BIOINFORMATICS

Vol. 26 ISMB 2010, pages i124–i131
doi:10.1093/bioinformatics/btq202

Phylogenetic networks do not need to be complex: using fewer reticulations to represent conflicting clusters

Leo van Iersel[1,*], Steven Kelk[2], Regula Rupp[3] and Daniel Huson[3]

[1]Department of Mathematics and Statistics, University of Canterbury, Private Bag 4800, Christchurch, New Zealand, [2]Centrum voor Wiskunde en Informatica (CWI), Life Sciences, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands and [3]Center for Bioinformatics ZBIT, Tübingen University, Sand 14, 72076 Tübingen, Germany

ABSTRACT

Phylogenetic trees are widely used to display estimates of how groups of species are evolved. Each phylogenetic tree can be seen as a collection of *clusters*, subgroups of the species that evolved from a common ancestor. When phylogenetic trees are obtained for several datasets (e.g. for different genes), then their clusters are often contradicting. Consequently, the set of all clusters of such a dataset cannot be combined into a single phylogenetic tree. *Phylogenetic networks* are a generalization of phylogenetic trees that can be used to display more complex evolutionary histories, including *reticulate events*, such as hybridizations, recombinations and horizontal gene transfers. Here, we present the new Cass algorithm that can combine any set of clusters into a phylogenetic network. We show that the networks constructed by Cass are usually simpler than networks constructed by other available methods. Moreover, we show that Cass is guaranteed to produce a network with at most two reticulations per biconnected component, whenever such a network exists. We have implemented Cass and integrated it into the freely available Dendroscope software.

Contact: l.j.j.v.iersel@gmail.com

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

1 INTRODUCTION

Phylogenetics studies the reconstruction of evolutionary histories from genetic data of currently living organisms. A (rooted) *phylogenetic tree* is a representation of such an evolutionary history in which species evolve by mutation and speciation. The leaves of the tree represent the species under consideration and the root of the tree represents their most recent common ancestor. Each internal node represents a speciation: 1 species splits into several new species. Thus, mathematically speaking, such a node has indegree 1 and outdegree at least 2. In recent years, a lot of work has been done on developing methods for computing (rooted) *phylogenetic networks* (Gambette, 2009; Gusfield *et al*., 2007b; D.H.Huson *et al*., submitted for publication; Nakleh, 2009; Semple, 2007), which form a generalization of phylogenetic trees. Next to nodes representing speciation, rooted phylogenetic networks can also contain *reticulations*: nodes with indegree at least 2. Such nodes can be used to represent the recombinations, hybridizations or horizontal gene transfers, depending on the biological context. In addition, phylogenetic networks can also be interpreted in a more abstract sense, as a visualization of contradictory phylogenetic information in a single diagram.

*To whom correspondence should be addressed.

Suppose we wish to investigate the evolution of a set $\mathcal{X}$ of taxa (e.g. species or strains). Each edge of a rooted phylogenetic tree represents a *cluster*: a proper subset of the taxon set $\mathcal{X}$. In more detail, an edge (u,v) represents the cluster containing those taxa that are descendants of v. Each phylogenetic tree T is uniquely defined by the set of clusters represented by T. Phylogenetic networks also represent clusters. Each of their edges represents one 'hardwired' and at least one 'softwired' cluster. An edge (u,v) of a phylogenetic network *represents* a cluster $C \subset \mathcal{X}$ *in the hardwired sense* if C equals the set of taxa that are descendants of v. Furthermore, (u,v) *represents* C *in the softwired sense* if C equals the set of all taxa that can be reached from v when, for each reticulation r, exactly one incoming edge of r is 'switched on' and the other incoming edges of r are 'switched off'. An equivalent definition states that a phylogenetic network N *represents* a cluster C *in the softwired sense* if there exists a tree T that is displayed by N (formally defined below) and represents C. In this article, we will always use 'represent' in the softwired sense. It is usually the clusters in a tree that are of more interest, and less the actual trees themselves, as clusters represent putative monophyletic groups of related species. For a complete introduction to clusters see D.H.Huson *et al*. (submitted for publication).

In phylogenetic analysis, it is common to compute phylogenetic trees for more than one dataset. For example, a phylogenetic tree can be constructed for each gene separately, or several phylogenetic trees can be constructed using different methods. To accurately reconstruct the evolutionary history of all considered taxa, one would preferably like to use the set $\mathcal{C}$ of all clusters represented by at least one of the constructed phylogenetic trees. In general, however, some of the clusters of the different trees will be incompatible, which means that there will be no single phylogenetic tree representing $\mathcal{C}$. Therefore, several recent publications have studied the construction of a phylogenetic *network* representing $\mathcal{C}$. Huson and Rupp (2008) describe how a phylogenetic network can be constructed that represents $\mathcal{C}$ in the hardwired sense (a *cluster network*). A network is a *galled network* if it contains no path between two reticulations that is contained in a single *biconnected component* (a maximal subgraph that cannot be disconnected by removing a single node, see Fig. 1). Huson and Klöpper (2007) and Huson *et al*. (2009) describe an algorithm for constructing a galled network representing $\mathcal{C}$ in the softwired sense.

Related literature describes the construction of phylogenetic networks from phylogenetic trees or *triplets* (phylogenetic trees on three taxa). A tree or triplet T is *displayed* by a network N if there is a subgraph T' of N that is a subdivision of T (i.e. T' can be obtained from T by replacing edges by directed paths). Computing the minimum number of reticulation required in a

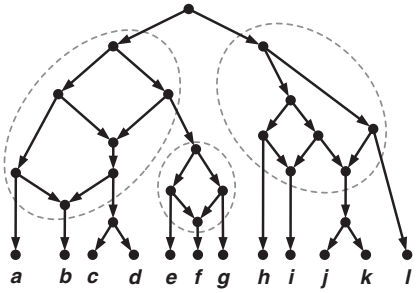

Fig. 1. Example of a phylogenetic network with five reticulations. The encircled subgraphs form its biconnected components. This binary network is a level-2 network since each biconnected component contains at most two reticulations.

phylogenetic network displaying two input trees (on the same set of taxa) was shown to be APX-hard by Bordewich and Semple (2007). Bordewich *et al.* (2007) proposed an exact exponential-time algorithm for this problem and Linz and Semple (2009) showed that it is fixed parameter tractable (FPT), if parameterized by the minimum number of reticulations. The downside of these algorithms is that they are very rigid in the sense that one generally needs very complex networks in order to display the given trees.

