An algorithm for reconstructing level-2 phylogenetic networks from trinets

van Iersel, Leo; Kole, Sjors; Moulton, Vincent; Nipius, Leonie

DOI
10.1016/j.ipl.2022.106300

Publication date
2022

Document Version
Final published version

Published in
Information Processing Letters

Information Processing Letters 178 (2022) 106300

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

An algorithm for reconstructing level-2 phylogenetic networks from trinets

Leo van Iersel [a,*,1], Sjors Kole [a], Vincent Moulton [b], Leonie Nipius [a]

[a] Delft Institute of Applied Mathematics, Delft University of Technology, Mekelweg 4, Delft, 2628 CD, the Netherlands
[b] School of Computing Sciences, University of East Anglia, NR4 7TJ, Norwich, United Kingdom

ARTICLE INFO

Article history:
Received 23 September 2021
Received in revised form 1 February 2022
Accepted 28 June 2022
Available online 5 July 2022
Communicated by Leah Epstein

Keywords:
Directed graph
Phylogenetic network
Polynomial-time algorithm
Subnetworks
Graph algorithms

ABSTRACT

Evolutionary histories for species that cross with one another or exchange genetic material can be represented by leaf-labelled, directed graphs called *phylogenetic networks*. A major challenge in the burgeoning area of phylogenetic networks is to develop algorithms for building such networks by amalgamating small networks into a single large network. The *level* of a phylogenetic network is a measure of its deviation from being a tree; the higher the level of a network, the less treelike it becomes. Various algorithms have been developed for building level-1 networks from small networks. However, level-1 networks may not be able to capture the complexity of some data sets. In this paper, we present a polynomial-time algorithm for constructing a rooted binary level-2 phylogenetic network from a collection of 3-leaf networks or *trinets*. Moreover, we prove that the algorithm will correctly reconstruct such a network if it is given all of the trinets in the network as input. The algorithm runs in time $O(t \cdot n + n^4)$ with t the number of input trinets and n the number of leaves. We also show that there is a fundamental obstruction to constructing level-3 networks from trinets, and so new approaches will need to be developed for constructing level-3 and higher level-networks.

1. Introduction

Phylogenetic networks are a generalization of phylogenetic trees that are commonly used to represent the evolutionary histories of species that cross with one another or exchange genetic material, such as plants and viruses. There are several classes of phylogenetic networks and various ways have been devised to build them – see e.g. [2,15] for recent surveys. Mathematically speaking, a *phylogenetic network* on a set of species X is basically a directed acyclic graph, with a single source or *root*, such that every sink or *leaf* has indegree 1 and the set of leaves is equal to X. In this paper, we shall only consider *recoverable, binary* phylogenetic networks, which we call *networks* for short. See Section 2 for formal definitions and Fig. 1 for examples.

Recently, there has been growing interest in the problem of building a network with leaf-set X from a collection of networks each of which having leaf-set equal to some subset of X in such a way that the input networks are each contained in the final network. Early work on this so-called *supernetwork problem* focused on building up networks from *phylogenetic trees*, that is, phylogenetic networks whose underlying graph is a tree. Several results have been presented for this problem, including algorithms for constructing networks from triplets, which are 3-leaved phylogenetic trees, (e.g. [6]) and from collections of phylogenetic trees all on leaf-set X (e.g. [17]) – for a recent summary of these approaches see [14]. However, an important issue with this strategy is that phylogenetic trees

* Corresponding author.
E-mail address: V.Moulton@uea.ac.uk (V. Moulton).

[1] Research funded in part by the Netherlands Organisation for Scientific Research (NWO) Vidi grant 639.072.602.

https://doi.org/10.1016/j.ipl.2022.106300

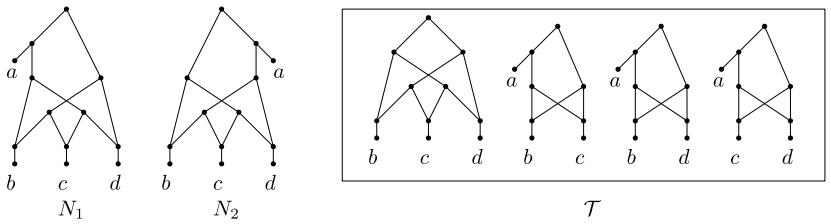

Fig. 1. Left: Two distinct level-3 networks N_1 and N_2 on the set $X = \{a, b, c, d\}$. Right: The set of trinets $\mathcal{T}$ that is contained in both N_1 and N_2.

do not necessarily *encode* phylogenetic networks, i.e., there are examples of distinct (non-isomorphic) networks that contain the same set of phylogenetic trees (see e.g. [3]), making it impossible to uniquely reconstruct such networks from their trees.

Motivated by this issue, in [4] it was proposed to build networks from collections of 3-leaved networks, or *trinets*. In that paper, the authors focused on building level-1 networks[2] where, in general, *level-k networks* are networks that can be converted into a tree by deleting at most k arcs from each biconnected component. In particular, they showed that level-1 networks are encoded by the trinets that they contain, and gave an algorithm for constructing a level-1 network on X from its trinets that is polynomial in $|X|$ (see also [13] for a more general algorithm). In [9] the encoding result was extended to the more general class of level-2 networks, and also to the distinct and quite broad class of so-called *tree-child* networks. Recently, in [14] it was also shown that *orchard* networks, which generalise tree-child networks, are encoded by their trinets, and an algorithm was given for constructing an orchard network from its trinets that is polynomial in the size of the vertex set of the network (whose size is not necessarily polynomial in $|X|$).

