Constructing Level-2 Phylogenetic Networks from Triplets

Leo van Iersel, Judith Keijsper, Steven Kelk, Leen Stougie, Ferry Hagen, and Teun Boekhout

Abstract—Jansson and Sung showed that, given a dense set of input triplets T (representing hypotheses about the local evolutionary relationships of triplets of taxa), it is possible to determine in polynomial time whether there exists a *level-1 network* consistent with T, and if so, to construct such a network [24]. Here, we extend this work by showing that this problem is even polynomial time solvable for the construction of *level-2* networks. This shows that, assuming density, it is tractable to construct plausible evolutionary histories from input triplets even when such histories are heavily nontree-like. This further strengthens the case for the use of triplet-based methods in the construction of phylogenetic networks. We also implemented the algorithm and applied it to yeast data.

Index Terms—Phylogenetic networks, level-2, triplets, reticulations, polynomial time algorithms.

1 Introduction

Phylogenetics is the field at the interface of biology, mathematics, and computer science, which studies the (re)construction of plausible evolutionary scenarios when confronted with incomplete and/or error-prone biological data. Until recently, almost all research effort was directed at finding evolutionary trees. The most well-known techniques are Maximum Parsimony (MP), Maximum Likelihood (ML), Bayesian methods, Distance-based methods (such as Neighbor Joining and UPMGA), and quartet-based methods, as well as various (meta)combinations of these [4], [12], [25], [31]. However, biologically, especially for lower order species, evolution does not necessarily exhibit a tree structure. Thus, the quest for models and methods for more general evolutionary structures than trees has emerged naturally. This forms the subject of this paper.[1]

Quartet methods apply to the construction of unrooted evolutionary trees; less well studied is the problem of constructing *rooted* evolutionary trees, where the branches of the tree are directed to reflect the direction of evolution. The analog of quartet methods for rooted evolutionary trees are *triplet* methods: here, we are given not unrooted trees on four leaves, but rooted binary trees on three leaves, as in Fig. 1. The unique *rooted triplet* (*triplet* for short) on a leaf set $\{x, y, z\}$ in which the parent of x and y is a proper descendant of the lowest common ancestor of x and z is denoted by $xy|z$ (which is identical to $yx|z$). The triplet in the figure is $xy|z$. We say that a directed network containing, amongst others, leaves x, y, and z is consistent with triplet $xy|z$ if it contains a subdivision of $xy|z$. (This will be formalized in Section 2.)

In contrast to the quartet problem, which is NP-hard [32], Aho et al. showed that, given a set of triplets, it is possible to construct in polynomial time a rooted tree consistent with all the input triplets or decide that no such tree exists [1]. Variations in cases where a tree does not exist have been studied in [2], [8], [20], [21], [35]. A well-studied one, albeit NP-hard [20], is to find a tree that maximizes the number of input triplets it is consistent with.

In recent years, attention has turned toward the construction of evolutionary scenarios that are not tree-like. This has been motivated by the fact that biological phenomena such as hybridization, horizontal gene transfer, recombination, and gene duplication can cause lineages, which earlier in time diversified from a common ancestor, to once again intersect with each other later in time. Such evolutionary events are called *reticulation events* and lead to evolutionary scenarios where the underlying undirected graph potentially contains cycles. In this study, we define a *phylogenetic network* (*network* for short) as a directed acyclic graph containing a single root vertex (indegree 0 and outdegree 2), leaf vertices (indegree 1 and outdegree 0) that are labeled in one-to-one correspondence to a set of taxa, split vertices (indegree 1 and outdegree 2), and reticulation vertices (indegree 2 and outdegree 1). We note that in the literature, various other definitions for phylogenetic networks have been proposed along with methods for constructing them (see, for example, [3], [13], and [28]), and we refer the reader to [14], [29], and [30] for excellent surveys of the overall field of phylogenetic networks. Unlike most other approaches, however, we utilize the

1. A provisional version of this paper, without proofs and technical details, can be found in the conference proceedings of RECOMB 2008. The conclusions of the present version have also been updated with respect to the RECOMB version.

- *L. van Iersel is with the Department of Mathematics and Statistics, University of Canterbury, Private Bag 4800, Christchurch, New Zealand. E-mail: l.j.j.v.iersel@gmail.com.*
- *J. Keijsper is with the Department of Computer Science, Technische Universiteit Eindhoven, HG 9.10, PO Box 513, 5600 MB Eindhoven, The Netherlands. E-mail: j.c.m.keijsper@tue.nl.*
- *S. Kelk is with the Centrum voor Wiskunde en Informatica (CWI), Postbus 94079, 1090 GB Amsterdam, The Netherlands. E-mail: s.m.kelk@cwi.nl.*
- *L. Stougie is with the Centrum voor Wiskunde en Informatica (CWI) and the Department of Economics and Business Administration, Free University, De Boelelaan 1105, 1081 HV Amsterdam, The Netherlands. E-mail: lstougie@feweb.vu.nl.*
- *F. Hagen and T. Boekhout are with the Centraalbureau voor Schimmelcultures (CBS), Fungal Biodiversity Center, Uppsalalaan 8, 3584 CT Utrecht, The Netherlands. E-mail: {f.hagen, t.boekhout}@cbs.knaw.nl.*

Manuscript received 8 July 2008; revised 22 Dec. 2008; accepted 3 Feb. 2009; published online 10 Feb. 2009.
For information on obtaining reprints of this article, please send e-mail to: tcbb@computer.org, and reference IEEECS Log Number TCBB-2008-07-0126. Digital Object Identifier no. 10.1109/TCBB.2009.22.

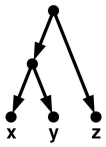


Fig. 1. One of the three possible triplets on the leaves x, y, and z.

concept of *level* to categorize the complexity of a phylogenetic network.

Informally, the level of a phylogenetic network restricts how interwoven the reticulations in the network are. For the formal definition, we need the concept of biconnectivity of a graph. A graph is said to be *biconnected* if it does not contain a single vertex whose removal disconnects[2] the graph. A biconnected subgraph H of a graph G is said to be a *biconnected component* if there is no biconnected subgraph $H' \neq H$ of G that contains H. A biconnected component with at most two vertices is *trivial*. The nontrivial biconnected components can thus be seen as the nontree-like parts of the network. Formally, a phylogenetic network is a *level-k* network if each biconnected component contains at most k reticulation vertices. For example, the network in Fig. 2 is a level-2 network since both its nontrivial biconnected components, which are colored gray, contain two reticulation vertices. Note that each phylogenetic network is a level-k network for some k. We also note that a *phylogenetic tree* is a level-0 network.

Jansson et al. considered constructing a *level-1* network (also known as *galled tree* [10], [34] or *galled network* [6], [11], [22]) consistent with a given set of triplets [22], [24]. In general, the level-1 problem is NP-hard. However, it can be solved in polynomial time when the input is *dense*, meaning that for each set of the three taxa, there is at least one triplet in the input. Density is a reasonable assumption if high-quality triplets can be constructed for all subsets of the three taxa. Various upper bounds, lower bounds, and approximation algorithms for the general case are given in [22]. Related problems comprize the construction of level-1 networks from ultrametric distance matrices [6] and building level-1 networks, where certain input triplets are *forbidden* [11]. Huson and Klöpper [15] consider a generalization of level-1 networks that they call *galled networks* (defined differently than that in [6], [11], [22]).

Level-1 networks can only describe situations with independent reticulations: All cycles are disjoint. Other hypothesized biological relations require more general phylogenetic networks. The concept of level imposes a hierarchy on the space of networks: lower level networks are more tree-like than higher level ones. They might therefore be easier to construct and easier to analyze and interpret. The parsimony principle can be used to argue that lower level networks should be preferred over higher level networks, when both are able to explain the data equally well. Level can also be used in an implicit context, e.g., if we expect the solution to be a tree, but can only find higher level networks, this suggests that data errors lie in the regions corresponding to the biconnected components.

In this paper, we extend considerably the work of Jansson and Sung by showing that, when the input set is dense, we can construct in polynomial time a *level-2* network consistent with all input triplets or decide that no such network exists. The general problem (i.e., that with a possibly nondense input triplet set) we have proven to be NP-hard [16]. (See also [18] for a proof for level-k for any fixed $k \geq 1$.) We omit giving the proof, which is a nontrivial extension of the NP-hardness proof for level-1 networks [22]. In Section 3, we present our algorithm and show that it runs in (slightly better than) cubic time in the number of input triplets. This extends significantly the power of triplet methods since it further extends the complexity of the evolutionary scenarios that can be constructed. Our result and the way it is proved make it tempting to conjecture that, for fixed k, the level-k problem with dense triplet sets is polynomial-time solvable. However, it is not clear whether the pivotal theorem in Section 3, Theorem 3, generalizes to level-3 and higher networks.

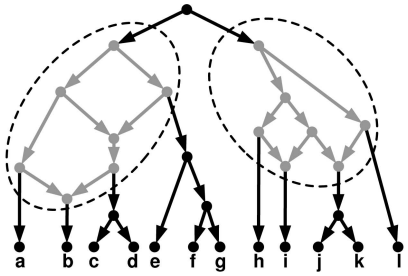


Fig. 2. An example of a level-2 network. Its nontrivial biconnected components are colored gray and have been circled.

An implementation of our algorithm in Java has been made publicly available and placed in an open-source repository [27]. We applied it to sequences from the yeast *Cryptococcus gattii*. We constructed a level-2 network for different isolates of this, potentially dangerous, yeast and are planning to use this network to find the origin of a *C. gattii* outbreak on the West Coast of Canada. The results are reported in Section 4. In Section 5, we discuss conclusions and many fascinating remaining open problems.

2 Preliminaries

Recall the definition of a level-k phylogenetic network given in Section 1. Denote the set of leaves in a network N by L_N. For a set T of triplets, define $L(T) = \bigcup_{t \in T} L_t$ and $n(T) = |L(T)|$. A set T of triplets is called *dense* if for each $\{x, y, z\} \subseteq L(T)$, at least one of $xy|z$, $xz|y$, and $yz|x$ belongs to T. Furthermore, for a set of triplets T and a set of leaves $L' \subseteq L(T)$, we denote by $T|L'$ the triplets $t \in T$ with $L_t \subseteq L'$. If the triplet set is clear from the context, we use L and n to denote $L(T)$ and $n(T)$.

To avoid "redundant" networks, in this paper, we only allow networks for which every nontrivial biconnected component has at least three outgoing arcs (including arcs leading to leaves). We also note here that whenever we refer to a *path* in a network, we mean a directed path.

Definition 1. *A triplet $xy|z$ is* consistent *with a network N (interchangeably: N is consistent with $xy|z$) if N contains a subdivision of $xy|z$, i.e., if N contains vertices $u \neq v$ and pairwise internally vertex-disjoint paths $u \to x$, $u \to y$, $v \to u$, and $v \to z$.*

2. A directed acyclic graph is *connected* if it is possible to reach any vertex from any other vertex when the orientations of the arcs are ignored.

By extension, a set of triplets T is said to be *consistent* with N (interchangeably: N is consistent with T) if every triplet in T is consistent with N. To clarify triplet consistency, we observe that the network in Fig. 2 is consistent with (amongst others) $ab|c$, $bc|a$, and $dg|k$, but not consistent with (for example) $ah|f$ or $hk|i$.