The *level* of a binary network is the maximum number of reticulations in a biconnected component,[1] and thus provides a measure of network complexity. Given an arbitrary number of trees on the same set of taxa, Huynh *et al*. (2005) describe a polynomial-time algorithm that constructs a level-1 phylogenetic network that displays all trees and has a minimum number of reticulations, if such a network exists (which is unlikely in practice). Given a triplet for each combination of three taxa, Jansson and coworkers (Jansson and Sung, 2006; Jansson *et al*., 2006) and give a polynomial-time algorithm that constructs a level-1 network displaying all triplets, if such a network exists. The algorithm by van Iersel and Kelk (2009) can be used to find such a network that also minimizes the number of reticulations. These results have later been extended to level-2 (van Iersel and Kelk, 2009; van Iersel *et al*., 2009a) and more recently to level-k, for all $k \in \mathbb{N}$ (To and Habib, 2009). Although this work on triplets is theoretically interesting, it has the practical drawback that biologists do not work with triplets (but rather with trees or clusters) and that it is rather difficult to intuitively convey what it means for a triplet to be 'in' a network. An additional drawback is that these triplet algorithms need at least one triplet in the input for each combination of three taxa, while some triplets might be difficult to derive correctly. If, for example, one induces triplets from a set of trees, then this is likely not to give you a triplet for each combination of three taxa, if one or more input trees are not fully resolved or if some input trees do not have exactly the same set of taxa.

In this article, we present the algorithm Cass,[2] which takes any set $\mathcal{C}$ of clusters as input and constructs a phylogenetic network that represents $\mathcal{C}$ (in the softwired sense). Furthermore, the algorithm aims at minimizing the level of the constructed network and in this sense Cass is the first algorithm to combine the flexibility of clusters with the power of level minimization. Cass constructs a phylogenetic tree representing $\mathcal{C}$ whenever such a tree exists.

[1]In Section 2, we generalize the notion of *level* to non-binary networks.
[2]Named after the Cass Field Station in New Zealand.

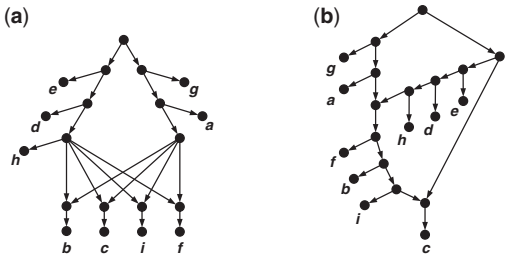

Fig. 2. (**a**) The output of the galled network algorithm (Huson *et al*., 2009) for $\mathcal{C} = \{\{a,b,f,g,i\}, \{a,b,c,f,g,i\}, \{a,b,f,i\}, \{b,c,f,i\}, \{c,d,e,h\}, \{d,e,h\}, \{b,c,f,h,i\}, \{b,c,d,f,h,i\}, \{b,c,i\}, \{a,g\}, \{b,i\}, \{c,i\}, \{d,h\}\}$ and (**b**) the network constructed by Cass for the same input.

Moreover, we prove that Cass constructs a level-1 or level-2 network representing $\mathcal{C}$ whenever there exists a level-1 or level-2 network representing $\mathcal{C}$, respectively. Experimental results show that also when no level-2 network representing $\mathcal{C}$ exists, Cass usually constructs a network with a significantly lower level and lower number of reticulations compared with other algorithms. In fact, we conjecture that similar arguments as in our proof for level-2 can be used to show that Cass always constructs a level-k network with minimum k. We prove a decomposition theorem for level-k networks that supports this conjecture. Finally, we prove that Cass runs in polynomial time if the level of the output network is bounded by a constant.

We have implemented Cass and added it to our popular tree-drawing program Dendroscope (Huson *et al*., 2007), where it can be used as an alternative for the cluster network (Huson and Rupp, 2008) and galled network (Huson *et al*., 2009) algorithms. Experiments show that, although Cass needs more time than these other algorithms, it constructs a simpler network representing the same set of clusters. For example, Figure 2a shows a set of clusters and the galled network with four reticulations constructed by the algorithm in Huson *et al*. (2009). However, for this dataset also a level-2 network with two reticulations exists, and Cass can be used to find this network, see Figure 2b. Dendroscope now combines the powers of Cass and the two previously existing algorithms for constructing galled- and cluster networks.

2 LEVEL-*K* NETWORKS AND CLUSTERS

Consider a set $\mathcal{X}$ of taxa. A *rooted (phylogenetic) network* (on $\mathcal{X}$) is a directed acyclic graph with a single root and leaves bijectively labeled by $\mathcal{X}$. The indegree of a node v is denoted $\delta^-(v)$ and v is called a *reticulation* if $\delta^-(v) \geq 2$. An edge (u,v) is called a *reticulation edge* if its head v is a reticulation and is called a *tree edge* otherwise. We assume without loss of generality that each reticulation has outdegree at least 1. Consequently, each leaf has indegree 1. When counting reticulations in a phylogenetic network, we count reticulations with more than two incoming edges more than once because, biologically, these reticulations represent several reticulate evolutionary events. Therefore, we formally define the *reticulation number* of a phylogenetic network $N = (V,E)$ as

$$\sum_{v \in V : \delta^-(v) > 0} (\delta^-(v) - 1) = |E| - |V| + 1.$$

A directed acyclic graph is *connected* (also called 'weakly connected') if there is an undirected path (ignoring edge

orientations) between each pair of nodes. A node (edge) of a directed graph is called a *cut-node* (*cut-edge*) if its removal disconnects the graph. A directed graph is *biconnected* if it contains no cut-nodes. A biconnected subgraph B of a directed graph G is said to be a *biconnected component* if there is no biconnected subgraph $B' \neq B$ of G that contains B.

A phylogenetic network is said to be a *level-k network* if each biconnected component has reticulation number at most k.[3] A phylogenetic network is called *binary* if each node has either indegree at most 1 and outdegree at most 2 or indegree at most 2 and outdegree at most 1. Note that the above definition of level generalizes the original definition (Choy *et al*., 2005) for binary networks. A level-k network is called a *simple level-$\leq k$ network* if the head of each cut-edge is a leaf. A simple level-$\leq k$ network is called a *simple level-k network* if its reticulation number is precisely k. For example, Figure 2a is a simple level-4 network and Figure 2b is a simple level-2 network. A *phylogenetic tree* (on $\mathcal{X}$) is a phylogenetic network (on $\mathcal{X}$) without reticulations, i.e. a level-0 network.