Intriguingly, in [8] it was shown that, as with trees, trinets do *not* encode networks in general. Indeed, in [14, p. 28] it was shown that even level-4 networks are not encoded by their trinets and, since level-2 networks are encoded by their trinets (see above), it was asked whether or not level-3 networks are encoded by their trinets (see also [1]). In the first result of this paper we answer this question – in particular, the two networks N_1 and N_2 in Fig. 1 are level-3 and are easily seen to be distinct and to contain the same set of trinets (see [12]). Hence, level-k networks are encoded by their trinets only if $k \leq 2$. As the algorithm in [4] can be used to uniquely reconstruct a level-1 network from its trinets, this leaves open the question of finding a polynomial algorithm for building a level-2 network from its trinets, which is the purpose of the rest of this paper. In particular, we shall present an algorithm which constructs a level-2 network on X from *any* set of trinets $\mathcal{T}$ whose leaf-set union is X that runs in $O(|\mathcal{T}||X| + |X|^4)$ time (Algorithm 1) and that is guaranteed to reconstruct a level-2 network from its set of trinets (Theorem 3). We now proceed by presenting some preliminaries, after which we shall describe our level-2 algorithm. We will conclude with a brief discussion of our results.

[2] In fact they considered the somewhat more general class of 1-nested networks.

2. Preliminaries

We refer the reader to [15, Chapter 10] for more information on the terminology and basic results on phylogenetic networks that we summarise in this section.

Definition 1. Let X be some finite set (corresponding to a set of species, say). A *binary phylogenetic network (on X)* is a directed acyclic graph with the following types of vertices: a single *root* with indegree 0 and outdegree 2; *tree-vertices* with indegree 1 and outdegree 2; *reticulations* with indegree 2 and outdegree 1; and *leaves* with indegree 1 and outdegree 0, where the leaves are in one-to-one correspondence with the elements of X.

Let N be a binary phylogenetic network on X, and suppose that u, v are two vertices in the vertex set of N. If there is a directed path from u to v (including the case that $u = v$), then we say that u is an *ancestor* of v and that v is a *descendant* of u. When (u, v) is an arc, we say that u is a *parent* of v and that v is a *child* of u. We say that (u, v) is a *cut-arc* if deleting (u, v) disconnects N. A set $A \subseteq X$ is called a *cut-arc set* in N if $A = X$ or A is the set of descendant leaves of v for some cut-arc (u, v). A cut-arc set A is *minimal* if $|A| > 1$ and there is no cut-arc set B with $|B| > 1$ and $B \subsetneq A$. A network is *simple* if it has no minimal cut-arc set except for X.

Now, suppose $A \subseteq X$. A *lowest stable ancestor (LSA)* of A in N is a vertex v such that, for all $a \in A$, all paths from the root to a contain v, and such that there is no descendant u of v with $u \neq v$ that satisfies this property. It is not difficult to see that the lowest stable ancestor is always unique for any $A \subseteq X$ [15, p. 263]. We say that N is *recoverable* if $LSA(X)$ is the root of X. In this paper, for simplicity, we shall call a recoverable, binary phylogenetic network on X a *network*. Only in statements of theorems we will mention these restrictions explicitly.

A *biconnected component* of a network is a maximal subgraph not containing any cut-arcs. A network is *level-k* if each biconnected component contains at most k reticulations. A level-k network is *strictly level-k* if it is not level-k' for any $k' < k$. This paper will mainly focus on level-2 networks; see Fig. 3 for an example.

A network on A is a *trinet* if $|A| = 3$ and a *binet* if $|A| = 2$. If T is a trinet or binet on A then we also use $L(T)$ to denote the set A. Furthermore, for a set of trinets and/or binets $\mathcal{T}$, we define $L(\mathcal{T}) = \cup_{T \in \mathcal{T}} L(T)$. We will now define the restriction of a network to a subset of X, which will be used to define the set of trinets contained in a network.

Definition 2. Let N be a network on X and $A \subseteq X$. The *restriction* of N to A, denoted $N|A$, is the network on A obtained from N by deleting all vertices that are not on a path from $LSA(A)$ to an element of A and subsequently replacing parallel arcs by single arcs and suppresssing indegree-1 outdegree-1 vertices, until neither of these operations is applicable.

The set of trinets $\mathcal{T}(N)$ of a network N on X is defined as $\{N|A \mid A \subseteq X, |A| = 3\}$. The set of binets and trinets $\overline{\mathcal{T}}(N)$ of a network N on X is defined as $\{N|A \mid A \subseteq X, 2 \leq |A| \leq 3\}$. Observe that $\overline{\mathcal{T}}(N)$ can be obtained from $\mathcal{T}(N)$.

We say that two networks N, N' on X are *equal* and write $N = N'$ if there is an isomorphism $f : V(N) \to V(N')$ such that, for all $x \in X$, $f(x)$ has the same label as x.

The following theorem forms the basis for our new level-2 algorithm.

Theorem 1 *([9]). Let N be a recoverable, binary level-2 network on X with $|X| \geq 3$. Then there exists no recoverable network $N' \neq N$ with $\mathcal{T}(N) = \mathcal{T}(N')$.*

2.1. Generators

Our algorithm will make heavy use of the underlying structure of biconnected components, which is called a "generator" (introduced in [16]) and defined as follows.

Definition 3. Let N be a simple network. The *underlying generator* of N is the directed multigraph G obtained from N by deleting all leaves and suppressing all indegree-1 outdegree-1 vertices. The arcs and indegree-2 outdegree-0 vertices of G are called *sides*. The arcs are also called *arc sides* and the indegree-2 outdegree-0 vertices also *reticulation sides*. We say that leaf x is *on side S* (or that side S *contains* x) if either

- S is a reticulation side of G and the parent of x in N, or
- S is an arc side of G obtained by suppressing indegree-1 outdegree-1 vertices of a path P in N and the parent of x lies on path P.

See Fig. 2 for all underlying generators of simple level-1 and level-2 networks.

To *attach* leaf x to a reticulation side S means adding x with an arc from S to x. To *attach* a list $(x_1, \ldots, x_l)$ of leaves to an arc side S means subdividing S to a path with l internal vertices $p_1, \ldots, p_l$ and adding leaves $x_1, \ldots, x_l$ with arcs $(p_1, x_1), \ldots, (p_l, x_l)$.