Note that the definition of triplet consistency in [22] ("$xy|z$ is *consistent* with N if $xy|z$ is an embedded subtree of N (i.e., if the lowest common ancestor of x and y is a proper descendant of the lowest common ancestor of x and z)") is only usable for trees and not for general networks. Personal communication with the author [19] has clarified that Definition 1 is the definition he actually meant.

We will now define SN-sets (short for "Side Network" sets), introduced in [24], which will play a crucial role in the rest of the paper. For a triplet set T, a subset X of L is an *SN-set* if there is no triplet $xy|z \in T$ with $x, z \in X$ and $y \notin X$. An SN-set X is *maximal* with respect to a triplet set T if $X \neq L$ and L is the only SN-set that is a strict superset of X. Note that a leaf is always contained in some maximal SN-set, as any singleton set is, by definition, an SN-set. Consider a dense triplet set T. In [24], it is shown that the family of SN-sets of such a triplet set T is laminar, i.e., any two SN-sets are either disjoint or one is included in the other. The *SN-tree* is, by definition, the directed tree associated with this laminar family, so every SN-set of T corresponds to the set of leaves reachable from a vertex of the SN-tree. All SN-sets of T can be found by constructing the SN-tree in $O(n^3)$ time [22].

We call an arc $a = (u, v)$ of a network N a *cut-arc* if its removal disconnects N and call it *trivial* if v is a leaf. A cut-arc is *highest* if there does not exist a cut-arc $a' = (u', v')$ such that u is reachable from v'. We say that a vertex is *below* a or *below* v, if it is reachable from v. A little thought should make it clear that for each cut-arc a in a network N consistent with a dense triplet set T, the set S of leaves below a is an SN-set of T and that the sets of leaves below the highest cut-arcs partition L_N. The following lemma reveals a crucial characteristic that we exploit in our algorithm.

Lemma 1. *Let N be a network consistent with a dense triplet set T. Then each maximal SN-set S of T is the union of sets of leaves below the highest cut-arcs in N.*

Proof. Suppose S is a maximal SN-set without this property. Since the set of leaves below a cut-arc is always an SN-set, maximality of S implies that S is not a strict subset of the leaves below some single highest cut-arc a. Thus, S intersects with the leaves below at least two highest cut-arcs. It follows that there exist leaves $x, z \in S$ and $y \notin S$ such that x and y are below the highest cut-arc a_1 and z is below the highest cut-arc $a_2 \neq a_1$. However, in this case, the only triplet on x, y, z consistent with N is $xy|z$, implying that $y \in S$, a contradiction. □

3 Constructing Level-2 Networks from Dense Triplet Sets

This section describes our main result: a polynomial time algorithm that constructs a level-2 network from a dense triplet set T if such a network exists. The algorithm is recursive. The main idea is visualized in Fig. 3. (For ease of viewing, all the figures from this point onward omit arrow heads on the arcs. Arcs are always directed away from the root.) Suppose we know the correct partition $L = S_1 \cup \ldots \cup S_7$ of the leaves. Then the algorithm replaces each set S_i by a single (meta)leaf and constructs a "simple" level-2 network (in black) on these metaleaves. In Section 3.1, we formally introduce simple level-2 networks and show how they can be constructed. The complete level-2 network can then be obtained by replacing each metaleaf S_i by a recursively created level-2 network. In spirit, this procedure resembles that for level-1 networks [24]. However, next to simple level-2 networks being more complex, finding the right partition and proving correctness of the algorithm is far more involved than in the level-1 case. Unlike level-1, there does not, for example, always exist a level-2 network, where the sets of leaves below the highest cut-arcs correspond to the maximal SN-sets. The recursion and finding a correct partition is described in Section 3.2.

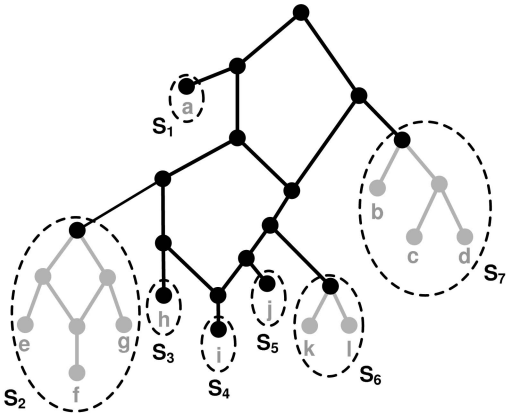


Fig. 3. Construction of level-2 network by recursively constructing "simple" networks, given the partition $L = S_1 \cup \ldots \cup S_7$. Note that, from this point onward, arrow heads are omitted from arcs: arcs are always directed away from the root.

3.1 Simple Level-2 Networks

We now introduce the class of level-2 networks that we name *simple* level-2 networks. Informally, these are the basic building blocks of level-2 networks in the sense that each biconnected component of a level-2 network is, in essence, a simple level-2 network. We first introduce a *simple level-k generator*.

Definition 2. *A simple level-k generator is a directed acyclic biconnected multigraph, which has a single root (indegree 0 and outdegree 2), precisely k reticulation vertices (indegree 2 and outdegree at most 1), and apart from that only split vertices (indegree 1 and outdegree 2), where vertices with indegree 2 and outdegree 0 as well as all arcs are labeled and called* sides.

Lemma 2. *There are one simple level-1 generator and four simple level-2 generators, which are shown in Fig. 4.*

Proof. The proof, based on simple case checking, is left to the reader. □

Definition 3. *A simple level-k network N, for $k \geq 1$, is a network obtained by applying the following transformation*

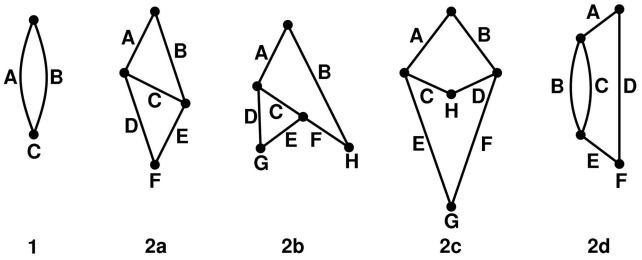


Fig. 4. The only simple level-1 generator and all four simple level-2 generators.

("leaf hanging") to some simple level-k generator such that the resulting graph is a valid network:

1. *replace each arc X by a path and for each internal vertex v of the path, add a new leaf x and an arc (v, x); we say that "leaf x is on side X"; and*
2. *for each vertex Y of indegree 2 and outdegree 0, add a new leaf y and an arc (Y, y); we say that "leaf y is on side Y."*

A simple level-k network is thus created by hanging leaves from sides of a simple level-k generator, i.e., from labeled vertices and labeled arcs. Note that it is not necessary to hang a leaf from each arc, since the path used to replace the arc is allowed to contain no internal vertices. An exception to this, however, is that whenever there are multiple arcs, we replace at least one of them by a path of at least three vertices. A simple level-2 network built by hanging leaves from generator $2a$, $2b$, $2c$, or $2d$ is called a *network of type* $2a$, $2b$, $2c$, or $2d$, respectively.

We emphasize that the simple level-k generators *do not* have leaves, while the simple level-k networks and general level-k networks *do* have leaves. Also note that reticulation vertices in generators are allowed to have outdegree 0, while in networks, reticulation vertices always have outdegree 1.

We do not attempt to define simple level-0 networks; instead, we introduce the *basic tree* which we define as the directed graph on three vertices $\{v_1, v_2, v_3\}$ with arc set $\{(v_1, v_2), (v_1, v_3)\}$. For the sake of convenience, we say that the basic tree, simple level-1 networks, and simple level-2 networks are all *simple level-≤ 2 networks.*

A level-k network that is not a level-$(k-1)$ network is called a *strict level-k network*. The following lemma gives a nice and simple characterization of simple level-k networks.

Lemma 3. *A strict level-k network is a simple level-k network if and only if it contains no nontrivial cut-arcs.*

Proof. A simple level-k network contains no nontrivial cut-arcs because simple level-k generators are biconnected. Now take a strict level-k network N with no nontrivial cut-arcs. It remains to show that N is a simple level-k network. Delete all leaves from N and subsequently suppress all vertices with indegree and outdegree equal to one. The resulting graph still contains exactly k reticulation vertices and contains no cut-arcs. It follows that this graph is biconnected because any graph with degree at most three containing a cut-vertex also contains a cut-arc. Therefore, it is a simple level-k generator. □

All simple level-1 networks on dense triplet sets can be found by an algorithm by Jansson et al. [22]. We designed Algorithm 1 to find all simple level-2 networks consistent with a dense set of triplets T. Before describing the algorithm, we will first give an analysis of the structure of simple level-2 networks upon which the algorithm is based. We use the notion *reticulation leaf* for a leaf whose parent is a reticulation vertex and we say that a network is a *caterpillar* if the deletion of all leaves gives a directed path. Consider any simple level-2 network. If we remove a reticulation leaf x and its parent x', there is one reticulation vertex y' left and below it is a caterpillar (or a single leaf or nothing). If we now remove the last reticulation vertex and everything below it, we obtain a tree, which is unique [23] and can be constructed using the algorithm of Aho et al. [1]. Algorithm 1 first identifies this tree and then reconstructs the simple level-2 network from it by reinserting the caterpillar and reticulation leaf. Since on some sides of the simple level-2 network, there might be no leaves, we have to consider adding a dummy reticulation leaf (for when there are no leaves between the two reticulation vertices) and add a dummy root (for when there are no leaves on a side connected to the root). How this can be done is explained in the pseudocode of Algorithm 1.

Algorithm 1. SL2

1: $Net := \emptyset$
2: **for** each leaf $x \in L$ **do**
3: /* *guess that x is a reticulation leaf and remove it:* */ $L' := L \setminus \{x\}$, $T' := T|L'$
4: $Cat := FindCaterpillarsets(T')$.
5: **for** each set C in Cat **do**
6: /* *guess that the leaves in C form the caterpillar below the remaining reticulation and remove C:* */ $L'' := L' \setminus C, T'' := T'|L''$
7: build the unique tree $N'' = (V, A)$ consistent with T'' if it exists using [1]
8: /* *add a dummy root r':* */ $V := V \cup \{r'\}$; $A := A \cup \{(r', r)\}$, with r the root of N''
9: **for** every arc $a_1 = (u_1, v_1)$ and every low or high arc $a_2 = (u_2, v_2)$ in A **do**
10: **if** $|C| \geq 2$ **then**
11: construct the caterpillar (V_{cat}, A_{cat}) consistent with $T|C$ and let y be its root
12: **else if** $|C| = 1$ **then**
13: /* *construct a caterpillar consisting of a single leaf:* */
14: $V_{cat} := C$, $A_{cat} := \emptyset$ and let y be such that $C = \{y\}$.
15: **else**
16: /* *construct a caterpillar consisting of a single dummy leaf:* */
17: $V_{cat} := \{y\}$, $A_{cat} := \emptyset$, with y a dummy leaf
18: /* *subdivide a_1 and a_2 by new vertices w_1 and w_2, create a new reticulation y' below w_1 and w_2 and put the caterpillar below y':* */ $V' := V \cup V_{cat} \cup \{w_1, w_2, y'\}$, $A' := A \cup A_{cat} \setminus \{a_1, a_2\} \cup \{(u_i, w_i), (w_i, v_i), (w_i, y') | i = 1, 2\} \cup \{(y', y)\}$
19: **for** every two arcs $a_3 = (u_3, v_4)$ and $a_4 = (u_4, v_4)$ in A' **do**
20: /* *subdivide a_3 and a_4 by new vertices w_3 and w_4,*

*create a new reticulation x' below w_3 and w_4 and put x below x': */* $V'' := V' \cup \{w_3, w_4, x', x\}$, $A'' := A' \setminus \{a_3, a_4\} \cup \{(u_i, w_i), (w_i, v_i), (w_i, x') | i = 3, 4\} \cup \{(x', x)\}$, $N := (V'', A'')$

21: **if** $C = \emptyset$ **then**

22: */* remove the dummy leaf: */* $N := N \setminus \{y\}$

23: suppress the former parent of y if it has in- and outdegree both equal to 1

24: */* remove the dummy root: */* $N := N \setminus \{r'\}$

25: **if** N is a simple level-2 network consistent with T **then**

26: $Net := Net \cup N$

27: **return** Net

A task of the algorithm is to find the caterpillar below reticulation vertex y'. For this, we define a subset L' of the leaves to be a *caterpillarset* if there exists a network consistent with the input triplets that contains a caterpillar with leaves L' as subgraph (or if $L' = \emptyset$). Note that this caterpillar has to be a valid (sub)network and that this subgraph is therefore not allowed to have vertices with indegree and outdegree both 1.

