An Obstruction Approach to Reconstruct Phylogenies and Level-k Networks from Triplets

Research report - LIRMM ... - December 11, 2008

Philippe Gambette, Vincent Berry, Christophe Paul

Département informatique, L.I.R.M.M., C.N.R.S. - Université Montpellier II.
{gambette,berry,paul}@lirmm.fr

Abstract

Warning: this manuscript is outdated. Results of Section 2 have been improved and appeared in our article at CPM'09, and Section 3 contains an error which was pointed out by Sylvain Guillemot (Appendix F cannot be used to reduce the complexity of the FPT algorithm) who provides simpler obstructions and a valid FPT algorithm, as well as a polynomial kernel for the MCSRT problem in a 2009 article with Matthias Mnich. This manuscript will remain available as long as the results of Section 4 are not published elsewhere.

We propose a new approach for the reconstruction of level-k phylogenetic networks from their set of triplets when all triplets of the network to reconstruct are considered to be in the input set. For level-0 networks, i.e. trees, we give obstructions on four leaves which characterize dense triplet sets incompatible with a tree. We deduce a linear time certifying algorithm for tree reconstruction, and an FPT algorithm for finding the minimum number of triplets to edit to make a dense triplet set compatible with a tree. For level-1 networks, i.e. galled trees, we give a similar reconstruction algorithm to decide whether the triplet set $\mathcal{T}$ is *extremely dense*, that is when $\mathcal{T}$ is the set of all triplets compatible with the reconstructed network. This approach is based on level-k generators. We give rules to build them automatically, and a decomposition theorem of level-k networks based on these generators.

1 Introduction

Context and definitions: Phylogenetic networks have been introduced in phylogenetics to generalize the tree model of evolution which can only represent *speciation* events. In a phylogenetic network, additional branches join vertices already connected by a path, hence defining *reticulations*. This enables to represent hybridization [Gra71, LR04], recombination [Hud83, WZZ01] or lateral gene transfer events [HL01, MCDB05]. Many algorithms have been developed to reconstruct such objects from different kinds of input: sequences, splits, triplets, quartets, rooted or unrooted trees or networks [Hus07, Gam]. The fact that networks are generally hard to handle gave rise to many different restrictions on their structure in order to get tractable algorithms.

A phylogenetic network is said to be *explicit* when all reticulations can be interpreted as precise biological events. In this case, the phylogenetic network is a rooted directed graph, where all vertices have degree at most 3: speciation vertices have indegree 1 and outdegree 2 and reticulation vertices have indegree 2 and outdegree 1. To cover all such explicit phylogenetic networks, the *level-k* hierarchy was introduced in [CJSS05]. An phylogenetic network can indeed be viewed as a blobbed-tree [GB05], that is a network with tree-like parts and non tree-like ones called *blobs*. The *level* of a network reflects the complexity of its blobs: it is defined as the maximum number of reticulations inside a blob of the network.

Galled trees, a restriction of phylogenetic networks which attracted a lot of attention since their introduction in 2001 [WZZ01, GEL03] can be defined as level-1 networks. Many polynomial algorithms

exist on this class of networks, and the introduction of the level-k hierarchy can also be seen as a framework to generalize these algorithms for galled trees to all explicit phylogenetic networks.

Related works: The level-k phylogenetic network reconstruction problem has been studied when the input consists of a *triplet* set, that is a set of rooted binary trees on three leaves. For level-0 networks, i.e. trees, this problem is solved using a graph structure introduced by Aho et al [ASSU81]. This graph structure also yields inference rules for triplet sets compatible with a tree [Bry97].

Recently, *dense* input triplet sets have been considered, meaning that for all sets of three leaves, at least one triplet on those leaves is included in the triplet set. Characterizations of the compatibility of a dense triplet set with a tree were found independently by [Dre97], and [GB07] who also gave a linear certifying tree reconstruction algorithm. Although reconstructing level-k phylogenetic networks from a triplet set $\mathcal{T}$ is NP-hard in general [JNS06, vIKM07], when restricted to dense input sets, the problem was solved for level-1 in $O(|\mathcal{T}|)$ [JNS06], using the new concept of SN-sets thereby extending Aho et al's approach. SN-sets were also used in a generalization for level-2 to provide a $O(|\mathcal{T}|^{\frac{8}{3}})$ algorithm [vIKK$^+$08]. However, the complexity of this problem for higher fixed levels remains unknown.

Another restriction was proposed to help reconstruct the network: considering an *extremely dense* triplet set $\mathcal{T}$, which means it consists of all triplets of the network N to reconstruct. There is an $O(|\mathcal{T}|^{k+1})$ algorithm to determine whether a triplet set (dense, obviously) is extremely dense for some level-k network. Note that the complexity of the problem of finding the smallest k such that a level-k phylogenetic network is compatible with a dense triplet set T is still unknown [vI], even with extreme density.

Although level-k networks has recently attracted a lot of attention, their combinatorial structure has not been studied in details. However, *level-k generators* were introduced to understand the structure of a subclass, *simple* level-k networks [vIKK$^+$08]. A case analysis to build all level-2 generators was also detailed, and later generalized as a brute force algorithm to build all 65 level-3 generators [Kel].

For trees, the problem of Maximum Compatible Subset of Rooted Triples, which aims at editing the minimum number of triplets on n leaves to make it compatible with a tree, is NP-complete [Bry97, Jan01, Wu04], and the current best time complexity to solve it is $O\big((|\mathcal{T}| + n^2)3^n\big)$ [Wu04]. This triplet edition problem is NP-complete, even with the density restriction, on level-k networks for any k [vIKK$^+$08]. The problem of removing the minimum number l of leaves to make a dense triplet set compatible with a tree, known as the SMAST problem for rooted triples, was proved NP-complete [BN06] and FPT in l [GB07].

In the unrooted context, the analogue of triplets are quartets. Our approach can be linked with the results by Bandelt & Dress [BD86] who provided local characterizations of dense quartet sets compatible with an unrooted tree. Using these local conflicts led to a fixed-parameter algorithm in time $O(4^t n + n^4)$ for reconstructing a tree whose quartet set differs from the input dense set by at most t quartets [GN03].

Our results: Here we focus on the problem of level-k phylogenetic network reconstruction from an extremely dense set of triplets, meaning we know the set of all triplets of the network to reconstruct. We aim at reducing the $O(|\mathcal{T}|^{k+1})$ complexity by making better use of their combinatorial structure. We provide a new approach which we show to work for level 0 and 1, and could be generalized to upper levels. This explains why we detail some algorithms obtained with our approach, which have already appeared in the literature with the same complexity.

The global idea of our approach is to express locally the complexity of the global structure. So we first provide a structure decomposition of level-k phylogenetic networks into generators which express the local complexity of the network. We give rules to build these generators and show that they can be used in practice to build all level-4 generators.

We also prove a new characterization of dense triplet sets compatible with a tree, based on obstructions on four leaves. A first use of this characterization provides a certifying algorithm in $O(|\mathcal{T}|) = O(n^3)$ which reconstructs the tree compatible with a dense set $\mathcal{T}$ if possible, or finds a

local obstruction otherwise. Moreover, the obstruction characterization enables us to show that when parameterized by the number t of triplets to edit, the triplet edition problem on n leaves is tractable. We provide an $O(3^t n + n^4)$ bounded search tree algorithm for this practical problem.

To decide whether a triplet set is extremely dense for a level-1 network, we propose an $O(|\mathcal{T}|)$ algorithm (based on the decomposition into generators) which we think can be extended to upper levels with a better complexity than $O(|\mathcal{T}|^{k+1})$.

Outline: The decomposition theorems of level-k generators and level-k phylogenetic networks are given in Section 2. In Section 3, for level-0 networks, we present the triplet obstructions, and the reconstruction, triplet edition, and leaf deletion algorithms. In Section 4, for level-1 networks, we give a recognition algorithm for an extremely dense triplet set. Finally, we give some details on why our results are interesting in a practical point of view.

2 Decomposing Level-k Phylogenetic Networks

A *phylogenetic tree* is a rooted binary tree with directed arcs and distinctly labeled leaves. A *phylogenetic network* is a generalization of a phylogenetic tree, defined as a directed acyclic graph[1] in which exactly one vertex has indegree 0 and outdegree 2 (the root) and all other vertices have either indegree 1 and outdegree 2 (*split vertices*), indegree 2 and outdegree ≤ 1 (*hybrid vertices*) or indegree 1 and outdegree 0 (leaves). The leaves are distinctly labeled. A directed acyclic graph is *biconnected* if it contains no vertex whose removal disconnects the graph. A *biconnected component*, or *blob*, of a pylogenetic network, is a maximal biconnected subgraph. For any arc (u, v) of N, u is a father of v, and v a son of u.

A phylogenetic network is called a *level-k network* [JS06] if each biconnected component contains at most k hybrid vertices. A level-k network which is not a level-$(k-1)$ network is called a *strict level-k phylogenetic network*. A level-0 phylogenetic network is a phylogenetic tree, and a level-1 network is usually called a *galled tree*. Many hard problems can be solved in polynomial time on these classes of networks, which motivates the study of upper levels.

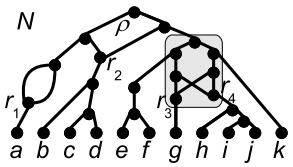
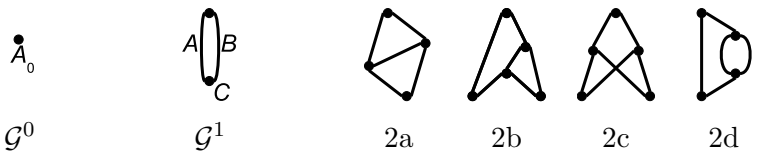


Figure 1: A level-2 network N of root ρ and leaf set $\{a, b, c, d, e, f, g, h, i, j, k\}$. All unlabeled vertices are split vertices. The gray area is a biconnected component with two hybrid vertices r_3 and r_4, and the arc from r_2 to its son is a cut-arc. Triplets $b|cd$, $a|ce$ and $e|ac$ are compatible with N.

A *level-k generator* [vIKK+08] (see Fig. 2) is a biconnected strict level-k network. Vertices of outdegree 0 and arcs of a level-k generator are called its *sides*, they are *empty* if no subtree is hanging from them. We define the following partial order $\succ_N$ on the sides of the level-k generator N: $Y \succ_N X$ if the source of arc Y (or Y itself, if Y is a vertex) can be reached from the target of arc X (or X itself, if X is a vertex).