Consider a set of taxa $\mathcal{X}$. Proper subsets of $\mathcal{X}$ are called *clusters*. We say that two clusters $C_1, C_2 \subset \mathcal{X}$ are *compatible* if either $C_1 \cap C_2 = \emptyset$ or $C_1 \subset C_2$ or $C_2 \subset C_1$. Consider a set of clusters $\mathcal{C}$. We say that a set of taxa $X \subseteq \mathcal{X}$ is *separated* (by $\mathcal{C}$) if there exists a cluster $C \in \mathcal{C}$ that is incompatible with X. The *incompatibility graph IG*($\mathcal{C}$) of $\mathcal{C}$ is the undirected graph (V, E) that has node set $V = \mathcal{C}$ and edge set

$$E = \{\{C_1, C_2\} \mid C_1 \text{ and } C_2 \text{ are incompatible clusters in } \mathcal{C}\}.$$

3 DECOMPOSING LEVEL-*K* NETWORKS

In this section, we describe the general outline of our algorithm Cass. We show how the problem of determining a level-k network can be decomposed into a set of smaller problems by examining the incompatibility graph. Our algorithm will first construct a simple level- $\leq$k network for each connected component of the incompatibility graph and subsequently merge these simple level-$\leq k$ networks into a single level-k network on all taxa.

We first give a formal description of the algorithm, which is illustrated by an example in Figure 3. After that we will explain why we can use this approach.

Consider a set of taxa $\mathcal{X}$ and a set $\mathcal{C}$ of input clusters. We assume that all singletons (sets $\{x\}$ with $x \in \mathcal{X}$) are clusters in $\mathcal{C}$. Our algorithm proceeds as follows.

Step 1. Find the non-trivial connected components $\mathcal{C}_1, \ldots, \mathcal{C}_p$ of the incompatibility graph IG($\mathcal{C}$). For each $i \in \{1, \ldots, p\}$, let $\mathcal{C}_i'$ be the result of collapsing unseparated sets of taxa as follows. Let $\mathcal{X}_i = \bigcup_{C \in \mathcal{C}_i} C$. For each maximal subset $X \subset \mathcal{X}_i$ that is not separated by $\mathcal{C}_i$, replace, in each cluster in $\mathcal{C}_i$, the elements of X by a single new taxon X, e.g. if $X = \{b, c\}$ then a cluster $\{a, b, c, d\}$ is modified to $\{a, \{b, c\}, d\}$.

Step 2. For each $i \in \{1, \ldots, p\}$, construct a simple level-$\leq k$ network N_i representing $\mathcal{C}_i'$.

Step 3. Let $\mathcal{C}^*$ be the result of applying the following modifications to $\mathcal{C}$, for each $i \in \{1, \ldots, p\}$: remove all clusters that are in $\mathcal{C}_i$, add a cluster $\mathcal{X}_i$ and add each maximal subset $X \subset \mathcal{X}_i$ that is not separated by $\mathcal{C}_i$. Construct the unique phylogenetic tree T on $\mathcal{X}$ representing precisely those clusters in $\mathcal{C}^*$. (Notice that each trivial connected component of the incompatibility graph is also a cluster in $\mathcal{C}^*$.)

Step 4. For each $i \in \{1, \ldots, p\}$, replace in T the lowest common ancestor v_i of $\mathcal{X}_i$ by the simple level-$\leq k$ network N_i as follows. Delete all edges leaving v_i and merge T with N_i by identifying the root of N_i with v_i and identifying each leaf of N_i labeled X by the lowest common ancestor of the leaves labeled X in T. Output the resulting network.

Notice that Steps 1, 3 and 4 are similar to the corresponding steps in algorithms for constructing galled trees (i.e. level-1 networks) and galled networks (Huson and Klöpper, 2007; Huson *et al*., 2009; D.H.Huson *et al*., submitted for publication). The reason why we use the same set-up in our algorithm, is outlined by Theorem 1. It shows that, when constructing a level-k network displaying a set of clusters, we can restrict our attention to level-k networks that satisfy the *decomposition property* (D.H.Huson *et al*., submitted for publication), which intuitively says that the biconnected components of the network correspond to the connected components of the incompatibility graph. We now repeat the formal definition.

Since a cluster $C \in \mathcal{C}$ can be represented by more than one edge in a network N, an *edge assignment* ϵ is defined as a mapping that chooses for each cluster $C \in \mathcal{C}$ a single tree edge $\epsilon(C)$ of N that represents C. A network N representing $\mathcal{C}$ is said to satisfy the *decomposition property* w.r.t. $\mathcal{C}$ if there exists an edge assignment ϵ such that:

- for any two clusters $C_1, C_2 \in \mathcal{C}$, the edges $\epsilon(C_1)$ and $\epsilon(C_2)$ are contained in the same biconnected component of N if and only if C_1 and C_2 lie in the same connected component of the incompatibility graph IG($\mathcal{C}$).

Theorem 1. *Let $\mathcal{C}$ be a set of clusters. If there exists a level-k network representing $\mathcal{C}$, then there also exists such a network satisfying the decomposition property w.r.t. $\mathcal{C}$.*

Proof. Let $\mathcal{C}$ be a set of input clusters and N a level-k network representing $\mathcal{C}$. Let $\mathcal{C}_1, \ldots, \mathcal{C}_p$ be the non-trivial connected components of the incompatibility graph IG($\mathcal{C}$). For each $i \in \{1, \ldots, p\}$, we construct a simple level-$\leq k$ network N_i as follows. Let $\mathcal{X}_i = \bigcup_{C \in \mathcal{C}_i} C$ as before. For each maximal subset $X \subset \mathcal{X}_i$ (with $|X| > 1$) that is not separated by $\mathcal{C}_i$, replace in N an arbitrary leaf labeled by an element of X by a leaf labeled X and remove all other leaves labeled by elements of X. In addition, remove all leaves with labels that are not in X_i. We tidy up the resulting graph by repeatedly applying the following five steps until none is applicable: (i) delete unlabeled nodes with outdegree 0; (ii) suppress nodes with indegree and outdegree 1 (i.e. contract one edge incident to the node); (iii) replace multiple edges by single edges, (iv) remove the root if it has outdegree 1 and (v) contract biconnected components that have only one outgoing edge. This leads to a level-k network N_i. Let $\mathcal{C}_i'$ be defined as in Step 1 of the algorithm. By its construction, N_i represents $\mathcal{C}_i'$. Furthermore, N_i is a simple level-$\leq k$ network, because if it would contain a cut-edge e whose head is not a leaf, then the set of taxa labeling leaves reachable from e would not be separated by $\mathcal{C}_i'$ and would hence have been collapsed. Finally, the networks $N_1, \ldots, N_p$ can be merged into a level-k network representing $\mathcal{C}$ and satisfying the decomposition property by executing Steps 3 and 4 of the algorithm. ■

Intuitively, Theorem 1 tells us that whenever there exists a level-k network N representing $\mathcal{C}$, there also exists such a network N' whose

[3]Note that to determine the reticulation number of a biconnected component one only counts edges inside this biconnected component.

