Generation of Binary Tree-Child phylogenetic networks

Gabriel Cardona, Joan Carles Pons, Celine Scornavacca

Generation of Binary Tree-Child phylogenetic networks

Gabriel Cardona[1*], Joan Carles Pons[1], Celine Scornavacca[2]

1 Department of Mathematics and Computer Science, University of the Balearic Islands, Ctra. de Valldemossa, km. 7.5, E-07122 Palma, Spain
2 Institut des Sciences de l'Evolution (ISE-M), Université de Montpellier, CNRS, IRD, EPHE, 34095 Montpellier Cedex 5, France

* gabriel.cardona@uib.es

Abstract

Phylogenetic networks generalize phylogenetic trees by allowing the modelization of events of reticulate evolution. Among the different kinds of phylogenetic networks that have been proposed in the literature, the subclass of binary tree-child networks is one of the most studied ones. However, very little is known about the combinatorial structure of these networks.

In this paper we address the problem of generating all possible binary tree-child (BTC) networks with a given number of leaves in an efficient way via reduction/augmentation operations that extend and generalize analogous operations for phylogenetic trees, and are biologically relevant. Since our solution is recursive, this also provides us with a recurrence relation giving an upper bound on the number of such networks.

We also show how the operations introduced in this paper can be employed to extend the evolutive history of a set of sequences, represented by a BTC network, to include a new sequence.

An implementation in python of the algorithms described in this paper, along with some computational experiments, can be downloaded from `https://github.com/bielcardona/TCGenerators`.

Author summary

Phylogenetic networks are widely used to represent evolutionary scenarios with reticulated events, and among them, the class of binary tree-child (BTC for short) networks is one of the most studied ones. Despite its importance, BTC networks, as mathematical objects, are not yet fully understood. In this paper we introduce two operations (reduction and augmentation) on the set of BTC networks that generalize well known operations on phylogenetic trees, and show how they can be used to analyze and synthesize any BTC network. Apart from the mathematical formulation of the problem, we exhibit how these operations can be used in biological applications to add a new sequence to a given BTC network. This can be useful, for instance, to update the network without redoing the whole search, or in a phylogenetic placement perspective. We also obtain a recursive formula for a bound on the number of such networks. We have implemented the algorithms in this paper, made them available on a public repository, and used this implementation to perform some computational simulations.

Introduction

Phylogenetic networks are, mathematically, a generalization of phylogenetic trees that, containing nodes with more than one ancestor, permit to model reticulated evolutionary events such as recombinations, lateral gene transfers and hybridizations.[1]

In this paper, we shall focus on directed phylogenetic networks (see [3] for a short survey on the phylogenetic network paradigm also covering undirected phylogenetic networks). Mathematically, such networks are, in the broadest sense, directed acyclic graphs with a single node with no incoming arcs –the *root*– representing the common ancestor of all the Operational Taxonomic Units (OTUs for short) under study, which are represented by the nodes with no outgoing arcs –the *leaves*– of the graph; internal nodes represent either (hypothetical) speciations or (hypothetical) reticulated events. Nodes with a single incoming arc –*tree* nodes– model extant or non-extant OTUs, and arcs between tree nodes model direct descent through mutation; nodes with two incoming arcs –*hybrid* nodes– model reticulated events involving the OTUs corresponding to the two parents of the node under consideration, and whose resulting OTU is modeled as its single child. Unfortunately, this definition is too broad, both for representing biologically-meaningful evolutionary scenarios, and for giving objects that can be efficiently handled.

So far, several restrictions on this general definition have been introduced in the literature. A few of them are based on biological considerations, while the majority have been introduced to artificially narrow the space of networks under study. This led to the introduction of a panoply of different classes of phylogenetic networks, such as time-consistent networks [4], regular networks [5], orchard networks [6], galled trees [7] and galled networks [8], level-k networks [9], tree-sibling networks [10], tree-based networks [11] and LGT networks [12], just to name a few.

In this paper, we shall focus on binary tree-child networks (BTC networks, for short), which were introduced by [10] and are one of the most studied classes of phylogenetic networks [13–16]. Mathematically, being tree-child means that every internal node is compelled to have at least a child node that is a tree node. BTC networks have been introduced in order to adjust a complex biological reality in a computationally tractable way. Although the original motivation for these networks is not biological, and hence they present some limitations, the mathematical constraint on BTC networks translates biologically as follows: every non-extant OTU is required to have at least an offspring species that evolved only through mutation. This means that not all biologically-meaningful evolutionary scenarios can be modeled with BTC networks. For example, the scenarios depicted in Fig 1(a,e) are not allowed since, in these cases, the node labeled with u has no child with a single incoming arc. Still, BTC networks are one of the most permissive classes of phylogenetic networks and they permit to model quite a lot of meaningful scenarios, and those that cannot be modeled can be approximated pretty well, see Fig 1.

The combinatorial study of phylogenetic networks is nowadays a challenging and active field of research. Nevertheless, the problem of counting how many phylogenetic networks are in a given subclass of networks is still open even for long-established classes. More precisely, this problem has been only recently solved for galled networks [18]; for other classes, including tree-child networks, we only have asymptotic results [19, 20]. Associated to the problem of counting networks, we find the problem of their "injective" generation, i.e. without having to check for isomorphism between pairs of constructed networks.

[1] We note here that other representations, for example gene tree-species tree reconciliations [2], permit to model scenarios including other classes of evolutionary events such as duplications, losses and transfers of genes.

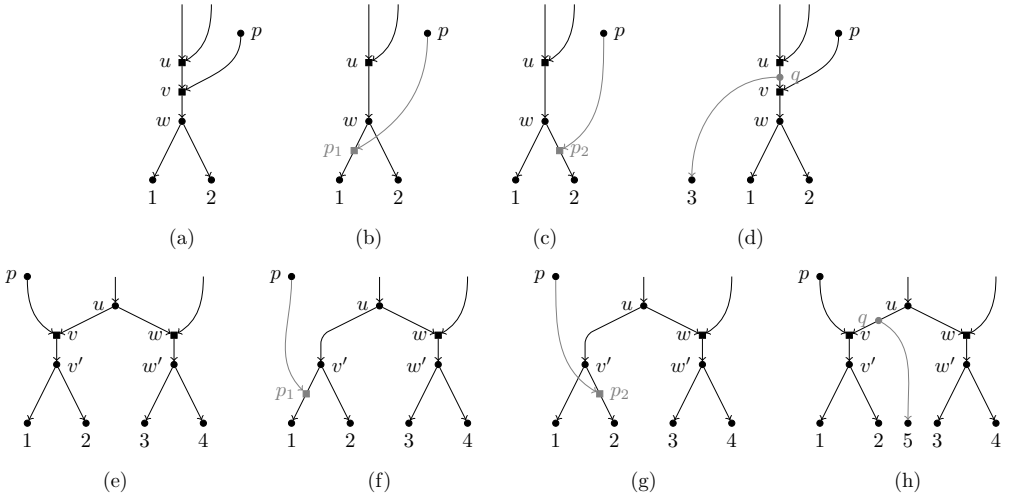

Fig 1. Limitations of BTC networks. The scenarios in (a) and (e) are not BTC networks since in both cases the node labeled with u has no tree-node child. Still, the scenario in (a) can be approximated either by the scenario in (b) or by that in (c), both scenarios being BTC networks. Also, if we are lucky enough to find an OTU between the hybrid event represented by the node u and that represented by v, e.g. the node q in (d), then the hybrid event in v can be modeled. The same reasoning holds for the scenario in (e) and those in (f,g,h). Thus, if the "true" network is not BTC, we can always find a BTC network those topology is not far from the true one. In our example, the networks in (b,c) are both a head-moving rSPR [1] away from the true network in (a). The same holds for the networks in (f,g) w.r.t. the one in (e). In general, each violation of the TC property, i.e. each hybrid node that has only hybrid children, moves the reconstructible network a head-moving rSPR away from the true one. Note that the configuration in (a) is known to generate severe indistinguishability issues [17].

The main result of this paper is a systematic way of recursively generating, with unicity, all BTC networks with a given number of leaves. This generation relies on a pair of reduction/augmentation operations –both producing BTC networks– where reductions decrease by one the number of leaves in a network, and augmentations increase it. The idea of using pairs of operations has already been used to deal either with other classes of phylogenetic networks [21, 22], or for BTC networks but without the unicity feature [6].

In order to give a biological meaning to these augmentation operations, assume that the evolutive history of a given group of species is known and modeled by a BTC network, and a new species has to be taken into account. The augmentation operation determines exactly how the phylogenetic network has to be modified, and what is the minimum information needed to establish this modification, in order to model the evolution of the group of species with the newly incorporated one.

As an interesting side product, this procedure gives a recursive formula providing an upper bound on the number of BTC networks. Note also that being able to generate all BTC networks with a given number of leaves may also be interesting as part of a divide-and-conquer framework to reconstruct phylogenetic networks, where we start by computing BTC networks on 3/5 leaves that are then combined together, as done for example in [23, 24].