Algorithm 2. FindCaterpillarsets

1: $Cat := \{\emptyset\}$

2: compute all SN-sets by the algorithm in [22].

3: **for** each singleton SN-set S **do**

4: $C := S$

5: $Cat := Cat \cup \{C\}$

6: **while** there is an SN-set $C \cup \{x\}$ with $x \notin C$ **do**

7: $C := C \cup \{x\}$

8: $Cat := Cat \cup \{C\}$

9: **return** Cat

Lemma 4. *Algorithm 2 finds $O(n)$ sets, including all caterpillarsets w.r.t. a dense triplet set T in $O(n^3)$ time.*

Proof. Suppose $L' \neq \emptyset$ is a caterpillarset w.r.t. dense triplet set T. Then there exists a network N that contains a caterpillar with leaves L' as subgraph. Write $L' = \{\ell_1, \ldots, \ell_k\}$ such that ℓ_1 has distance $k-1$ and $\ell_i (2 \leq i \leq k)$ has distance $k-i+1$ from the root of the caterpillar. The set $\{\ell_1, \ldots, \ell_i\}$ occurs below a cut-arc, and thus, forms an SN-set of T, for all $1 \leq i \leq k$. From the pseudocode in Algorithm 2, it is clear that this algorithm will find all sets of this form, i.e., all sets $L' = \{\ell_1, \ldots, \ell_k\}$ that have the property that $\{\ell_1, \ldots, \ell_i\}$ is an SN-set of T, for all $1 \leq i \leq k$. The number of such sets is $O(n)$, since the family of SN-sets is laminar. The algorithm can be implemented to run in $O(n^3)$ time by using the SN-tree, which can be constructed in $O(n^3)$ time [22]. □

Note that we do not claim that the algorithm finds only caterpillarsets; for us it is enough to know that it returns $O(n)$ sets, including all caterpillarsets.

We say that z is a *low leaf* of a network N if its parent has outdegree zero in $N \setminus L$. A low leaf is thus either a reticulation leaf or the child of a split vertex, where both children of the split vertex are leaves. If arc a enters a low leaf z, we say that a is a *low arc* of N. An arc leaving the root or a child of the root is called a *high arc*.

Theorem 1. *Algorithm 1 finds all simple level-2 networks consistent with a dense triplet set.*

Proof. Suppose triplet set T is consistent with some simple level-2 network N. At some iteration, the algorithm will choose the right leaf and caterpillarset to remove and the right arcs to subdivide to arrive at network N, provided that the algorithm indeed checks all relevant possible combinations of arcs to be subdivided, which we prove to be the case below. Furthermore, the algorithm checks (line 25) for each constructed network whether it is a simple level-2 network consistent with T. We conclude that the algorithm will find exactly all simple level-2 networks consistent with T. The example shown in Figs. 9a, 9b, and 9c and the text above it at the end of Section 3.2, interpreting the sets $S_1, \ldots, S_7$ as the taxa (while they are SN-sets there), helps clarify how Algorithm 1 constructs a simple level-2 network.

If in line 9, we choose all pairs of arcs instead of restricting one of the two to be a high or a low arc, then it is obvious that indeed all possible combinations will be checked. This would lead in the analysis below to an $O(n^9)$ running time of the algorithm.

It remains to be shown that restricting to choosing one of the two arcs to be subdivided being a high or a low arc does not exclude any essential combination, i.e., a combination leading to a different consistent level-2 network. First, consider a type $2a$ network. We can first remove the leaf on side F and then the caterpillarset consisting of all leaves (possibly none) on side E. If there are at least two leaves on side B, then one of the arcs we choose to subdivide is a low arc (we subdivide the arc leading to the leaf that in the complete network was the "lowest" leaf on side B, i.e., its parent is not an ancestor of any other leaf on side B). If, on the other hand, there is at most one leaf on side B, then one of the arcs we subdivide is a high arc (we subdivide the arc leaving the dummy root if there are no leaves on side B and we subdivide an arc leaving the child of the dummy root if there is exactly one leaf on side B).

Now consider a type $2b$ network. We remove the leaf on side G and the caterpillarset consisting of the leaf on side H. Then, we can argue just like with $2a$ that if there are at least two leaves on side B, we subdivide a low arc and otherwise a high arc. In a type $2c$ network, we remove the leaf on side G and the caterpillarset consisting of the leaf on side H. If on one of the sides C, D, E, or F, there are at least two leaves, we can subdivide a low arc on this side. Otherwise there is at most one leaf on each of the sides C, D, E, and F, and therefore, all arcs we want to subdivide are low. Finally, in a type $2d$ network, we remove the leaf on side F and the caterpillarset consisting of the leaves on side E. Then, we can always subdivide a low arc unless there are no leaves on sides B and C, which is not allowed. This concludes the proof. □

Theorem 2. *Algorithm 1 can be implemented to run in time $O(n^8)$.*

Proof. A little thought should make it clear that any level-2 network with n leaves contains $O(n)$ arcs. The number of caterpillarsets is also $O(n)$, since the SN-sets form a laminar family. The number of high arcs in any network is at most six. In a network obtained from a simple level-2

network by removing a reticulation leaf and the caterpillar below the remaining reticulation vertex, there are at most eight low arcs. For example, if the network is of type $2a$ and we remove the leaves on sides E and F, then there are at most six low arcs: two leading to leaves on side B, two to leaves on side C, and two to leaves on side D. Thus, both the number of high and low arcs is $O(1)$.

Line 25 can be executed in time $O(n^3)$ since biconnectivity can be checked in linear time [33] and triplet consistency can be checked in $O(n^3)$ time [5]. Therefore, the algorithm can be implemented to run in time $O(n^8)$.□

3.2 From Simple to General Level-2 Networks

This section explains how to build general level-2 networks by recursively building simple level-≤ 2 networks. We need some definitions. For any network N, denote by $\mathrm{Collapse}(N)$ the network obtained from N by collapsing the sets of leaves below the highest cut arcs. More precisely, for each highest cut-arc $a = (u, v)$ in N, replace v and everything reachable from it by a single new leaf V, which we identify with the set of leaves below a in N. Thus, if N has q highest cut-arcs $a_1, \ldots, a_q$, then $\mathrm{Collapse}(N)$ has leaf set $\mathcal{C} = \{C_1, \ldots, C_q\}$, where C_i is the set of leaves below a_i in N (and hence, $\mathcal{C}$ is a partition of L_N). Note that if N is a level-2 network, then $\mathrm{Collapse}(N)$ is a simple level-≤ 2 network, by Lemma 3. For example, if N is the network in Fig. 3, where $S_1, \ldots, S_7$ are the sets of leaves below the highest cut-arcs, then $\mathrm{Collapse}(N)$ is the black part of the network (with $S_1, \ldots, S_7$ viewed as leaves). Thus, in $\mathrm{Collapse(N)}$, each gray part of the network is replaced by a single leaf, e.g., the gray part containing e, f, and g is replaced by a single leaf S_2.

For a dense triplet set T on leaf set L, a partition $\mathcal{C}$ of L is a *correct partition* for T if there exists a level-2 network N consistent with T such that the leaf sets below the highest cut-arcs in N are given by $\mathcal{C}$, i.e., such that $\mathrm{Collapse}(N)$ has leaf set $\mathcal{C}$. For any triplet set T and any partition $\mathcal{C} = \{C_1, \ldots, C_q\}$ of L, define the *induced* triplet set $T\nabla\mathcal{C}$ as the set of triplets $C_iC_j|C_k$ such that there exist $x \in C_i$, $y \in C_j$, and $z \in C_k$, with $xy|z \in T$ and i, j, and k all distinct. Note that $T\nabla\mathcal{C}$ is a triplet set on $\mathcal{C}$ and that it is dense if T is dense. Note that if N is a network consistent with T such that the leaf sets below the highest cut-arcs in N are given by a partition $\mathcal{C}$ of L, then $\mathrm{Collapse}(N)$ is consistent with $T\nabla\mathcal{C}$.

The rough idea of our algorithm is to first determine, if it exists, a correct partition $\mathcal{C} = \{C_1, \ldots, C_q\}$ for the given dense triplet set T, then construct a simple level-≤ 2 network $N_{\mathcal{C}}$ consistent with $T\nabla\mathcal{C}$ using Algorithm 1, and then replace every leaf C_i of $N_{\mathcal{C}}$ by a recursively created level-2 network consistent with $T|C_i$. The resulting network N is consistent with T by the following lemma. Note that in the lemma, we do not require $N_{\mathcal{C}}$ to be simple.

Lemma 5. *Let $\mathcal{C} = \{C_1, \ldots, C_q\}$ be a correct partition for a dense triplet set T and $N_{\mathcal{C}}$ be a level-2 network consistent with $T\nabla\mathcal{C}$. Let N be a network obtained from $N_{\mathcal{C}}$ by replacing each leaf C_i of $N_{\mathcal{C}}$ by a level-2 network N_i consistent with $T|C_i$. Then, N is a level-2 network consistent with T.*

Proof. Consider a triplet $xy|z \in T$. First, suppose $x, y, z \in C_i$ for some i. Then, $xy|z \in T|C_i$, and hence, N is consistent with $xy|z$ because its subnetwork N_i is. Since $\mathcal{C}$ is a correct partition for T, there exists some level-2 network N' consistent with T such that C_i are the sets of leaves below the highest cut-arcs of N'. This means that for no $i \neq j$, it holds that $x, z \in C_i$ while $y \in C_j$, as N' is consistent with $xy|z$. If $x, y \in C_i$ and $z \in C_j$ for some $i \neq j$, then $xy|z$ is consistent with N since C_i and C_j are below cut-arcs of N that are not reachable from each other (as they are both expanded leaves of $N_{\mathcal{C}}$). Finally, suppose $x \in C_i$, $y \in C_j$, and $z \in C_k$, with i, j, k distinct. From $xy|z \in T$, it follows that $C_iC_j|C_k \in T\nabla\mathcal{C}$, and hence, $N_{\mathcal{C}}$ is consistent with $C_iC_j|C_k$. But then, N is consistent with $xy|z$. □

The main subject of the rest of this section is how to find a correct partition of the leaf set. In a consistent level-1 network, the collection of maximal SN-sets forms a correct partition $\mathcal{S}$, as follows from the algorithm in [22]. For level-2, the situation is more complicated, but maximal SN-sets can still be used to obtain correct partitions.