We define S_k as the set of generators of level at most k. Note that phylogenetic networks have been defined above such that a level-k generator is a level-k phylogenetic network (contrary to [vIKK+08] we allow phylogenetic networks to contain hybrid vertices of outdegree 0). The level-0 (respectively level-1) generator is called $\mathcal{G}^0$ (resp. $\mathcal{G}^1$). In [vIKK+08], the level-2 generators are found by a case analysis which can also be applied to compute the 65 level-3 generators [Kel]. Here we provide rules to compute level-$(k+1)$ generators from level-k generators.

[1] ... in which we allow multiple edges, as is shown by the blob containing r_1 in Figure 1. Note that choosing whether to allow this configuration (an "empty" cycle in the network) in the definition of a phylogenetic network is just technical: here we allow it to be able to define level-k generators as level-k phylogenetic networks.

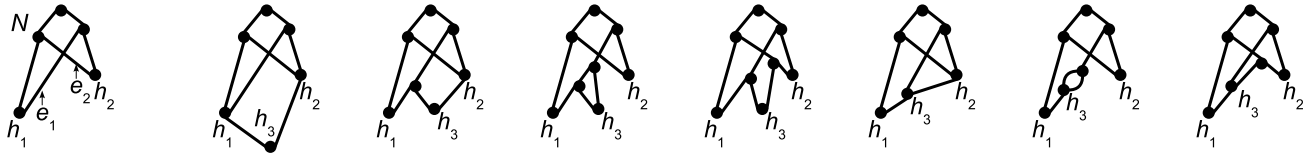


$\mathcal{G}^0$ $\mathcal{G}^1$ 2a 2b 2c 2d

Figure 2: Level-0 generator $\mathcal{G}^0$ (a), level-1 generator $\mathcal{G}^1$ (b) and level-2 generators, called 2a, 2b, 2c, 2d in [vIKK+08]. All arcs are directed downward but orientation is not displayed for the sake of readability.

Definition 1. *N is a level-k generator. The network $R1(N, X, Y)$ is obtained by choosing two sides X and Y of N, such that if $X = Y$ then X is not a hybrid vertex (i.e. it is an arc), and hanging a new hybrid vertex under X and Y (see Fig. 3). The network $R2(N, X, Y)$ is obtained by choosing a side X of N and an arc $Y \not\succ_N X$ of N, and putting an arc from X to Y, which creates a new hybrid vertex "inside" arc Y.*

Note that X and Y have a symmetric role for rule $R1$ but not for rule $R2$. When we build $R1(N, X, Y)$ from N, we say that we apply rule $R1$ on X and Y (and the same for $R2$).



(a) (b) (c) (d) (e) (f) (g) (h)

Figure 3: Results of applying rules $R1$ and $R2$ on a level-2 generator N (a) depending on the type of side (arc or hybrid vertex) where it is applied: $R1(N, h_1, h_2)$ (b), $R1(N, h_1, e_2)$ (c), $R1(N, e_2, e_2)$ (d), $R1(N, e_1, e_2)$ (e), $R2(N, h_1, e_2)$ (f), $R2(N, e_1, e_1)$ (g), $R2(N, e_2, e_1)$ (h).

Property 1. *For any level-k generator N, and any two sides X and Y of N, if $R1(N, X, Y)$ (resp. $R2(N, X, Y)$) exists, then $R1(N, X, Y)$ (resp. $R2(N, X, Y)$) is a level-$(k+1)$ generator.*

Proof: The definitions of $R1$ and $R2$ (especially the fact that $Y \not\succ_N X$ to apply rule $R2(N, X, Y)$) ensure that acyclicity of the graph is preserved. Thus, we just have to show that for any type of side X and Y (as detailed in Fig. 3), applying rule $R1$ or $R2$ always adds exactly one hybrid vertex, with outdegree ≤ 1, and split vertices.

We first check what happens when applying rule $R1$ to get $R1(N, X, Y)$:

- if $N = \mathcal{G}^0$, then applying $R1$ gives the level-1 generator $\mathcal{G}^1$.
- if X and Y are both hybrid vertices (distinct), they have outdegree 0 as they are sides of N, so applying $R1$ will just give them outdegree 1, and create a new hybrid vertex of outdegree 0 (Fig. 3(b)).
- if X is a hybrid vertex, and Y is an arc, then applying $R1$ gives X outdegree 1, adds a new hybrid vertex of outdegree 0 and creates an new split vertex "inside" Y (whose father is the upper extremity of Y, and whose sons are the lower extremity of Y and the new hybrid vertices created), as shown in Fig. 3(c). By symmetry we also get a valid generator if X is an arc and Y is a hybrid vertex.
- if X and Y are both arcs (possibly the same as in Fig. 3(d)) then applying $R1$ creates two split vertices, one inside X and the other inside Y (Fig. 3(e)).

In all cases $R1(N, X, Y)$ corresponds to N augmented by a hybrid vertex and possibly some split vertices, so if N is a level-k generator then $R1(N, X, Y)$ is a level-$(k+1)$ generator.

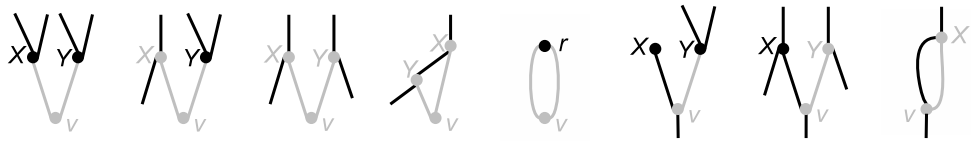
We now check what happens when applying rule $R2$ to get $R2(N, X, Y)$:

- if X is a hybrid vertex, and Y is an arc, then applying $R2$ gives X outdegree 1, and creates an new hybrid vertex of outdegree 1 "inside" Y (whose fathers are X and the upper extremity of Y, and whose son is the lower extremity of Y), as shown in Fig. 3(f).
- if X and Y are both arcs (possibly the same, see Fig. 3(g)) then applying $R1$ creates a split vertex inside X and a hybrid vertex of outdegree 0 inside Y (Fig. 3(h)).

In all cases $R2(N, X, Y)$ corresponds to N augmented by a hybrid vertex and possibly some split vertices, so if N is a level-k generator then $R2(N, X, Y)$ is a level-$(k+1)$ generator. □

Property 2. *For any level-$(k+1)$ generator N, there exists a level-k generator N', and some sides X and Y of N' such that $N = R1(N', X, Y)$ or $N = R2(N', X, Y)$.*

Proof: The proof works by induction, and consists in "reversing the rules" to remove a hybrid vertex.



(a) (b) (c) (d) (e) (f) (g) (h)

Figure 4: Different possible cases to "reverse" rules $R1$ and $R2$, depending on whether the rule has created an outdegree 0 hybrid vertex (a-e) which corresponds to rule $R1$ or an outdegree 1 hybrid vertex (f-h) which corresponds to rule $R2$: gray arcs and vertices are to be deleted to reverse the rule.

First we define operations to reverse rules $R1$ and $R2$. We make a case analysis to define the network N' obtained by *removing* a hybrid vertex v (which is not one of the two sons of the root) from a network N:

- if v has outdegree 0, then five cases can happen:
  - (a) both fathers of v are distinct hybrid vertices X and Y, then if we delete v, vertices X and Y get outdegree 0, and no other vertex is changed, as shown in Fig. 4(a). If we call N' the network obtained after deletion then we note $N = R1(N', X, Y)$,
  - (b) a father of v, say Y, is a hybrid vertex and the other, X, is a split vertex, then, as shown in Fig. 4(b), by deleting v, X, and joining the father of X to the second son of X (other than v) thanks to arc e_X, we get a network N' such that $N = R1(N', Y, e_X)$,
  - (c) both fathers of v are split vertices X and Y such that X is neither a son nor the father of Y, then, as shown in Fig. 4(c), by deleting v, X, Y, and joining the father of X to the second son of X (other than v) thanks to arc e_X, and the father of Y to the second son of Y (other than v) thanks to arc e_Y, we get a network N' such that $N = R1(N', e_X, e_Y)$.
  - (d) both fathers of v are split vertices X and Y where X is the father of Y then, as shown in Fig. 4(d), by deleting v, X, Y, and joining the father of X to the second son of Y (other than v) thanks to arc e_{XY}, we get a network N' such that $N = R1(N', e_{XY}, e_{XY})$.
  - (e) v is the only son of the root, then N has to be $\mathcal{G}^1$, as shown in Fig. 4(e), we remove v and its two incoming arcs to get the level-0 generator $N' = \mathcal{G}^0$ with one vertex A_0 and $N = R1(N', A_0, A_0)$.
- if v has outdegree 1, then three cases can happen:
  - (f) at least one father of v, say Y, is a hybrid vertex. Then by deleting v, and joining X to the son of v thanks to arc e_X, vertex Y gets outdegree 0 and the degree of no other vertex is changed, as shown in Fig. 4(f), we get a network N' such that $N = R2(N', Y, e_X)$,

(g) both fathers of v are different split vertices X and Y, then, as shown in Fig. 4(g), by deleting v, Y, and joining the father of Y to the second son of Y (other than v) thanks to arc e_Y, and joining X to the son of v thanks to arc e_X, we get a network N' such that $N = R2(N', e_Y, e_X)$.

(h) v only has one father which is the split vertex X, then, as shown in Fig. 4(h), by deleting v, X, and joining the father of X to the son of v thanks to arc e_X, we get a network N' such that $N = R2(N', e_X, e_X)$.

Note that using rules $R1$ and $R2$ here is an abuse of notation as we have no guarantee, when we write $N = Ri(N', X, Y)$, that N' is a generator, and X and Y are sides.

We now prove by induction on k that for any level-$(k+1)$ generator, there exists a hybrid vertex v (which is either the only son of the root, or not a son of the root) such that removing v gives a level-k generator.

Base case:

Call A_0 the only vertex of $\mathcal{G}^0$, A and B the two arcs of $\mathcal{G}^1$, and C its hybrid vertex. Then we can check the base cases for $k \leq 1$ (see Fig. 2) as $\mathcal{G}^1 = R1(\mathcal{G}^0, A_0, A_0)$ (vertex C is removed, we are in case e), and for $k = 1$ we remove hybrid vertices which are not sons of the root: $2a = R1(\mathcal{G}^1, B, C)$ (case b), $2b = R1(\mathcal{G}^1, B, B)$ (case d), $2c = R1(\mathcal{G}^1, A, B)$ (case c) and $2d = R2(\mathcal{G}^1, B, B)$ (case h).