A trinet $T \in \mathcal{T}(N)$ is called a *crucial* trinet of a simple network N if it contains a leaf on each reticulation side of the underlying generator G of N and, for each pair of parallel arcs in G, a leaf on at least one of these two sides. Crucial trinets are of special interest because they have the same underlying generator as the network N.

Two reticulation sides u, v of a generator $G = (V, A)$ are *symmetric* if there exists an automorphism $f : V \to V$ of G with $f(u) = v$. The equivalence classes under this notion of symmetry are called *sets of symmetric reticulation sides*.

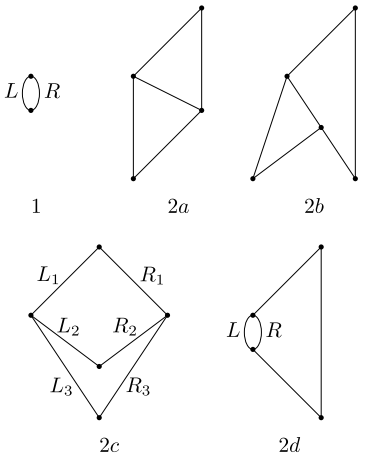

Fig. 2. The only underlying generator of a simple level-1 network and the four underlying generators of simple level-2 networks [9]. Generator 2c has three sets of symmetric arc sides $\{L_1, R_1\}, \{L_2, R_2\}, \{L_3, R_3\}$ while generators 1 and 2d have one set of symmetric arc sides $\{L, R\}$. Generator 2c is the only level-2 generator with symmetric reticulation sides.

Two arc sides (u, v), (u', v') of a generator $G = (V, A)$ are *symmetric* if there exists an automorphism $f : V \to V$ of G with $f(r) = r$ for each reticulation side r and such that $u' = f(u)$ and $v' = f(v)$. The equivalence classes under this notion of symmetry are called *sets of symmetric arc sides*, see Fig. 2. The idea behind this definition is that the reticulation sides of G are parents of leaves in N. In our algorithm, we will make heavy use of crucial trinets, which contain those leaves. Since they are labelled, we can distinguish them.

3. Algorithm

3.1. Outline

We work with multisets of trinets and binets because these may arise when collapsing or restricting trinet sets. Hence, let $\mathcal{T}$ be a multiset of binets and trinets. The high-level idea of the algorithm is to first find a minimal cut-arc set A. Then we construct $\mathcal{T}^*$ by collapsing A to a single leaf a^* and find a network N^* for $\mathcal{T}^*$ recursively. The next step is to construct $\mathcal{T}'$ from $\mathcal{T}$ by restricting to the taxa in A and to find a simple network N' for $\mathcal{T}'$. Finally, we construct N from N^* and N' by replacing a^* by N'. The pseudo code is in Algorithm 1.

Within our explanation of the algorithm we will also explain why in case the underlying set of $\mathcal{T}$ is $\mathcal{T}(N)$ for some recoverable level-2 network N, the algorithm correctly reconstructs N.

3.2. Finding a minimal cut-arc set

We first find a minimal cut-arc set of the level-2 network that we are constructing from $\mathcal{T}$. We find these sets using the following digraphs $\Omega_i(\mathcal{T})$ (see Fig. 3), which were introduced in [13] for level-1 networks.

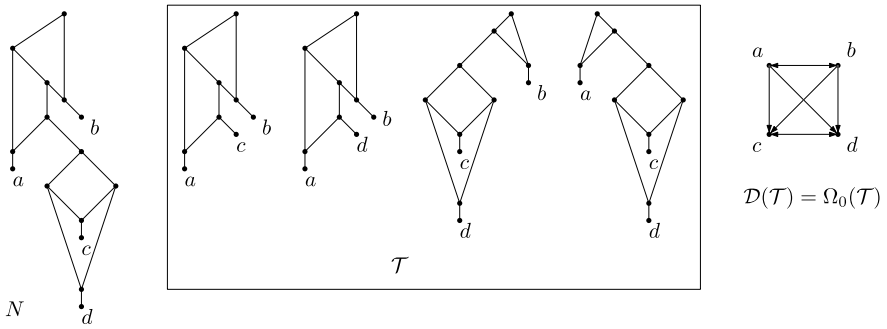

Fig. 3. A level-2 network N, its set of trinets $\mathcal{T} = \mathcal{T}(N)$ and the digraph $\Omega_0(\mathcal{T}) = \mathcal{D}(\mathcal{T})$. The set $\{c, d\}$ is the only minimal sink set in $\Omega_0(\mathcal{T})$ and the only minimal cut-arc set in N.

Algorithm 1: Constructing level-2 networks from trinets.

Data: Multiset $\mathcal{T}$ of level-2 trinets and (possibly) binets on taxon set X.
Result: Level-2 phylogenetic network N on X.
Find a cut-arc set A using Algorithm 2;
`//Find network N* with A collapsed`
Initialize $\mathcal{T}^* = \emptyset$ and let $a^* \notin X$ be a new taxon;
for $T \in \mathcal{T}$ *with* $L(T) \setminus A \neq \emptyset$ **do**
  if $L(T) \cap A = \emptyset$ **then** Add T to $\mathcal{T}^*$;
  else
    Pick $a \in L(T) \cap A$;
    Construct $T|(L(T) \setminus A \cup \{a\})$;
    Relabel a to a^* and add the resulting trinet or binet to $\mathcal{T}^*$;
Construct N^* from $\mathcal{T}^*$ by recursively running Algorithm 1;
`//Find simple network N' on A`
$\mathcal{T}' = \{T|(L(T) \cap A) \mid T \in \mathcal{T}, |L(T) \cap A| \geq 2\}$;
Construct a simple network N' for $\mathcal{T}'$ using Algorithm 3;
`//Combine N' and N*`
if $A \neq X$ **then**
  return *the network constructed from* N^* *and* N' *by identifying* a^* *with the root of* N'
else return N' ;

Definition 4. Given a multiset $\mathcal{T}$ of binets and trinets and $i \geq 0$, $\Omega_i(\mathcal{T})$ is the digraph with vertex set $L(\mathcal{T})$ and an arc (x, y) if at most i trinets $T \in \mathcal{T}$ with $x, y \in L(T)$ have a minimal cut-arc set not containing y.