The paper is organized as follows. In Section Methods, we review the basic definitions that will be used throughout the paper. The main part of the paper is in Section Results, which is split between different subsections. Subsection Reduction of

Networks is devoted to the reduction procedure, while in Subsection Generation of Networks we introduce the augmentation operation and prove that any BTC network can be obtained, in a unique way, via a sequence of augmentation operations applied to the trivial network with one leaf. In Subsection Bounding the Number of Networks, we show how to relax the conditions for the applicability of the augmentation operation to obtain a recursive formula providing an upper bound on the number of BTC networks. In Subsection An Application to Phylogenetic Reconstruction, we give a concrete biological application of the methods we have developed. In Subsection Computational Experiments, we introduce the implementation of the algorithms presented in the paper, and some experimental results, including the exhaustive generation of all BTC networks with up to six leaves and an upper bound of their number up to ten leaves. Finally, in Section Conclusion we discuss how our reduction/augmentation operations extend and generalize analogous operations for phylogenetic trees.

Methods

In this section we introduce the mathematical notations that are used in the rest of the paper.

Throughout this paper, a *tree node* in a directed graph is a node u whose pair of degrees $d(u) = (\text{indegree}\, u, \text{outdegree}\, u)$ is $(1, 0)$ for the *leaves*, $(0, 2)$ for the *roots*, or $(1, 2)$ for *internal* tree nodes; a *hybrid node* is a node u with $d(u) = (2, 1)$. If two nodes u and v are linked by an arc (u, v) we say that u is a *parent* of v, or that v is a *child* of u. Also, two nodes are *siblings* if they have a common parent.

A *binary phylogenetic network* over a set X of taxa is a directed acyclic graph with a single root such that all its nodes are either tree nodes or hybrid nodes, and whose leaf set is bijectively labeled by the set X. In the following, we will implicitly identify every leaf with its label. A binary phylogenetic network is *tree-child* if every node either is a leaf or has at least one child that is a tree node [10]; in particular, the single child of a hybrid node must be a tree node. We will denote by $\mathcal{BTC}_n$ the set of binary tree-child phylogenetic networks over the set $[n] = \{1, \ldots, n\}$.

An *elementary node* in a directed graph is a node u with $d(u) = (1, 1)$ or $d(u) = (0, 1)$. An *elementary path* p is a path $u_1, \ldots, u_k$ composed of elementary nodes such that neither the single parent of u_1 (if it exists) nor the single child of u_k are elementary. We call these last two nodes respectively the *grantor* (if this node is well-defined) and *heir* of the nodes in the elementary path. In case of an elementary node, its grantor and heir are those of the nodes in the single elementary path that contains the given node. The *elimination* of an elementary path p consists in deleting all nodes in p, together with their incident arcs, and adding an arc between the grantor and the heir of p (provided that the grantor exists; otherwise, no arc is added). The elimination of an elementary node is defined as the elimination of the elementary path that contains the given node.

Given a node u, we can *split* it by adding a new node $\tilde{u}$, an arc $(\tilde{u}, u)$, and replacing every arc (v, u) with $(v, \tilde{u})$. If u is a tree node, then $\tilde{u}$ is an elementary node whose heir is u, and the elimination of $\tilde{u}$ recovers the original network. The successive splitting (say k times) of a tree node u generates an elementary path formed by k nodes, whose heir is u, and whose elimination recovers the original network. Fig 2 illustrates the definitions given in this section.

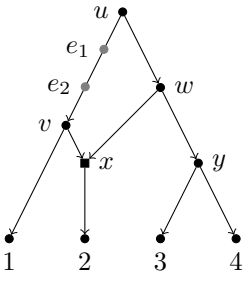

Fig 2. The definitions introduced in the Methods Section. For the network N in the figure (black nodes and arcs only), we have the following: $X = \{1, 2, 3, 4\}$ is the set of taxa, u is the root, x is a hybrid node and all other nodes are internal tree nodes. If we split v twice by adding the elementary nodes e_1 and e_2 in grey, we have that (e_1, e_2) is an elementary path with grantor and heir equal respectively to u and v. N is a binary tree-child network since both parents v and w of the only hybrid node x have another child (1 and y, respectively) that is a tree node.

Results

Reduction of Networks

The goal of this subsection is to define a reduction procedure on BTC networks that can be applied to any such network, and producing a BTC network with one leaf less. By successive application of this procedure, any BTC network can thus be reduced to the trivial network with a single leaf.

We start by associating to each leaf ℓ a path whose removal will produce the desired reduction (up to elementary paths).

Let ℓ be a leaf of a BTC network N. A *pre-TH-path* for ℓ is a path $u_1, \ldots, u_r = \ell$ such that (see Fig 3):

1. Each node u_i in the path is a tree node.
2. For each $i = 1, \ldots, r-1$, the child of u_i different from u_{i+1}, denoted by v_i, is a hybrid node.
3. For each $i \neq j$, we have that $v_i \neq v_j$.

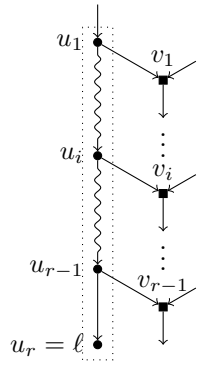

Fig 3. A pre-TH-path for the leaf ℓ. Tree nodes are represented by circles and hybrid nodes by squares; snake arrows represent paths. The path inside the dotted box is a pre-TH-path for ℓ.

A *TH-path* is a maximal pre-TH-path, i.e. a pre-TH-path that cannot be further extended. Note that, since all nodes in a pre-TH-path p are tree nodes, if p can be extended by prepending one node, then this extension is unique. Hence, starting with the trivial pre-TH-path formed by the leaf ℓ alone, and extending it by prepending the parent of the first node in the path as many times as possible, we obtain a TH-path that is unique by construction. Let $u_1, \dots, u_r = \ell$ be a TH-path; different possibilities may arise that make it maximal: (1) u_1 is the root of N; (2) the parent of u_1, call it x, is a hybrid node; (3) x is a tree node whose both children are tree nodes; (4) x is a parent of v_i for some $i \in [r-1]$. We shall see in Lemma 1 that the first case cannot hold; the other three possibilities are depicted in Fig 4.

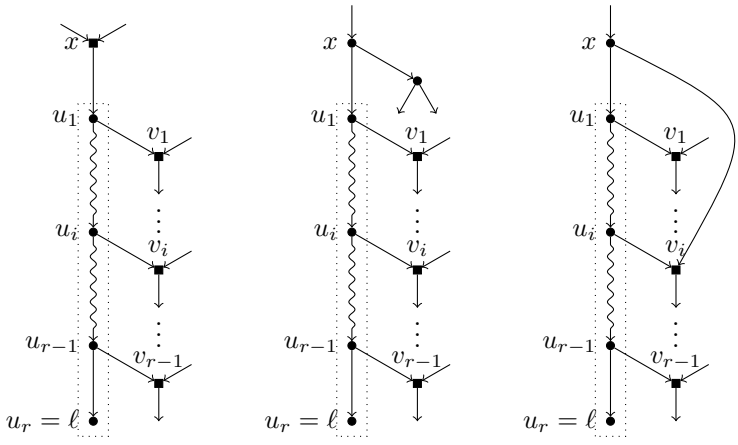

Fig 4. The different possibilities for the TH-path of a leaf ℓ. Depiction of the conditions (2), (3) and (4), respectively, under which a pre-TH-path cannot be extended, making the path inside the dotted box a TH-path for ℓ.

For each leaf ℓ, we denote by $\mathrm{TH}(\ell)$ its single TH-path and by $\mathrm{TH}(\ell)_1$ the first node of this path. Note that we allow the case $r = 1$. In this case, if we are not in a trivial BTC network (i.e. a network consisting of a single node), the parent of ℓ is either a hybrid node, or a tree node whose two children are tree nodes.

Lemma 1. *Let N be a non-trivial BTC network and let ℓ be any of its leaves. Then, $\mathrm{TH}(\ell)_1$ cannot be the root of N.*

Proof. Let $u_1, \dots, u_r = \ell$ be the path $\mathrm{TH}(\ell)$ and assume for the sake of contradiction that u_1 is the root of N. For each $i = 1, \dots, r-1$, let v_i be the hybrid node that is a child of u_i and x_i the parent of v_i different from u_i (see Fig 5); recall that x_i does not belong to $\mathrm{TH}(\ell)$ by the definition of a pre-TH-path. Since u_1 is the root of N, every node of N either belongs to the path $\mathrm{TH}(\ell)$ or is descendant of a node in $\{v_i \mid i \in [r-1]\}$. In particular, for each $i \in [r-1]$, there exists some $\sigma(i) \in [r-1]$ such that x_i is descendant of $v_{\sigma(i)}$, and since this node is descendant of $x_{\sigma(i)}$, x_i is descendant of $x_{\sigma(i)}$. Hence, starting with x_1 we get a sequence $x_1, x_{\sigma(1)}, x_{\sigma(\sigma(1))}, \dots$ where each node in the sequence is a descendant of the following one. Since there is a finite number of nodes, at some point we find a repeated node, which means that N contains a cycle and hence we have a contradiction. □

We say that a leaf ℓ is of *type T* (resp. of *type H*) if the parent of $\mathrm{TH}(\ell)_1$ is a tree node (resp. a hybrid node). If ℓ is of type H, we indicate by $\overline{\mathrm{TH}}(\ell)$ the path obtained by prepending to $\mathrm{TH}(\ell)$ the parent of $\mathrm{TH}(\ell)_1$. For convenience, we let $\overline{\mathrm{TH}}(\ell) = \mathrm{TH}(\ell)$ if ℓ is of type T.