Inductive step:

We now fix $k \geq 2$. We suppose that the expected property is true for any level-j generator, with $j < k$ and prove it for level k. So consider a level-$k+1$ generator N. It contains at least three hybrid vertices, so at least one of the three, say v, is not an out-neighbor of the root. So we can remove it, that is we get a level-k network N' such that $N = Ri(N', X, Y)$, where $i \in \{1, 2\}$, and X and Y are arcs or vertices of N'.

If removing v gives a biconnected network N', then N' is a level-k network as the changes made when removing v deleted exactly one hybrid vertex, so N' is a level-k generator and the property is true.

Otherwise, N' is not biconnected. Let N'' be a biconnected component of N' which does not contain the root. Removing v from N did not create any vertex of indegree 1 and outdegree 0 in N', i.e. there is no leaf in N', therefore N'' contains at least two vertices (so at least one hybrid vertex).

Also note that in N', there are at most two cut-arcs incident to vertices of N'', otherwise N could not be biconnected. One of those cut-arcs leads to the root of N'', as N'' does not contain the root, so there is at most one cut-arc hanging from N'' in N', i.e. at most one vertex of indegree 1 and outdegree 1 in N''. We call G'' the network obtained by deleting this vertex, if it exists, and joining its father to its son. G'' is a level-j generator, with $j < k$.

We can apply the induction hypothesis on G'': it contains a hybrid vertex v'' that can be removed. Let X'' and Y'' be the fathers of v'' in G'' (we name X'' and Y'' similarly to X and Y in the case analysis to remove v). We now have to carefully check what reverse rule was used to remove v'' from G'' and show that v'' can also be removed from N.

Indeed, in N, the arc which is a cut-arc hanging from N'' in N' could be hanging below X'' and Y'' and over v''. Another possibility is that the arc leading to v hangs here (if v was removed from N according to case b for example), or that even v itself is placed here (if v was removed according to case g for example).

We first consider case a. If v'' also has outdegree 0 in N then:

- if both fathers of v'' in N are hybrid vertices, then remove v'' from N according to case a.
- if exactly one of the fathers of v'' in N is not a hybrid vertex, then remove v'' from N according to case b.
- if no father of v'' in N is a hybrid vertex, then remove v'' from N according to case c.

Otherwise, v'' has outdegree 1 in N then remove v'' according to case f. Note that it is impossible that neither X'' nor Y'' is the father of v'' in N: as there is already an arc below v'' in N, there can

only be one other arc (leading to v, or cut-arc in N') hanging from N'' (thus creating a split vertex below one of v'''s father and over v'') in N.

We now consider case b. If v'' also has outdegree 0 in N then the same applies as in case a. Otherwise, v'' has outdegree 1 in N:

- if one of the fathers of v'' in N is a hybrid vertex, then remove v'' in N according to case f.
- otherwise remove v'' from N according to case g.

We now consider case c. If v'' also has outdegree 0 in N then:

- if exactly one of the fathers of v'' in N is a hybrid vertex, then remove v'' from N according to case b.
- otherwise remove v'' from N according to case c.

Otherwise, v'' has outdegree 1 in N, the same applies as in case b.

We now consider case d. If v'' also has outdegree 0 in N then, if X'' and Y'' are still the fathers of v'' in N then remove v'' from N according to case d, otherwise:

- if exactly one of the fathers of v'' in N is a hybrid vertex, then remove v'' from N according to case b.
- otherwise remove v'' from N according to case c.

Otherwise, v'' has outdegree 1 in N, the same applies as in case b.

We now consider case e. If v'' also has outdegree 0 in N then, the case where v'' still has only one father in N cannot happen (otherwise N would not be biconnected), so:

- if exactly one of the fathers of v'' in N is a hybrid vertex, then remove v'' from N according to case b.
- otherwise remove v'' from N according to case c.

Otherwise, v'' has outdegree 1 in N. If v'' still has only one father in N then remove v'' according to case h, otherwise the same applies as in case b.

We now consider cases f and g:

- if one of the fathers of v'' in N is a hybrid vertex, then remove v'' in N according to case f.
- otherwise remove v'' from N according to case g.

We finally consider case h: if v'' still has only one father in N then remove v'' according to case h, otherwise the same applies as in case f.

We can also check in all these cases that removing n'' from N maintained the biconnectivity ensured when removing n'' from G.

In any case, we have found a hybrid vertex which can be removed to get a level-k generator, therefore the property is true. □

Property 3. *The number g_k of level-k generators is bounded by $k!^2 50^k$.*

Proof: Call a_k the maximum number of arcs and n_k the maximum number of hybrid vertices of a level-k generator. By noticing that each application of rule $R1$ or $R2$ adds at most four arcs and one hybrid vertex (bounds reached for example in Fig. 3(e)), we get that $n_k \leq n_{k-1} + 1$ and $a_k \leq a_{k-1} + 4$, so $n_k \leq k$ and $a_k \leq 4k$. When applying the k^{th} rule $R1$ or $R2$, we choose a pair of vertices or arcs, so there are at most $(a_k + n_k)^2$ possibilities. Thus $g_{k+1} \leq 2(a_k + n_k)^2 g_k \leq 50k^2 g_k$, so finally $g_k \leq k!^2 50^k$. □

Note that this bound is far from tight, as different sequences of rules may produce isomorphic level-k generators. To use the rules practically, after computing level-$(k+1)$ generators from the set of level-k generators, isomorphic ones should be removed using a digraph isomorphism algorithm [McK81]. However, the bound for g_{k+1} from g_k and the fact that $g_3 = 65$ [Kel] shows that it is possible to generate automatically level-4 and 5 generators at least.

Indeed, we used them to build all level-4 generators: from the 65 level-3 generators, 8501 level-4 generators were generated using rules $R1$ and $R2$. Isomorphic generators were remained throughout the process, giving a total of 1993 non-isomorphic level-4 generators. The list of these generators, the program to build them, its source, as well as implementation notes, are available at `http://www.lirmm.fr/~gambette/ProgGenerators.php`. Note that the sequence 1,4,65,1993 is not present in the On-Line Encyclopedia of Integer Sequences [Slo08].

When generators were introduced in [vIKK+08], they were just intended to generate some restrictions of level-k phylogenetic networks, called *simple*, which contain no cut-arc except the *trivial* ones leading to leaves. We give an explicit decomposition theorem which shows how they can be used to generate any level-k network, and exhibits the link with the blobbed-tree structure of phylogenetic networks, as they are constructed here by just connecting blobs in a tree-like manner (using just one cut-arc for the connection).

Definition 2. *Given a set S_k of generators of level at most k, and a phylogenetic network N, we define the following rules:*

- *$SplitRoot_k(G_1, G_2)$ is obtained by hanging G_1 and $G_2 \in S_k$ below a root.*
- *$Attach_k(v, G, N)$ is the network obtained by adding an arc from hybrid vertex $v \in N$ of outdegree 0 to a copy of a generator $G \in S_k$.*
- *$Attach_k(e, G, N)$ is the network obtained by subdividing arc e (i.e. adding a vertex of indegree 1 and outdegree 1 inside e) and adding an arc from the created vertex to a copy of $G \in S_k$.*

Theorem 1. *N is a level-k network iff there exists a sequence of $r \in \mathbb{N}$ locations (arcs or hybrid vertices) $(\ell_j)_{j\in[1,r]}$ and a sequence of generators $(G_j)_{j\in[0,r]}$ in S_k, such that:*

$$N = Attach_k(\ell_r, G_r, Attach_k(\dots Attach_k(\ell_2, G_2, Attach_k(\ell_1, G_1, G_0))\dots)),$$
$$\text{or } N = Attach_k(\ell_r, G_r, Attach_k(\dots Attach_k(\ell_2, G_2, SplitRoot_k(G_1, G_0))\dots)).$$

Proof: $\Leftarrow$: This implication is trivial, as any of the above rules, for any level-i generator G_j with $i \leq k$ creates a cut-arc to a new biconnected component with k hybrid vertices or less, so it gives a level-k network.

$\Rightarrow$: We prove by induction on p, that for any k, a level-k phylogenetic network N with p vertices can be obtained by repeated applications of rule *Attach* after one possible application of rule *SplitRule*.

We fix k.

Base case: if $p = 1$ then the only possible network is $\mathcal{G}^0$, which corresponds to not applying (take $r = 0$ in the definition) the rule *Attach* to the level-0 generator $\mathcal{G}^0$.

Inductive step: now suppose that all networks with strictly less than p vertices verify the desired property, let N be a network with p vertices.

If N contains a leaf l, then:

- either it has at least one grand-father u, then:
  - either its father is a split vertex, then delete l, its father, and connect its grand-father u to its sibling v. The network N' obtained has less than p vertices, so the induction hypothesis applies, and
    $$N = Attach_k((u, v), \mathcal{G}^0, N'),$$
    which gives the desired property.

– either its father is a hybrid vertex h, then delete l, and the arc from its father. The network N' obtained has less than p vertices, so the induction hypothesis applies, and $N = Attach_k(h, \mathcal{G}^0, N')$, which gives the desired property.

- or it has no grand-father, that is its father is the root. Then the network N' obtained by considering the sibling u of l and the subnetwork rooted at u has less than p vertices, so we can apply the induction hypothesis and get:

  – either

  $$N' = Attach_k(\ell_r, G_r, Attach_k(\dots \\ Attach_k(\ell_2, G_2, Attach_k(\ell_1, G_1, G_0))\dots)),$$

  then

  $$N = Attach_k(\ell_r, G_r, Attach_k(\dots \\ Attach_k(\ell_2, G_2, Attach_k(\ell_1, G_1, SplitRoot_k(\mathcal{G}^0, G_0)))\dots))$$

  – or

  $$N' = Attach_k(\ell_r, G_r, Attach_k(\dots \\ Attach_k(\ell_2, G_2, SplitRoot_k(G_1, G_0))\dots)).$$

  then

  $$N = Attach_k(\ell_r, G_r, Attach_k(\dots \\ Attach_k(\ell_2, G_2, Attach_k(\ell', G_1, SplitRoot(G_0, \mathcal{G}^0)))\dots)),$$

  where ℓ' is the arc from the root to G_0 in $SplitRoot(G_0, \mathcal{G}^0)$.

Otherwise, N only contains a root, vertices of indegree 2, and vertices of indegree 1 outdegree 2.

Either N is biconnected, then it is a generator, and it has k hybrid vertices or less (as N is level-k), so the expected property is true.

