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

Leo van Iersel1,∗, Steven Kelk2, Regula Rupp3 and Daniel Huson3

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

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: one 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.

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 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
phyllogenetic 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, 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 ∈ 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, which takes any set C of clusters as input and constructs a phylogenetic network that represents 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 C whenever such a tree exists.

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

Moreover, we prove that Cass constructs a level-1 or level-2 network representing C whenever there exists a level-1 or level-2 network representing C, respectively. Experimental results show that also when no level-2 network representing 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 Cass 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 X of taxa. A rooted (phylogenetic) network (on X) is a directed acyclic graph with a single root and leaves bijectively labeled by X. The indegree of a node v is denoted δ+(v) and v is called a reticulation if δ+(v) ≥ 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

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

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 \( k \). 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 and Huson, 2008) for binary networks. A level-\( k \) network is called a simple level-\( k \) network if it has only one outgoing edge. This leads to 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, an edge assignment \( \epsilon \) is defined as a mapping that chooses for each cluster \( C \in \mathcal{C} \) a single tree edge \( \epsilon(C) \) that represents \( C \). A network \( N \) representing \( \epsilon \) is said to satisfy the decomposition property with respect to \( \mathcal{C} \) if it exists an edge assignment \( \epsilon \) 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 \( IG(C) \).

**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 \( C_1, \ldots, C_p \) be the non-trivial connected components of the incompatibility graph \( IG(C) \). For each \( i \in \{1, \ldots, p\} \), we construct a simple level-\( k \) network \( N_i \) as follows. Let \( X_i = \bigcup_{C \in C_i} C \) as before. For each maximal subset \( X \subseteq X_i \) (with \( |X| > 1 \)) that is not separated by \( C_0 \), 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 \( X \). 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 leaf if it has outdegree 1 and (v) contract biconnected components that have only one outgoing edge. This leads to a level-\( k \) network \( N \). Let \( C_i^* \) be defined as in Step 1 of the algorithm. By its construction, \( N_i \) represents \( C_i^* \). Furthermore, \( N_i \) is a simple level-\( k \) 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 \( C_i^* \) 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 representing \( \mathcal{C} \), there also exists such a network \( N' \) whose
Then the algorithm ‘decollapses’, i.e. it replaces each leaf labeled a hold, as shown in Huson et al network’ by ‘network with k \leq 4 SIMPLE LEVEL-phylogenetic tree T network representing a given set of clusters. We say that a k 4 SIMPLE LEVEL-of the incompatibility graph. Since Fig. 3. subgraph of the in- and outdegree of outdegree of 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 = 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 C(S) to denote the result of removing all elements of S from each cluster in \mathcal{C} and we use C(S) to denote C(S) \setminus S (the restriction of \mathcal{C} to S). We say that a set S 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 C(S) are compatible. An ST-set 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. 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\}, \{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-k network for each connected component of the incompatibility graph. Subsequently, we can merge these simple level-k networks into a level-k network representing 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 Gustfield 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
The actual implementation is slightly more complex and much more complicated than the pseudocode in Algorithm 1. First, when \( \text{Cass}(k) \) 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 \( t \) not in \( X \), let \( N' \) be a copy of \( N' \) and an edge from \( t \) to \( N' \) with a leaf labeled \( t \) and an edge from \( t \) to \( N' \).

Let us first mention that the actual algorithm is slightly more complicated than the pseudocode in Algorithm 1. First, when \( \text{Cass}(k) \) 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 \( t \) not in \( X \), let \( N' \) be a copy of \( N' \) and an edge from \( t \) to \( N' \) with a leaf labeled \( t \) and an edge from \( t \) to \( N' \).

Lemma 1. Given a set of clusters \( C \), such that \( IG(C) \) is connected and any \( X \subseteq X \) is separated, \( \text{Cass}(1) \) and \( \text{Cass}(2) \) construct a simple level-1, respectively, simple level-2 network representing \( C \), 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-2 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. \( C \setminus \{x\} \). 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-1 network \( N' \) representing \( C \), there exists a simple level-2 network \( N \) representing \( C \) such that the sets of leaf-labels of the maximal strict subtrees of \( N' \setminus \{x\} \) are the maximal \( ST \)-sets w.r.t. \( C \setminus \{x\} \), 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.

Let us first mention that the actual algorithm is slightly more complicated than the pseudocode in Algorithm 1. First, when \( \text{Cass}(k) \) 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 \( t \) (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 \( C \). 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 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. \( C \setminus \{x\} \). 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 \( t \) that is not reachable by a directed path from the reticulation of \( N' \setminus \{x\} \). We can replace \( t \) by a strict subtree on \( X \) that represents \( C \setminus X \). Such