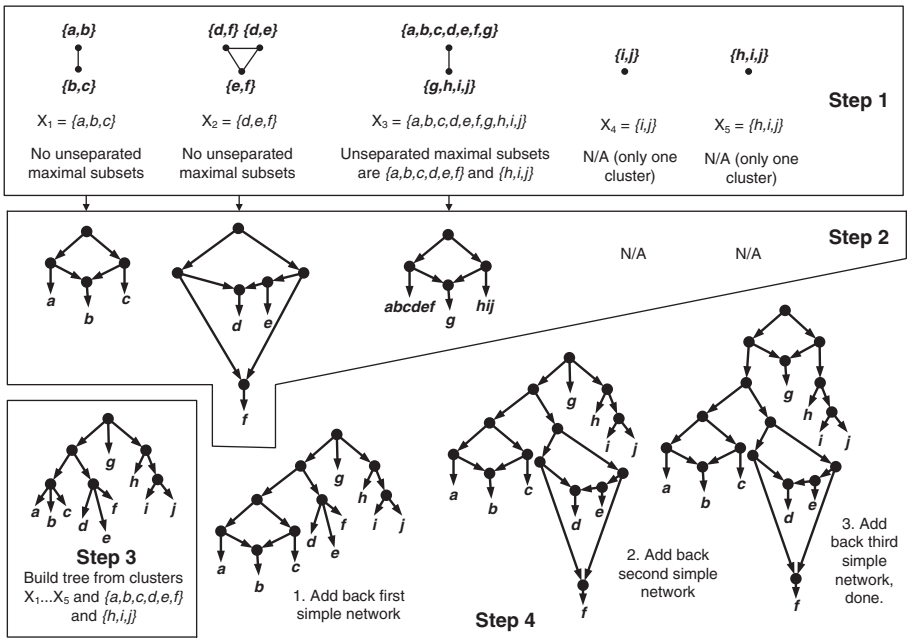

Fig. 3. How the four-step decomposition algorithm of CASS (Section 3) constructs a level-2 network from the clusters $\{a,b\}$, $\{b,c\}$, $\{d,e\}$, $\{d,f\}$, $\{e,f\}$, $\{a,b,c,d,e,f,g\}$, $\{g,h,i,j\}$, $\{i,j\}$, $\{h,i,j\}$. Section 4 describes how the simple networks in Step 2 are created.

biconnected components correspond to the connected components of the incompatibility graph. Since N' has level k, each biconnected component has level at most k. Hence, we can construct a simple level-$\leq k$ network for each connected component of the incompatibility graph. Subsequently, we can merge these simple level-$\leq k$ networks into a level-k network representing $\mathcal{C}$. This is precisely what the set-up described above does.

Note finally that the statement obtained by replacing 'level-k network' by 'network with k reticulations' in Theorem 1 does not hold, as shown in Huson *et al*. (2009), based on Gusfield *et al*. (2007a).

4 SIMPLE LEVEL-*K* NETWORKS

This section describes how one can construct a simple level-k network representing a given set of clusters. We say that a phylogenetic tree T is a *strict subtree* of a network N if T is a subgraph of N and for each node v of T, except its root, it holds that the in- and outdegree of v in T are equal to the in- and (respectively) outdegree of v in N.

Informally, our method for constructing simple level-k networks operates as follows. See Figure 4 for an example. CASS loops over all taxa x. For each choice of x, CASS removes it from each cluster and subsequently collapses all maximal 'ST-sets' ('strict tree sets', defined below) of the resulting cluster set. The algorithm repeats this step k times, after which all leaves will be collapsed into just two taxa and the second phase of the algorithm starts. CASS creates a network consisting of a root with two children, labeled by the only two taxa. Then the algorithm 'decollapses', i.e. it replaces each leaf labeled by an ST-set by a strict subtree. Subsequently, CASS adds a new leaf below a new reticulation and labels it by the latest removed taxon. Since it does not know where to create the new reticulation, CASS tries adding the reticulation below each pair of edges. The algorithm continues with a new decollapse step followed by hanging the next leaf below a reticulation. These steps are also repeated k times. For each constructed simple level-k network, CASS checks whether it represents all input clusters. If it does, the algorithm outputs the resulting network, after contracting any edges that connect two reticulations.

The idea behind this strategy is as follows. Observe that any simple level-k network N ($k \geq 1$) contains a leaf whose parent is a reticulation (since we assume that each reticulation has outdegree at least 1). If we remove this leaf and reticulation from N, the resulting network might contain one or more strict subtrees. To reconstruct the network, we need to identify these strict subtrees from the set of clusters. We will see below that each strict subtree corresponds to an ST-set. Moreover, for the case $k \leq 2$, we prove that (without loss of generality) each maximal strict subtree corresponds to a *maximal* ST-set. CASS collapses the maximal ST-sets because it assumes that these correspond to the strict subtrees. Now observe that collapsing each maximal strict subtree of the network leads to a (not necessarily simple) level-$(k-1)$ network, which again contains a leaf whose parent is a reticulation. It follows that it is indeed possible to repeat the described steps k times. Finally, CASS checks if all clusters are represented and only outputs networks for which this is the case.

Let us now formalize this algorithm. Given a set $S \subseteq \mathcal{X}$ of taxa, we use $\mathcal{C} \setminus S$ to denote the result of removing all elements of S from each cluster in $\mathcal{C}$ and we use $\mathcal{C}|S$ to denote $\mathcal{C} \setminus (\mathcal{X} \setminus S)$ (the restriction of $\mathcal{C}$ to S). We say that a set $S \neq \mathcal{X}$ is an *ST-set* (strict tree set) w.r.t. $\mathcal{C}$, if S is not separated by $\mathcal{C}$ and any two clusters $C_1, C_2 \in \mathcal{C}|S$ are compatible. An ST-set S is *maximal* if there is no ST-set T with $S \subsetneq T$. Informally, the maximal ST-sets are the result of repeatedly collapsing pairs of unseparated taxa for as long as possible.