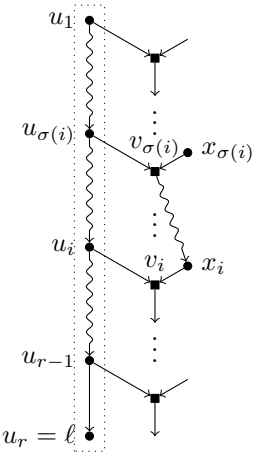

Fig 5. The first node in a TH-path cannot be the root. Illustration of the nodes involved in the proof of Lemma 1.

Definition 1. Let ℓ be a leaf in a BTC network N. We define the *reduction of N with respect to ℓ* as the result of the following procedure (see Figs 6 and 7):

1. Delete all nodes in $\overline{\mathrm{TH}}(\ell)$ (together with any arc incident on them).
2. Eliminate all elementary nodes.

We indicate this reduction by $R(N, \ell)$. If we want to emphasize the type of the deleted leaf, we indicate the reduction by $T(N, \ell)$ and say it is a T-reduction if ℓ is of type T, or by $H(N, \ell)$ and say that it is a H-reduction if ℓ is of type H.

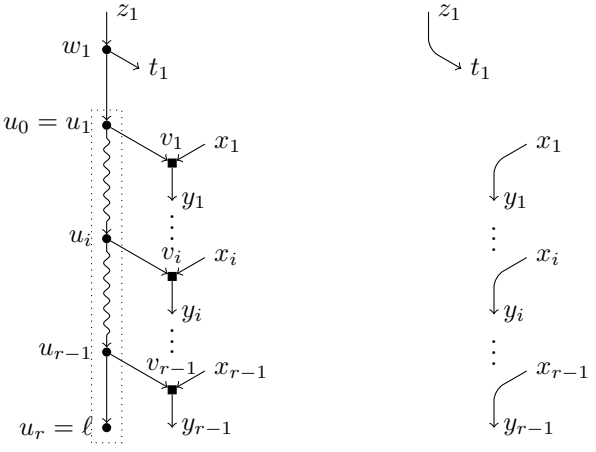

Fig 6. Reduction operation of type T. Depiction of the reduction operation $T(N, \ell)$. The nodes inside the dotted box form $\overline{\mathrm{TH}}(\ell)$ and will be removed, which will create elementary nodes that will be substituted by arcs.

To ease of reading, we shall introduce some notations that will be used hereafter and are also illustrated in Figs 6 and 7:

Definition 2. Let $u_1, \ldots, u_r = \ell$ be the path $\mathrm{TH}(\ell)$ and let u_0 be the first node in $\overline{\mathrm{TH}}(\ell)$. For each $i \in [r-1]$, v_i is the hybrid child of u_i, x_i the parent of v_i different from u_i, and y_i the single child of v_i. The parent(s) of u_0 is w_1 (are w_1, w_2); the node w_j is

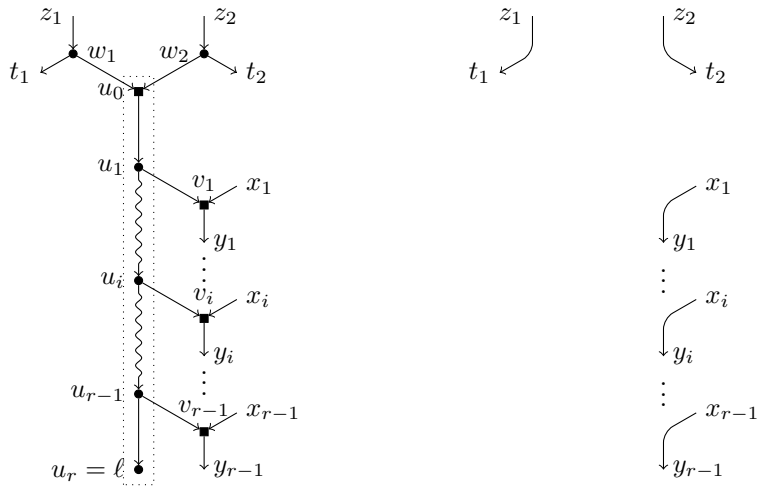

Fig 7. Reduction operation of type H. Depiction of the reduction operation $H(N, \ell)$. The nodes inside the dotted box form $\overline{\mathrm{TH}}(\ell)$ and will be removed, which will create elementary nodes that will be substituted by arcs.

always a tree node, z_j is its parent (if it exists, since w_j could be the root of N), and t_j its child different from u_0, where $j = 1$ for T-reductions and $j \in [2]$ for H-reductions.

Remark 1. Since N is tree-child, the nodes y_i are always tree nodes, and so are t_1 and t_2 in case of an H-reduction. In case of a T-reduction, by definition of a TH-path, t_1 is either a tree node or coincides with one of the hybrid nodes v_i. Also, the removal of the arcs of the form (u_i, v_i) and (w_j, u_0) makes nodes v_i and w_j elementary in $N \setminus \overline{\mathrm{TH}}(\ell)$, where $i \in [r-1]$, and $j = 1$ for T-reductions and $j \in [2]$ for H-reductions. Since no other arc is removed, no other node can be elementary. In order to find the heirs of nodes v_i and w_j, we must analyse under which circumstances two of these elementary nodes are adjacent in $N \setminus \overline{\mathrm{TH}}(\ell)$.

1. If we had that two nodes v_i and v_j were connected by an arc in $N \setminus \overline{\mathrm{TH}}(\ell)$, then the single child of a hybrid node in N would be also a hybrid. This contradicts the fact that N is tree-child.

2. The existence of an arc (v_i, w_j) would imply the existence of a cycle in N, which is impossible.

3. Consider now the case of an arc (w_j, v_i). In case of an H-reduction, it would imply that both children of w_j are hybrid nodes, which is impossible. However, such an arc can be present in a T-reduction: when t_1 is equal to v_i. In this last case, w_1 and v_i form an elementary path in $N \setminus \overline{\mathrm{TH}}(\ell)$ and their common heir is y_i (see Fig 8).

4. Finally, in case of an H-reduction, it can exist an arc between w_1 and w_2, say that the arc is (w_1, w_2) (which implies, $t_1 = w_2$, $z_2 = w_1$). In this case, w_1 and w_2 form an elementary path in $N \setminus \overline{\mathrm{TH}}(\ell)$ and their common heir is t_2 (see Fig 9).

In all other cases, the elementary nodes v_i and w_j are isolated, and their respective heirs are y_i and t_j.

We study now what we call the *recovering data* of a reduction. This information will be used in the next subsection to recover the original network from its reduction.

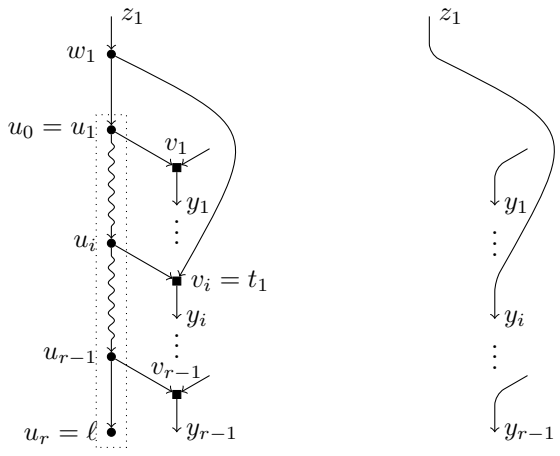

Fig 8. Reduction operation of type T (particular case). Particular case of the reduction operation of type T when $t_1 = v_i$ for some $i \in [r-1]$.

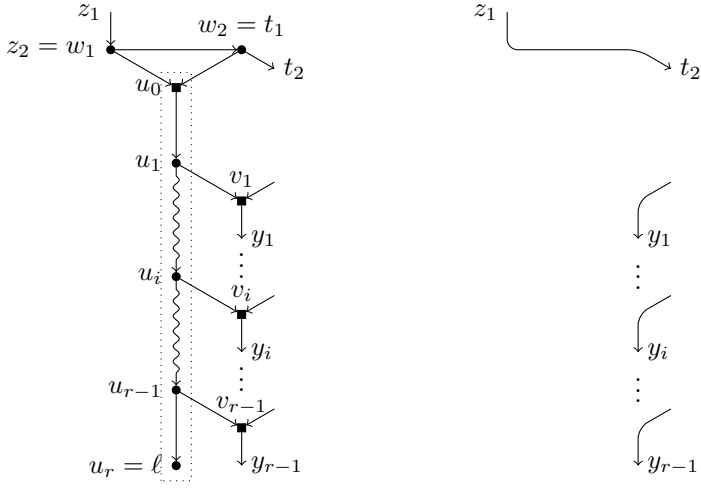

Fig 9. Reduction operation of type H (particular case). Particular case of the reduction operation of type H when w_1 and w_2 are linked by an arc.