Suppose dense triplet set T is consistent with a level-2 network N. We show in Theorem 3 that we have to split at most one maximal SN-set of T to obtain a correct partition of the leaves. We first show in Lemma 6 that, if N is a simple level-≤ 2 network, we can "push" any single maximal SN-set, apart from one, below a highest cut-arc, maintaining consistency with T. In Lemma 8, this is extended to *all* maximal SN-sets, and in Theorem 3, to general level-2 networks. For the latter two results, we need Lemma 7, describing the relation between the maximal SN-sets of T and the maximal SN-sets of $T\nabla\mathcal{C}$, for a correct partition $\mathcal{C}$.

Given a simple level-2 network N and an SN-set S, we say that (N, S) is *exceptional* if:

1. N is of type $2c$ and S consists of the two reticulation leaves in N, or
2. N is of type $2a$ or $2d$ and S consists of all the leaves on side E and at least one leaf on side C.

Lemma 6. *Let N be a simple* level-≤ 2 *network consistent with dense triplet set T and S an SN-set of T. Then, unless (N, S) is exceptional, the partition $\mathcal{C}' := \{S\} \cup \{\{l\} \mid l \in L \setminus S\}$ of L is a correct partition for T.*

Proof. We explicitly construct a level-2 network N' consistent with T such that S equals the set of leaves below the highest cut-arc of N', and all other sets of leaves below the highest cut-arcs are the same as in N, i.e., they are singletons (as N is simple).

The case that S is a singleton is trivial. Thus, assume that S is not exceptional and contains at least two leaves. We say that u is the *lowest common ancestor (LCA)* of x and y if u is an ancestor of both x and y and no proper descendant of u has this property. We give three critical observations as follows:

Observation 1. No two leaves in S have the root as the lowest common ancestor.

That is, if two leaves in S have the root as LCA, then all leaves are in S, since S is an SN-set.

Observation 2. If $x \in S$ and there is a path not containing any reticulation vertices from the parent of x to another leaf l, then l is also in S.

That is, if $y \neq x$, $y \in S$, then $l \in S$, since N is not consistent with $xy|l$.

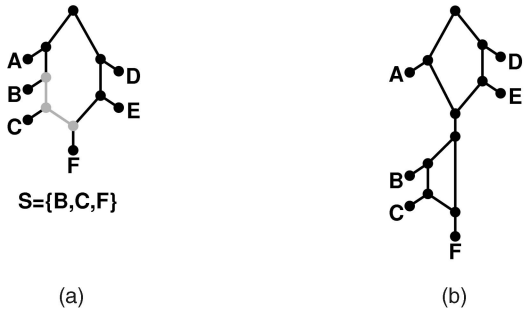


Fig. 5. Construction of (b) N' in case (a) N is a simple level-1 network and the parents of S form a path (in gray) ending in the reticulation vertex.

Observation 3. If two leaves $x, y \in S$ have exactly one LCA u, then all leaves z that have a parent on a path from u to x are in S, unless N is of type $2a$, x is on side F, y is on side C, and z on side E. In this case, no leaves of S are on sides E or B.

Proof. The proof of this observation is less obvious and deferred to the Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCBB.2009.22. □

As a warming-up for the reader, we prove the lemma for the case that N is a simple level-1 network. The proof is constructive. For simple level-2 networks, the proof is essentially the same but more involved. In case N is simple level-1, Observations 1 and 2 imply that the parents of S form a path P starting below the root and ending in either the reticulation vertex or in a parent of the reticulation vertex. We construct the network N' as follows. Let N_c denote the result of contracting in N the path P to a single vertex v_c. Let N_c^* denote the result of removing all leaves in S from N_c and N^* denote the result of removing all leaves not in S from N. We combine N_c^* and N^* by adding an arc from v_c to the root of N^*. To obtain a valid network N' with only labeled leaves, we remove any unlabeled vertices with outdegree zero and suppress any vertices with indegree and outdegree one. For example, see Fig. 5. It is not too difficult to check that N' is consistent with all triplets that are consistent with N, except for triplets of the form $xy|z$ with $x, z \in S$, and $y \notin S$. Since the latter type of triplets are not in T, this concludes the level-1 case.

From now on, we assume that N is a simple level-2 network. The main differences with the level-1 case are that in the level-2 case, the parents of S form a connected subgraph that is not necessarily a path, and that there are some exceptional cases, which complicate the analysis.

Claim 1. There are at most two (directed) paths in N such that each leaf in S has a parent on one of these paths, unless N is of type $2b$ and there is a leaf on side D, a leaf on side E, and a leaf on side F or H in S.

Proof. Deferred to the Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCBB.2009.22. □

We first deal with a particular case: N is of type $2a$ and (some) leaves on sides C and F are in S and no leaves on side B or E are in S. The lemma holds for this case as can be seen by the construction of a network N' as indicated in Fig. 6, where all the leaves in S (including possibly leaves on side A and/or D) are put below the nontrivial cut-arc and all other leaves above it. Note that if there are leaves on side A or D in S, then all leaves on side D must be in S. It is not too difficult to check that in this case, N' is consistent with all triplets that are consistent with N and not of the form $xy|z$ with $x, z \in S$ and $y \notin S$.

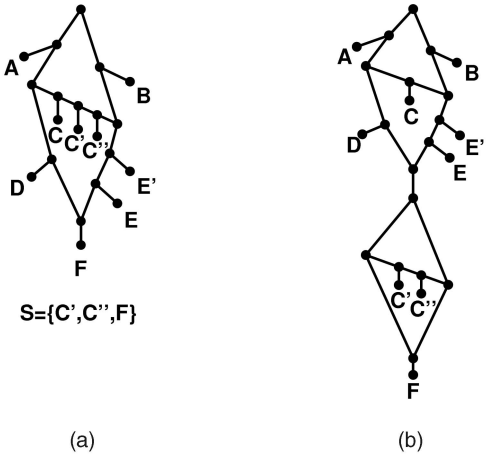


Fig. 6. Construction of (b) N' in case (a) N is of type $2a$, and leaves on sides C and F are in S and no leaves on side E or B are in S.

Thus, from here on, this case is excluded, implying that Observation 3 holds without exception.

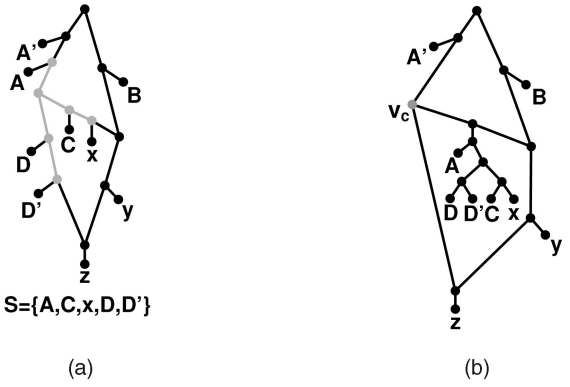
For any subgraph H of N without leaves, let $d^-(H)$ denote the number of arcs entering H and let $d^+(H)$ denote the number of arcs going out of H not leading to leaves in S.[3]

Claim 2. There exists a connected subgraph of N containing the parents of leaves in S and no parents of other leaves. There exists a subgraph $2P$, which is minimal with respect to these properties, does not contain the root, and has either $d^-(2P) \leq 3$ and $d^+(2P) \leq 1$ or $d^-(2P) = 1$ and $d^+(2P) = 2$.

Proof. Deferred to the Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCBB.2009.22. □

Let N_c denote the result of contracting in N the connected subgraph $2P$ to a single vertex v_c. Let N_c^* denote the result of removing all leaves in S from N_c. Let N^* denote the result of removing all leaves not in S from N. We construct N' by combining N_c^* and N^*. How this is done depends on the indegree and outdegree of v_c in N_c^*. The idea is, as in the level-1 case, to create a new outgoing arc from v_c to the root of N^*. But in order to obtain a valid network, we only add such an arc if v_c has outdegree zero or indegree and outdegree one. If v_c has indegree at least two and outdegree one, we subdivide the arc leaving v_c by a new vertex v_n and create an arc from v_n to the root of N^*, unless N is of type $2c$ and all leaves on sides E, C, and H are in S or N is of type $2b$ and all leaves on sides E, G, and F are in S. In the former case, we construct N' by removing the leaves in S from N (we do not contract $2P$) and replacing side H by N^*. In the latter case, we construct N' by removing the leaves in S from N (we do not contract $2P$) and replacing side G by N^*. If v_c has indegree one and

3. Arcs with both endpoints in H do not contribute to $d^+(H)$ or $d^-(H)$.



(a) (b)

Fig. 7. Construction of (b) N' if (a) N is of type $2a$ with all leaves on sides C and D and some on side A in S.

outdegree two, we subdivide the arc entering v_c by a new vertex v_n and create an arc from v_n to the root of N^*, unless N is of type $2a$ with all leaves on sides C and D and some on side A in S. In the latter excluded case, we apply the construction from Fig. 7.

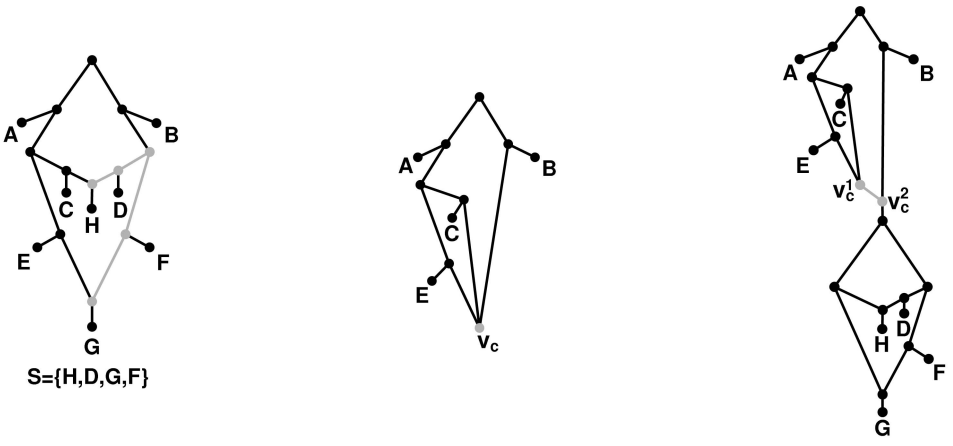
To obtain a valid network, we remove again all unlabeled vertices with outdegree zero and suppress vertices with indegree and outdegree one. Moreover, if v_c has indegree three, we replace it by two vertices v_c^1 and v_c^2 and connect two former parents of v_c to v_c^1, one former parent of v_c to v_c^2 and v_c^1 to v_c^2.