A *sink set* in a digraph $D = (V, A)$ is a set $U \subseteq V$ such that there is no arc $(u, v) \in A$ with $u \in U$ and $v \notin U$. A sink set U is *minimal* if $|U| > 1$ and there is no sink set W with $|W| > 1$ and $W \subsetneq U$. A *strongly connected component* of a digraph is a maximal subgraph $D' = (V', A')$ containing, for any $u, v \in V'$, a directed path from u to v and from v to u.

If N is a level-1 network, minimal sink sets in $\Omega_i(\mathcal{T}(N))$ correspond to minimal cut-arc sets in N [13]. To extend this result to level-2 networks, we will use the following theorem, which is a special case of [5, Theorem 7.3]. It uses the *closure digraph* $\mathcal{D}(\mathcal{T})$ of a set $\mathcal{T}$ of trinets, which was introduced in [13] and is defined as follows. Its vertex set is $X = \cup_{T \in \mathcal{T}} L(T)$ and it has an arc (x, y) if, for all $z \in X \setminus \{x, y\}$, there exists a trinet on $\{x, y, z\}$ in $\mathcal{T}$ in which y is a descendant of $LSA(x, z)$.

Theorem 2. *[5] Let N be a binary level-2 network on X and $A \subseteq X$. Then A is minimal cut-arc set of N if and only if A is a minimal sink set of the closure digraph $\mathcal{D}(\mathcal{T}(N))$.*

The next lemma shows that the closure digraph $\mathcal{D}(\mathcal{T})$ is equal to $\Omega_0(\mathcal{T})$ if $\mathcal{T}$ is the set of trinets of some network.

Lemma 1. *If $\mathcal{T} = \mathcal{T}(N)$ for some network N on X, then $\Omega_0(\mathcal{T}) = \mathcal{D}(\mathcal{T})$.*

Proof. First let (x, y) be an arc of $\Omega_i(\mathcal{T})$. Assume that (x, y) is not an arc of $\mathcal{D}(\mathcal{T})$. Then there exists a $z \in X \setminus \{x, y\}$ such that y is not a descendant of $LSA(x, z)$ in the trinet T on $\{x, y, z\}$. We now claim that the arc entering $LSA(x, z)$ is a cut-arc of T. If it is not, then there is some arc (u, v) of T with $v \neq LSA(x, z)$ such that u is not a descendant of $LSA(x, z)$ and v is a descendant of $LSA(x, z)$. This arc (u, v) must lie on a path from the root to at least one of x, y, z. However, it cannot be on a path from the root to x or z because each such path passes through $LSA(x, z)$. Also, it cannot be on a path from the root to y because such a path does not contain any descendants of $LSA(x, z)$. Hence, we can conclude that $\{x, z\}$ is a cut-arc set, which contradicts the assumption that (x, y) is an arc of $\Omega_i(\mathcal{T})$.

Now let (x, y) be an arc of $\mathcal{D}(\mathcal{T})$ and let $z \in X \setminus \{x, y\}$. Then y is a descendant of $LSA(x, z)$ in the trinet on $\{x, y, z\}$ in $\mathcal{T}$. Hence, $\{x, z\}$ is not a cut-arc set. Since a minimal cut-arc set contains at least two leaves, it follows that T has no minimal cut-arc set not containing y. It now follows that (x, y) is an arc of $\Omega_0(\mathcal{T})$. □

Since we consider trinet sets that are not necessarily exactly the trinet set of some network, we cannot always simply use the digraph $\Omega_0(\mathcal{T}) = \mathcal{D}(\mathcal{T})$. In particular, it may happen that $\Omega_0(\mathcal{T})$ has no arcs. We therefore use the strategy described in Algorithm 2, based on [13], which finds a minimal sink set in the digraph $\Omega_i(\mathcal{T})$ for the smallest i for which $\Omega_i(\mathcal{T})$ contains at least one arc.

From Theorem 2 and Lemma 1 follows that Algorithm 2 produces a minimal cut-arc set if the input set is equal to $\mathcal{T}(N)$ for some level-2 network N. Since $\Omega_0(\mathcal{T})$ is not affected by binets or multiple copies of trinets, the same holds when $\mathcal{T}$ is a multiset of binets and trinets with underlying set $\overline{\mathcal{T}}(N)$.

Algorithm 2: Finding a cut-arc set.

Data: Multiset $\mathcal{T}$ of level-2 trinets and (possibly) binets on taxon set X.
Result: Set $A \subseteq X$.
for $i = 0, \ldots, |X| - 2$ **do**
  Construct $\Omega_i(\mathcal{T})$ (see Definition 4);
  if $\Omega_i(\mathcal{T})$ *has at least one arc* **then**
    Let $\mathcal{S}$ be the set of strongly connected components of $\Omega_i(\mathcal{T})$;
    if $\mathcal{S}$ *contains a minimal sink set* **then return** *a smallest such set* ;
    else
      For $S \in \mathcal{S}$, let $v(S)$ be the set of vertices of $\Omega_i(\mathcal{T})$ that are a descendant of a vertex in S;
      return *a smallest such set* $v(S)$ *containing at least two elements*

For a general input multiset of binets and trinets, the output of Algorithm 2 is a minimal cut-arc set of the network that will be constructed (by Algorithm 1).