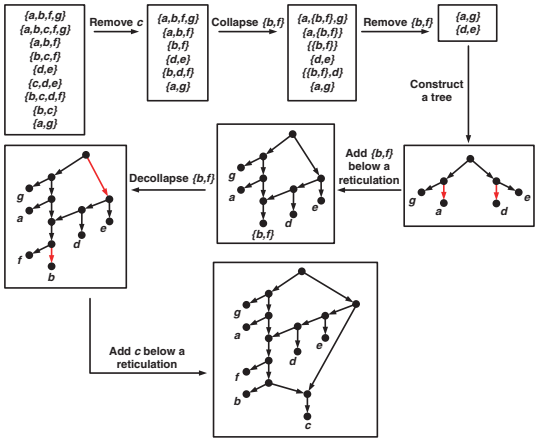

Fig. 4. Construction of a simple level-2 network by the CASS algorithm. The edges e_1, e_2 that will be subdivided are colored red. Singleton clusters have been omitted, as well as the last collapse step, for simplicity.

We use COLLAPSE($\mathcal{C}$) to denote the result of collapsing each maximal ST-set S into a single taxon S. More precisely, for each cluster $C \in \mathcal{C}$ and maximal ST-set S of $\mathcal{C}$, we replace C by $C \setminus S \cup \{\{S\}\}$. For example (omitting singleton clusters), if

$$\mathcal{C} = \{\ \{1,2\},\ \{2,3,4\},\ \{3,4\}\ \},$$

then $\{3,4\}$ is the only non-singleton maximal ST-set and

$$\text{COLLAPSE}(\mathcal{C}) = \{\ \{1,2\},\ \{2,\{3,4\}\}\ \}.$$

The set of taxa of a (collapsed) cluster set $\mathcal{C}$ is denoted $\mathcal{X}(\mathcal{C})$. Thus, for the above example, $\mathcal{X}(\text{COLLAPSE}(\mathcal{C})) = \{1,2,\{3,4\}\}$. We are now ready to give the pseudocode of CASS(k) in Algorithm 1. The actual implementation is slightly more complex and much more space efficient.

Figure 4 shows how the CASS(2) algorithm, for example, constructs a simple level-2 network. We will now show that CASS(1) and CASS(2) will indeed construct a simple level-1, respectively, level-2 network whenever this is possible.

LEMMA 1. *Given a set of clusters $\mathcal{C}$, such that IG($\mathcal{C}$) is connected and any $X \subsetneq \mathcal{X}$ is separated,* CASS(1) *and* CASS(2) *construct a simple level-1, respectively, a simple level-2 network representing $\mathcal{C}$, if such a network exists.*

PROOF. The general idea of the proof is as follows. Details have been omitted due to space constraints. Assume $k \leq 2$. It is clear that any (simple) level-k network N contains a reticulation r with a leaf, say labeled x, as child. Let $N \setminus \{x\}$ denote the network obtained by removing the reticulation r and the leaf labeled x from N. This network might contain one or more strict subtrees. By the definition of ST-set, the set of leaf labels of each maximal strict subtree corresponds to an ST-set w.r.t. $\mathcal{C} \setminus \{x\}$. However, in general not each such set needs to be a *maximal* ST-set. This is critical, because the total number of ST-sets can be exponentially large. Therefore, the main ingredient of our proof is the following. We show that whenever there exists a simple level-k network representing $\mathcal{C}$, there exists a simple level-k network N' representing $\mathcal{C}$ such that the sets of leaf-labels of the maximal strict subtrees of $N' \setminus \{x\}$ are the maximal ST-sets w.r.t. $\mathcal{C} \setminus \{x\}$, with x the label of some leaf whose parent is a reticulation in N'. This is clearly true for $k = 1$. For $k = 2$, we sketch our proof below.

Algorithm 1 CASS(k): constructing a simple level-k network from clusters

input $(\mathcal{C}, \mathcal{X}, k, k')$
output CASS$(\mathcal{C}, \mathcal{X}, k, k')$
// in the initial call to the algorithm, $k' = k$
$\mathcal{N} := \emptyset$
if $k' = 0$ **then**
  return the unique tree representing exactly those clusters in $\mathcal{C}$ or return $\emptyset$ if no such tree exists
for $x \in \mathcal{X} \cup \{\delta\}$ **do**
  // δ is a dummy taxon not in $\mathcal{X}$
  remove leaf: $\mathcal{C}' := \mathcal{C} \setminus \{x\}$
  collapse: $\mathcal{C}'' := $ COLLAPSE$(\mathcal{C}')$
  recurse: $\mathcal{N}' :=$ CASS$(\mathcal{C}'', \mathcal{X}(\mathcal{C}''), k, k'-1)$
  for each network N' in $\mathcal{N}'$ **do**
    decollapse: replace each leaf of N' labeled by a maximal ST-set S w.r.t. $\mathcal{C}'$ by the tree on S representing exactly those clusters in $\mathcal{C}'|S$
    for each pair of edges e_1, e_2 (not necessarily distinct) **do**
      let N'' be a copy of N'
      add leaf below reticulation: create in N'' a reticulation t, a leaf l labeled x and an edge from t to l; then, for $i = 1, 2$, insert in N'' a node v_i into e_i and add an edge from v_i to t;
      if N'' represents $\mathcal{C}$ **then**
        save network: $\mathcal{N} := \mathcal{N} \cup \{N''\}$
if $k = k'$ **then**
  return any simple level-k network in $\mathcal{N}$, after removing each leaf labeled δ and contracting each edge connecting two reticulations
else
  return $\mathcal{N}$

Let us first mention that the actual algorithm is slightly more complicated than the pseudocode in Algorithm 1. First, when CASS(k) constructs a tree, it adds a new 'dummy' root to this tree and creates an edge from this dummy root to the old root. Such a dummy root is removed before outputting a network. Second, whenever the algorithm removes a dummy taxon δ (which we use to model the situation when the previous leaf removal caused more than one reticulation to disappear), it makes sure that it does not collapse in the previous step.

Suppose there exists some level-2 network representing $\mathcal{C}$. It can be shown that any such network is simple and that there exists at least one binary such network, say N. Since N is a binary simple level-2 network, there are only four possibilities for the structure of N (after removing leaves), see van Iersel *et al.* (2009a). These structures are called *generators*. In each case, $N \setminus \{x\}$ contains at most two maximal strict subtrees that have more than one leaf. Furthermore, $N \setminus \{x\}$ contains exactly one reticulation r', below which hangs a strict subtree T_r with set of leaf labels X_r (possibly, $|X_r| = 1$ or $|X_r| = 0$).