Or N is not biconnected, and N has a hybrid vertex of outdegree 0. Consider its biconnected component tree. Consider a leaf of this tree, that is one of the "lowest" biconnected components. Let C be this biconnected component, i.e. C is a level-k generator. We treat it exactly like leaf l in the beginning, by replacing $\mathcal{G}^0$ by C in the decomposition formulas. □

This decomposition is not unique, as two different sequences of rules may lead to the same phylogenetic network (typically, by just changing the order of the applied rules). However, the theorem expresses the decomposition of phylogenetic networks into generators and can be rephrased to ensure a one-to-one correspondance between phylogenetic networks and their decomposition, a tree with vertices labeled by generators, and edges labeled as well to store where they should get attached on the slides of the generators. From such a decomposition, and the set of level-k generators, counting or exhaustive generation of level-k phylogenetic networks is possible, which would extend currently known results on the number of unicyclic networks and galled trees [SS04].

3 Triplet Obstructions for Trees

A *triplet* $x|yz$ is a phylogenetic tree on three leaves x, y and z where x, and the father of y and z are the sons of the root. A triplet $x|yz$ is compatible with a phylogenetic network N (or N is compatible with $x|yz$) if N contains two vertices u and v and pairwise internally vertex-disjoint paths from u to y, u to z, v to u and v to x. A triplet set is compatible with a phylogenetic network if all its triplets are. Computing the set $\mathcal{T}$ of all triplets compatible with a phylogenetic network can be done in time $O(|\mathcal{T}|) = O(n^3)$ with dynamic programming [BGHK08]. We recall that a triplet set is dense if there exists a triplet on any set of three leaves, and note $\mathcal{T}[L]$ the set of all triplets in $\mathcal{T}$ whose leaves are all in L.

3.1 Characterization of Trees and Obstructions

The keypoint of our reconstruction approach for trees is to know where to place a new leaf x when we consider that a triplet $a|bc$ has already been placed.

Definition 3. *Consider a triplet $a|bc$, a leaf x, a dense triplet set $\mathcal{T}$. We define the five following zones A, B, C, D, E (see Fig. 5) depending on $\mathcal{T}_x = \mathcal{T}[a, b, c, x]$:*

- *zone A:* $L_A = \{x \mid \mathcal{T}_x = \{a|bc, b|ax, c|ax, x|bc\}\}$, *leaf* $\alpha_A = a$,
- *zone B:* $L_B = \{x \mid \mathcal{T}_x = \{a|bc, a|bx, a|cx, c|bx\}\}$, *leaf* $\alpha_B = b$,
- *zone C:* $L_C = \{x \mid \mathcal{T}_x = \{a|bc, a|bx, a|cx, b|cx\}\}$, *leaf* $\alpha_C = c$,
- *zone D:* $L_D = \{x \mid \mathcal{T}_x = \{x|ab, x|ac, x|bc, a|bc\}\}$, *leaf* $\alpha_D = b$,
- *zone E:* $L_E = \{x \mid \mathcal{T}_x = \{a|bc, a|bx, a|cx, x|bc\}\}$, *leaf* $\alpha_E = b$,

and $\mathcal{T}_X = T[L_X \cup \{\alpha_X\}]$ for $X \in \{A, B, C, D, E\}$.

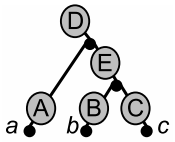


Figure 5: Zones defined by the different possible cases for triplet sets on 4 leaves.

Theorem 2. *For a dense triplet set $\mathcal{T}$, the following properties are equivalent:*

(i) $\mathcal{T}$ *is compatible with a tree.*

(ii) $\mathcal{T}$ *contains exactly one triplet on each set of three leaves, and for any triplet set on four leaves $\{a, b, c, d\}$, $ab|d, bc|d \in \mathcal{T} \Rightarrow ac|d$, and $ab|c, bc|d \in \mathcal{T} \Rightarrow ac|d$ [Dre97],*

(iii) $\mathcal{T}$ *contains exactly one triplet on each set of three leaves, and for any triplet set on four leaves $\{a, b, c, d\}$, $ab|c, bc|d \in \mathcal{T} \Rightarrow ab|d, ac|d \in \mathcal{T}$ [GB07],*

(iv) $\mathcal{T}$ *contains exactly one triplet on each set of three leaves, and any triplet set on four leaves is isomorphic either to $\{x_1|x_2x_3,\ x_1|x_2x_4,\ x_1|x_3x_4,\ x_2|x_3x_4\}$ (case 1), or to $\{x_1|x_2x_3,\ x_2|x_1x_4,\ x_3|x_1x_4,\ x_4|x_2x_3\}$ (case 2),*

(v) $\mathcal{T}$ *contains no set of triplets isomorphic to any of the four following obstructions:* ① $\{a|bc, c|ab\}$, ② $\{a|bc, c|bd, d|ab\}$, ③ $\{a|bc, c|bd, d|ac\}$, ④ $\{a|bc, a|bd, d|ac\}$.

Proof: $i \Leftrightarrow ii$ and $i \Leftrightarrow iii$ were proved independently respectively in[Dre97] and [GB07]. Any of these properties could be used to prove that $i \Leftrightarrow iv$. However, we give a new proof as some algorithms we provide are deduced directly from the structure of this proof.

$i \Rightarrow iv$: we check that there are only two possible tree shapes for rooted binary trees on 4 leaves: the caterpillar tree corresponds to case 1 and the balanced tree to case 2; none allows 2 different triplets on the same 3 leaves.

$iv \Rightarrow i$: we proceed by induction on the number of leaves. The result is direct on 4 leaves. Consider a triplet set $\mathcal{T}$ with $n > 4$ leaves. We start with a triplet $a|bc$. For each other leaf x, we only have case 1 or 2 for non-isomorphic sets of triplets on 4 leaves, with the bijective labeling $f : \{x_1, x_2, x_3, x_4\} \to \{a, b, c, x\}$, we get 5 possibilities depending on the value of $f^{-1}(x)$, which correspond exactly to the 5 zones introduced in Definition 3. Thus, to each leaf x we can affect a zone X. The sets $\mathcal{T}_X$ are dense, and their size is strictly lower than the original triplet set, so we can apply the induction hypothesis and know that each of them is compatible with a tree which can be included

in the tree structure of Fig. 5 to obtain T. We also need to check that all triplets which are not in $\mathcal{T}_A \cup \mathcal{T}_B \cup \mathcal{T}_C \cup \mathcal{T}_D \cup \mathcal{T}_E$ are compatible with T.

Leaves in different zones:
In the following of the proof, when we use capital letter X we mean any leaf in L_X. Triplets with leaves in different zones can be of different kinds:

- on leaves $\{x, y, Z\}$, where $\{x, y\} \in \{a, b, c\}$, $Z \in \{A, B, C, D, E\}$. By definition of the areas those triplets are ok.
- on leaves $\{x, Y, Z\}$, where:
  - $x = a$ and:
    - $Y = A$ and:
      - $Z = B$: we know that we have triplets $b|aA$ and $a|bB$, so we have to be in case 2, so we also have triplets $B|aA$, which is ok, and $A|bB$.
      - $Z = C$: symmetric to the previous case ($C \leftrightarrow B, c \leftrightarrow b$).
      - $Z = D$: we have triplets $b|aA$ and $D|ab$, so we have to be in case 1, and we also have triplet $D|aA$, which is ok, and $D|Ab$.
      - $Z = E$: we have triplets $b|aA$ and $a|bE$, so we have to be in case 2, and we also have $E|aA$, which is ok, and $A|bE$.
    - $Y = B$ and:
      - $Z = C$: we have $a|bB$ and $a|bC$, so we are in case 1, and we also have $a|BC$, which is ok.
      - $Z = D$: we have $a|bB$ and $D|ab$, so we are in case 1, and we also have $D|aB$, which is ok, and $D|bB$.
      - $Z = E$: we have $a|bB$ and $a|bE$, so we are in case 1, and we also have $a|BE$, which is ok.
    - $Y = C$: symmetric to the case $Y = B$ ($C \leftrightarrow B, c \leftrightarrow b$).
    - $Y = D$ and $Z = E$: we have $D|ab$ and $a|bE$, so we are in case 1, and we also have $D|aE$, which is ok, and $D|bE$.
  - $x = b$ and:
    - $Y = A$ and:
      - $Z = B$: we have already shown that we have $A|bB$, which is ok.
      - $Z = C$: we know that we have $b|aA$ and $a|bC$, so we are in case 2, so we also have $A|bC$, which is ok.
      - $Z = D$: we have already shown that we have $D|Ab$, which is ok.
      - $Z = E$: we have already shown that we have $A|bE$, which is ok.
    - $Y = B$ and:
      - $Z = C$: we know that we have triplets $b|cC$ and $c|bB$, so we are in case 2, so we also have $C|bB$, which is ok, and $B|cC$.
      - $Z = D$: we have already shown that we have $D|bB$, which is ok.
      - $Z = E$: we have $c|bB$ and $E|bc$, so we are in case 1, so we also have $E|bB$, which is ok, and $E|Bc$.
    - $Y = C$ and:
      - $Z = D$: we have $a|bC$ and $D|ab$, so we are in case 2, so we also have $D|bC$, which is ok.
      - $Z = E$: we have $a|bC$ and $a|bE$, so we are in case 1, so we also have $a|EC$, which is ok.

        * $Y = D$ and $Z = E$: we have already shown that we have $D|bE$, which is ok.
    - $x = c$: symmetric to the previous case where $x = b$ ($c \leftrightarrow b, C \leftrightarrow B$).

- on leaves $\{X, Y, Z\}$, where:
    - $X = A$ and:
        * $Y = B$ and:
            · $Z = C$: we have shown that we have triplets $a|BC$, $B|aA$ and $C|aA$, so we are in case 2 and also have triplet $A|BC$, which is ok.
            · $Z = D$: we have shown that we have $A|Bc$, $D|Ac$ and $D|Bc$, so we are in case 1 and also have $D|AB$, which is ok.
            · $Z = E$: we have shown that we have $B|aA$, $E|aA$ and $a|BE$, so we are in case 2 and also have $A|BE$, which is ok.
        * $Y = C$: symmetric to the previous case where $Y = B$ ($c \leftrightarrow b, C \leftrightarrow B$).
        * $Y = D$ and $Z = E$: we have shown that we have $D|aA$, $D|aE$ and $E|aA$, so we are in case 1 and also have $D|AE$, which is ok.
    - $X = B$ and:
        * $Y = C$ and:
            · $Z = D$: we have shown that we have $D|Bb$, $D|Cb$ and $B|Cb$, so we are in case 1 and also have $D|BC$, which is ok.
            · $Z = E$: symmetric to the previous case where $Z = D$ ($E \leftrightarrow D$).
        * $Y = D$ and $Z = E$: we have shown that we have $D|aA$, $D|aE$ and $E|aA$, so we are in case 1 and also have $D|AE$, which is ok.
    - $X = C$: symmetric to the previous case where $X = B$ ($c \leftrightarrow b, C \leftrightarrow B$).

