	23	9	33	6	17
22	10	4	12	4	21	10	33	6	15
42.3	18.2	5.8	14.7	5.5	31	11	57.9	7	24.7

Note: r and c are defined as in Table 2.

Table 4. Results of LNETWORK for the simulation data generated by the PHYL-O-GEN tool

LNETWORK											
$\|\mathcal{C}\|$	$\|\mathcal{X}\|$	k	r	c	t	$\|\mathcal{C}\|$	$\|\mathcal{X}\|$	k	r	c	t
288	148	7	39	71	5 s	524	240	8	78	197	11 m 26 s
286	120	10	39	84	14 m 6 s	488	220	14	74	221	48 h
206	100	7	30	87	11.6 s	292	150	8	43	119	27 s
185	100	8	29	87	35.8 s	341	150	14	51	163	25 h 38 m
189	100	11	33	90	5 m 9 s	343	150	10	47	113	5 m 14 s
218	100	10	32	91	3 m 35 s	372	150	9	47	75	2 m 36 s

Note: k, r, c and t are defined as in Table 2.

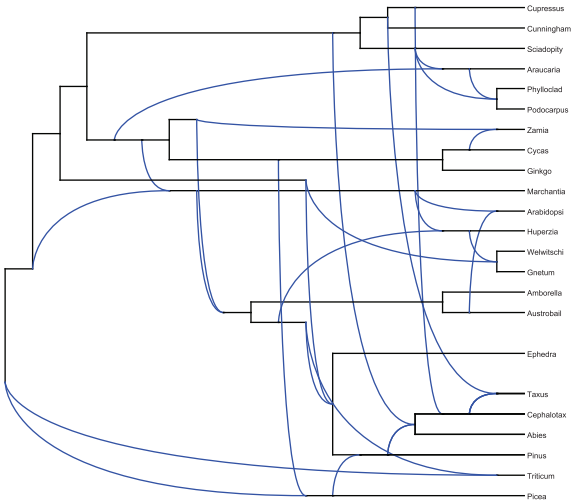

Fig. 2. A level-15 network with 23 taxa, 15 reticulations and 1236 redundant clusters constructed by LNETWORK for the four gene trees of the Pinaceae datasets, within 15 days. All of the clusters that were present in at least one of the four gene trees were used. CASS did not finish the process within 30 days. For the same input, the galled network produced a level-16 network with 16 reticulations and 3197 redundant clusters, and the cluster network produced a level-31 network with 31 reticulations and >5239 redundant clusters

maximum (max) of tripartition distances of those networks; the last row gives their average values. Our conclusions about the superiority of LNETWORK are based on the following observations. First, the number of different networks constructed by LNETWORK is less than the number of different networks constructed by CASS for the same data with different input orders. Second, the tripartition distance between the networks constructed by LNETWORK is less than that between the networks constructed by CASS; this demonstrates that, if LNETWORK outputs more than one network when the input order of the data is changed, the networks constructed by LNETWORK are more similar to each other than the networks constructed by CASS. Thus, LNETWORK effectively weakens the influence of input data order compared with CASS.

Table 2 compares the results of LNETWORK with the results of CASS for several artificial datasets. Table 2 shows the level k, the reticulation number r, the redundant cluster number c and the running time t in hours (h), minutes (m) and seconds (s) for the two algorithms; the last row gives their average values. The comparison shows that LNETWORK takes less time than CASS, and that the level and the reticulation numbers of the networks constructed by LNETWORK are the same as the level and the reticulation numbers of the networks constructed by CASS in almost every case. Furthermore, the networks constructed by LNETWORK have fewer redundant clusters than the networks constructed by CASS in almost every case.

The study by van Iersel *et al.* compared HYBRID INTERLEVE (Collins *et al.*, 2011) and Parsimonious Inference of Reticulate Network (PIRN) (Wu, 2010) with CASS. In this study, we compare LNETWORK with CASS, the galled network (Huson *et al.*, 2009) and the cluster network (Huson and Rupp, 2008) using practical data. Table 3 shows that the average reticulation number of LNETWORK is slightly more than that of CASS and less than that of the other two programs. The networks constructed by LNETWORK represent fewer redundant clusters than those constructed by the other programs. In terms of running time, LNETWORK is faster than CASS in general, but is slower than the cluster network and the galled network algorithms. Even so, the running time of LNETWORK is acceptable. Thus LNETWORK is superior to the other algorithms when we consider running time, number of reticulations and number of redundant clusters. Tables 2 and 3 show that when the constructed networks are binary, the level and the reticulation number of the network constructed by LNETWORK are the same as the level and the reticulation number of the network constructed by CASS when the same input data are used.