3.3. Constructing a simple network

Once we have found a minimal cut-arc set A, we need to construct the part of the network below this cut-arc. To do this, we restrict $\mathcal{T}$ to $\mathcal{T}' = \{T|(L(T) \cap A) \mid T \in \mathcal{T}, |L(T) \cap A| \geq 2\}$ and find a simple network for $\mathcal{T}'$.

If the underlying set of $\mathcal{T}$ is $\overline{\mathcal{T}}(N)$ with N a level-2 network and A is a minimal cut-arc set of N, then the underlying set of $\mathcal{T}'$ is $\overline{\mathcal{T}}(N')$ with N' either a tree with two leaves or a simple network.

3.3.1. The number of reticulations

Let $\mathcal{T}''$ be the set containing all trinets from $\mathcal{T}'$ and let p_2 be the fraction of the trinets in $\mathcal{T}''$ that are strictly level-2 and let $n = |L(\mathcal{T}')|$. If $n = 2$, we construct a network equal to a binet with maximum multiplicity in $\mathcal{T}'$. Otherwise, if $p_2 < \frac{n-2}{2\binom{n}{3}}$, we set the number of reticulations k to 1, else we set k to 2.

Suppose $\mathcal{T}'$ has underlying set $\overline{\mathcal{T}}(N')$ with N' either a tree with two leaves or a level-2 network that is simple (note that it may also be level-1). If N' has two leaves then all binets in $\mathcal{T}'$ are equal to N' and the algorithm correctly constructs N'. This holds in particular when N' is a tree with two leaves. Now assume $n \geq 3$. If N' is a simple level-1 network, then $p_2 = 0$, so the algorithm correctly sets the number of reticulations to 1. Finally, suppose N' is a simple strictly level-2 network. Then, $|\mathcal{T}''| = \binom{n}{3}$. Furthermore, observe that in a level-2 generator the number of reticulation sides plus the number of parallel arcs is at most 2. Hence, there are at least $n-2$ crucial trinets because there are $n-2$ choices for the third leaf. Since each crucial trinet is strictly level-2, at least $n-2$ trinets in $\mathcal{T}''$ are strictly level-2. Therefore, we have $p_2 \geq \frac{n-2}{\binom{n}{3}} \geq \frac{n-2}{2\binom{n}{3}}$ and the algorithm correctly sets the number of reticulations to 2.

3.3.2. Leaves on reticulation sides

Let k be the number of reticulations determined in the previous subsection. Let G be a generator that is the underlying generator of the maximum number of strictly level-k trinets in $\mathcal{T}'$. Let $\mathcal{T}_G$ be the set of trinets in $\mathcal{T}'$ that have underlying generator G.

For each $x \in L(\mathcal{T}_G)$ and for each set of symmetric reticulation sides C of G, let $p_{x,C}$ denote the fraction of trinets in $\mathcal{T}_G$ that have leaf x on a side in C. We proceed greedily as follows. Pick x, C maximizing $p_{x,C}$ over all leaves x that have not been assigned to a side yet and over all C containing at least one side that has not been assigned a leaf yet. Assign x to an arbitrary side in C. Repeat until all reticulation sides have been assigned a leaf. Attach each leaf assigned to a reticulation side to this side.

Let $\mathcal{T}_G^r$ be the set of trinets in $\mathcal{T}'$ that have underlying generator G and that have an automorphism such that each reticulation side of G contains its assigned leaf. From now on, we assume that each reticulation side of the generator of each trinet in $\mathcal{T}_G^r$ contains its assigned leaf.

Suppose the underlying set of $\mathcal{T}'$ is $\overline{\mathcal{T}}(N)$ for some simple, strictly level-k, network N. Then all strictly level-k trinets have the same underlying generator as N. Moreover, for each set C of symmetric reticulation sides, $p_{x,C} = 1$ for all leaves x that are on a side in C in N and $p_{x,C} = 0$ otherwise. Hence, the algorithm correctly assigns leaves to sets of symmetric reticulation sides. It can assign leaves to an arbitrary side within this set since level-2 generators have at most one set of symmetric reticulation sides (see Fig. 2), and those are symmetric.

3.3.3. Leaves per set of symmetric arc sides

For each leaf $x \in L(\mathcal{T}_G^r)$ that has not been assigned to a reticulation side, assign x to a set of symmetric arc sides C of G, maximizing the fraction of trinets in $\mathcal{T}_G^r$ that have leaf x on a side in C.

Suppose the underlying set of $\mathcal{T}'$ is $\overline{\mathcal{T}}(N)$ for some simple level-2 network N. Then it can be argued as in the previous subsection that the algorithm assigns each leaf to the set of symmetric arc sides corresponding to its location in N.

3.3.4. Leaves per arc side

Consider a set of symmetric arc sides C and the set of leaves X_C assigned to C. For $x, y \in X_C$, let $\mathcal{T}_{xy}$ denote the set of simple trinets in $\mathcal{T}'$ containing both x and y, and let q_{xy} denote the fraction of trinets in $\mathcal{T}_{xy}$ in which x and y are on the same side of the underlying generator, with $q_{xx} = 1$. We define the following score for $x \neq y$:

$$r_{xy} = 3 \sum_{z \in X_C} \min\{q_{xz}, q_{yz}\} - \sum_{z \in X_C} q_{xz} - \sum_{z \in X_C} q_{yz}.$$

The main idea of this score function is that, assuming the trinets come from some level-2 network, $r_{xy} \geq 0$ if and only if x and y are on the same side.

The algorithm proceeds as follows. Create a partition $\mathcal{P}_C$ of X_C, initially consisting of only singletons. While $|\mathcal{P}_C| > |C|$ or there exist $x \neq y$ with $r_{xy} > 0$, pick a pair $X, Y \in \mathcal{P}_C$ maximizing

$$r_{XY} = \frac{1}{|X||Y|} \sum_{x \in X, y \in Y} r_{xy}. \tag{1}$$

Merge sets X and Y in $\mathcal{P}_C$.