This whole procedure is illustrated in Fig. 8. An example network N is displayed in Fig. 8a with $2P$ in gray. After contracting $2P$ to a vertex v_c (in gray) and removing the leaves from $S = \{H, D, G, F\}$, we get the network N_c^* (Fig. 8b). Since v_c has indegree three and outdegree zero, we replace it by two vertices of indegree two and create an outgoing arc to N^*. After suppressing all vertices with indegree and outdegree both one, we get the network N' (Fig. 8c). Note that N is of type $2c$, but $\mathrm{Collapse}(N')$ is of type $2d$ in this example.

To check that N' is consistent with T, consider three leaves x, y, z. If $x, y, z \in S$ or $x, y, z \notin S$, then any triplet on these leaves that is consistent with N is clearly also consistent with N'. If $x, y \in S$ and $z \notin S$, then, by the definition of SN-set, $xy|z$ is the only triplet in T on these three leaves and this triplet is consistent with N', as x and y appear below one cut-arc in N' together, whereas z is not below that arc.

Now consider the case that $x \in S$ and $y, z \notin S$, and suppose a triplet t over $\{x, y, z\}$ is consistent with N but not with N'. Note that since t is not consistent with N', it is also not consistent with N_c. First, suppose $t = yz|x$. Because t is consistent with N, there are vertices u and v of N such that there exist internally vertex disjoint paths $u \to v$, $v \to y$, $v \to z$, and $u \to x$. Because t is not consistent with N_c, either one of these paths has been contracted and is thus contained in $2P$, or a path between internal vertices of $u \to x$ and $v \to z$ (or $v \to y$) has been completely contracted, stopping $u \to x$ and $v \to z$ from being internally vertex disjoint. First, suppose that a path from an internal vertex w_1 of $u \to x$ to an internal vertex w_2 of $v \to z$ has been completely contracted. In this case, w_2 is a reticulation and there must be another leaf $x_2 \in S$ below the path $w_2 \to z$, because, otherwise, $w_1 \to w_2$ would not be in $2P$. Since z is not in S, there must be a second reticulation on the path $w_2 \to z$. The picture that arises is a network of type $2a$ or $2d$ with S consisting of the leaves on side E and some of the leaves on side C, but this case has been excluded from the lemma. Now suppose that a path from an internal vertex of $v \to z$ to an internal vertex of $u \to x$ has been completely contracted. The picture emerges of a network of type $2b$ with z on side G, x on side H and y on side D or with x on side G, z on side H and y on side C or of a network of type $2c$ with z and x on sides G and H, but in these cases, it can be checked that the triplet $yz|x$ is also consistent with N'. Thus, we conclude that one of the paths $u \to v$, $v \to y$, $v \to z$ and $u \to x$ has been contracted. This must be the path $u \to v$ since $2P$ does not contain any leaves. Hence, in N_c, there are three internally disjoint paths from v_c to x, y, and z. It follows that v_c has outdegree two, and thus, indegree one in N_c^*. This means that network N' is either constructed as in Fig. 7 or the arc leading to v_c is subdivided by v_n and there is an arc from v_n to the root of N^*. In both cases, inconsistency of N' with $yz|x$ is contradicted.

Now consider the case that $t = xy|z$ (the case $= xz|y$ is symmetric). In the same way as above, we argue that as t is consistent with N but not with N_c, there exist vertices u and v of N, internally vertex disjoint paths $u \to v$, $v \to x$, $v \to y$, and $u \to z$ denoted by P_{uv}, P_{vx}, P_{vy}, and P_{uz}, respectively, and



(a) (b) (c)

Fig. 8. Networks (a) N, (b) N_c^*, and (c) N' as an example of the construction in the proof of Lemma 6.

that $2P$ contains either the path P_{uv}, or a path from an internal vertex of P_{uz} to an internal vertex of P_{vx} (or P_{vy}), or a path from an internal vertex of P_{vx} (P_{vy}) to an internal vertex of P_{uz}. In case $2P$ contains a path from an internal vertex of P_{uz} to an internal vertex of P_{vx}, the picture arises of a network of type $2c$ with all leaves on sides E, C, and H in S (with x on H, z on G and y on B, D, or F) or a network of type $2b$ with all leaves on sides E, F, and H in S (with x on H, y on B and z on G), or a network of type $2b$ with all leaves on sides E, F, and G in S (with x on G, y on D and z on H). In these cases, it can be checked that N' is consistent with $xy|z$.

In case $2P$ contains a path from an internal vertex of P_{vx} to an internal vertex of P_{uz}, the picture arises of a network of type $2a$ with S consisting of all the leaves on side E and some of the leaves on side C (with x on C, y on A, C, or D and z on F), or with S consisting of all the leaves on side E and some on side B (x is on B, y is on B above x, and z is on F), or a network of type $2d$ with S consisting of all the leaves on E and some on B (x is on B, y is on B above x and z is on F). The first possibility has been excluded from the lemma, and for the other possibilities, it is not hard to check that N' is consistent with $xy|z$.

Moreover, the case that $2P$ contains a path from an internal vertex w_1 of P_{uz} to an internal vertex w_2 of P_{vy} does not occur, since it would imply that w_2 is a reticulation vertex different from a second reticulation vertex below w_1 on P_{uz} (which exists because $z \notin S$) and different from yet another reticulation vertex below w_2 on P_{vy} (which exists because $y \notin S$), contradicting the fact that N is a simple *level-*≤ 2 network. For similar reasons $2P$ does not contain a path from an internal vertex of P_{vy} to an internal vertex of P_{uz}. Let us then consider the remaining case i.e. that $2P$ contains the path P_{uv}. The fact that $2P$ is a minimal subgraph covering all parents of leaves in S implies that u is either an LCA of two leaves in S or a descendant of such an LCA. We know that $2P$ does not contain the root, hence, u is not the root. Observe that u is also not the parent of a leaf since $z \notin S$ would be the only candidate for such a leaf and $2P$ contains no parents of leaves that are not in S. Hence, u is a split vertex that is not equal to the root and does not have a leaf as child.

Below, we prove a claim stating that there exists a path Q_{rz} from the root of N to z that does not intersect $2P$. This path also exists in N_c. If it does not intersect the path from v to y in N, and hence, the path from v_c to y in N', then this implies immediately that $xy|z$ is consistent with N', a contradiction. Otherwise, Q_{rz} joins P_{vy} at a reticulation vertex, r_1 say. Moreover, Q_{rz} must join P_{uz} at a reticulation vertex r_2, which is a descendant of r_1. The facts that u is a split vertex without a leaf child and there exist internally vertex disjoint paths $u \to r_1$ (consisting of P_{uv} and part of P_{vy}) and $u \to r_2$ (consisting of the first part of P_{uz}) imply that N is of type $2a$ with z on side F and y on side E. Moreover, u is its unique split vertex without leaf child and v is on side C (and hence, the upper part of C is in S, and by Observation 2, all of C is in S). Because $2P$ contains the parent of $x \in S$ and because Q_{rz} is completely disjoint from $2P$, it follows that x is on C and no leaf of E is in S. Since $2P$ is a minimal subgraph covering S, and since u is contained in $2P$, also the lower part of A or the upper part of D is in S. In any case, this implies that all of D is in S (by Observation 2 or by the fact that $ac|d$ is not consistent with N for a on A, c on C, and d on D). In this case, N' is constructed as in Fig. 7, again contradicting inconsistency with $xy|z$.

Claim 3. There exists a path from the root of N to z that does not intersect $2P$.

Proof. Observation 2 immediately implies that the claim is true if z is not below any reticulation vertex. In case $z \notin S$ is on side E or F of $2a$, side H of $2b$, side G or H of $2c$, or on side F of $2d$, the claim follows basically from the fact that no two leaves in S have the root as LCA (Observation 1). For $2a$, we need the additional argument that if $2P$ contains part of E (above z) and part of either A or D, then it also contains the whole lower part of E (by Observation 2) and it contains F (as $ae|f$, $de|f$ are not consistent) contradicting that $z \notin S$.

Now suppose z is on side G of $2b$. Then $2P$ cannot contain part of side A, since through Observation 2, this would imply that all of D, C, and E would be in S, and therefore, also all of G, contradicting $z \notin S$. If both D and C or E contain vertices in S, then for $a \in D$, $b \in C$, or E, the triplet $ab|z$ cannot be consistent with N, implying that $z \in S$, a contradiction. Finally, suppose z is on side E of $2d$. If there is no path from the root to z which is disjoint from $2P$, then $2P$ contains part of A or part of C and part of B or both. In any case, no triplet $ab|z$ with $a, b \in S$ is consistent with N, a contradiction. □

The proof of this claim completes the proof of Lemma 6.

The lemma essentially shows that any single maximal SN-set (with some well-specified exceptions) can be "pushed" below the highest cut-arc in a simple level-≤ 2 network without losing consistency with the triplet set. We wish to argue, by induction, that all maximal SN-sets (with the same exceptions) can be "pushed" below the highest cut-arcs. We first need the next lemma.

Lemma 7. *Let $\mathcal{C}$ be a correct partition for dense triplet set T. There is a bijection between the maximal SN-sets of T and the maximal SN-sets of $T\nabla\mathcal{C}$ mapping a maximal SN-set $\{C_1, \ldots, C_\ell\}$ of $T\nabla\mathcal{C}$ to the maximal SN-set $C_1 \cup \ldots \cup C_\ell$ of T.*

Proof. Let N be a level-2 network consistent with T whose sets of leaves below the highest cut-arcs are given by $\mathcal{C}$. First, consider an SN-set of T that is a union $S = C_1 \cup \ldots \cup C_\ell$ of sets from $\mathcal{C}$. Define S' as $\{C_1, \ldots, C_\ell\}$ and suppose S' is not an SN-set of $T\nabla\mathcal{C}$. Then there exist distinct $C_i, C_k \in S'$ and $C_j \notin S'$ such that $C_iC_j|C_k \in T\nabla\mathcal{C}$. This implies the existence of leaves $x \in C_i$, $y \in C_j$, $z \in C_k$, and a triplet $xy|z \in T$ such that $x, z \in S$ and $y \notin S$, contradicting the fact that S is an SN-set. Thus, S' is an SN-set of $T\nabla\mathcal{C}$.

Second, consider an SN-set $S' = \{C_1, \ldots, C_\ell\}$ of $T\nabla\mathcal{C}$. If $S := C_1 \cup \ldots \cup C_\ell$ is not an SN-set of T, then there exists a triplet $xy|z \in T$ with $x, z \in S$, and $y \notin S$. Because $y \notin S$, y is not in the same partition element C_j as x or z, or in other words, y is not below the same highest cut-arc as x or z in N. If x and z were both in the same partition element C_i, then they were both below the same highest cut-arc in the network N. But this contradicts the fact that N is consistent with $xy|z \in T$. So $x \in C_i$, $y \in C_j$, and

$z \in C_k$, where $C_i, C_k \in S'$ is distinct and $C_j \notin S'$. But then $xy|z \in T$ implies $C_iC_j|C_k \in T\nabla\mathcal{C}$, contradicting the fact that S' is an SN-set.