Definition 3. The *recovering data* of the reduction $N' = R(N, \ell)$ is the pair (S_1, S_2), where:

- S_1 is the multiset of the nodes of N' that are heirs of the nodes w_j. The cardinality of S_1 (as a multiset) is either 1 or 2, depending on the type of the reduction, and will be denoted by $|S_1|$.
- S_2 is the tuple $(y_1, \ldots, y_{r-1})$ of nodes of N', which are the heirs of the nodes v_i. This tuple could be empty, corresponding to the case $r = 1$.

We introduce now a set of conditions on multisets and tuples of nodes, and prove that the recovering data associated to any of the defined reductions satisfies them.

Definition 4. Given a BTC network N' and a pair (S_1, S_2) with

- S_1 a multiset of tree nodes of N',
- $S_2 = (y_1, \ldots, y_{r-1})$, with $r \geq 1$, a (potentially empty) tuple of $r-1$ tree nodes of N',

consider the following set of conditions:

1. For every $i, j \in [r-1]$ with $i \neq j$, the nodes y_i and y_j are different, and if they are siblings, then $y_i \in S_1$ or $y_j \in S_1$.
2. For every $i \in [r-1]$, if y_i is the child of a hybrid node or has a hybrid sibling, then $y_i \in S_1$.
3. No node in S_1 is a proper descendant of any node in S_2.

4T. $|S_1| = 1$.

4H. $|S_2| = 2$ and no node of S_1 appears in S_2.

We say that (S_1, S_2) is *T-feasible* if it satisfies conditions 1, 2, 3, and 4T, and *H-feasible* if it satisfies conditions 1, 2, 3, and 4H. Finally, we say that (S_1, S_2) is *feasible* if it is either T-feasible or H-feasible.

Proposition 2. *Let $N' = T(N, \ell)$ be a T-reduction of a BTC network N. Then, its recovering data $(\{\tau_1\}, (y_1, \ldots, y_{r-1}))$ is T-feasible.*

Proof. First, note that, by Remark 1, all nodes in $(\{\tau_1\}, (y_1, \ldots, y_{r-1}))$ are tree nodes and that Condition 4T holds trivially. Note also that τ_1 is equal to y_i if $t_1 = v_i$, or to t_1 if this node is different from all the nodes v_i. We now prove that Conditions 1, 2 and 3 hold:

1. If $y_i = y_j$, then in N we have $v_i = v_j$, which is impossible by definition of TH-path. If y_i and y_j are siblings in N' but none of these nodes is equal to τ_1, then v_i and v_j are siblings in N, which implies that their common parent has two hybrid children, which is impossible in a BTC network.
2. If y_i is the child in N' of a hybrid node and $\tau_1 \neq y_i$, then in N we have that v_i, which is a hybrid node, is the child of a hybrid node, which is impossible in a tree-child network. Analogously, if y_i has a sibling in N' which is a hybrid node, and $y_i \neq \tau_1$, then in N we have that v_i is sibling of another hybrid node, which is again impossible.
3. The existence of a non-trivial path in N' from y_i to τ_1 would, by construction, imply the existence of a path from y_i to w_1 in N. Since there exists also a path in N from w_1 to y_i, this would contradict the fact that N is a DAG.

□

Proposition 3. *Let $N' = H(N, \ell)$ be an H-reduction of a BTC network N. Then, its recovering data $(\{\tau_1, \tau_2\}, (y_1, \ldots, y_{r-1}))$ is H-feasible.*

Proof. Again we have, by Remark 1, that all nodes in the recovering data are tree nodes. Additionally, by the same remark, we have that $|S_1| = 2$ –and hence the first part of Condition 4H holds– and if (w_1, w_2) is an arc of N, then $S_1 = \{t_2, t_2\}$, otherwise $S_1 = \{t_1, t_2\}$ with $t_1 \neq t_2$. Note that Condition 3 implies that Conditions 1 and 2 can be simplified as follows: for all $i, j \in [r-1]$ with $i \neq j$, y_i and y_j are neither equal nor siblings, and for all $i \in [r-1]$, y_i is neither the child nor the sibling of a hybrid node.

Conditions 1 and Conditions 2 and 3 in their simplified form follow using the same arguments as in the previous proposition. As for the condition 4H, the nodes τ_1 and τ_2 are different from the nodes y_i since the parents of τ_1 and τ_2 in N are tree nodes, while the parent of each of the nodes y_i is hybrid. □

The following proposition is the main result of this subsection, since it shows that the reduction that we have defined, when applied to a BTC network, gives another BTC network with one leaf less. Hence, successive applications of these reductions reduce any BTC network to the trivial BTC network.

Proposition 4. *Let N be a BTC network over X and ℓ one of its leaves. Then, $R(N,\ell)$ is a BTC network over $X \setminus \{\ell\}$.*

Proof. First, it is easy to see that, since no new path is added, the resulting directed graph is still acyclic.

Then, we need to check that $R(N,\ell)$ is binary. To do so, we start noting that every node in $N \setminus \overline{\mathrm{TH}}(\ell)$ is either a tree node, a hybrid node, or an elementary node. Indeed, the removal of $\overline{\mathrm{TH}}(\ell)$ (Phase 1 of Definition 1) only affects the nodes adjacent to this path, that is the nodes v_i and w_i, which, as shown in Remark 1, become elementary. The elimination of all elementary nodes (Phase 2 of Definition 1) does not affect the indegree and outdegree of any other node, apart when the root ρ of $N \setminus \overline{\mathrm{TH}}(\ell)$ is elementary. In such a case, the heir of ρ becomes the new root. Hence, $R(N,\ell)$ is binary and rooted.

Note also that the set of leaves of $R(N,\ell)$ is $X \setminus \{\ell\}$, since in $N \setminus \overline{\mathrm{TH}}(\ell)$ no node becomes a leaf and the only leaf that is removed is ℓ.

Finally, we need to prove that $R(N,\ell)$ is tree-child. Note that, from what we have just said about how the reduction affects indegrees and outdegrees of the nodes that persist in the network, it follows that each hybrid node of $R(N,\ell)$ is also a hybrid node of N, and that its parents in $R(N,\ell)$ are the same as in N. It follows that no node in $R(N,\ell)$ can have that all its children are hybrid, since this would imply that N is not tree-child, a contradiction. □

Corollary 5. *Let $N \in \mathcal{BTC}_n$ be a BTC network over $[n]$. Let $N_n = N$ and define recursively $N_i = R(N_{i+1}, i+1)$ for each $i = n-1, n-2, \ldots, 1$. Then, N_i is a BTC network over $[i]$. In particular, N_1 is the trivial BTC network with its single node labeled by 1.*

We finish this subsection with the computation of the number of tree nodes and hybrid nodes that the reduced network has, both in terms of the original network and of the reduction operation that has been applied. But before, we give an absolute bound on the number of these nodes in terms of the number of leaves.

Lemma 6. *Let N be BTC network over $[n]$ with t tree nodes and h hybrid nodes. Then $t - h = 2n - 1$, $h \leq n - 1$ and $t \leq 3n - 2$.*

Proof. The equality $t - h = 2n - 1$ follows easily from the handshake lemma taking into account the number of roots, internal tree nodes, leaves and hybrid nodes in N, and their respective indegrees and outdegrees. The inequality $h \leq n - 1$ is shown in Proposition 1 in [10], and the last inequality is a simple consequence of the equality and the inequality already proved. □

Proposition 7. *Let N be a BTC network and ℓ one of its leaves, and $N' = R(N,\ell)$. Let t, h (resp. t', h') be the number of tree nodes and hybrid nodes of N (resp. of N'). Then*

$$t' = t - |\overline{\mathrm{TH}}(\ell)| - 1, \qquad h' = h - |\overline{\mathrm{TH}}(\ell)| + 1,$$

where $|\overline{\mathrm{TH}}(\ell)|$ is the number of nodes in $\overline{\mathrm{TH}}(\ell)$.

Proof. Since the number of tree nodes and hybrid nodes are linked by the equality in Lemma 6, it is enough to prove that $h' = h - |\overline{\mathrm{TH}}(\ell)| + 1$. From the discussion in Remark 1, it is straightforward to see that the number of hybrid nodes in N that are not in N' is $r - 1$ if ℓ is of kind T, and r otherwise. Hence, in both cases we have $h' = h - (|\overline{\mathrm{TH}}(\ell)| - 1)$ and the result follows. □

Generation of Networks

In this subsection, we consider the problem of how to revert the reductions defined in the previous subsection, taking as input the reduced network and its recovering data. This will allow us to define a procedure that, starting with the trivial BTC network with one leaf, generates all the BTC networks with any number of leaves in a unique way.