First, we assume that X_r is not a maximal ST-set w.r.t. $\mathcal{C} \setminus \{x\}$. In that case it follows that there is some maximal ST-set X that contains X_r and also contains at least one taxon labeling a leaf ℓ that is not reachable by a directed path from the reticulation of $N \setminus \{x\}$. We can replace ℓ by a strict subtree on X that represents $\mathcal{C}|X$. Such

a tree exists because X is an ST-set. We remove all leaves that label elements of X and are not in this strict subtree. Since there are now no leaves left below the reticulation, we can remove this reticulation as well. It is easy to see that the resulting network is a tree representing $\mathcal{C}\setminus\{x\}$. Moreover, we show that in each case a leaf labeled x can be added below a new reticulation (possibly with indegree 3) in order to obtain a network N' that represents $\mathcal{C}$. Since N' contains just one reticulation, it is clear that the maximal strict subtrees of $N'\setminus\{x\}$ are the maximal ST-sets w.r.t. $\mathcal{C}\setminus\{x\}$. Cass(2) reconstructs such a network with an indegree-3 reticulation by removing x, removing a dummy taxon δ, constructing a tree, adding a leaf labeled δ below a reticulation, adding a leaf labeled x below a reticulation, removing the leaf labeled δ and contracting the (now redundant) edges between the two reticulations. Note that this works because Cass(2) does not collapse in this case.

It remains to consider the possibility that X_r is a maximal ST-set w.r.t. $\mathcal{C}\setminus\{x\}$. In this case, we modify network N to N' in such a way that also the other maximal ST-sets w.r.t. $\mathcal{C}\setminus\{x\}$ appear as the leaf-sets of strict subtrees in $N'\setminus\{x\}$. We again use a case analysis to show that this is always possible in such a way that the resulting network N' represents $\mathcal{C}$. ■

Lemma 2. Cass *runs in time* $O(|\mathcal{X}|^{3k+2}\cdot|\mathcal{C}|)$*, if* k *is fixed.*

Proof. Omitted due to space constraints. ■

Theorem 2. *Given a set of clusters* $\mathcal{C}$*,* Cass *constructs in polynomial time a level-2 network representing* $\mathcal{C}$*, if such a network exists.*

Proof. Follows from Lemmas 1 and 2 and Theorem 1. ■

We conclude this section by showing that for each $r\geq 2$, there exists a set of clusters $\mathcal{C}_r$ such that any galled network representing $\mathcal{C}_r$ needs at least r reticulations, while Cass constructs a network with just two reticulations, which also represents $\mathcal{C}_r$. This follows from the following lemma.

Lemma 3. *For each* $r\geq 2$*, there exists a set* $\mathcal{C}_r$ *of clusters such that there exists a network with two reticulations that represents* $\mathcal{C}_r$ *while any galled network representing* $\mathcal{C}_r$ *contains at least* r *reticulations.*

Proof. Omitted due to space constraints. ■

5 PRACTICE

Our implementation of the Cass algorithm is available as part of the Dendroscope program (Huson *et al*., 2007). To use Cass, first load a set of trees into Dendroscope. Subsequently, run the algorithm by choosing 'options' and 'network consensus'. The program gives you the option of entering a threshold percentage t. Only clusters that appear in more than t percent of the input trees will be used as input for Cass. Choose 'minimal network' to run the Cass algorithm to construct a phylogenetic network representing all clusters that appear in more than t percent of the input trees.

Cass computes a solution for each biconnected component separately. If the computations for a certain biconnected component take too long, you can choose to 'skip' the component, in which case the program will quickly compute the cluster network (Huson and Rupp, 2008) for this biconnected component, instead. Alternatively, you can choose to construct a galled network, or to increase the threshold percentage t. See van Iersel *et al*. (2009b) for a user guide for Cass and all datasets used for this article. See Huson *et al*. (2007) for more information on using Dendroscope.

Table 1. Results of Cass compared with GalledNetwork for several example cluster sets with $|\mathcal{C}|$ clusters and $|\mathcal{X}|$ taxa

Data		GalledNetwork			Cass		
$\lvert\mathcal{C}\rvert$	$\lvert\mathcal{X}\rvert$	t	k	r	t	k	r
30	5	0 s	6	6	1 s	4	4
62	6	0 s	8	8	7 s	5	5
126	7	0 s	10	10	28 s	6	6
254	8	6 s	12	12	4 m 3 s	7	7
42	10	0 s	4	4	6 s	4	4
38	11	0 s	7	7	14 s	5	5
61	11	0 s	6	6	47 s	5	5
77	22	0 s	9	9	36 s	3	3
75	30	0 s	11	11	5 s	2	2
89	31	0 s	16	16	27 m 32 s	4	4
180	51	0 s	11	11	30 s	2	2
193	57	0 s	1	4	1 s	1	4
270	76	0 s	16	16	4 m 52 s	2	2
404	122	1 s	2	2	21 m 10 s	2	2
135.8	**31.9**	**1s**	**8.5**	**8.7**	**4 m 19 s**	**3.7**	**3.9**

For each algorithm, the level k and reticulation number r of the output network are given as well as the running time t in minutes (m) and seconds (s) on a 1.67 GHz 2 GB laptop. The last row gives the average values.

We have tested Cass on both practical and artificial data and compared Cass with other programs. The results (using $t=0$) are summarized in Table 1 and Figure 5. For Table 1, several example datasets have been used, which have been selected in such a way as to obtain a good variation in number of taxa, number of clusters and network complexity. The first four datasets are the sets containing *all* possible clusters on 5, 6, 7 and 8 taxa, respectively. The other datasets have been constructed by taking the set of clusters in (arbitrary) networks of increasing size and complexity. Mostly networks with just one biconnected component have been used because, for networks with more biconnected components, both algorithms use the same method to decompose into biconnected components and then both find a solution for each biconnected component separately. For each dataset, we have constructed one network using Cass, which we call the Cass network, and one galled network using the algorithm in Huson *et al*. (2009). Two conclusions can be drawn from the results. First, Cass uses more time than the galled network algorithm. Nevertheless, the time needed by Cass can still be considered acceptable for phylogenetic analysis. Second, Cass constructs a much simpler network in almost all cases. For three datasets, the Cass network and the galled network have the same reticulation number and the same level. For all other datasets, the Cass network has a significantly smaller reticulation number, and also a lower level, than the galled network.