Finally, the one-to-one correspondence we have just established between SN-sets of $T\nabla\mathcal{C}$ and SN-sets of T of the form $S = C_1 \cup \ldots \cup C_\ell$ together with the fact that, by Lemma 1, any maximal SN-set of T is of that form, proves the lemma. □

We are ready to extend Lemma 6 to the collection of all SN-sets.

Lemma 8. *If N is a simple* level-≤ 2 *network consistent with a dense triplet set T, then there exists a level-2 network N' consistent with T such that each maximal SN-set is equal to the set of leaves below a highest cut-arc, except for at most one maximal SN-set S for which* $(\mathrm{Collapse}(N'), S)$ *is exceptional.*

Proof. The proof is by induction on the number i of maximal SN-sets that are not equal to a set of leaves below a highest cut-arc of N. Lemma 6 deals with the induction basis $i = 1$.

To prove the induction step, suppose $i \geq 2$. By the definition of exceptional, there can not be two maximal SN-sets S_1, S_2 such that both (N, S_1) and (N, S_2) are exceptional. Hence, there is at least one maximal SN-set S' that is not equal to the set of leaves below a highest cut-arc and for which (N, S') is not exceptional. By Lemma 6, there exists a level-2 network N'' consistent with T in which S' is the set of leaves below a highest cut-arc a and the other sets of leaves below highest cut-arcs are the same as in N, i.e., they are singletons. Let $N_{S'}$ be the subnetwork of N'' below highest cut-arc a. Let $\mathcal{C}$ be the collection of sets of leaves below highest cut-arcs of N''. $\mathrm{Collapse}(N'')$ is a simple level-≤ 2 network consistent with $T\nabla\mathcal{C}$. Since, by Lemma 7, $\mathrm{Collapse}(N'')$ has $i - 1$ maximal SN-sets of $T\nabla\mathcal{C}$ that are not below highest cut-arcs, the induction hypothesis can be applied. It follows that there exists a network N_{ind} consistent with $T\nabla\mathcal{C}$ such that each maximal SN-set of $T\nabla\mathcal{C}$ is equal to the set of leaves below a highest cut-arc, except for at most one maximal SN-set S for which $\mathrm{Collapse}(N_{ind}, S)$ is exceptional. We construct N' from N_{ind} by replacing the leaf S' by the network $N_{S'}$. The lemma follows. □

We extend the above result to general (nonsimple) level-2 networks. The following theorem describes how to construct a correct partition from the collection of maximal SN-sets. Given a level-2 network N and an SN-set S, we use $\mathrm{Collapse}(N, S)$ to denote the result of replacing, in both N and S, each set of leaves below a highest cut-arc of N by a single leaf.

Theorem 3. *Let T be a dense triplet set consistent with some level-2 network N. Then, there exists a level-2 network N' consistent with T such that each maximal SN-set is equal to the set of leaves below a highest cut-arc, except for at most one maximal SN-set S for which* $\mathrm{Collapse}(N', S)$ *is exceptional.*

Proof. Let N be a level-2 network consistent with T. Suppose $a_1, \ldots, a_q$ are the highest cut-arcs of N and C_i is the set of leaves below a_i. Then, $\mathcal{C} := \{C_1, \ldots, C_q\}$ is a correct partition for T. Denote the subnetwork of N below a_i by $N(C_i)$. First, note that $\mathrm{Collapse}(N)$ is a simple level-≤ 2 network on leaf set $\mathcal{C}$ consistent with $T\nabla\mathcal{C}$. By Lemma 8, there exists a level-2 network $N_{\mathcal{C}}$ on leaf set $\mathcal{C}$, consistent with $T\nabla\mathcal{C}$, and having all maximal SN-sets of $T\nabla\mathcal{C}$, with at most one exception, below the highest cut-arcs. The exception occurs if there is a maximal SN-set S of $T\nabla\mathcal{C}$ such that $(\mathrm{Collapse}(N), S)$ is exceptional: in that case, one maximal SN-set of $T\nabla\mathcal{C}$ is equal to the union of at least two sets of leaves below the highest cut-arcs in $N_{\mathcal{C}}$. Let N' be the result of replacing each leaf C_i in $N_{\mathcal{C}}$ by the subnetwork $N(C_i)$ of N. Then N' is consistent with T by Lemma 5.

Lemma 7 describes a bijection between the maximal SN-sets of $T\nabla\mathcal{C}$ and the maximal SN-sets of T. Since the maximal SN-sets of $T\nabla\mathcal{C}$, with one possible exception, occur below the highest cut-arcs of $N_{\mathcal{C}}$, the corresponding maximal SN-sets of T occur below the highest cut-arcs of N', also with one possible exception. An exceptional SN-set of $T\nabla\mathcal{C}$ consists of several leaves below the highest cut-arcs in $N_{\mathcal{C}}$, and hence, the corresponding exceptional SN-set of T is the union of several sets of leaves below the highest cut-arcs in N'. □

Now suppose an input triplet set T is consistent with some level-2 network. The above theorem implies that, after possibly splitting one maximal SN-set, we obtain a correct partition and the problem then essentially reduces to constructing a simple level-≤ 2 network for the induced triplet set. The following lemma tells us how to handle the exceptional cases.

Lemma 9. *Let T be a dense set of triplets and N' be a network with the properties described in Theorem 3. If there is an SN-set S' of T such that* $\mathrm{Collapse}(N', S')$ *is exceptional, then there exists an SN-set $S'' \subset S'$ such that S'' together with the maximal sets in $\{X | X \text{ is an SN-set}, S'' \cap X = \emptyset\}$ forms a correct partition for T.*

Proof. First consider the case that $\mathrm{Collapse}(N')$ is of type $2c$ and S' consists of the leaves that are in N' below the two reticulations of $\mathrm{Collapse}(N')$. In this case, the set of leaves below either of the two reticulations forms an SN-set and the lemma follows.

Now consider the case that $\mathrm{Collapse}(N')$ is of type $2a$ or $2d$ and S' consists of the leaves that are in N' below the paths corresponding to sides E and C of $\mathrm{Collapse}(N')$. Let S'' be the set of leaves that are in N' below the path corresponding to side E of $\mathrm{Collapse}(N')$. Consider the set $\mathcal{R}$ consisting of S'' and the maximal sets in $\{X | X \text{ is an SN-set}, S'' \cap X = \emptyset\}$ and observe that $(\mathrm{Collapse}(N'), X)$ is not exceptional for any $X \in \mathcal{R}$. Thus, each set in $\mathcal{R}$ can be pushed below a single cut-arc by Lemma 6 and the lemma follows. □

This lemma shows that whenever we have to split an SN-set, we can modify Algorithm SL2 to find a simple level-2 network faster, since we either already know the two reticulation leaves or we know which leaves are on side E.

The general outline of our Algorithm 3, which constructs level-2 networks from dense triplet sets, is as follows: First, we compute the maximal SN-sets. If there are precisely two maximal SN-sets, then we recursively create two level-2 networks for the two maximal SN-sets and connect their roots to a new root. Otherwise, we try splitting each maximal SN-set in turn and we try the case where no maximal SN-set is split. If $\mathcal{S}$ is the obtained set of SN-sets, then we compute the induced set of triplets $T\nabla\mathcal{S}$ and try to construct a simple level-1 or level-2 network N consistent with $T\nabla\mathcal{S}$ using Algorithm 1. We recursively create level-2 networks for each SN-set in $\mathcal{S}$ and replace

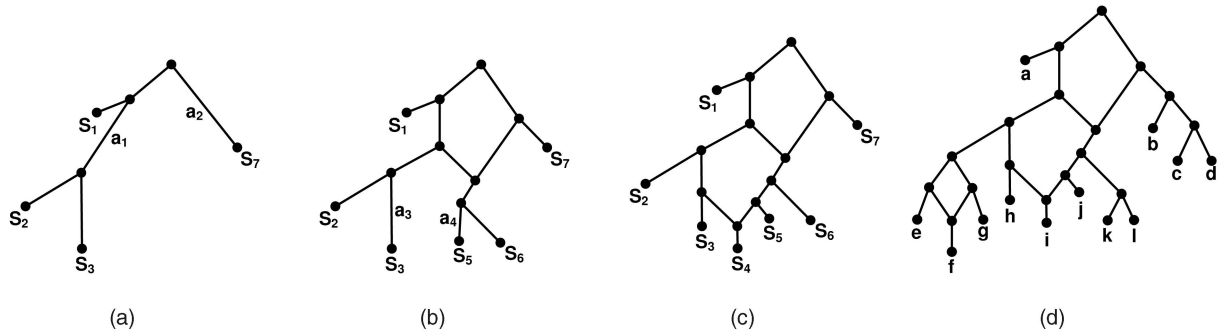


Fig. 9. Example of the construction of a level-2 network by Algorithm 3.

each leaf of N by the corresponding, recursively created, level-2 network. The whole algorithm is presented in pseudocode in Algorithm 3.

Algorithm 3. LEVEL2
1: $N := \emptyset$
2: compute the set SN of SN-sets and $maxSN$ of maximal SN-sets of T
3: **if** $|maxSN| = 2$ **then**
4: N consists of a root connected to two leaves: the elements of $maxSN$
5: **else**
6: **if** $T\nabla maxSN$ is consistent with a simple level-1 network **then**
7: let N be such a network
8: **else if** $T\nabla maxSN$ is consistent with a simple level-2 network **then**
9: let N be such a network
10: **else**
11: **for** $S' \in maxSN$ **do**
12: **for** $S'' \in SN$ with $S'' \subseteq S'$ **do**
13: compute the set $maxSN'$ consisting of S'' and maximal sets in $\{X \in SN, S''X = 0\}$
14: **if** $T\nabla maxSN'$ is consistent with a simple level-2 network of type 2c where the elements of S' are the two reticulation leaves or with a simple level-2 network of type 2a or 2d where the elements of S' are on side E and (part of) side C **then**
15: let N be such a network
16: **for** each leaf V of N **do**
17: recursively create a level-2 network N_V consistent with $T|V$
18: **if** $N \neq \emptyset$ and all $N_X \neq \emptyset$ **then**
19: replace each leaf V of N by the recursively created N_V.
20: **return** N
21: **else**
22: **return** $\emptyset$

Theorem 4. *Algorithm 3 constructs, in $O(|T|^{\frac{8}{3}})$ time, a level-2 network consistent with a dense set of triplets T if and only if such a network exists.*

Proof. Correctness follows from the above. For the running time, recall that the algorithm recursively constructs simple level-1 and -2 networks, which become the biconnected components of the network. If such a biconnected component has a outgoing arcs, then the simple level-1 algorithm takes $O(a^3)$ time [22] and the simple level-2 algorithm takes $O(a^8)$ time by Theorem 2. If no solution is found, the algorithm loops over all maximal SN-sets (line 11) and loops for one, say S', through all SN-sets S'' included in it. This results in a collection of at most a SN-sets, and finding a simple level-2 network with leaves corresponding to those SN-sets (line 15) now takes only $O(a^6)$, since we can adapt Algorithm 1 to run much faster for the exceptional cases. First, if S' consists of just two elements S_1, S_2, we simply try choosing $x = S_1$ and $C = S_2$ in lines 2 and 5. Indeed, the network we are looking for now is of type $2c$ and has reticulation leaves S_1 and S_2. Second, we try choosing S'' as the caterpillar in line 5. In this case, we also know that we have to subdivide at least two low edges, because we know that x has to be below S'' and that S'' has to be below the other elements of S'. Since there are $O(n)$ SN-sets in total, the adapted Algorithm 1 is repeated at most $O(n)$ times. Hence, constructing a single biconnected component with a outgoing arcs takes at most $O(a^8 + n \cdot a^6)$ time.