Finally, assign, injectively at random, the parts of $\mathcal{P}_C$ to the sides in C.

Suppose the underlying set of $\mathcal{T}'$ is $\overline{\mathcal{T}}(N)$ for some simple level-2 network N. The only level-2 generators with symmetric arc sides (see Fig. 2) are 1 and 2d with $C = \{L, R\}$ and 2c with $C = \{L_i, R_i\}$, $i \in \{1, 2, 3\}$. If x, y are on the same side then $q_{xy} = 1$ and otherwise we have $q_{xy} = 0$. We can now see that if x, y are on the same side then r_{xy} is equal to the number of leaves on that side (since each of the three sums is equal to the number of leaves on that side) which is at least 2. If, on the other hand, x, y are on different sides, then $r_{xy} \leq -2$ (since the first sum is 0 and the other two sums are at least 1). Hence, the algorithm correctly splits the leaves in X_C into two sets corresponding to the leaves on side L_i and R_i (or L and R). For generators 1 and 2d it does not matter which set is assigned to which side, by symmetry. For generator 2c, this does matter. It is done randomly here and corrected if necessary in the next subsection.

3.3.5. Side alignment

The following is only necessary when the underlying generator G is generator 2c, see Fig. 2, since it contains more than one set of symmetric arc sides. Call its sets of symmetric arc sides $C_1 = \{L_1, R_1\}$, $C_2 = \{L_2, R_2\}$ and $C_3 = \{L_3, R_3\}$. We have to consider swapping sides L_2, R_2 and/or L_3, R_3 (i.e., assign the leaves assigned to L_2 to R_2 and vice versa and/or assign the leaves assigned to L_3 to R_3 and vice versa). From the four possibilities, we choose the one maximizing the following score:

$$u_{L_1,L_2} + u_{L_1,L_3} + u_{L_2,L_3} + u_{R_1,R_2} + u_{R_1,R_3} + u_{R_2,R_3} \tag{2}$$

with

$$u_{S,T} = \sum_{x \in X_S y \in X_T} q_{xy} - |X_S||X_T|, \tag{3}$$

and X_U the set of leaves assigned to side U.

Suppose the underlying set of $\mathcal{T}'$ is $\overline{\mathcal{T}}(N)$ for some simple level-2 network N with underlying generator 2c. Then we have that $q_{xy} = 1$ if $x \in L_i, y \in L_j$ or $x \in R_i, y \in R_j$, and $q_{xy} = 0$ if $x \in L_i, y \in R_j$ or vice versa. Hence, $u_{L_iL_j} = u_{R_iR_j} = 0$ and $u_{L_iR_j}, u_{R_iL_j} < 0$. Therefore, choosing the assignment maximizing (2), out of all possible assignments, chooses the assignment corresponding to N.

3.3.6. Ordering the leaves on the arc sides

Consider an arc side S and the set of leaves X_S assigned to side S. Let $\mathcal{T}^s_{xy}$ denote the set of simple trinets in $\mathcal{T}$ containing both x and y and both on the same side of the underlying generator. Let a_{xy} denote the fraction of trinets in $\mathcal{T}^s_{xy}$ in which the parent of x is an ancestor of y. Let π be an ordered list of leaves, which is initially empty. Find a leaf $x \in X_S \setminus \pi$ maximizing

$$\sum_{y \in X_S \setminus \pi} a_{xy} - a_{yx}. \tag{4}$$

Append leaf x to π and continue until π is a permutation of X_S. The permutation π then describes the ordering of the leaves on side S. Attach the list of leaves π to side S.

Suppose the underlying set of $\mathcal{T}'$ is $\overline{\mathcal{T}}(N)$ for some simple level-2 network N. For two leaves x, y on the same arc side S of N, we have that $a_{xy} = 1$ if the parent of x is an ancestor of y and $a_{xy} = 0$ otherwise. Hence, (4) is equal to the number of leaves that have not been added to the permutation π yet and are below x on side S, minus the number of leaves that have not been added to the permutation π yet and are above x on side S. Therefore, the algorithm constructs the ordering π of leaves on side S in N.

The pseudo code for constructing a simple network is in Algorithm 3.

Algorithm 3: Constructing a simple level-2 network.

Data: Multiset $\mathcal{T}'$ of level-2 trinets and (possibly) binets on taxon set X.
Result: Simple level-2 network N' on X.

```
//Determine the level k
n = |L(T')|;
T'' = the set of trinets contained in T';
p_2 = the fraction of trinets in T'' that are strictly level-2;
if n = 2 then
    return an arbitrary network with maximum multiplicity in T'
if p_2 < (n-2)/(2*binom(n,3)) then
    k = 1
else
    k = 2
//Determine the generator
G = the underlying generator of the maximum number of strictly level-k trinets in T';
N' = G;
//Assign leaves to reticulation sides
T_G = the set of trinets in T' that have underlying generator G;
while there is a reticulation side of G that has not been assigned a leaf do
    Let p_{x,C} be the fraction of trinets in T_G that have leaf x on a side in set C;
    Find x ∈ L(T_G) that has not been assigned to a side and a set of symmetric reticulation sides C that have not all been assigned a leaf, maximizing p_{x,C};
    Assign x to an arbitrary side in C and attach x to this side in N';
T^r_G = the set of trinets in T' that have underlying generator G and that have an automorphism such that each reticulation side of G contains its assigned leaf;
Relabel the sides of the generators of the trinets in T^r_G such that each reticulation side contains its assigned leaf;
//Assign leaves to sets of symmetric arc sides
for each leaf x that has not been assigned to a reticulation side do
    Assign x to a set of symmetric arc sides C maximizing the fraction of trinets in T^r_G that have leaf x on a side in C;
```

//Continued on Page 7

3.4. Theoretical result

The following theorem shows that the algorithm is guaranteed to reconstruct a level-2 network from its set of trinets.