Figure 5 summarizes the results of an application of Cass to practical data. This dataset consists of six phylogenetic trees of grasses of the *Poaceae* family, originally published by the Grass Phylogeny Working Group (2001) and reanalyzed in Schmidt (2003). The phylogenetic trees are based on sequences from six different gene loci, ITS, ndhF, phyB, rbcL, rpoC and waxy, and

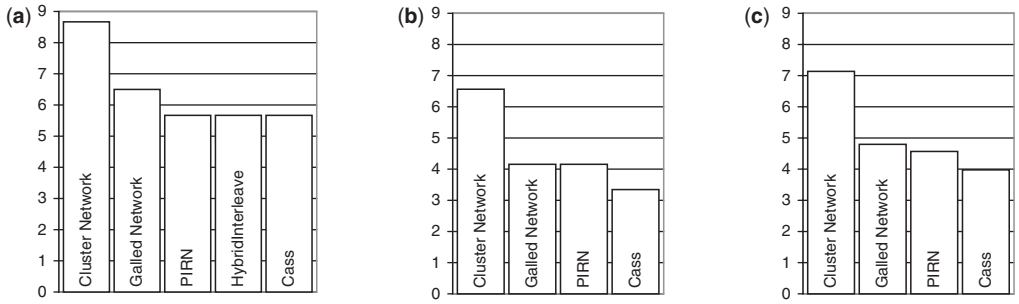

Fig. 5. (**a**) The average number of reticulations used by the compared programs, ranging over all combinations of two gene trees from the six trees in the *Poaceae* grass dataset, and restricted to those combinations for which all programs terminated within 5 min. (**b**) As in (a), but ranging over combinations containing three or more gene trees. (**c**) As in (a), but ranging over all combinations of two or more trees.

contain 47, 65, 40, 37, 34 and 19 taxa, respectively. We have compared the results of Cass not only with the galled network and the cluster network algorithms, but also with the very recently developed algorithms HybridInterleave (J.Collins *et al.*, submitted for publication) and PIRN (Y.Wu, submitted for publication). HybridInterleave computes the minimum number of reticulations required to combine two binary phylogenetic trees (on the same set of taxa) into a phylogenetic network that displays both trees. PIRN has the same objective as HybridInterleave but has the advantage that it can accept more than two trees as input (which are still required to be binary). On the other hand, HybridInterleave has the advantage that it is guaranteed to find an optimal solution. For this experiment, we compiled PIRN with the ILP (Integer Linear Programming) solver CPLEX 10.2. We considered all possible subsets of at least two of the six gene trees; 57 in total. For each subset, we first restricted the trees to the taxa present in all trees in the subset to make the input data compatible with HybridInterleave and PIRN. Then, we executed each program for a maximum of 5 min on a 2.83 GHz quad-core PC with 8 GB RAM and recorded the best solution it could find in that time frame. The full results are available in Table 2 in the Supplementary Material. Results for HybridInterleave (which could only be applied to pairs of trees) differ from the results reported in (J.Collins *et al.*, submitted for publication) because there trees with a different rooting were used. Our results show that Cass always found a solution (within 5 min) when the minimum level was at most 4, and sometimes when the minimum level was 5 or 6. We also see that, in all these cases, no program found a solution using fewer reticulations than Cass.

To obtain each of the graphs in Figure 5, we averaged over those subsets where all the programs had terminated within 5 min (which was the majority). Several conclusions can be drawn from these graphs. The main conclusion is that Cass on average required fewer reticulations than each of the other programs. That Cass uses fewer reticulations than PIRN can be explained by the fact that PIRN (as well as HybridInterleave) requires the output network to display all input trees. The networks constructed by Cass do not necessarily display the input trees, but still represent all clusters from the trees, and in general use fewer reticulations to do so. Figure 5a is noteworthy in this regard. It turns out that, when restricted to subsets of exactly two trees, Cass, PIRN and HybridInterleave always achieved the same optimum. This turns out not to be coincidence, but a mathematical consequence of extracting clusters from exactly two binary trees on the same taxa set (L.J.J.van Iersel and S.M.Kelk, in preparation). The advantages of Cass clearly become most visible when larger subsets of trees are used.

In terms of running time, PIRN and HybridInterleave are in general faster than Cass, but Cass has the significant flexibility that it is not restricted to binary (i.e. fully resolved) input trees and is not restricted to trees on the same taxa set. Compared with HybridInterleave, Cass also has the advantage that it is not restricted to two input trees and that it constructs an actual network rather than to only report the number of reticulations. Finally, because Cass is not restricted to binary trees, the user is free to choose only well-supported clusters from the input trees. Figure 6 is a nice example of this: this is the output of Cass when given all clusters that were in at least three of the six gene trees (i.e. $t = 34\%$), without having to first restrict to those taxa common to all six trees (in this case, only four taxa were common to all six input trees). This example also illustrates that, when there exists a solution with a low level, Cass can handle large numbers of taxa and reticulations.

6 DISCUSSION

We have introduced the Cass algorithm, which can be used to combine any set of clusters into a phylogenetic network representing those clusters. We have shown that the algorithm performs well on practical data. It provides a useful addition to existing software, because it usually constructs a simpler network representing the same set of input clusters. Furthermore, we have shown that Cass provides a polynomial-time algorithm for deciding whether a level-2 phylogenetic network exists that represents a given set of clusters. This algorithm is more useful in practice than algorithms for similar problems that take triplets as input (Jansson and Sung, 2006; Jansson *et al.*, 2006; To and Habib, 2009; van Iersel and Kelk, 2009; van Iersel *et al.*, 2009a), because clusters are more biologically relevant than triplets and because the latter algorithms need at least one triplet for each combination of three taxa as input, while Cass can be used for any set of input clusters. Furthermore, Cass is also not restricted to two input trees, as the algorithms in Bordewich *et al.* (2007); J.Collins *et al.*, (submitted for publication); Linz and Semple (2009) and not to fully resolved trees on identical taxa sets, as the algorithms in Bordewich *et al.* (2007), J.Collins *et al.* (submitted for publication), Y.Wu *et al.* (submitted for publication). Finally, we remark that Cass can also be used when one or more multi-labeled trees are given as input. In this case, Dendroscope first computes