Two leaves in the same zone:
We consider the triplets on two leaves in zone X and one in a different zone Y. An important remark is that the sets $\mathcal{T}_X$ have been chosen such that triplets not in $\mathcal{T}_X$ give no other information on the position of leaves inside X than $\mathcal{T}_X$ already provides.

Two cases occur:

- either zone Y is *lower than* zone X (i.e. there exists a directed path from the root to zone X through zone Y). Then leaves inside Y may influence the location of some leaves inside X. We show below that in this case this information is compatible for all leaves inside, Y, and compatible with the information given by the leaf α_X.

- otherwise (if X is lower than Y or none can be reached by a path from the root through the other), we show below that triplets are compatible with the tree reconstructed.

Case where Y lower than X: let us prove that for any two leaves x_1 and x_2 in zone X and y_1 in lower zone Y_1, the triplet on $\{x_1, x_2, y_1\}$ is compatible with the triplet on $\{x_1, x_2, \alpha_X\}$ (i.e. $x_1|x_2y_1 \in \mathcal{T} \Rightarrow x_1|x_2\alpha_X \in \mathcal{T}$, $x_2|x_1y_1 \in \mathcal{T} \Rightarrow x_2|x_1\alpha_X \in \mathcal{T}$, and $y_1|x_1x_2 \in \mathcal{T} \Rightarrow \alpha_X|x_1x_2 \in \mathcal{T}$).

We start with zone E, recall that $\alpha_E = b$. Consider two leaves $e_1, e_2 \in L_E$. In the first part of the proof (*leaves in different zones*), we have proved that we always have triplets $e_1|bx$ and $e_2|bx$, for $x \in B \cup C \cup \{c\}$ (B and C are the two zones lower than E). The triplet on leaves $\{b, e_1, e_2\}$ can be either:

- $b|e_1e_2$: then we have to be in case 1 for the triplet set on leaves $\{b, e_1, e_2, x\}$, so we have triplet $x|e_1e_2$,

- or $e_1|be_2$: then we have to be in case 2 for the triplet set on leaves $\{b, e_1, e_2, x\}$, and we have triplet $e_1|xe_2$,

- or $e_2|be_1$, which is symmetric to the previous case and forces the presence of triplet $e_2|xe_1$.

In all cases the triplet on leaves $\{b, e_1, e_2\}$ is compatible with the one on leaves $\{x, e_1, e_2\}$.

The proof is exactly the same when we consider two leaves $d_1, d_2 \in L_D$ and a leaf $x \in A \cup B \cup C \cup E \cup \{a, c\}$, which concludes the proof.

Case where Y not lower than X:

- If $X = A$, the leaves are called a_1, a_2: if $Y = B$, then we have proved in the first part of the proof that we have $B|aa_1$ and $B|aa_2$ so we are in case 1 and we have $B|a_1a_2$. The same works for $Y \in \{C, D, E, b, c\}$.
- If $X = B$, the leaves are called b_1, b_2: if $Y = A$, then we have proved in the first part of the proof that we have $A|bb_1$ and $A|bb_2$ so we are in case 1 and we have $A|b_1b_2$. The same works for $Y \in \{C, D, E, b, c\}$.
- If $X = C$, the case is symmetric to $X = B$.
- If $X = D$, no zone is lower than D.
- If $X = E$, the leaves are called e_1, e_2. If $Y = A$, we have proved that $A|be_1$ and $A|be_2$ are present, so we are in case 1 and we have $A|e_1e_2$.

Three leaves in the same zone:
By using the induction hypothesis, we know that the triplet is compatible with T.

Finally, we have shown that all triplets of $\mathcal{T}$ are compatible with T.

iii $\Rightarrow$ *iv*: any triplet set on four leaves is of a certain type (case 1 or 2) which does not contain any of the obstructions ② to ④, and obstruction ① is explicitly forbidden by forcing exactly one triplet on each set of three leaves.

iv $\Rightarrow$ *iii*: we consider any triplet set on four leaves. Because of obstruction ① it does not contain two different triplets on the same leaves. So we enumerate all possible sets of 4 triplets on 4 leaves (where two triplets differ by one leaf), that is, up to isomorphism:

- $\mathcal{T}_1$: $\{x_1|x_2x_3, x_1|x_2x_4, x_1|x_3x_4, x_2|x_3x_4\}$
- $\mathcal{T}_2$: $\{x_1|x_2x_3, x_2|x_1x_4, x_3|x_1x_4, x_4|x_2x_3\}$
- $\{x_1|x_2x_3, x_2|x_3x_4, x_3|x_1x_4, x_4|x_1x_2\}$: impossible as we can find obstruction ③ in this triplet set $(x_1 \to d, x_2 \to a, x_3 \to c, x_4 \to b)$
- $\{x_1|x_2x_3, x_1|x_3x_4, x_4|x_1x_2, x_4|x_2x_3\}$: impossible as we can find obstruction ④ in this triplet set $(x_1 \to a, x_2 \to c, x_3 \to b, x_4 \to d)$
- $\{x_1|x_2x_3, x_1|x_2x_4, x_3|x_1x_4, x_2|x_3x_4\}$: impossible as we can find obstruction ④ in this triplet set $(x_1 \to a, x_2 \to b, x_3 \to d, x_4 \to c)$
- $\{x_1|x_2x_3, x_1|x_2x_4, x_3|x_1x_4, x_4|x_2x_3\}$: impossible as we can find obstruction ④ in this triplet set $(x_1 \to a, x_2 \to b, x_3 \to d, x_4 \to c)$

Finally all triplet sets of size 4 are of type $\mathcal{T}_1$ or $\mathcal{T}_2$ which are exactly case 1 and case 2 □

3.2 A Certifying Algorithm for Tree Reconstruction

The proof of Theorem 2 provides the framework for a certifying tree reconstruction algorithm from triplets. Although such an algorithm with the same complexity was already given in [GB07], we detail this one which is extended to level 1 in Section 4.

Theorem 3. *For a dense set of triplets $\mathcal{T}$, there is an algorithm in time $O(|\mathcal{T}|)$ which either reconstructs the tree T compatible with $\mathcal{T}$, or returns a set of at most three triplets which proves that $\mathcal{T}$ is compatible with no tree.*

Proof: We call this algorithm BuildTree and give its pseudo-code in Algorithm 1. Its correctness directly follows from Theorem 2. We now detail the analysis of the $O(|\mathcal{T}|) = O(n^3)$ time complexity.

A call to *recursiveBuildTree* takes $O(n)$ time. We now prove that this function is executed $O(n)$ times during the algorithm. Let T_i be the tree computed by the i^{th} call of this function. We label all its arcs by i (see Fig. 6 where i is represented by a color), so that in the end of the algorithm, all arcs of the reconstructed tree are labeled by a set of integers. Note that all trees T_i are either included in another, or arc-disjoint, but they never overlap on an arc. Thus, this family of trees can be considered as a family of non-overlapping sets of arcs, which has a size linear in the number of those arcs, which is linear in the number of the leaves of the reconstructed tree, i.e. $O(n)$. So all the calls to *recursiveBuildTree* have been labeled by $O(n)$ different trees, so there has been $O(n)$ recursive calls to this function.

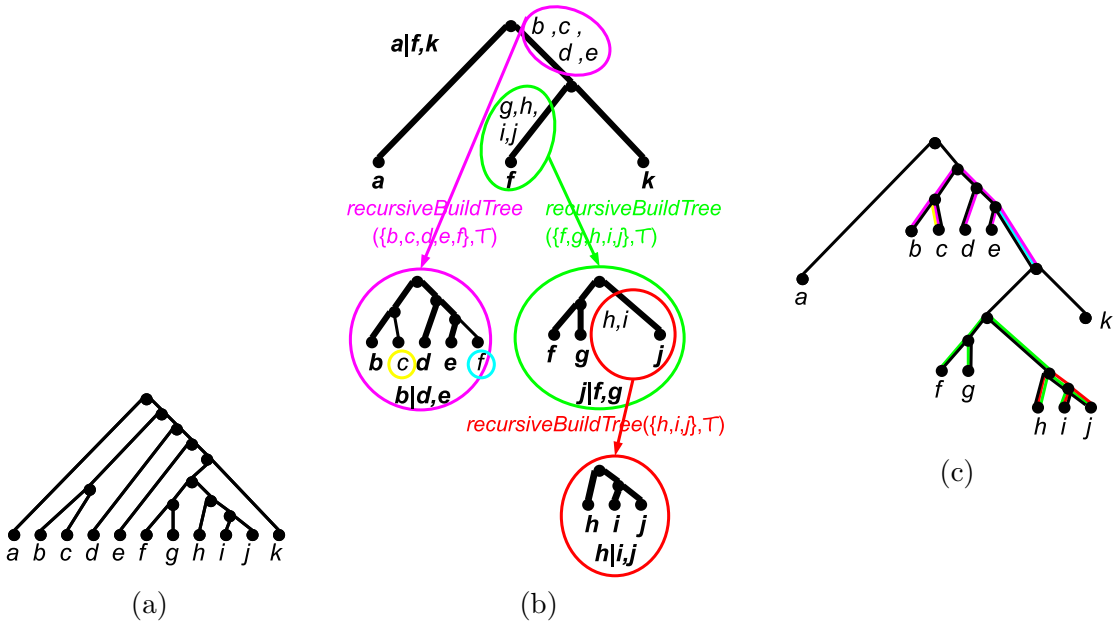


Figure 6: Algorithm 1 applied on the set of triplets compatible with some tree (a): we can see the recursive calls to *recursiveBuildTree* (b) and how the recursively computed subtrees are attached together (c). If each subtree corresponding to one call of *recursiveBuildTree* gets its arcs colored with the same color, then the sets of arcs of the same color in the reconstructed tree is a set of non arc-overlapping subtrees (they are either disjoint or included in another), which proves that function *recursiveBuildTree* is called only a linear number of times during the execution of the algorithm.

Then, triplet consistency in *BuildTree* can be checked in time $O(n^3)$, either by precomputing the lowest common ancestor in the obtained tree for all pairs of leaves [GT83], and checking in constant time for each triplet if it is compatible with the reconstructed tree, or with the consistency checking algorithm in [BGHK08] which extends to networks.