Suppose N is the level-2 network we are constructing. Let $a_1, \ldots, a_m$ denote the numbers of arcs leaving its m nontrivial biconnected components. It is easy to see that N contains $O(n)$ arcs and hence that $a_1 + \ldots + a_m = O(n)$. Because

$$(a_1^8 + n \cdot a_1^6) + \ldots + (a_m^8 + n \cdot a_m^6) \leq (a_1 + \ldots + a_m)^8 + n \cdot (a_1 + \ldots + a_m)^6,$$

the total time needed for constructing biconnected components in lines 6-15 is $O((a_1 + \ldots + a_m)^8 + n \cdot (a_1 + \ldots + a_m)^6)$ and hence $O(n^8)$. The computation of maximal SN-sets in line 2 takes $O(n^3)$ time [22] and is executed $O(n)$ times. All other computations can also be done in $O(n^3)$ time and are executed at most $O(n^2)$ times. Thus, Algorithm 3 runs in time $O(n^8) = O(|T|^{\frac{8}{3}})$ since T is dense. □

An example of the overall algorithm is displayed in Fig. 9. Suppose the input triplet set T is consistent with the level-2 network N in Fig. 3. Then, $\mathrm{Collapse}(N)$ is equal to the simple level-2 network in Fig. 9c. Since this network does not correspond to one of the exceptional cases, it is not necessary to split maximal SN-sets in this case. Let us assume that Algorithm 4 finds maximal SN-sets $SN = \{S_1, \ldots, S_7\}$ (it is also possible that the maximal SN-sets are different, in which case, the algorithm constructs a slightly different network

that is also consistent with T). Then, the induced triplet set $T\nabla SN$ is consistent with the simple level-2 network in Fig. 9c. Algorithm 1 finds this simple level-2 network as follows. In some iteration, it removes leaf S_4 and caterpillar-set $\{S_5, S_6\}$ and constructs the tree (Fig. 9a) consistent with the remaining triplets. (For clarity, and because it is, in this example, not necessary, we have not shown the dummy root.) At some iteration, SL2 will subdivide arcs a_1 and (high arc) a_2 and hang the caterpillar with leaves S_5, S_6 below the new vertices, giving the network in Fig. 9b. By subsequently subdividing a_3 and a_4 and hanging leaf S_4 below the new vertices, SL2 indeed obtains the simple level-2 network in Fig. 9c. Replacing each leaf S_i with a recursively constructed level-2 network consistent with $T|S_i$ gives the complete level-2 network in Fig. 9d.

4 Experimental Results

The algorithm has been implemented in Java and made publicly available in an open-source repository [27]. It has been applied to experimental data consisting of sequences from different isolates of the yeast *Cryptococcus gattii*. This yeast is potentially dangerous and an ongoing outbreak on the West Coast of Canada [26], which started in 1999, has caused many infections and even some fatalities. We have constructed a phylogenetic network for these isolates as a tool to find the origin of this *C. gattii* outbreak. We have blinded the names of the isolates here since the biological part of the research has not yet reached a conclusion.

We have used *C. gattii* sequences consisting of six genes and 3,356 nucleotides in total. Given the *C. gattii* sequences, we have constructed a set of triplets as follows. First, all identical sequences are combined into a single *sequence type (ST)*. From 71 sequences, we obtained 30 STs. One of the sequence types (which is only distantly related to the others) is used as an outgroup and we have applied the Maximum Likelihood method of PHYML [9] to each subset of four sequence types that includes the outgroup. Each output tree of PHYML gives us one input triplet for our Algorithm 3. Running our algorithm for all STs tells that there exists no level-2 network consistent with all triplets. Therefore, we have applied our algorithm to a set containing as many STs as possible (where certain important STs get priority over others) without destroying level-2 realizability. This set has been found by searching through all the subsets. The subset contains 21 of the 30 STs, including 12 of the 14 STs to which we gave priority because they were expected to be involved in recombinations. Given this subset and all triplets, the execution of Algorithm 3 took 0.8 second on a Pentium 4, 3-GHz PC with 1 GB memory. The resulting level-2 network is displayed in Fig. 10. This network is consistent with all 1,330 triplets that were generated over this subset of the taxa. The figure displays both a reticulate pattern and a dichotomous and tree-like structure. Our method is able to differentiate and visualize these.

Including the search for a level-2-realizable subset of the STs and the construction of all input triplets, the total time needed was 19 minutes and 12 seconds. The long time is caused by the brute-force search for a large level-2-realizable subset. In the future, we suggest using a local search technique to search through the space of subsets. If, on the other hand, a set of taxa is considered that is already level-2-realizable, then our LEVEL2 algorithm can be used directly and constructs a network very fast. We note that, even in cases where we can find a level-2 network for only a subset of the taxa, our level-2 networks could give a better representation of reality than a tree or a level-1 network.

5 Conclusion and Open Questions

In this paper, we have shown that in polynomial time, we can decide whether a dense triplet set is consistent with a level-2 network, and if so, construct such a network. It brings more complex, interwoven forms of evolution within reach of triplet methods. The method has shown to be useful in practice. Many open questions and challenges remain as follows:

1. **Applicability.** First, practical experiments, one of which we reported in Section 4, show that the implementation we have made is fast and accurate. It remains interesting to test it on other phylogenetic data. Certain parts of the community criticize the validity of many quartet-based methods, saying that it is, in practice, far harder to generate high-quality input quartets than is often claimed. The *short quartet method* [7] has been discussed as a way of addressing this critique. This debate needs to be addressed in the context of this paper.
2. **Complexity.** Is the dense version polynomial time solvable for all fixed k? In this regard, generalizing the crucial Theorem 3 will probably be difficult, because it is at this moment not clear whether the technique of "pushing" maximal SN-sets below the cut-arcs generalizes to level 3 and higher. It is also interesting to improve the running time of our algorithm and see how far this is necessary for further applications. At the moment, the running time $O(n^8)$ seems fast enough in practice.

   Recent research has shown that, in the nondense case, the level-k problem is NP-hard for all fixed $k \geq 1$ [18]. For dense triplet sets, the complexity of the problem for nonfixed k is still open. The related question about the computational complexity of computing the smallest k for which there exists a level-k network that is consistent with a dense set of triplets is also still open.
3. **Bounds.** Jansson et al. [22] determine constructive lower and upper bounds on the value τ for which the following statement is true: for each set of triplets T, not necessarily dense, there exists some level-1 network N that is consistent with at least $\tau|T|$ triplets in T. Recent research has proven that, for level-2 networks, $\tau \geq 0.61$ [5]. Computing the maximum value of τ for level 2 remains an open problem, as does the computation of lower and upper bounds on τ for level-3 networks and higher.
4. **Building all networks.** It is not clear whether it is possible to adapt our algorithm to generate *all* level-2 networks consistent with the input triplet set. If so, then such an adaptation could (even in the case that exponentially many networks are produced) be very useful for comparing the plausibility and/or relative similarity of the various solutions.
5. **Properties of constructed networks.** Under what conditions on the triplet set T is there only one network N consistent with T? Under what conditions does T permit some solution N such that the set of all

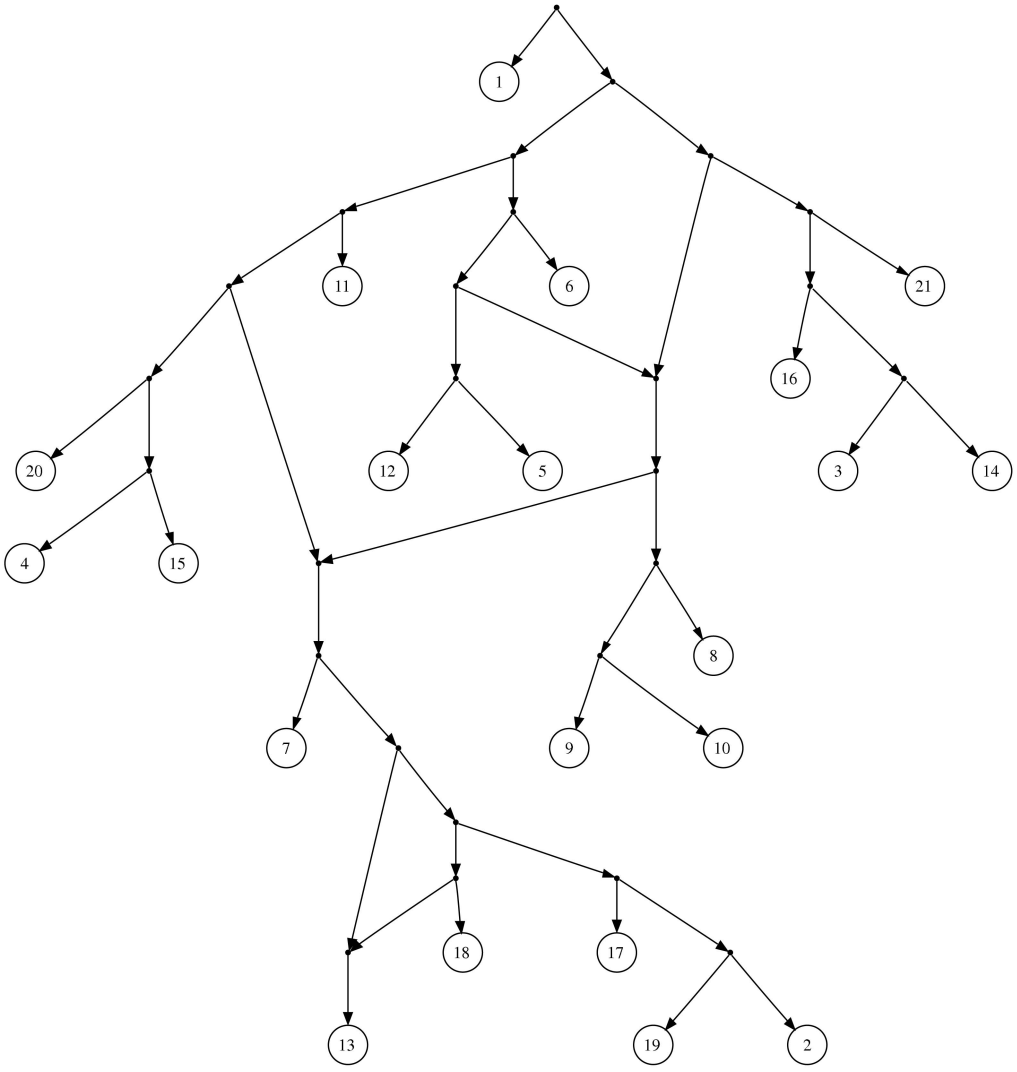


Fig. 10. The network constructed by Algorithm 3 for the triplets based on the yeast data set.

triplets consistent with N is exactly equal to T? These questions are also valid for level-1 networks.