We ran LNETWORK and CASS on the simulation data randomly generated by the PHYL-O-GEN tool (http://tree.bio.ed.ac.uk/software/phylogen/). Table 4 shows the results of only LNETWORK as CASS did not finish within 10 days. These results suggest that LNETWORK can construct networks for datasets that

Table 5. Results of LNETWORK, CASS, the cluster network and the galled network for the simulation data generated by the Recodon tool

Method	SIZE	FP	FN	MDFP	MDFN
LNETWORK	**2.7869**	**0.1235**	**0.0650**	**0.1605**	**0.0481**
CASS	2.8006	0.1368	0.0863	0.1753	0.0576
Galled network	2.9410	0.1432	0.0899	0.1832	0.0579
Cluster network	11.1292	0.5489	0.1849	0.7724	0.1797

Note: The best value in every column is displayed in bold.

need high levels and many reticulations. This conclusion is illustrated in Figure 2, which shows the output of LNETWORK when it is given real data consisting of all of the clusters in at least one of the four gene trees from a Pinaceae dataset.

Finally, we compare the performance of LNETWORK, CASS, the galled network and the cluster network using simulation data generated by the Recodon tool (Arenas and Posada, 2007) (`http://darwin.uvigo.es/`). We use the tool to randomly generate 700 datasets, and then run these datasets using each program. This tests how well the LNETWORK algorithm constructs networks for input tree sets by comparing the topology structure of the input trees with the trees represented by the resulting constructed network. We first list the trees represented by the constructed network. Let $\mathcal{T}$ be the input tree set and $\mathcal{N}$ be the set of trees represented by the constructed network for the input tree set. We then compute the Robinson–Foulds (RF) distance (Robinson and Foulds, 1981) between any one tree in $\mathcal{N}$ and any one tree in $\mathcal{T}$. For any two trees T_1 and T_2, the RF distance between them is denoted as $\mathrm{RF}(T_1, T_2)$. Two trees are identical if the RF distance between them is zero. The false positive (FP) is the fraction of trees in $\mathcal{N}$ that do not exist in $\mathcal{T}$. The false negative (FN) is the fraction of trees in $\mathcal{T}$ that do not exist in $\mathcal{N}$. For any one tree T in $\mathcal{N}$, the degree of false positive (DFP) of T, denoted by $\mathrm{DFP}(T)$, is defined as $\mathrm{DFP}(T) = \min\{\mathrm{RF}(T, T_0) | T_0 \in \mathcal{T}\}$. For any one tree T in $\mathcal{T}$, the degree of false negative (DFN) of T, denoted by $\mathrm{DFN}(T)$, is defined as $\mathrm{DFN}(T) = \min\{\mathrm{RF}(T, T_0) | T_0 \in \mathcal{N}\}$. For each of the 700 datasets, we compute the size of $\mathcal{N}$ (SIZE), FP, FN, mean DFP (MDFP) and mean DFN (MDFN). Table 5 shows the average of those values. The mean size of the input tree sets is 2.3652. The table shows that the mean size of trees represented by the networks constructed by LNETWORK is closer to the mean size of input tree sets than the mean size of trees represented by the networks constructed by the other programs, and the mean FP, the mean FN, the mean MDFP and the mean MDFN of the networks constructed by LNETWORK are less than those of the networks constructed by the other programs. Thus, the networks constructed by LNETWORK are closer to the input tree sets than the networks constructed by the other programs.

6 CONCLUSIONS

We have presented LNETWORK, which can construct a phylogenetic network for a given set of clusters. We have shown that LNETWORK effectively weakens the influence of input data order compared with CASS, and that the execution time of LNETWORK is shorter than that of CASS. Moreover, we show that the levels and the reticulation numbers of networks constructed by LNETWORK are the same as those of networks constructed by CASS for most datasets, and that the networks constructed by LNETWORK represent fewer redundant clusters than other available methods, for most datasets. We also show that the trees represented by the networks constructed by LNETWORK are closer to the input trees than those represented by the networks constructed by all other tested programs.

ACKNOWLEDGEMENT

We thank L.J.J. van Iersel for providing us with the source code of the CASS(k) program and the test data.

Funding: This work was supported by the Natural Science Foundation of China (60932008, 61172098 and 61271346) and the Specialized Research Fund for the Doctoral Program of Higher Education of China (20112302110040).

Conflict of Interest: none declared.

REFERENCES

Arenas,M. and Posada,D. (2007) Recodon: coalescent simulation of coding DNA sequences with recombination, migration and demography. *BMC Bioinformatics*, **8**, 458.