We start by defining two augmentation procedures that take as input a BTC network and a feasible pair, and produce a BTC network with one leaf more.

Definition 5. Let N be a BTC network over X, ℓ a label not in X, and $(\{\tau_1\}, (y_1, \dots, y_{r-1}))$ a T-feasible pair. We apply the following operations to N (see Fig 10):

1. Create a path of new nodes $u_1, \dots, u_r$.
2. Split the node τ_1 creating one elementary node w_1 and add an arc (w_1, u_1).
3. For each node y_i, split it introducing one elementary node v_i and add an arc (u_i, v_i).
4. Label the node u_r by ℓ.

We denote by $T^{-1}(N, \ell; \{\tau_1\}, (y_1, \dots, y_{r-1}))$ the resulting network and say that it has been obtained by an *augmentation operation of type* T.

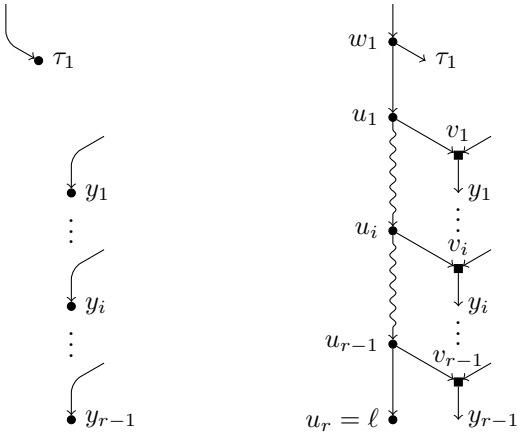

Fig 10. Augmentation operation of type T. Depiction of the augmentation operation $T^{-1}(N, \ell; \{\tau_1\}, (y_1, \dots, y_{r-1}))$ when $\tau_1 \neq y_i$ for all $i \in [r-1]$.

Note that the order in which steps 2 and 3 are done is relevant in the case that $\tau_1 = y_i$ for some $i \in [r-1]$. In such a case, two nodes w_1 and v_i are created, linked by an arc (w_1, v_i) (see Fig 11).

Proposition 8. *Using the notations of Definition 5, the network*

$$\tilde{N} = T^{-1}(N, \ell; \{\tau_1\}, (y_1, \dots, y_{r-1}))$$

is a BTC network over $X \cup \{\ell\}$. Moreover, if N has h hybrid nodes, then $\tilde{N}$ has $h + r - 1$ hybrid nodes.

Proof. We first check that the resulting directed graph is acyclic. Let us assume that $\tilde{N}$ contains a cycle. If we define $U_1 = \{u_1, \dots, u_r\}$ and $U_2 = V(\tilde{N}) \setminus U_1$, we have that the only arcs connecting U_1 with U_2 are (u_i, v_i) (with $i = 1, \dots, r-1$), and (w_1, u_1) is the

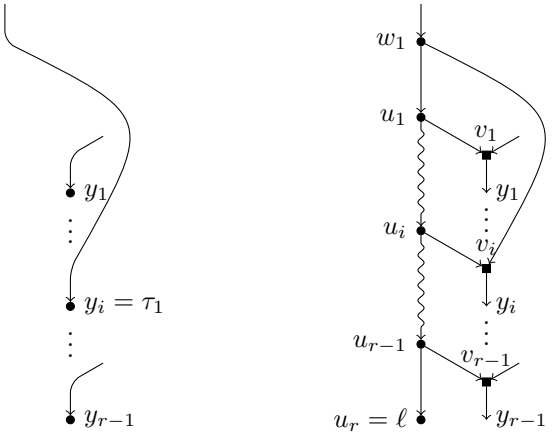

Fig 11. Augmentation operation of type T (particular case). Depiction of the augmentation operation $T^{-1}(N, \ell; \{\tau_1\}, (y_1, \ldots, y_{r-1}))$ when $\tau_1 = y_i$.

only arc connecting U_2 with U_1. The cycle can be contained neither inside U_1, since these nodes are linked by a single path, nor inside U_2, since otherwise N would contain a cycle. Hence, the cycle must contain at least the arc (w_1, u_1) and an arc (u_i, v_i). This implies the existence of a path from v_i to w_1 visiting only nodes in U_2, which in turn means that N contains a path from y_i to τ_1, against Condition 3 of Definition 4.

Note that the nodes in U_1 are tree nodes by construction. Also by construction, the node w_1 is a tree node, the nodes v_i are hybrid nodes and u_r is a leaf which is labelled with ℓ. Finally, the other nodes keep the same degrees they had in N and hence $\tilde{N}$ is a binary phylogenetic network over $X \cup \{\ell\}$ with $h + r - 1$ hybrid nodes.

Since N is tree-child, in order to check that $\tilde{N}$ is also tree-child, we only need to check the newly added hybrid nodes, which are the parents of the nodes v_i.

Let us first consider the case that $\tau_1 \neq y_i$ for all $i \in [r-1]$. For each node v_i, its parents are u_i and the parent x_i of y_i in N. The node u_i is by construction a tree node whose other child is u_{i+1}, which, in turn, is a tree node. Since $\tau_1 \neq y_i$, by Condition 2 of Definition 4, y_i can have neither a hybrid parent nor a hybrid sibling, and it cannot be a sibling of any other node y_j with $j \in [r-1]$. This latter restriction implies that y_i has the same sibling $\tilde{x}_i$ in N and $\tilde{N}$. Thus both x_i and $\tilde{x}_i$ are not hybrid nodes, and the network is tree-child.

Let us now consider the case that $\tau_1 = y_i$ for a single choice of $i \in [r-1]$. The hybrid node v_i in $\tilde{N}$ has as parents the nodes w_1 and u_i, and these two nodes have as respective children u_1 and u_{i+1}, which are tree nodes. For each other node v_j with $j \neq i$ and such that y_j is a not sibling of y_i, the same argument as in the previous case proves that both parents of v_j have a tree child. If y_j is a sibling of y_i, it is easy to see that the parent of v_j is still tree-child since it has w_1 as child. □

Definition 6. Let N be a BTC network over X, ℓ a label not in X, and $(\{\tau_1, \tau_2\}, (y_1, \ldots, y_{r-1})$ a H-feasible pair. We apply the following operations to N (see Fig 12):

1. Create a path of new nodes $u_0, u_1, \ldots, u_r$.
2. Split each of the nodes τ_i introducing one elementary node w_i and add an arc from w_i to u_0. Note that, if $\tau_1 = \tau_2$, two consecutive elementary nodes must be created (see Fig 13 for this case).
3. For each node y_i, split it introducing one elementary node v_i and add an arc (u_i, v_i).

4. Label the node u_r by ℓ.

We denote by $H^{-1}(N, \ell; \{\tau_1, \tau_2\}, (y_1, \ldots, y_{r-1}))$ the resulting network and say that it has been obtained by an augmentation operation of type H.

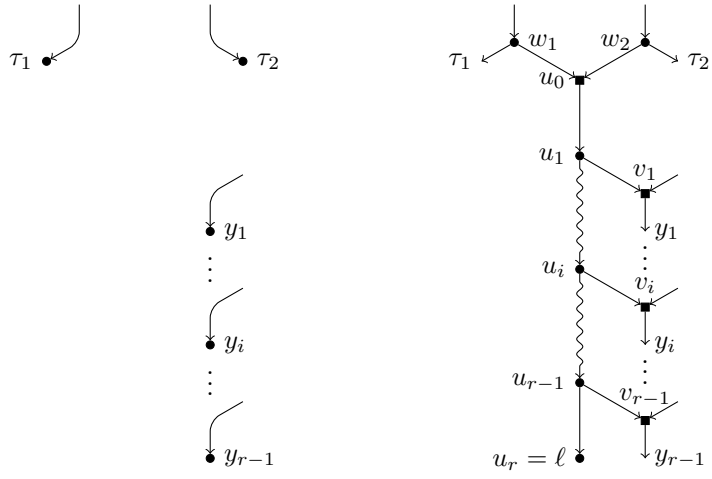

Fig 12. Augmentation operation of type H. Depiction of the augmentation operation $H^{-1}(N, \ell; \{\tau_1, \tau_2\}, (y_1, \ldots, y_{r-1}))$ when $\tau_1 \neq \tau_2$.