6. **Different triplet restrictions.** Density is only one of very many possible restrictions on the input triplets. Interesting alternatives are what we call *minimal density* and *extreme density*. A minimally dense triplet set has exactly one triplet for every combination of three leaves. In the extreme dense case, we assume that the set of input triplets is exactly equal to the set of triplets consistent with some network.
7. **Confidence.** At the moment, all input triplets are assumed to be correct. Is there scope for attaching a confidence measure to each input triplet and optimizing on this basis? This is related to the problem of ensuring that certain triplets are *excluded* from the output network, as explored in [11].

ACKNOWLEDGMENTS

The authors thank Katharina Huber for her useful ideas and many interesting discussions, and Thu-Hien To for spotting an ommission in our original case analysis. Part of this research was funded by the Dutch BSIK/BRICKS project.

REFERENCES

[1] A.V. Aho, Y. Sagiv, T.G. Szymanski, and J.D. Ullman, "Inferring a Tree from Lowest Common Ancestors with an Application to the Optimization of Relational Expressions," *SIAM J. Computing,* vol. 10, no. 3, pp. 405-421, 1981.

[2] D. Bryant, "Building Trees, Hunting for Trees, and Comparing Trees: Theory and Methods in Phylogenetic Analysis," PhD dissertation, Univ. of Canterbury, 1997.

[3] D. Bryant and V. Moulton, "Neighbor-Net: An Agglomerative Method for the Construction of Phylogenetic Networks," *Molecular Biology and Evolution,* vol. 21, no. 2, pp. 255-265, 2004.

[4] D. Bryant and M. Steel, "Constructing Optimal Trees from Quartets," *J. Algorithms,* vol. 38, no. 1, pp. 237-259, 2001.

[5] J. Byrka, P. Gawrychowski, K.T. Huber, and S. Kelk, "Worst-Case Optimal Approximation Algorithms for Maximizing Triplet Consistency within Phylogenetic Networks," arXiv:0710.3258v3 [q-bio.PE], 2008.

[6] H.-L. Chan, J. Jansson, T.W. Lam, and S.-M. Yiu, "Reconstructing an Ultrametric Galled Phylogenetic Network from a Distance Matrix," *J. Bioinformatics and Computational Biology,* vol. 4, no. 4, pp. 807-832, 2006.

[7] P.L. Erdös, M.A. Steel, L.A. Szekely, and T. Warnow, "A Few Logs Suffice to Build (Almost) All Trees (Part II)," *Theoretical Computer Science,* vol. 221, no. 1, pp. 77-118, 1999.

[8] L. Gaasieniec, J. Jansson, A. Lingas, and A. Östlin, "On the Complexity of Constructing Evolutionary Trees," *J. Combinatorial Optimization,* vol. 3, pp. 183-197, 1999.

[9] S. Guindon and O. Gascuel, "A Simple, Fast, and Accurate Algorithm to Estimate Large Phylogenies by Maximum Likelihood," *Systematic Biology,* vol. 52, no. 5, pp. 696-704, 2003.
[10] D. Gusfield, S. Eddhu, and C. Langley, "Optimal, Efficient Reconstruction of Phylogenetic Networks with Constrained Recombination," *J. Bioinformatics and Computational Biology,* vol. 2, pp. 173-213, 2004.
[11] Y.-J. He, T.N.D. Huynh, J. Jansson, and W.-K. Sung, "Inferring Phylogenetic Relationships Avoiding Forbidden Rooted Triplets," *J. Bioinformatics and Computational Biology,* vol. 4, no. 1, pp. 59-74, 2006.
[12] M. Holder and P.O. Lewis, "Phylogeny Estimation: Traditional and Bayesian Approaches," *Nature Rev. Genetics,* vol. 4, pp. 275-284, 2003.
[13] K.T. Huber, B. Oxelman, M. Lott, and V. Moulton, "Reconstructing the Evolutionary History of Polyploids from Multilabeled Trees," *Molecular Biology and Evolution,* vol. 23, no. 9, pp. 1784-1791, 2006.
[14] D.H. Huson and D. Bryant, "Application of Phylogenetic Networks in Evolutionary Studies," *Molecular Biology and Evolution,* vol. 23, no. 2, pp. 254-267, 2006.
[15] D.H. Huson and T.H. Klöpper, "Beyond Galled Trees—Decomposition and Computation of Galled Networks," *Proc. Int'l Conf. Research in Computational Molecular Biology (RECOMB '07),* pp. 211-225, 2007.
[16] L.J.J. van Iersel, J.C.M. Keijsper, S.M. Kelk, and L. Stougie, "Constructing Level-2 Phylogenetic Networks from Triplets," arXiv:0707.2890v1 [q-bio.PE], 2007.
[17] L.J.J. van Iersel, J.C.M. Keijsper, S.M. Kelk, L. Stougie, F. Hagen, and T. Boekhout, "Constructing Level-2 Phylogenetic Networks from Triplets," *Proc. 12th Ann. Int'l Conf. Research in Computational Molecular Biology (RECOMB '08),* pp. 450-462, 2008.
[18] L.J.J. van Iersel, S.M. Kelk, and M. Mnich, "Uniqueness, Intractability and Exact Algorithms: Reflections on Level-k Phylogenetic Networks," to be published in *J. Bioinformatics and Computational Biology.*
[19] J. Jansson, personal communication, Kyushu Univ., 2007.
[20] J. Jansson, "On the Complexity of Inferring Rooted Evolutionary Trees," *Proc. Brazilian Symp. Graphs, Algorithms and Combinatorics (GRACO '01),* pp. 121-125, 2001.
[21] J. Jansson, J.H.-K. Ng, K. Sadakane, and W.-K. Sung, "Rooted Maximum Agreement Supertrees," *Algorithmica,* vol. 43, pp. 293-307, 2005.
[22] J. Jansson, N.B. Nguyen, and W.-K. Sung, "Algorithms for Combining Rooted Triplets into a Galled Phylogenetic Network," *SIAM J. Computing,* vol. 35, no. 5, pp. 1098-1121, 2006.
[23] J. Jansson and W.-K. Sung, "Inferring a Level-1 Phylogenetic Network from a Dense Set of Rooted Triplets," *Proc. Conf. Computing and Combinatorics (COCOON '04),* pp. 462-472, 2004.
[24] J. Jansson and W.-K. Sung, "Inferring a Level-1 Phylogenetic Network from a Dense Set of Rooted Triplets," *Theoretical Computer Science,* vol. 363, pp. 60-68, 2006.
[25] T. Jiang, P.E. Kearney, and M. Li, "A Polynomial Time Approximation Scheme for Inferring Evolutionary Trees from Quartet Topologies and Its Application," *SIAM J. Computing,* vol. 30, no. 6, pp. 1942-1961, 2000.
[26] S. Kidd, F. Hagen, R. Tscharke, M. Huynh, K. Bartlett, M. Fyfe, L. MacDougall, T. Boekhout, K.J. Kwon-Chung, and W. Meyer, "A Rare Genotype of Cryptococcus Gattii Caused the Cryptococcosis Outbreak on Vancouver Island (British Columbia, Canada)," *Proc. Nat'l Academy of Sciences USA,* vol. 101, pp. 17258-17263, 2004.
[27] "LEVEL2: A Fast Method for Constructing Level-2 Phylogenetic Networks from Dense Sets of Rooted Triplets," http://homepages.cwi.nl/~kelk/level2triplets.html and http://level2.sourceforge.net/, 2009.
[28] V. Makarenkov, "T-REX: Reconstructing and Visualizing Phylogenetic Trees and Reticulation Networks," *Bioinformatics,* vol. 17, no. 7, pp. 664-668, 2001.
[29] V. Makarenkov, D. Kevorkov, and P. Legendre, "Phylogenetic Network Reconstruction Approaches," *Applied Mycology and Biotechnology,* vol. 6. pp. 61-97, 2006.
[30] B.M.E. Moret, L. Nakhleh, T. Warnow, C.R. Linder, A. Tholse, A. Padolina, J. Sun, and R. Timme, "Phylogenetic Networks: Modeling, Reconstructibility, and Accuracy," *IEEE/ACM Trans. Computational Biology and Bioinformatics,* vol. 1, no. 1, pp. 13-23, Jan.-Mar. 2004.
[31] C. Semple and M. Steel, *Phylogenetics.* Oxford Univ. Press, 2003.
[32] M. Steel, "The Complexity of Reconstructing Trees from Qualitative Characters and Subtrees," *J. Classification,* vol. 9, pp. 91-116, 1992.
[33] R.E. Tarjan and U. Vishkin, "An Efficient Parallel Biconnectivity Algorithm," *SIAM J. Computing,* vol. 14, no. 4, pp. 862-874, 1985.
[34] L. Wang, K. Zhang, and L. Zhang, "Perfect Phylogenetic Networks with Recombination," *J. Computational Biology,* vol. 8, no. 1, pp. 69-78, 2001.
[35] B.Y. Wu, "Constructing the Maximum Consensus Tree from Rooted Triples," *J. Combinatorial Optimization,* vol. 8, pp. 29-39, 2004.

Leo van Iersel received the MS degree in applied mathematics from the Universiteit Twente, The Netherlands, in 2004. He received the PhD degree from the Technische Universiteit Eindhoven, The Netherlands, in January 2009. He now works as a postdoctoral researcher at the University of Canterbury, Christchurch, New Zealand. His research is concerned with the design of algorithms for phylogenetics and other areas of computational biology.

Judith Keijsper received the master's and PhD degrees from the Universiteit van Amsterdam in 1994 and 1998, respectively, where she worked with Lex Schrijver on combinatorial algorithms for graph problems. She worked as a postdoctoral researcher at Leibniz-IMAG, Grenoble, France, and as an assistant professor at the Universiteit Twente, The Netherlands, for a short period of time, and then moved to the Technische Universiteit Eindhoven, The Netherlands, in 2000, where she is currently an assistant professor.

Steven Kelk is a postdoctoral researcher at the Centrum voor Wiskunde en Informatica in Amsterdam, working on applications of combinatorics to problems arising in computational biology. His current research interests are phylogenetic networks, elementary flux mode analysis, approximation algorithms, and computational complexity.

Leen Stougie has been a full professor in operations research at the Free University of Amsterdam, since 1 November 2008. Since 2000, he has been affiliated part-time to the Centrum voor Wiskunde en Informatica (CWI) in Amsterdam, where he works mainly on combinatorial problems related to computational biology. From 2004 until 2009, he was a project manager of "Algorithms and Processes in Life Sciences" within the Dutch BSIK-BRICKS project.

Ferry Hagen received the bachelor of applied sciences degree in molecular biology from Hogeschool Utrecht, The Netherlands, in 2005. Since 2006, he has been working toward the PhD degree in the Yeast Research Group at the CBS Fungal Biodiversity Center, Utrecht, The Netherlands. He investigates the epidemiology and virulence of the human pathogenic yeasts Cryptococcus gattii and Cryptococcus neoformans using a set of molecular and microbiological tools.

Teun Boekhout has a PhD degree and is currently working as a senior scientist at the CBS Fungal Diversity Center, Utrecht, The Netherlands. His main research interest is to understand fungal diversity from the evolutionary and functional points of view. One of his research topics is to understand the Cryptococcus neoformans/Cryptococcus gattii complex that comprises important human and animal pathogens. In order to do so, a wide diversity of molecular and phylogenetic approaches is applied.