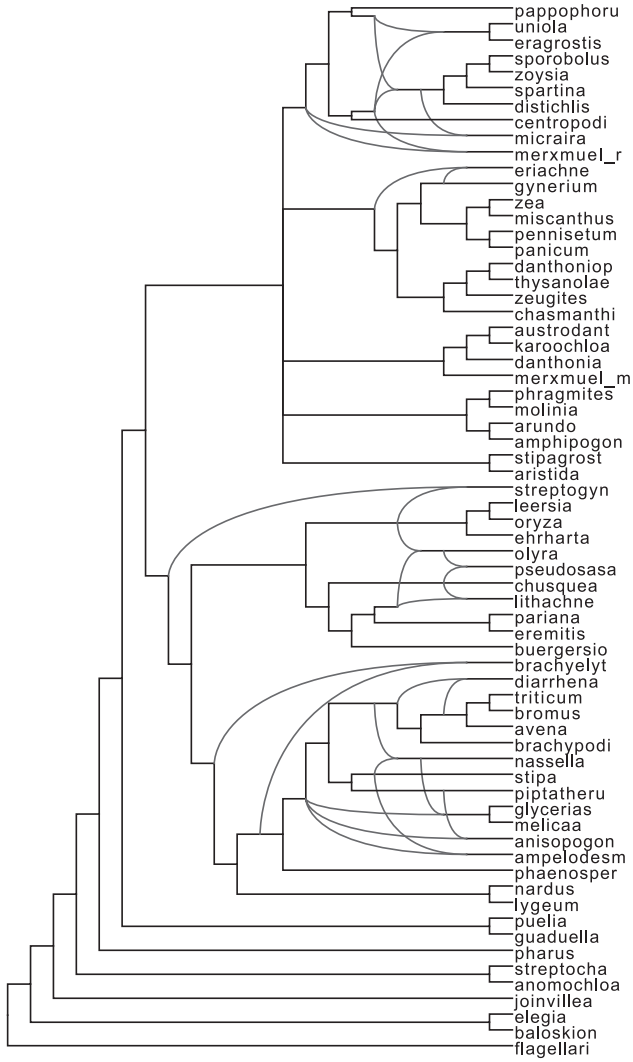

Fig. 6. Level-4 network with 66 taxa and 15 reticulations constructed by Cass for the six gene trees of the *Poaceae* grass dataset, within 5 s. Clusters were used that were present in at least three of the six gene trees. For the same input GalledNetwork produced a level-5 network with 17 reticulations, and the cluster network algorithm produced a level-11 network with 32 reticulations.

all clusters in the multi-labeled tree(s) and subsequently uses Cass to find a phylogenetic network representing these clusters. Several theoretical problems remain open. First of all, does Cass always construct a minimum-level network, even if this minimum is three or more? Second, what is the complexity of constructing a minimum level network, if the minimum level k is not fixed but part of the input? Is this problem FPT when parameterized by k? Finally, it would be interesting to design an algorithm that finds a network representing a set of input clusters that has a minimum reticulation number. So far, not even a non-trivial exponential-time algorithm is known for this problem.

ACKNOWLEDGEMENTS

We thank Mike Steel for organizing the Cass workshop in the Cass Field Station in February 2009, where we started this work. We thank Yufeng Wu for providing us with the source code for his PIRN program.

Funding: Allan Wilson Centre for Molecular Ecology and Evolution (to L.V.I.); Computational Life Sciences grant of The Netherlands Organisation for Scientific Research (NWO to S.K.); Regula Rupp by the Deutsche Forschungsgemeinschaft (PhyloNet project to S.K.).

Conflict of Interest: none declared.

REFERENCES

Bordewich,M. and Semple,C. (2007) Computing the minimum number of hybridization events for a consistent evolutionary history. *Discrete Appl. Math.*, **155**, 914–928.

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

Choy,C. *et al*. (2005) Computing the maximum agreement of phylogenetic networks. *Theor. Comput. Sci.*, **335**, 93–107.

Gambette,P. (2009) Who's who in phylogenetic networks, 2009. Available at http://www.lirmm.fr/~gambette/PhylogeneticNetworks/ (last accessed date April 24, 2010).

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

Gusfield,D. *et al*. (2007a) A decomposition theory for phylogenetic networks and incompatible characters. *J. Comput. Biol.*, **14**, 1247–1272.

Gusfield,D. *et al*. (2007b) An efficiently computed lower bound on the number of recombinations in phylogenetic networks: theory and empirical study. *Discrete Appl. Math.*, **155**, 806–830.

Huson,D.H. and Klöpper,T.H. (2007) Beyond galled trees - decomposition and computation of galled networks. In *Research in Computational Molecular Biology (RECOMB)*, Vol. 4453 of *Lecture Notes in Computer Science*, Springer, Berlin, pp. 221–225.

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. *et al*. (2007) Dendroscope: an interactive viewer for large phylogenetic trees. *BMC Bioinformatics*, **8**, 460.

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

Huynh,T. *et al*. (2005) Constructing a smallest refining galled phylogenetic network. In *Research in Computational Molecular Biology (RECOMB)*, Vol. 3500 of *Lecture Notes in Bioinformatics*, Springer, Berlin, pp. 265–280.

Jansson,J. and Sung,W.-K. (2006) Inferring a level-1 phylogenetic network from a dense set of rooted triplets. *Theor. Comp. Sci.*, **363**, 60–68.

Jansson,J. *et al*. (2006) Algorithms for combining rooted triplets into a galled phylogenetic network. *SIAM J. Comput.*, **35**, 1098–1121.

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

Nakleh,L. (2009) Evolutionary phylogenetic networks: models and issues. In Heath,L. and Ramakrishnan,N. (eds), *The Problem Solving Handbook for Computational Biology and Bionformatics*, Springer.

Schmidt,H.A. (2003) Phylogenetic trees from large datasets. PhD Thesis, Heinrich-Heine-Universität, Düsseldorf.

Semple,C. (2007) Hybridization networks. In Gascuel,O. and Steel,M. (eds), *Reconstructing Evolution - New Mathematical and Computational Advances*, Oxford University Press, pp. 277–314.

To,T.-H. and Habib,M. (2009) Level-k phylogenetic networks are constructable from a dense triplet set in polynomial time. In *Combinatorial Pattern Matching (CPM)*, Vol. 5577 of *Lecture Notes in Computer Science*, pp. 275–288.

van Iersel,L.J.J. and Kelk,S.M. (2009) Constructing the simplest possible phylogenetic network from triplets. *Algorithmica* [Epub ahead of print, doi:10.1007/s00453-009-9333-0, July 7, 2009].

van Iersel,L.J.J. *et al*. (2009a) Constructing level-2 phylogenetic networks from triplets. *IEEE/ACM Trans. Comput. Biol. Bioinform.*, **6**, 667–681.

van Iersel,L.J.J. *et al*. (2009b) Cass: Combining phylogenetic trees into a phylogenetic network, Available at http://sites.google.com/site/cassalgorithm/ (last accessed date April 24, 2010).