Cardona,G. (2009) Metrics for phylogenetic networks I: generalizations of the Robinson-Foulds metric. *IEEE/ACM Trans. Comput. Biol. Bioinform.*, **6**, 46–61.

Collins,J. *et al.* (2011) Quantifying hybridization in realistic time. *J. Comput. Biol.*, **18**, 1305–1318.

Doolittle,W.F. (1999) Phylogenetic classification and the universal tree. *Science*, **284**, 2124–2128.

Gusfield,D. and Bansal,V. (2005) A fundamental decomposition theory for phylogenetic networks and incompatible characters. In: *Proceedings of the Ninth International Conference on Research in Computational Molecular Biology (RECOMB 2005)*. LNCS 3500, Springer, Cambridge, MA, pp. 217–232.

Huson,D.H. (2005) Introduction to phylogenetic networks. In: *13th Annual Intelligent Systems for Molecular Biology Conference*. Detroit, MI, USA.

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

Huson,D.H. and Rupp,R. (2008) Summarizing multiple gene trees using cluster networks. In: *Algorithms in Bioinformatics (WABI)*. Vol. 5251 *of Lecture Notes in Bioinformatics*, Springer, Berlin, pp. 296–305.

Huson,D.H. and Scornavacca,C. (2010) A survey of combinatorial methods for phylogenetic networks. *Genome Biol. Evol.*, **3**, 23–35.

Huson,D.H. and Scornavacca,C. (2012) Dendroscope 3-An interactive viewer for rooted phylogenetic trees and networks. *Syst. Biol.*, **61**, 1061–1067.

Huson,D.H. *et al.* (2004) Phylogenetic super-networks from partial trees. *IEEE/ACM Trans. Comput. Biol. Bioinform.*, **1**, 151–158.

Huson,D.H. *et al.* (2009) Computing galled networks from real data. *Bioinformatics*, **25**, i85–i93.

Huson,D.H. *et al.* (2011) Clusters and rooted phylogenetic networks. In: *Phylogenetic Networks: Concepts, Algorithms and Applications*. Cambridge University Press, Cambridge, UK.

Kelly,L.J. *et al.* (2010) Intragenic recombination events and evidence for hybrid speciation in Nicotiana (Solanaceae). *Mol. Biol. Evol.*, **27**, 781–799.

Linder,C.R. and Rieseberg,L.H. (2004) Reconstructing patterns of reticulate evolution in plants. *Am. J. Bot.*, **91**, 1700–1708.

Linder,C.R. *et al.* (2004) Network (reticulate) evolution: biology, models, and algorithms. In: *Ninth Pacific Symposium on Biocomputing*. Hawaii, USA.

Linz,S. and Semple,C. (2009) Hybridisation in nonbinary trees. *IEEE/ACM Trans. Comput. Biol. Bioinform.*, **6**, 30–45.

Maddison,W.P. (1997) Gene trees in species trees. *Syst. Biol.*, **46**, 523–536.

Nakhleh,L. (2010) Evolutionary phylogenetic networks: models and issues. In: Heath,L.L.S. and Ramakrishnan,N. (eds) *The Problem Solving Handbook for Computational Biology and Bioinformatics*. Springer US.

Rieseberg,L.H. (1997) Hybrid origins of plant species. *Annu. Rev. Ecol. Evol. Syst.*, **28**, 359–389.

Robinson,D.F. and Foulds,L.R. (1981) Comparison of phylogenetic trees. *Math. Biosci.*, **53**, 131–147.

Schwarzer,J. *et al.* (2011) Speciation within genomic networks: a case study based on Steatocranus cichlids of the lower Congo rapids. *J. Evol. Biol.*, **25**, 138–148.

Song,Y.S. and Hein,J. (2005) Constructing minimal ancestral recombination graphs. *J. Comp. Biol.*, **12**, 147–169.

van Iersel,L. *et al.* (2010) Phylogenetic networks do not need to be complex: using fewer reticulations to represent conflicting clusters. *Bioinformatics*, **26**, i124–i131.

Wang,L. *et al.* (2012) Indehiscent sporangia enable the accumulation of local fern diversity at the Qinghai-Tibetan Plateau. *BMC Evol. Biol.*, **12**, 158–169.

Whitfield,J.B. *et al.* (2008) Filtered Z-closure supernetworks for extracting and visualizing recurrent signal from incongruent gene trees. *Syst. Biol.*, **57**, 939–947.

Woolley,S.M. *et al.* (2008) A comparison of phylogenetic network methods using computer simulation. *Plos One*, **3**, e1913.

Wu,Y. (2010) Close lower and upper bounds for the minimum reticulate network of multiple phylogenetic trees. *Bioinformatics*, **26**, i140–i148.