---

算法1 \( \text{Cass}(k) \): 构建一个简单级别的 \( k \) 网络从簇

1. 输入 \((C, X, k, k')\)
2. 输出 \( \text{Cass}(C, X, k, k') \)
3. \( k' = 0 \)
4. \( N' := \emptyset \)
5. 如果 \( k' = 0 \) 则
6. 返回唯一树代表恰好位于 \( C \) 中的每个簇，不返回 \( \emptyset \) 除非没有这样的树存在
7. 重复对于每个边 \( e_1, e_2 \) (不必要地不同) 执行
8. \( \delta \) 是一个不在 \( X \) 中的 dummy 税号
9. 删除叶: \( C' := C \setminus \{x\} \)
10. 崩溃: \( C' := \text{Collapse}(C') \)
11. 递归: \( X' := \text{Cass}(C', X(C'), k', k' - 1) \)
12. 对每个网络 \( N' \) 在 \( N' \) 中做
decollapse: 替换每个叶的 \( N' \) 由一个最大级别 \( ST \)-set \( S \) w.r.t. \( C' \) 由树在 \( S \) 代表确切的那些簇在 \( C' \)
14. 对每个边的 \( e_1, e_2 \) (不必要地不同) 做
do add leaf below reticulation: 创建一个在 \( N' \) 中的\( \delta \) 一个 reticulation \( r \), 一个叶 \( l \) 标签的 \( x \) 而且一个边从 \( t \) 到 \( l \);
16. 对每个边的 \( e_1, e_2 \) (不必要地不同) 做
do add leaf below reticulation: 创建一个在 \( N' \) 中的\( \delta \) 一个 reticulation \( r \), 一个叶 \( l \) 标签的 \( x \) 而且一个边从 \( t \) 到 \( l \);
17. 对每个边的 \( e_1, e_2 \) (不必要地不同) 做
do add leaf below reticulation: 创建一个在 \( N' \) 中的\( \delta \) 一个 reticulation \( r \), 一个叶 \( l \) 标签的 \( x \) 而且一个边从 \( t \) 到 \( l \);
18. 对每个边的 \( e_1, e_2 \) (不必要地不同) 做
do add leaf below reticulation: 创建一个在 \( N' \) 中的\( \delta \) 一个 reticulation \( r \), 一个叶 \( l \) 标签的 \( x \) 而且一个边从 \( t \) 到 \( l \);
19. 对每个边的 \( e_1, e_2 \) (不必要地不同) 做
do add leaf below reticulation: 创建一个在 \( N' \) 中的\( \delta \) 一个 reticulation \( r \), 一个叶 \( l \) 标签的 \( x \) 而且一个边从 \( t \) 到 \( l \);
20. 对每个边的 \( e_1, e_2 \) (不必要地不同) 做
do add leaf below reticulation: 创建一个在 \( N' \) 中的\( \delta \) 一个 reticulation \( r \), 一个叶 \( l \) 标签的 \( x \) 而且一个边从 \( t \) 到 \( l \);
21. 对于每个网络 \( N' \) 在 \( N' \) 中做
22. 递归 \( N' \)
23. 返回 \( N' \)

图4. 构造一个简单级别的 \( k = 2 \) 网络的 \( \text{Cass}(k) \) 算法。最上层的 \( e_1, e_2 \) 将被细分为彩色的红色。单个群都已省略，作为最后的缩合步骤，以简化。
Theorem 4. Omitted due to space constraints.

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 \( C_r \) such that any galled network representing \( C_r \) needs at least \( r \) reticulations, while \textsc{Cass} constructs a network with just two reticulations, which also represents \( C_r \). This follows from the following lemma.

Lemma 3. For each \( r \geq 2 \), there exists a set \( C_r \) of clusters such that there exists a network with two reticulations that represents \( C_r \) while any galled network representing \( C_r \) contains at least \( r \) reticulations.

Proof. Omitted due to space constraints.

5 PRACTICE

Our implementation of the \textsc{Cass} algorithm is available as part of the Dendroscope program (Huson et al., 2007). To use \textsc{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 \textsc{Cass}. Choose ‘minimal network’ to run the \textsc{Cass} algorithm to construct a phylogenetic network representing all clusters that appear in more than \( t \) percent of the input trees.

\textsc{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 \textsc{Cass} and all datasets used for this article. See Huson et al. (2007) for more information on using Dendroscope.

We have tested \textsc{Cass} on both practical and artificial data and compared \textsc{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 \textsc{Cass}, which we call the \textsc{Cass} network, and one galled network using the algorithm in Huson et al. (2009). Two conclusions can be drawn from the results. First, \textsc{Cass} uses more time than the galled network algorithm. Nevertheless, the time needed by \textsc{Cass} can still be considered acceptable for phylogenetic analysis. Second, \textsc{Cass} constructs a much simpler network in almost all cases. For three datasets, the \textsc{Cass} network and the galled network have the same reticulation number and the same level. For all other datasets, the \textsc{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 \textsc{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 realigned in Schmidt (2003). The phylogenetic trees are based on sequences from six different gene loci, ITS, ndhF, phyB, rbcL, rpoC and waxy, and

<table>
<thead>
<tr>
<th>Data</th>
<th>\textsc{GallenedNetwork}</th>
<th>\textsc{Cass}</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>X</td>
<td>)</td>
</tr>
<tr>
<td>30</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>62</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>126</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>254</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>42</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>38</td>
<td>11</td>
<td>7</td>
</tr>
<tr>
<td>61</td>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td>77</td>
<td>22</td>
<td>9</td>
</tr>
<tr>
<td>75</td>
<td>30</td>
<td>11</td>
</tr>
<tr>
<td>89</td>
<td>31</td>
<td>16</td>
</tr>
<tr>
<td>180</td>
<td>51</td>
<td>11</td>
</tr>
<tr>
<td>193</td>
<td>57</td>
<td>1</td>
</tr>
<tr>
<td>270</td>
<td>76</td>
<td>16</td>
</tr>
<tr>
<td>404</td>
<td>122</td>
<td>2</td>
</tr>
<tr>
<td>135.8</td>
<td>31.9</td>
<td>8.5</td>
</tr>
</tbody>
</table>

For each algorithm, the level \( t \) 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.

Table 1. Results of \textsc{Cass} compared with \textsc{GallenedNetwork} for several example cluster sets with \(|C|\) clusters and \(|X|\) taxa.
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 PI RN (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. PI RN 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 PI RN 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 PI RN. 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 PI RN can be explained by the fact that PI RN (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, PI RN 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, PI RN 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. 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 to 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
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 IRN 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
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).

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 there existed 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.