An obstruction can be found in $O(n^4)$ with the naive algorithm (testing all triplet sets on four leaves for case 1 or 2 of Th. 2). Now we prove that knowing a triplet on leaves $\{x, y, z\}$ incompatible with the tree reconstructed by Algorithm 1, we can find an obstruction in constant time. If a different triplet on $\{x, y, z\}$ is present then we simply have obstruction ①.

For unmarked triplets, the key idea is to store for each pair of leaves $\{x, y\}$ its *separating set*, that is the three leaves that have been used by the algorithm to put them into different zones. More formally, if you consider the execution tree of *recursiveBuildTree*, if $recursiveBuildTree(L, \mathcal{T})$ calls $recursiveBuildTree(L_1, \mathcal{T})$ and $recursiveBuildTree(L_2, \mathcal{T})$, where $\{x, y\} \subset L$ and $x \notin L_2$ and $y \notin L_1$,

and the three leaves a, b and c were considered to define the zones, then $\{a, b, c\}$ is called the *separating set* of $\{x, y\}$. If $recursiveBuildTree(L, \mathcal{T})$ does not make any recursive call to $recursiveBuildTree$, then x and y have to be among the leaves considered to define the zones at this step, we call z the third leaf, and the separating set of $\{x, y\} \subset L$ is defined as $\{x, y, z\}$. For a set of three leaves $\{x, y, z\}$, we define its separating set as the separating set of the pair $\{x, y\}$, $\{x, z\}$ or $\{y, z\}$ which was separated first in the algorithm.

We claim that if a triplet on leaves $\{x, y, z\}$ of separating set $\{a, b, c\}$ is incompatible with the reconstructed tree, then there is an obstruction among the triplets in $\mathcal{T}[\{a, b, c\} \cup \{x, y, z\}]$. Indeed, in the proof of Theorem 2, we have shown that the marked triplets (the ones considered in the algorithm), and the possibility of only the two cases of Theorem 2 for triplet sets on four leaves, force all unmarked triplets. Thus, if a tree can be reconstructed from the input set of triplets on leaves $\{a, b, c\} \cup \{x, y, z\}$, it has to be the same than the one the algorithm reconstructed for this set of leaves, as it is compatible with the same triplet set. It is not the case, which implies that no tree can be reconstructed from the input set of triplets on leaves $\{a, b, c\} \cup \{x, y, z\}$, which has thus to contain an obstruction by Theorem 2.

Finally, to find an obstruction in constant time, after having stored the separating sets for each pair during the algorithm, we just have to compute the separating set $\{a, b, c\}$ for the leaves $\{x, y, z\}$ of the incompatible triplet, and then find an obstruction on four leaves among these $l \leq 6$ leaves (at most $\binom{6}{4} = 15$ possible such subsets of four leaves). □

Note that if it is previously known that the input triplet set is compatible with a tree, then executing *recursiveBuildTree* only reconstructs the tree, and the total time complexity is $O(n^2)$, which is however not as fast as the $O(n \log n)$ algorithm in [PT86]. It would be interesting to know whether this can be reduced to $O(n)$, as it is shown that any set of triplets on n leaves defining a tree T has a subset of cardinality $n - 2$ which also defines T [Ste92].

3.3 Two FPT Algorithms for Triplet Set Edition

Obstructions also directly provide a fixed-parameter algorithm called MaxTripletSubset, for the Maximum Compatible Subset of Rooted Triples [Bry97] in the dense case.

Maximum Compatible Subset of Rooted Triplets (MCSRT)
Instance: A set $\mathcal{T}$ of triplets, and $t \in [0, |\mathcal{T}|]$.
Question: Is there a subset $\mathcal{T}'$ of $\mathcal{T}$ such that $\mathcal{T}'$ is compatible with a tree and $|\mathcal{T}'| \geq |\mathcal{T}| - t$?

This problem can be seen as a minimization problem where we want to edit at most t triplets to get a set compatible with a tree. Note that contrary to the algorithm for the similar problem for quartets [GN03], we allow more than one triplet to be present for the same set of three leaves.

Theorem 4. *For a dense triplet set, MCSRT can be solved in $O(3^t n + n^4)$ time.*

Proof: We proceed with a bounded search tree: all triplets are initialized as unmarked, and the set S_O of all obstructions is found in time $O(n^4)$.

For each vertex of the search tree, of depth at most t:

- we consider the first obstruction in S_O, then there are 3 possible triplets to change (the unmarked ones, as it is useless to consider already changed triplets)
- each of them can be changed to two other possibilities, so 6 branches are created in the search tree. In fact, for each triplet in an obstruction, we prove in Lemma 1 that one of the two possibilities leads to a new obstruction on the same leaves, so considering this change is useless. Thus, only 3 branches need to be added to the search tree at the current vertex.
- for each branch created, update the set S_O of obstructions in time $O(n)$ by considering, for the edited triplet $a|bc$, whether an obstruction has been created or removed on leaves $\{a, b, c, x\}$ for each leaf x.

```
BuildTree(L: set of leaves, 𝒯: dense triplet set)
begin
    T ← recursiveBuildTree(L, 𝒯);
    if an unmarked triplet t = x|yz ∈ 𝒯 is not compatible with T then
        find an obstruction;
    else return T;
end
recursiveBuildTree(L: set of leaves, 𝒯: dense triplet set)
begin
    if |L| ≤ 2 then return the tree with a root connected to leaves in L;
    else
        choose a triplet t = a|bc in 𝒯[L], mark it;
        L_A ← {a}; L_B ← {b}; L_C ← {c}; L_D ← {b}; L_E ← {b}; ;
        for each leaf x in L do
            Mark all triplets in 𝒯[a, b, c, x];
            if x ∉ {a, b, c} then
                𝒯_x ← {triplets on leaves a, b, c and x};
                if 𝒯_x = {a|bc, b|ax, c|ax, x|bc} then  L_A ← L_A ∪ {x};
                else if 𝒯_x = {a|bc, a|bx, a|cx, c|bx} then  L_B ← L_B ∪ {x};
                else if 𝒯_x = {a|bc, a|bx, a|cx, b|cx} then  L_C ← L_C ∪ {x};
                else if 𝒯_x = {x|ab, x|ac, x|bc, a|bc} then  L_D ← L_D ∪ {x};
                else if 𝒯_x = {a|bc, a|bx, a|cx, x|bc} then  L_E ← L_E ∪ {x};
                else an obstruction has been found, so the triplet set is not compatible with a
                    tree;
        foreach X ∈ {A, B, C, D, E} do
            T_X ← recursiveBuildTree(L_X,𝒯);
        in T_D, replace leaf b by a vertex with arcs to T_A and T_E;
        in the resulting tree, replace leaf b by a vertex with arcs to T_B and T_C;
        return the resulting tree;
end
```

Algorithm 1: Certifying algorithm for tree reconstruction from a dense set of triplets $\mathcal{T}$ on leaves in L.

Finally, if there exists a vertex of depth at most t such that $S_O = \emptyset$ then there exists a tree compatible with at least $|\mathcal{T}| - t$ triplets from $\mathcal{T}$. Otherwise, the instance admits no solution. The search tree has $O(3^t)$ vertices, hence the total time complexity is $O(3^t n + n^4)$. □

Lemma 1 (Carnon Lemma)**.** *For any of the obstructions in Theorem 2, for each triplet in the obstruction, one of the two possibilities to edit the triplet leads to a new obstruction on the same leaves.*

Proof:

- Obstruction ① $\{a|bc, c|ab\}$:
  - if $a|bc$ is changed to $b|ac$ then we still have obstruction ①.
  - if $c|ab$ is changed to $b|ac$ then we still have obstruction ①.
- Obstruction ② $\{a|bc, c|bd, d|ab\}$:
  - if $a|bc$ is changed to $b|ac$ then we get obstruction ③.
  - the two other triplets give isomorphic cases, and each time one change also leads to obstruction ③.

- Obstruction ③ $\{a|bc, c|bd, d|ac\}$:
  - if $a|bc$ is changed to $c|ab$ then we get obstruction ④.
  - if $c|bd$ is changed to $b|cd$ then we get obstruction ②.
  - if $d|ac$ is changed to $c|ad$ then we get obstruction ④.
- Obstruction ④ $\{a|bc, a|bd, d|ac\}$:
  - if $a|bc$ is changed to $c|ab$ then we get obstruction ③.
  - if $a|bd$ is changed to $b|ad$ then we get obstruction ③.
  - if $d|ac$ is changed to $c|ad$ then we get obstruction ④.

□

This FPT algorithm is easy to implement[2], we believe some optimizations are possible to improve both their practical and theoretical time complexity, by looking for a polynomial kernel or trying iterative compression for instance.

Note that another exact approach exists for editing a triplet set to make it compatible with a tree: removing the minimum number l of leaves such that the triplet set induced on the remaining leaves becomes compatible, known as the SMAST problem for rooted triples. This is also an NP-complete problem, [BN06] and FPT in l [GB07].

4 Extreme Density for Level-1 Networks

We say that a set $\mathcal{T}$ of triplets is *extremely dense for a level-k phylogenetic network* if there exists a level-k phylogenetic network N such that $\mathcal{T}$ is the set of all triplets compatible with $\mathcal{T}$, and no network of lower level is compatible with $\mathcal{T}$. Note that this definition forces N to be a strict level-k network, and that this concept, although introduced as extreme density in [vIKK+08], was later denoted by: N *reflects* the triplet set $\mathcal{T}$ [KvI08].

Lemma 2. *If $\mathcal{T}$ is an extremely dense set of triplets for a level-1 network N, then there exist three leaves a, b, c such that $\{a|bc, b|ac\} \subset \mathcal{T}$, and $c|ab \notin \mathcal{T}$.*

Proof: As $\mathcal{T}$ is compatible with a strict level-1 network, then it contains a level-1 generator as a consequence to Theorem 1. Consider any set of three leaves, and their position with respect to this generator. An enumeration based on hanging leaves on a level-1 generator shows that no configuration is compatible with the 3 triplets $a|bc$, $b|ac$ and $c|ab$. It remains to prove that $a|bc$ and $b|ac$ (up to a renaming of the leaves) have to be present.

We consider subtrees hanging from the level-1 generator $\mathcal{G}^1$ (see Fig. 2). If side C is empty, then the cycle can be destroyed with no effect on the set of compatible triplets.