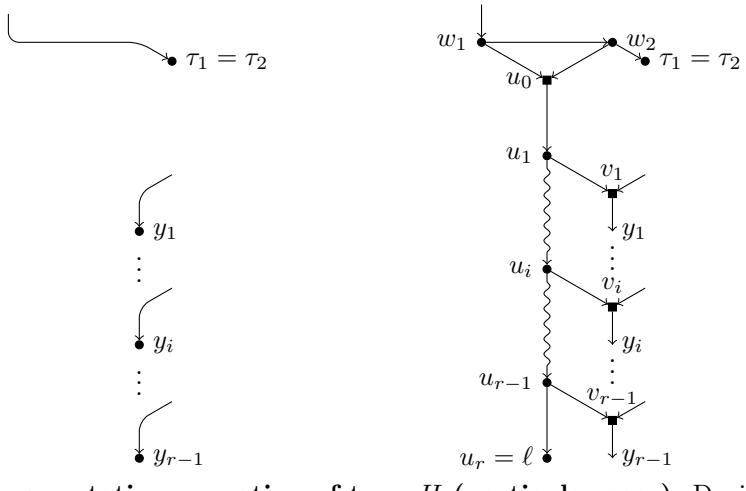

Fig 13. Augmentation operation of type H (particular case). Depiction of the augmentation operation $H^{-1}(N, \ell; \{\tau_1, \tau_2\}, (y_1, \ldots, y_{r-1}))$ when $\tau_1 = \tau_2$.

Proposition 9. *Using the notations of Definition 6, the network*

$$\tilde{N} = H^{-1}(N, \ell; \{\tau_1, \tau_2\}, (y_1, \ldots, y_{r-1}))$$

is a BTC network over $X \cup \{\ell\}$. If N has h hybrid nodes, then $\tilde{N}$ has $h + r$ hybrid nodes.

Proof. The proof is completely analogous to that of Proposition 8, taking into account that one extra hybrid node is created. □

Given a BTC network over X, a label $\ell \notin X$ and a feasible pair (S_1, S_2), in order to unify notations we define the augmented network $R^{-1}(N, \ell; S_1, S_2)$ as $T^{-1}(N, \ell; S_1, S_2)$, if $|S_1| = 1$, and as $H^{-1}(N, \ell; S_1, S_2)$, if $|S_1| = 2$. Also, we shall generically say that the *offspring* of a BTC network is the set of networks that can be obtained from it by means of augmentation operations.

Our next goal is to prove that different augmentation operations applied to a same BTC network or different BTC networks over the same set of taxa provide different networks. We start with the case of different networks.

Proposition 10. *Let $\tilde{N}_1$ and $\tilde{N}_2$ be two BTC networks, both obtained by one augmentation operation applied to two non-isomorphic BTC networks N_1 and N_2 over the same set of taxa X. Then $\tilde{N}_1$ and $\tilde{N}_2$ are not isomorphic.*

Proof. If $\tilde{N}_1$ and $\tilde{N}_2$ have different sets of labels, then it is clear that they are not isomorphic. We can therefore assume that both augmentation operations introduced the same new leaf ℓ. Suppose that $\tilde{N}_1 \simeq \tilde{N}_2$. Then $R(\tilde{N}_1, \ell) \simeq R(N_2, \ell)$. Now, from the definitions of the reductions and augmentations it is straightforward to check that $R(\tilde{N}_i, \ell) = N_i$ and we get that $N_1 \simeq N_2$, a contradiction. □

We treat now the case of applying different augmentation operations to the same BTC network. But first, we give a technical lemma that will be useful in the proof of the proposition.

Lemma 11. *Let N be a BTC network. Then, the identity is the only automorphism (as a leaf-labeled directed graph) of N.*

Proof. Let ϕ be any automorphism of N. Since ϕ is an automorphism of directed graphs and sends each leaf to itself, it follows that $\mu(u) = \mu(\phi(u))$ for each node u of N, where $\mu(u)$ is the μ-vector of u as defined in [10]. Then, by [10, Lemma 5c], it follows that u and $\phi(u)$ are either equal, or one of them is the single child of the other one; according to our definition of BTC networks, this last possibility implies that one of them is a hybrid node and the other one is a tree node, which is impossible if ϕ is an automorphism. Hence $\phi(u) = u$ for every node u. □

Proposition 12. *Let $\tilde{N}_1$ and $\tilde{N}_2$ be two BTC networks, both obtained by one augmentation operation applied to the same BTC network N. If either the kinds of operation or the feasible pairs used to construct $\tilde{N}_1$ and $\tilde{N}_2$ are different, then $\tilde{N}_1$ and $\tilde{N}_2$ are not isomorphic.*

Proof. Let us assume that $\tilde{N}_1$ and $\tilde{N}_2$ are isomorphic. Then, it is clear that they have the same set of labels, and exactly one of them, say ℓ, is not a label of N. Since $\tilde{N}_1$ and $\tilde{N}_2$ are isomorphic, the kind of ℓ is the same in both networks, which implies that the kind of augmentation operations used to construct $\tilde{N}_1$ and $\tilde{N}_2$ are the same. Also, since $\tilde{N}_1$ and $\tilde{N}_2$ are isomorphic, the nodes in the respective recovering data of the reductions $R(\tilde{N}_i, \ell)$ must be linked by an isomorphism of phylogenetic networks. Therefore, and since by Lemma 11 BTC networks do not have a nontrivial automorphism, the respective recovering data must be equal. □

The following proposition shows that the reduction procedure defined in the previous subsection can be reverted using the augmentation operations presented in this subsection.

Proposition 13. *Let N be a BTC network and ℓ a leaf of N. Let $N' = R(N, \ell)$, (S_1, S_2) its recovering data, and $\tilde{N} = R^{-1}(N', \ell; S_1, S_2)$. Then, N and $\tilde{N}$ are isomorphic.*

Proof. It is straightforward to see that the operations T^{-1} and H^{-1} reverse the effects of T and H, respectively. The only points worthy of attention correspond to the cases where the single node in S_1 appears in S_2 (for reductions/augmentations of type T) or where there is a single node in S_1 with multiplicity two (for reductions/augmentations of type H). In the first case, the augmentation process creates two elementary nodes, w_1 and v_i, connected by an arc (w_1, v_i), which is the same situation as in N after the removal of the nodes in $\overline{\mathrm{TH}}(\ell)$. In the second case, two elementary nodes τ_1 and τ_2 are created, connected by an arc, once again the same situation as in N after the removal of the nodes in $\overline{\mathrm{TH}}(\ell)$. □

A direct consequence of the results in this subsection is the following theorem, which can be used to generate in an effective way all BTC networks over a set of taxa. See Fig 14 for an example.

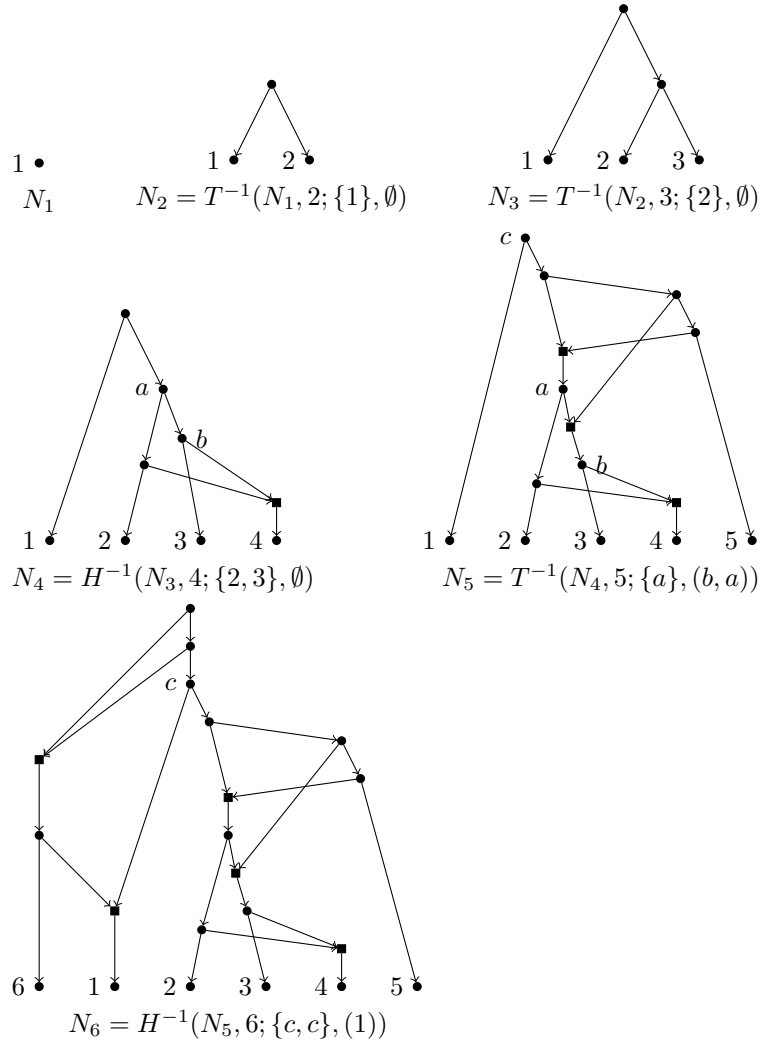

Fig 14. Construction of a BTC network. Example of a chain of augmentation operations that generate a BTC network.

Theorem 14. *Let $N \in \mathcal{BTC}_n$ be a BTC network over $[n]$. Then, N can be constructed from the trivial network in $\mathcal{BTC}_1$ (with one node labeled by 1) by application of $n-1$*

augmentation operations, where at each step i, the leaf $i+1$ is added. Moreover, these augmentation operations are unique.