Theorem 3. *If N is a recoverable, binary level-2 network on X with $|X| \geq 3$, then Algorithm 1 will output N when applied to input $\mathcal{T} = \mathcal{T}(N)$.*

```
    //Assign leaves to arc sides
21  for each set of symmetric arc sides C do
22      P_C = partition of X_C containing only singletons;
23      q_xy = the fraction of simple trinets containing x, y in
          which x, y are on the same side of the underlying generator;
24      r_xy = 3 Σ_{z∈X_C} min{q_xz, q_yz} − Σ_{z∈X_C} q_xz − Σ_{z∈X_C} q_yz;
25      r_XY = 1/(|X||Y|) Σ_{x∈X, y∈Y} r_xy;
26      while there exist X, Y ∈ P_C with r_XY > 0, or |P_C| > |C| do
27          Find a pair X, Y ∈ P maximizing r_XY;
28          Merge sets X and Y in P_C; update r_XY;
29      while there is a leaf in X_C that has not been assigned to a side do
30          Pick a set Z ∈ P_C containing a leaf that has not been
              assigned to a side;
31          Pick a side S ∈ C that has not been assigned any leaves;
32          Assign the leaves from Z to side S;
    //Align sides
33  if G is generator 2c from Fig. 2 then
34      Find bijections f : {L_2, R_2} → {L_2, R_2} and
          g : {L_3, R_3} → {L_3, R_3} maximizing u_{L_1,f(L_2)} + u_{L_1,g(L_3)} +
          u_{f(L_2),g(L_3)} + u_{R_1,f(R_2)} + u_{R_1,g(R_3)} + u_{f(R_2),g(R_3)};
35      with u_{S,T} = Σ_{x∈X_S, y∈X_T} q_xy − |X_S||X_T|;
36      Assign the leaves assigned to L_2, R_2, L_3, R_3 to
          f(L_2), f(R_2), g(L_3), g(R_3), respectively;
    //Order leaves on arc sides
37  for each arc side S with set X_S of assigned leaves do
38      T^s_xy = the set of simple trinets in T' containing x and y on
          the same side of the underlying generator;
39      a_xy = the fraction of trinets in T^s_xy in which the parent of x is
          an ancestor of y;
40      π = ();
41      while π is not a permutation of X_S do
42          Find a leaf x ∈ X_S \ π maximizing Σ_{y∈X_S\π} a_xy − a_yx and
              append x to π;
43      Attach the list of leaves π to side S in N';
44  return N'
```

Proof. We use induction on the number of vertices of N. The base case is that N is a tree with 3 leaves and 5 vertices, say $X = \{x, y, z\}$ and $\{x, y\}$ is the minimal cut-arc set. The algorithm will generate $A = \{x, y\}$ (see Section 3.2). The set $\mathcal{T}'$ contains only the tree on $\{x, y\}$ and hence this is constructed as N' (see Section 3.3.1). The set $\mathcal{T}^*$ contains only the tree on $\{a^*, z\}$ and hence N^* is this tree. Combining N' and N^* gives N (Section 3.1).

If N has at least 6 vertices, the algorithm finds a minimal cut-arc set A of N by Section 3.2. If $A \neq X$, let (u, v) be the corresponding cut-arc of N and let N' be the subnetwork of N rooted at v. If $A = X$, let $N' = N$. In either case, the underlying set of $\mathcal{T}'$ is $\overline{\mathcal{T}}(N')$. By Section 3.3, the algorithm constructs N' (which is either a tree with two leaves or a simple network) from $\mathcal{T}'$. If $A = X$ then this completes the proof. Otherwise, let N^* be the network obtained from N by deleting all vertices of N' except for v and labelling v by a^*. The underlying set of $\mathcal{T}^*$ is $\overline{\mathcal{T}}(N^*)$ and N^* contains fewer vertices than N. If N^* has at least three leaves, the algorithm constructs N^* from $\mathcal{T}^*$ by induction. If N^* has two leaves, then $\mathcal{T}^*$ only contains N^* and hence the algorithm constructs N^* (see Section 3.3.1). In both cases, combining N' and N^* gives N (see Section 3.1). □

Algorithm 2 can be implemented efficiently to run in $O(|\mathcal{T}| + |X|^2)$ time (similarly to [13] for level-1). The main idea here is to first compute $\phi(x, y)$, the number of trinets containing x and y that have a minimal cut-arc set not containing y. This can be done in $O(|\mathcal{T}| + |X|^2)$ time since we need to loop through the set of trinets only once and update the values $\phi(x, y)$ affected by this trinet T, i.e., with $x, y \in L(T)$. Finding a minimal cut-arc set in a trinet can be done in constant time as the size of each trinet is bounded by a constant (as any trinet that is not recoverable can be ignored). After that, the digraph Ω_i can be constructed in $O(|X|^2)$ time, and this only needs to be done for the smallest i for which $\phi(x, y) \leq i$ for at least one pair x, y. The condensed digraph can be found with Tarjan's algorithm for computing strongly connected components in $O(|X|^2)$ time. Since the number of generators, and the number of sides of each generator, is bounded by a constant, the bottleneck of Algorithm 3 is Line 26. The values q_{xy} can be computed in $O(|\mathcal{T}| + |X|^2)$ time and the values r_{xy} in $O(|X|^3)$ time. The values r_{XY} can be computed in $O(|X|^2)$ time by looping through all x, y and updating the values of r_{XY} with $x \in X$ and $y \in Y$. This last step has to be repeated $O(|X|)$ times. So Algorithm 3 takes $O(|\mathcal{T}| + |X|^3)$ time. Computing $\mathcal{T}'$ and $\mathcal{T}^*$ can be done in $O(|\mathcal{T}| + |X|)$ time since the size of the trinets is bounded by a constant. All of this has to be repeated $O(|X|)$ times. Hence, the algorithm runs in time $O(|\mathcal{T}||X| + |X|^4)$.