Otherwise, say leaf c is hanging in the subnetwork below side C. When B is empty, if only one subtree, N_{A1}, is hanging from side A, or if side A is also empty, then the cycle can be destroyed with no effect on the set of compatible triplets (cut the cycle below the vertex where N_{A1} is hanging). If two subnetworks, or more, are hanging on side A, say N_{A1} and N_{A2}, then let a_1 be a leaf of N_{A1}, a_2 be a leaf of N_{A2}, this configuration implies the presence of two different triplets on leaves $\{a_1, a_2, c\}$.

When A is empty, then by symmetry we get the same results. Otherwise, both sides A and B are non-empty. Let a be a leaf hanging in the subnetwork hanging from A and b from B. Both triplets $a|bc$ and $b|ac$ have to be present.

Finally, in all cases, either we find exactly two different triplets on the same three leaves, or we can remove a hybrid vertex per biconnected component. In this case $\mathcal{T}$ would become compatible with a level-0 network, which would contradict the fact that $\mathcal{T}$ is extremely dense for level-1. □

[2] A proof-of-concept program and its source are available at http://www.lirmm.fr/~gambette/ReTriplets.php.

Lemma 2 can be used as a starting point for an algorithm to decide, in a similar way as Algorithm 1, whether a dense set of triplets is extremely dense for some level-1 network.

Indeed, if we generate all possible configurations obtained by hanging three leaves a, b, c on a level-1 generator, such that the network obtained is compatible with triplet $a|bc$, we obtain 35 networks, among which only three are also compatible with triplet $b|ac$. They are shown on Fig. 7 and called $\mathcal{G}_1^1, \mathcal{G}_2^1, \mathcal{G}_3^1$. In each configuration there are eight zones that we formally define below (similarly to

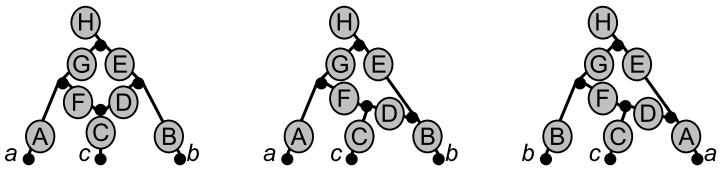


Figure 7: The eight possible zones for the three possible level-1 configurations.

Definition 3 for level-0 networks):

Definition 4. *Consider four leaves a, b, c, x, such that $\{a|bc, b|ac\} \subset \mathcal{T}$ and $c|ab \notin \mathcal{T}$, and an extremely dense triplet set $\mathcal{T}$. We define the following zones (see Fig. 7) depending on $\mathcal{T}_x = \mathcal{T}[a, b, c, x]$, which are mutually exclusive:*

- *zone A:* $L_A = \{x \mid \mathcal{T}_x = \{a|bc, b|ac, b|ax, b|cx, c|ax, x|bc\}\}$, $\alpha_A = a$,
- *zone B:* $L_B = \{x \mid \mathcal{T}_x = \{a|bc, a|bx, a|cx, b|ac, c|bx, x|ac\}\}$, $\alpha_B = b$,
- *zone C:* $L_C = \{x \mid \mathcal{T}_x = \{a|bc, a|bx, a|cx, b|ac, b|ax, b|cx\}\}$, $\alpha_C = c$,
- *zone H:* $L_H = \{x \mid \mathcal{T}_x = \{a|bc, b|ac, x|ab, x|ac, x|bc\}\}$, $\alpha_H = c$,
- *zone D1:* $L_{D1} = \{x \mid \mathcal{T}_x = \{a|bc, b|ac, a|bx, a|cx, x|ac, b|cx, c|bx\}\}$, $\alpha_{D1} = c$,
- *zone E1:* $L_{E1} = \{x \mid \mathcal{T}_x = \{a|bc, b|ac, a|bx, a|cx, x|ac, c|bx, x|bc\}\}$, $\alpha_{E1} = c$,
- *zone F1:* $L_{F1} = \{x \mid \mathcal{T}_x = \{a|bc, b|ac, b|ax, a|cx, c|ax, b|cx, x|bc\}\}$, $\alpha_{F1} = c$,
- *zone G1:* $L_{G1} = \{x \mid \mathcal{T}_x = \{a|bc, b|ac, b|ax, c|ax, x|ac, b|cx, x|bc\}\}$, $\alpha_{G1} = c$,
- *zone D2:* $L_{D2} = \{x \mid \mathcal{T}_x = \{a|bc, b|ac, a|bx, b|ax, a|cx, b|cx, c|bx\}\}$, $\alpha_{D2} = b$,
- *zone E2:* $L_{E2} = \{x \mid \mathcal{T}_x = \{a|bc, b|ac, a|bx, x|ab, x|ac, c|bx, x|bc\}\}$, $\alpha_{E2} = b$,
- *zone F2:* $L_{F2} = \{x \mid \mathcal{T}_x = \{a|bc, b|ac, a|bx, b|ax, a|cx, b|cx, x|bc\}\}$, $\alpha_{F2} = b$,
- *zone G2:* $L_{G2} = \{x \mid \mathcal{T}_x = \{a|bc, b|ac, b|ax, x|ab, x|ac, b|cx, x|bc\}\}$, $\alpha_{G2} = b$,
- *zone D3:* $L_{D3} = \{x \mid \mathcal{T}_x = \{a|bc, b|ac, a|bx, b|ax, a|cx, c|ax, b|cx\}\}$, $\alpha_{D3} = a$,
- *zone E3:* $L_{E3} = \{x \mid \mathcal{T}_x = \{a|bc, b|ac, b|ax, c|ax, x|ab, x|ac, x|bc\}\}$, $\alpha_{E3} = a$,
- *zone F3:* $L_{F3} = \{x \mid \mathcal{T}_x = \{a|bc, b|ac, a|bx, b|ax, a|cx, x|ac, b|cx\}\}$, $\alpha_{F3} = a$,
- *zone G3:* $L_{G3} = \{x \mid \mathcal{T}_x = \{a|bc, b|ac, a|bx, x|ab, a|cx, x|ac, x|bc\}\}$, $\alpha_{G3} = a$.

From this definition we can design an algorithm similar to Algorithm 1 to reconstruct a level-1 phylogenetic network, and even all possible level-1 phylogenetic networks compatible with an extremely dense triplet set. The main difference is that we do not have a short obstruction characterization for level-1 on such triplet sets yet, which explains why we have to check the consistency of the triplets with the structure in the end.

As for level-0 the recursive algorithm focuses on a set L of leaves. If L contains strictly less than three leaves then answer YES. Otherwise start by searching for two different triplets $a|bc$ and $b|ac$ on a set of three leaves. If no such triplets are found, then apply Algorithm 1: if it answers YES and the triplet set is dense, then answer YES, otherwise answer NO. When $a|bc$ and $b|ac$ are found, if triplet $c|ab$ is also present then answer NO.

Otherwise compute the zones for all other leaves $x \in L$ according to definition 4. If this fails for some leaf, then answer NO. Now we try to determine if one of the configurations $\mathcal{G}_1^1$, $\mathcal{G}_2^1$ or $\mathcal{G}_3^1$ is forced. If there exist $i \neq j \in [1,3]$ such that $L_{Xi} \neq \emptyset$ and $L_{Yj} \neq \emptyset$, then $\mathcal{T}$ is incompatible with a level-1 phylogenetic network, so answer NO. Otherwise, two cases can happen.

1) Either all zones Xi, for $i \in [1,3]$ and $X \in \{D, E, F, G\}$, are empty, then all three configurations are possible. Choose one (or store all three when aiming at building all level-1 networks compatible with $\mathcal{T}$).

2) Or one of the configurations among $\mathcal{G}_1^1, \mathcal{G}_2^1$ and $\mathcal{G}_3^1$ is forced.

Recursively apply the algorithm on leaf sets $L_X \cup \alpha_X$ for each zone X. Then connect the obtained subnetworks inside the configuration chosen. If the subnetwork for zone $F1$ contains a cycle between its root and leaf C, then connecting it inside $\mathcal{G}_1^1$ creates a level-2 generator with the greater $D1$-$E1$-$F1$-$G1$ cycle. Thus, we have to check that the network finally obtained is a level-1 network, with a biconnected component detection algorithm. The algorithm ends by checking that the set of all triplets encoded in the resulting network is equal to $\mathcal{T}$. This is a difference with Algorithm 1 as we do not have a short obstruction characterization for level 1 on triplet sets yet, which explains why we have to check the consistency of the triplets with the structure in the end.

This algorithm has complexity $O(|\mathcal{T}|) = O(n^3)$, because before the last check mentioned above, which is also in $O(|\mathcal{T}|)$, all triplets are considered once. The last step of the algorithm ensures the correction if a network has been reconstructed. If no network can be built, then the algorithm answers NO:

- either because it fails to place a leaf in one of the zones, in which case no level-1 network is compatible with $\mathcal{T}$ as ensured by our case study of all possible ways to attach leaves on a cycle.
- or because there exist $i \neq j \in [1,3]$ such that $x \in L_{Xi}$ and $y \in L_{Yj}$, then Xi forces configuration $\mathcal{G}_i^1$. $T[a,b,c,x]$ corresponds to zone Yj, so it corresponds to no other zone (as they are mutually exclusive) so leaf y cannot be placed in configuration $\mathcal{G}_i^1$. Hence no level-1 network is compatible with $\mathcal{T}$.
- the other cases of answer NO are obviously correct.

Note that this algorithm cannot be extended directly to *dense* level-1 triplet sets, as without extreme density, for instance, both positions $x \in H$ and $x \in D2$ are compatible with the presence of $\{x|ab, x|ac, x|bc\}$ in the input triplet set.

5 Practical Interest of our Results for Phylogenetics

The results presented here have diverse applications for biologists. Although sequences are usually the main input in phylogenetics, it was recently shown [DR06] that under a coalescent model, an inferred tree on strictly more than three taxa, is most likely to be wrong because of discrepancies in gene and species tree. This gives strong motivation to develop methods which take triplets as input as the ones we propose here.

The obstruction approach we provide is very useful when the input triplet set is not compatible with a tree. In the case of a dense input triplet set, contrary to the classical algorithm by Aho et al. [ASSU81], Algorithm 1 can output an obstruction, therefore giving feedback to the biologist about the conflicts in the data.

Algorithm MaxTripletSubset goes further in this direction, as it manages to make a triplet set compatible with a tree, by changing a minimum number of triplets. This algorithm should be used

when the triplets are supposed to globally fit into a tree, that is no recombination, hybridization, or lateral gene transfer is supposed to have taken place. Otherwise, it is better to apply the FPT algorithm for SMAST on triplets [GB07] to get rid of a minimum number of leaves involved in reticulation events.