Proof. The existence is a direct consequence of Corollary 5 and Proposition 13. Unicity comes from Propositions 10 and 12. □

It should be noted that very recently, other methods to generate all BTC networks over a set of taxa have been proposed [6], but, to our knowledge, this is the first time that the networks are generated with unicity. In previous attempts, an isomorphism check was needed after the generation phase.

Bounding the Number of Networks

In this subsection, we shall first give bounds for the number of BTC networks that can be obtained from a given one by means of augmentation operations. This will be done by bounding the number of feasible pairs in such a network. Then, we shall find bounds for the number of BTC networks with a fixed number n of leaves.

Let N be a BTC network over $[n]$ with h hybrid nodes. From Lemma 6 we know that it has $t = 2n + h - 1$ tree nodes, and that $h \leq n - 1$ and $t \leq 3n - 2$. In the following, we shall show how to compute the number of pairs (S_1, S_2) satisfying all conditions of Definition 4, except for Condition 3, via an auxiliary problem. Note that this will only give an upper bound for the number of networks, since the pairs we find can produce networks with cycles.

An Auxiliary Problem

Let $P(N, k)$ be the set of tuples of length k of tree nodes of N such that (1) no pair of them are equal or siblings, and (2) none of them has a hybrid parent or sibling. We indicate the number of such tuples as $p(N, k) = |P(N, k)|$, and since this number will only depend on n, h and k, we indicate it also by $p(n, h, k)$. We consider the problem of computing $p(n, h, k)$.

We compute first how many tree nodes are there that neither have a hybrid parent nor a hybrid sibling. Since the single child of a hybrid node must be a tree node, there are h tree nodes that have a hybrid parent. Note that each hybrid node has two siblings that must be tree nodes; also, a tree node cannot be sibling of two different hybrid nodes; hence, there are $2h$ tree nodes that have a hybrid sibling. Since there cannot be a tree node having the two properties (if it has a hybrid parent, then it does not have any hybrid sibling), there are $3h$ tree nodes that are either a child or a sibling of a hybrid node. Then, the number of tree nodes that neither have a hybrid parent nor a hybrid sibling is $t - 3h = 2n - 2h - 1$. Note that this set of nodes is composed by the root of the network and pairs of tree nodes that are siblings.

Consider now the problem of counting the number of tuples $(y_1, \ldots, y_k)$ in this set that are neither equal nor siblings. We distinguish two cases:

- If none of the nodes y_i is the root of N, we start having $2n - 2h - 2$ choices for y_1, and at each stage the number of choices decreases by two units. Hence, the number of choices is

$$\begin{aligned} p_0(n, h, k) &= (2n - 2h - 2)(2n - 2h - 4) \cdots (2n - 2h - 2k) \\ &= 2^k (n - h - 1)(n - h - 2) \cdots (n - h - k) \\ &= 2^k \frac{(n - h - 1)!}{(n - h - k - 1)!}. \end{aligned}$$

- If one of the nodes y_i is the root of N, then the process of constructing an element in $P(N,k)$ can be described as first choosing at which position i one puts the root, and then filling in the remaining $k-1$ positions with a tuple of the set $P(N,k-1)$ such that none of the nodes is the root (which is what we have just computed). Hence, the number of possibilities is

$$p_1(n,h,k) = k2^{k-1}\frac{(n-h-1)!}{(n-h-k)!}.$$

Then we get that

$$\begin{aligned} p(n,h,k) &= p_0(n,h,k) + p_1(n,h,k) \\ &= 2^k\frac{(n-h-1)!}{(n-h-k-1)!} + k2^{k-1}\frac{(n-h-1)!}{(n-h-k)!} \end{aligned}$$

Counting Pairs Satisfying Conditions 1, 2 and 4H

Consider pairs (S_1, S_2) satisfying Conditions 1, 2 and 4H. Recall that, since condition 4H implies that S_1 and S_2 cannot have elements in common, Conditions 1 and 2 are simplified: no pair of nodes in S_2 can be siblings and none of them can either be the child of a hybrid node or have a hybrid sibling. Hence, the problem is equivalent to finding a tuple $(y_1,\ldots,y_{r-1})$ in $P(N,r-1)$ and then either a tree node τ_1 or an unordered pair $\{\tau_1,\tau_2\}$ of different tree nodes, in either case disjoint from those in $(y_1,\ldots,y_{r-1})$. Once the tuple $(y_1,\ldots,y_{r-1})$ is fixed, the number of tree nodes available for choosing τ_1 and τ_2 is $t-r+1 = 2n+h-r$. Hence, the number of possible pairs is

$$F_H(n,h,r-1) = F_{H,1}(n,h,r-1) + F_{H,2}(n,h,r-1),$$

where

$$\begin{aligned} F_{H,1}(n,h,r-1) &= p(n,h,r-1)\cdot(2n+h-r), \\ F_{H,2}(n,h,r-1) &= p(n,h,r-1)\cdot\frac{1}{2}(2n+h-r)(2n+h-r-1). \end{aligned}$$

Counting Pairs Satisfying Conditions 1, 2 and 4T

The problem now is to count the ways of choosing $S_1 = \{\tau_1\}$ and a tuple $S_2 = (y_1,\ldots,y_{r-1})$ satisfying Conditions 1, 2 and 4T. Now τ_1 can appear in S_2, and different possibilities arise, since it allows that one of the nodes in S_2 has a sibling in S_2, or that it has a hybrid parent or sibling. We consider, thus, these different possibilities:

- $\tau_1 \neq y_i$ (for all i): This case is very similar to the one considered in the previous paragraph, specifically the case where only a single node τ_1 had to be taken. The number of possible pairs is

$$F_{T,1}(n,h,r-1) = p(n,h,r-1)\cdot(2n+h-r).$$

- $\tau_1 = y_i$ is a child or a sibling of a hybrid node: Choosing one of these pairs is equivalent to first choosing the position i, then filling the other $r-2$ positions with a tuple in $P(N,r-2)$, and then choosing a node that is a child or sibling of a hybrid node to be put in the position i. The number of ways to do this procedure is

$$F_{T,2}(n,h,r-1) = p(n,h,r-2)\cdot(r-1)\cdot 3h,$$

since each hybrid node has a single child and two siblings, and none of these $3h$ nodes appears twice, associated to two different hybrid nodes.

- $\tau_1 = y_i$ is a sibling of some other node y_j in S_2: In this case one has to choose the positions i and j where to put the pair of sibling nodes, fill the other $r-3$ positions with a tuple in $P(N, r-3)$, choose a pair of sibling tree nodes to take as y_i and y_j, and finally set $\tau_1 = y_i$. The choice of i and j can be done in $(r-1)(r-2)$ different ways. The choice of the tuple of length $r-3$ can be done in $p(n,h,r-3)$ ways; $p_1(n,h,r-3)$ of them contain the root of N (and $r-4$ tree nodes with a sibling tree node) and $p_0(n,h,r-3)$ do not contain the root (and contain $r-3$ tree nodes with a sibling tree node). Once this is done, the number of available pairs of sibling tree nodes is $n-h-1-(r-4) = n-h-r+3$, if the root of N was chosen, or $n-h-1-(r-3) = n-h-r+2$ otherwise. Hence, the total number of pairs is $F_{T,3}(n,h,r-1) = F_{T,3,A}(n,h,r-1) + F_{T,3,B}(n,h,r-1)$, corresponding to these two cases, with:

$$F_{T,3,A}(n,h,r-1) = (r-1)(r-2) \cdot p_1(n,h,r-3) \cdot (2n-2h-2r+6),$$
$$F_{T,3,B}(n,h,r-1) = (r-1)(r-2) \cdot p_0(n,h,r-3) \cdot (2n-2h-2r+4).$$

- $\tau_1 = y_i$ but none of the previous conditions hold: In this case one only has to take a tuple in $P(N, r-1)$ and choose which of the $r-1$ nodes to take as τ_1. The number of possible pairs is then

$$F_{T,4}(n,h,r-1) = p(n,h,r-1) \cdot (r-1).$$

Note that the four conditions above are mutually exclusive. Hence, the overall number of possible pairs (S_1, S_2) is the sum of all numbers found:

$$F_T(n,h,r-1) = F_{T,1}(n,h,r-1) + F_{T,2}(n,h,r-1) + F_{T,3}(n,h,r-1) + F_{T,4}(n,h,r-1).$$

Bounds for the Number of Networks

Each network $N \in \mathcal{BTC}_n$ with h hybrid nodes, appears as augmentation $R^{-1}(N', n, S_1, S_2)$ of a unique network $N' \in \mathcal{BTC}_{n-1}$ with h' hybrid nodes, where S_2 has length $r-1 = h-h'$, if the augmentation is of type T, or $r-1 = h-h'-1$ if it is of type H. If we call $B(n,h)$ the number of networks in $\mathcal{BTC}_n$ with h hybrid nodes, and since we have bounded the number of feasible pairs, we have that

$$B(n,h) \leq \sum_{h'=0}^{h} B(n-1,h') \cdot F_T(n-1,h',h-h') + \\ + \sum_{h'=0}^{h-1} B(n-1,h') \cdot F_H(n-1,h',h-h'-1)$$

Also, since the number of hybrid nodes in a BTC network with n leaves is at most $n-1$, we have that

$$|\mathcal{BTC}_n| = \sum_{h=0}^{n-1} B(n,h),$$

and the expression above allows us to compute a bound for this number of networks. See Subsection Computational Experiments for an experiment with these bounds.