4. Discussion

We have presented an algorithm that, for an input set $\mathcal{T}$ of trinets (and possibly binets) with leaf-set X, outputs a level-2 network on X with run time $O(|\mathcal{T}||X| + |X|^4)$ and that is guaranteed to reconstruct a level-2 network from its set of trinets. Note that a variant of this algorithm is presented in [11]. It should also be noted that our level-2 algorithm cannot be used to decide whether or not an arbitrary set of trinets is contained in some level-2 network or not. Indeed, if a set of level-1 trinets is input into the algorithm, then it will output a level-1 network. But it is known that deciding whether or not an arbitrary set of level-1 trinets is contained in a level-1 network is NP-complete [7].

In addition, our algorithm can be used to build level-1 networks for more general inputs than the level-1 TriLoNet algorithm described in [13], since TriLoNet's input is restricted to collections in which there is a trinet on every 3-subset of the leaf-set. The main innovation in our algorithm lies in Algorithm 3. The high-level idea is to split the process up in different stages: determine the level, determine the generator, assign leaves to reticulation sides, assign leaves to sets of symmetric arc sides, assign leaves to arc sides, align sides and order leaves on arc sides. Most of these steps are not necessary for the level-1 case, or are much simpler.

In terms of potential applications of our level-2 algorithm, in [13] a method is presented to derive collections of level-1 trinets from molecular sequence data; it would be interesting to see if this approach could be extended to derive level-2 trinets as well. We expect that this could be

quite complicated, and so it may be necessary to restrict the level-1/level-2 building blocks to some subset of the list of potential 3-leaved networks.

In another direction, in this paper we have shown that level-3 networks are not necessarily encoded by their trinets. However, Fig. 1 is essentially the only case in which a level-3 network is not encoded [12], and so it would be interesting to investigate if there is a polynomial-time algorithm for constructing level-3 networks from trinets modulo this symmetry. Alternatively, it can be shown that the collection of 4-leaved networks (or quarnets) contained in a level-3 network encode the network [12], and so new algorithms could be potentially developed to build level-3 networks from quarnets. Another interesting open question is whether a level-k network is always encoded by its $(k+1)$-nets. Some partial results are presented in [10].

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Magnus Bordewich, Britta Dorn, Simone Linz, Rolf Niedermeier, Algorithms and Complexity in Phylogenetics, Dagstuhl Reports, vol. 9, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2020.

[2] R.A. Leo Elworth, Huw A. Ogilvie, Jiafan Zhu, Luay Nakhleh, Advances in Computational Methods for Phylogenetic Networks in the Presence of Hybridization, Springer International Publishing, Cham, 2019, pp. 317–360.

[3] Philippe Gambette, Katharina T. Huber, On encodings of phylogenetic networks of bounded level, J. Math. Biol. 65 (1) (2012) 157–180.

[4] Katharina T. Huber, Vincent Moulton, Encoding and constructing 1-nested phylogenetic networks with trinets, Algorithmica 66 (3) (2013) 714–738.

[5] Katharina T. Huber, Vincent Moulton, Taoyang Wu, Hierarchies from lowest stable ancestors in nonbinary phylogenetic networks, J. Classif. 36 (2) (2019) 200–231.

[6] Katharina T. Huber, Leo van Iersel, Steven Kelk, Radoslaw Suchecki, A practical algorithm for reconstructing level-1 phylogenetic networks, IEEE/ACM Trans. Comput. Biol. Bioinform. 8 (3) (2010) 635–649.

[7] Katharina T. Huber, Leo van Iersel, Vincent Moulton, Celine Scornavacca, Taoyang Wu, Reconstructing phylogenetic level-1 networks from nondense binet and trinet sets, Algorithmica 77 (1) (2017) 173–200.

[8] Katharina T. Huber, Leo van Iersel, Vincent Moulton, Taoyang Wu, How much information is needed to infer reticulate evolutionary histories?, Syst. Biol. 64 (1) (2015) 102–111.

[9] Leo van Iersel, Vincent Moulton, Trinets encode tree-child and level-2 phylogenetic networks, J. Math. Biol. 68 (7) (2014) 1707–1729.

[10] Frank Janisse, Encoding level-k phylogenetic networks, MSc thesis, TU Delft, 2021, http://resolver.tudelft.nl/uuid:11939b58-b834-4073-8de8-b61d9a5f9a81.

[11] Sjors Kole, Constructing level-2 phylogenetic networks from trinets, MSc thesis, TU Delft, 2020, http://resolver.tudelft.nl/uuid:c699ea63-f8c8-40f7-8f07-11ac055c42e0.

[12] Leonie Nipius, Rooted binary level-3 phylogenetic networks are encoded by quarnets, BSc thesis, TU Delft, 2020, http://resolver.tudelft.nl/uuid:a9c5a8d4-bc8b-4d15-bdbb-3ed35a9fb75d.

[13] James Oldman, Taoyang Wu, Leo van Iersel, Vincent Moulton, TriLoNet: piecing together small networks to reconstruct reticulate evolutionary histories, Mol. Biol. Evol. 33 (8) (2016) 2151–2162.

[14] Charles Semple, Gerry Toft, Trinets encode orchard phylogenetic networks, J. Math. Biol. 83 (3) (2021) 1–20.

[15] Mike Steel, Phylogeny: Discrete and Random Processes in Evolution, SIAM, 2016.

[16] Leo van Iersel, Judith Keijsper, Steven Kelk, Leen Stougie, Ferry Hagen, Teun Boekhout, Constructing level-2 phylogenetic networks from triplets, IEEE/ACM Trans. Comput. Biol. Bioinform. 6 (4) (2009) 667–681.

[17] Stephen Willson, Regular networks can be uniquely constructed from their trees, IEEE/ACM Trans. Comput. Biol. Bioinform. 8 (3) (2010) 785–796.