Our results on level-1 networks (galled trees), give insight on their uniqueness according to a set of triplets. We have shown that even when all the triplets of the network to reconstruct are known, it is impossible to determine which subnetwork originated from a reticulation event if at most three subnetworks are hanging from a reticulation cycle. Otherwise, it becomes possible to direct the reticulation cycles properly (e.g. choose among Fig. 7 graphs) and determine the taxa originating from the hybrid vertices, which means that a unique level-1 network is compatible with precisely the input triplet set. Such a certificate of uniqueness gives more confidence in the obtained result to the biologist.

The zone characterizations for level-1 networks also allows to know the different possible positions for a leaf for which there are not enough input triplets to determine a single position.

6 Acknowledgments

We thank the French ANR project ANR-O6-BLAN-0148-01 for support. We thank Steven Kelk, Leo van Iersel, Katharina Huber and Matthias Mnich as well as anonymous referees, for their comments on earlier versions of this report, and David Bryant and Mike Steel for useful references. We thank Michael Rao for his help on the isomorphism test for level-k generators, and Thu-Hien To and Michel Habib for useful discussions.

References

[ASSU81] Alfred V. Aho, Yehoshua Sagiv, Thomas G. Szymanski, and Jeffrey D. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM Journal on Computing*, 10(3):405–421, 1981. 2, 19

[BD86] Hans-Jürgen Bandelt and Andreas Dress. Reconstructing the shape of a tree from observed dissimilarity data. *Advances in Applied Mathematics*, 7:309–343, 1986. 2

[BGHK08] Jaroslaw Byrka, Pawel Gawrychowski, Katharina T. Huber, and Steven Kelk. Worst-case optimal approximation algorithms for maximizing triplet consistency within phylogenetic networks, 2008. http://arxiv.org/abs/0710.3258. 9, 14

[BN06] Vincent Berry and François Nicolas. Improved parameterized complexity of the maximum agreement subtree and maximum compatible tree problems. *IEEE/ACM Transactions in Computational Biology and Bioinformatics*, 3(3):289–302, 2006. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.61.5408. 2, 17

[Bry97] David Bryant. *Building Trees, Hunting for Trees, and Comparing Trees: theory and method in phylogenetic analysis*. PhD thesis, Dept. Mathematics, University of Canterbury, New Zealand, 1997. http://citeseer.ist.psu.edu/685475.html. 2, 15

[CJSS05] Charles Choy, Jesper Jansson, Kunihiko Sadakane, and Wing-Kin Sung. Computing the maximum agreement of phylogenetic networks. *Theoretical Computer Science*, 335(1):93–107, 2005. http://www.df.lth.se/~jj/Publications/masn8_TCS2005.pdf. 1

[DR06] James H. Degnan and Noah A. Rosenberg. Discordance of species trees with their most likely gene trees. *PLoS Genetics*, 2(5):e68, 2006. http://dx.doi.org/10.1371/journal.pgen.0020068. 19

[Dre97] Andreas W. M. Dress. Towards a theory of holistic clustering. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 37:271–289, 1997. 2, 10

[Gam] Philippe Gambette. Who is who in phylogenetic networks: articles, authors and programs. http://www.lirmm.fr/~gambette/PhylogeneticNetworks. 1

[GB05] Dan Gusfield and Vikas Bansal. A fundamental decomposition theory for phylogenetic networks and incompatible characters. In *Proceedings of the ninth Annual International Conference on Research in Computational Molecular Biology (RECOMB'05)*, volume 3500 of *Lecture Notes in Computer*

Science, pages 217–232. Springer Verlag, 2005. http://wwwcsif.cs.ucdavis.edu/~gusfield/gusfieldrecomb.pdf. 1

[GB07] Sylvain Guillemot and Vincent Berry. Fixed-parameter tractability of the maximum agreement supertree problem. In *Proceedings of the* 18^{th} *Annual Symposium on Combinatorial Pattern Matching (CPM'07)*, volume 4580 of *Lecture Notes in Computer Science*, pages 274–285. Springer Verlag, 2007. http://hal-lirmm.ccsd.cnrs.fr/lirmm-00130406. 2, 10, 13, 17, 20

[GEL03] Dan Gusfield, Satish Eddhu, and Charles Langley. Efficient reconstruction of phylogenetic networks with constrained recombination. In *Proceedings of the 2003 IEEE Computational Systems Bioinformatics Conference (CSB2003)*, pages 363–374, 2003. http://wwwcsif.cs.ucdavis.edu/~gusfield/ieeefinal.pdf. 1

[GN03] Jens Gramm and Rolf Niedermeier. A fixed-parameter algorithm for minimum quartet inconsistency. *Journal of Computer and System Sciences*, 67(4):723–741, 2003. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.10.7652. 2, 15

[Gra71] Verne Grant. *Plant Speciation*, pages 300–320, 383–386. Columbia University Press, 1971. 1

[GT83] Harold N. Gabow and Robert E. Tarjan. A linear-time algorithm for a special case of disjoint set union. In *Proceedings of the 15th ACM Symposium on Theory of Computing (STOC'83)*, pages 246–251, 1983. http://dx.doi.org/10.1145/800061.808753. 14

[HL01] Mike Hallett and Jens Lagergren. Efficient algorithms for lateral gene transfers problems. In *Proceedings of the fifth Annual International Conference on Research in Computational Molecular Biology (RECOMB'01)*, pages 141–148, 2001. 1

[Hud83] Richard R. Hudson. Properties of the neutral allele model with intragenic recombination. *Theoretical Population Biology*, 23:183–201, 1983. http://dx.doi.org/10.1016/0040-5809(83)90013-8. 1

[Hus07] Daniel H. Huson. *Split networks and Reticulate Networks*, pages 247–276. Oxford University Press, 2007. similar to http://www-ab.informatik.uni-tuebingen.de/research/phylonets/GCB2006.pdf. 1

[Jan01] Jesper Jansson. On the complexity of inferring rooted evolutionary trees. In *Proceedings of the Brazilian Symposium on Graphs, Algorithms, and Combinatorics (GRACO 2001)*, volume 7 of *Electronic Notes in Discrete Mathematics*, pages 50–53, 2001. http://www.df.lth.se/~jj/Publications/023_GRACO2001.pdf. 2

[JNS06] Jesper Jansson, Nguyen Bao Nguyen, and Wing-Kin Sung. Algorithms for combining rooted triplets into a galled phylogenetic network. *SIAM Journal on Computing*, 35(5):1098–1121, 2006. http://www.df.lth.se/~jj/Publications/triplets_to_gn7_SICOMP2006.pdf. 2

[JS06] Jesper Jansson and Wing-Kin Sung. Inferring a level-1 phylogenetic network from a dense set of rooted triplets. *Theoretical Computer Science*, 363(1):60–68, 2006. http://www.df.lth.se/~jj/Publications/ipnrt8_TCS2006.pdf. 3

[Kel] Steven Kelk. http://homepages.cwi.nl/~kelk/lev3gen/. 2, 3, 8

[KvI08] Steven Kelk and Leo van Iersel. Constructing the simplest possible phylogenetic network from triplets. In *Proceedings of the* 19^{th} *International Symposium on Algorithms and Computation (ISAAC'08)*, volume 5369 of *Lecture Notes in Computer Science*, pages 472–483. Springer Verlag, 2008. http://arxiv.org/abs/0805.1859. 17

[LR04] C. Randal Linder and Loren H. Rieseberg. Reconstructing patterns of reticulate evolution in plants. *American Journal of Botany*, 91(10):1700–1708, 2004. http://www.amjbot.org/cgi/reprint/91/10/1700.pdf. 1

[MCDB05] Dave MacLeod, Robert L. Charlebois, W. Ford Doolittle, and Eric Bapteste. Deduction of probable events of lateral gene transfer through comparison of phylogenetic trees by recursive consolidation and rearrangement. *BMC Evolutionary Biology*, 5(27), 2005. http://dx.doi.org/10.1186/1471-2148-5-27. 1

[McK81] Brendan McKay. Practical graph isomorphism. *SIAM Journal on Computing*, 30:45–87, 1981. algorithm implemented in Nauty http://cs.anu.edu.au/~bdm/nauty/. 8

[PT86] Judea Pearl and Michael Tarsi. Structuring causal trees. *Journal of Complexity*, 2(1):60–77, 1986. ftp://ftp.cs.ucla.edu/tech-report/198_-reports/850029.pdf. 15

[Slo08] Neil J.A. Sloane. The on-line encyclopedia of integer sequences, 2008. Published electronically at http://www.research.att.com/~njas/sequences/. 8

[SS04] Charles Semple and Mike Steel. Unicyclic networks: compatibility and enumeration. *IEEE/ACM Transactions in Computational Biology and Bioinformatics*, 3:398–401, 2004. http://www.math.canterbury.ac.nz/~c.semple/papers/SS06.pdf. 9

[Ste92] Mike Steel. The complexity of reconstructing trees from qualitative characters and subtrees. *Journal of Classification*, 9:91–116, 1992. http://www.math.canterbury.ac.nz/~m.steel/Non_UC/filesresearch/complexity.pdf. 15

[vI] Leo van Iersel. Complexity of constructing a level-k network.... http://www.win.tue.nl/~liersel/overview.html. 2

[vIKK+08] Leo van Iersel, Judith Keijsper, Steven Kelk, Leen Stougie, Ferry Hagen, and Teun Boekhout. Constructing level-2 phylogenetic networks from triplets. In *Proceedings of the twelfth Annual International Conference on Research in Computational Molecular Biology (RECOMB'08)*, volume 4955 of *Lecture Notes in Computer Science*, pages 450–462. Springer Verlag, 2008. http://homepages.cwi.nl/~kelk/RECOMBrevised.pdf. 2, 3, 4, 8, 17

[vIKM07] Leo van Iersel, Steven Kelk, and Matthias Mnich. Uniqueness, intractability and exact algorithms: reflections on level-k phylogenetic networks, 2007. Submitted to JBCB, http://arxiv.org/pdf/0712.2932v2. 2

[Wu04] Bang Ye Wu. Constructing the maximum consensus tree from rooted triples. *Journal of Combinatorial Optimization*, 29:29–39, 2004. http://www.pws.stu.edu.tw/bangye/paper/cett_JISE.pdf. 2

[WZZ01] Lusheng Wang, Kaizhong Zhang, and Louxin Zhang. Perfect phylogenetic networks with recombination. In *Proceedings of the 16th ACM Symposium on Applied Computing (SAC'01)*, pages 46–50, 2001. http://dx.doi.org/10.1145/372202.372271. 1