The asymptotic formula $|\mathcal{BTC}_n| = 2^{2n \log n + O(n)}$ is given in [19], and both our experimental results in Subsection Computational Experiments for $n \leq 8$ and the bounds that we have computed for $n \leq 10$ are coherent with this expression. However, the problem of finding a closed expression for the asymptotic behaviour of our bounds is still open.

An Application to Phylogenetic Reconstruction

Several models of reticulate evolution on biological sequences have been proposed in the last decades, for example the displayed trees model [25], an extension of the multispecies coalescent (MSC) to phylogenetic networks [26] and the ancestral recombination graph model –ARG for short [27]– to only name a few. The associated problems are difficult to solve and big efforts have been done by the community to provide practitioners with fast algorithms.

Suppose we are given a BTC network N over a set of OTUs X, where each tree node is associated with a word in an alphabet (for instance a DNA sequence) $s(u) \in \Sigma^*$. The pair (N, s) can, for example, be the outcome of an ML search in the space of BTC networks given an alignment over X. Now, suppose we are given a new sequence and we want to update N to include it, ensuring that the resulting network is still BTC. We may want to do this, for instance, to update the network without redoing the whole ML search, or in a phylogenetic placement perspective (for example, we want to know where to place a given strain of a virus in N), or even because we use a heuristic algorithm that reconstructs a network by adding one sequence at the time.

We assume that a model of evolution is given, and we assume that we can compute the following probabilities:

1. Given $s, s' \in \Sigma^*$, $P_S(s, s')$ is the probability that the sequence s evolved by descent with modification giving as a result the sequence s'.

2. Given $s_1, s_2, s' \in \Sigma^*$, $P_H(s_1, s_2, s')$ is the probability that a hybridization between sequences s_1 and s_2 –possibly coupled with descent with modification– gives as result the sequence s'.

For each tree node t of N, we let $\phi_t : \Sigma^* \to [0, 1]$ be the function defined as follows. If t is the root of N, then ϕ_t is the constant function equal to 1. Otherwise, if the single parent p of t is a tree node, then $\phi_t(s) = P_S(s(p), s)$. If p is a hybrid node with parents g_1, g_2, then $\phi_t(s) = P_H(s(g_1), s(g_2), s)$. That is, $\phi_t(s)$ is the probability that a given sequence s is the result of the evolution of the sequences at the parent node (or grandparents, in case of hybrid parent) of t.

Now, we want to extend N to another BTC network in order to include an extant OTU $\ell \notin X$ identified by its sequence $s_\ell \in \Sigma^*$, while keeping the sequences associated to all tree nodes of N. According to the results presented in this paper, we need to identify the augmentation operation $R^{-1}(N, \ell; S_1, S_2)$ that has to be applied, and determine the sequences at the newly created tree nodes. If the operation to be applied is of type T, that is, $\tilde{N} = T^{-1}(N, \ell; S_1, S_2)$, we need to find certain nodes $\tau_1, y_1, \ldots, y_{r-1}$, with the additional condition that $S_1 = \{\tau_1\}$ and $S_2 = (y_1, \ldots, y_{r-1})$ form a T-feasible pair. Analogously, if it is of type H, $\tilde{N} = H^{-1}(N, \ell; S_1, S_2)$, then $S_1 = \{\tau_1, \tau_2\}$ and $S_2 = (y_1, \ldots, y_{r-1})$ must form a H-feasible pair.

Intuitively, the node τ_1 in case of an augmentation of type T, or the nodes τ_1 and τ_2 in case of type H, have to be chosen in order to maximize the probability of appearance of the new OTU, while the other nodes appear in order to give a better explanation of the corresponding sequences by means of hybridization with the lineage leading to ℓ.

We present here an heuristic to find the augmentation operation, together with the assignment of sequences to new tree nodes, that deploys this intuitive idea:

1. Assume that an augmentation of type T is going to take place. To determine τ_1, for each tree node t of N, we find a sequence $\sigma(t) \in \Sigma^*$ that maximizes

$$\pi(t) = \phi_t(\sigma(t)) \cdot P_S(\sigma(t), s(t)) \cdot P_S(\sigma(t), s_\ell).$$

   The rationale behind this expression is that we look for the best way to divide the arc entering t to add the new taxon with sequence s_ℓ as child of the newly created

node. See Fig 15(left) for a depiction of this. Then τ_1 is a node with the maximum value of π over all nodes of N, that is the best location where to hang s_ℓ in N. For future reference, let $\sigma^T = \sigma(\tau_1)$, $\pi^T = \pi(\tau_1)$ and $S_1^T = \{\tau_1\}$.

2. Assume that an augmentation of type H is going to take place. To determine τ_1, τ_2, for each unordered pair of tree nodes $\{t_1, t_2\} := \boldsymbol{t}$ of N, we find sequences $\sigma^1(\boldsymbol{t}), \sigma^2(\boldsymbol{t}) \in \Sigma^*$ that maximize

$$\pi(\boldsymbol{t}) = \phi_{t_1}(\sigma^1(\boldsymbol{t}))\cdot P_S(\sigma^1(\boldsymbol{t}), s(t_1))\cdot \phi_{t_2}(\sigma^2(\boldsymbol{t}))\cdot P_S(\sigma^2(\boldsymbol{t}), s(t_2))\cdot P_H(\sigma^1(\boldsymbol{t}), \sigma^2(\boldsymbol{t}), s_\ell)$$

See Fig 15(right). The rationale for this choice is the same as in the previous point.

Let $\boldsymbol{\tau} = \{\tau_1, \tau_2\}$ be a pair with maximum value of π. For future reference, let $\sigma^H = (\sigma_1^H, \sigma_2^H) = (\sigma^1(\boldsymbol{\tau}), \sigma^2(\boldsymbol{\tau}))$, $\pi^H = \pi(\boldsymbol{\tau})$ and $S_1^H = \boldsymbol{\tau}$.

3. If $\pi^T \geq \pi^H$, we opt for an augmentation of type T and we let $S_1 = S_1^T$; otherwise, we opt for an augmentation of type H and we let $S_1 = S_1^H$. The tree nodes in $R^{-1}(N, \ell; S_1, \emptyset)$ that were present in N keep their sequences. Moreover, in case of an augmentation of type T, we subdivide the arc entering τ_1 via a new node w_1 associated to the sequence σ^T and we add a new leaf ℓ with sequence s_ℓ as child of w_1. In case of an augmentation of type H, we subdivide the arcs entering τ_1 and τ_2 via two new nodes w_1, w_2 that are assigned to the two sequences σ_1^H, σ_2^H; then, we add a new hybrid node with parents w_1 and w_2 and having as child a new leaf ℓ with sequence s_ℓ.

4. For each $k \geq 1$, we assume that $y_1, \ldots, y_{k-1}$ are already determined. Let C be the set of tree nodes y such that $(S_1, (y_1, \ldots, y_{k-1}, y))$ is a feasible pair. To determine y_k we proceed as follows:

   - If $k = 1$ and we opted for a type H augmentation, for each $y \in C$ with parent[2] p, we find a sequence $\sigma(y)$ that maximizes

   $$\pi(y) = P_H(\sigma_1^H, \sigma_2^H, \sigma(y)) \cdot P_H(\sigma(y), s(p), s(y)) \cdot P_S(\sigma(y), s_\ell).$$

   See Fig 16(left). If $\kappa(y) = \pi(y) - P_S(s(p), s(y))\ P_H(\sigma_1^H, \sigma_2^H, s_\ell)$ is negative, we remove y from C since this means that the hypothesis of the existence of a hybridization just above y is less likely than its absence.

   - If $k > 1$ or we opted for a type T augmentation, for each $y \in C$ with parent p, we find a sequence $\sigma(y)$ that maximizes

   $$\pi(y) = P_S(\sigma(y_{k-1}), \sigma(y)) \cdot P_H(\sigma(y), s(p), s(y)) \cdot P_S(\sigma(y), s_\ell),$$

   where in case that $k = 1$ (and hence we opted for a type T augmentation), we let $y_0 = \tau_1$. See Fig 16(right). If $\kappa(y) = \pi(y) - P_S(s(p), s(y))P_S(\sigma(y_{k-1}), s_\ell)$ is negative, we remove y from C as in the previous case.

   Then, if C is empty we output the network $\tilde{N} = R^{-1}(N, \ell; S_1, (y_1, \ldots, y_{k-1}))$. Otherwise, let y_k the node that maximizes κ. Then, we create a new tree node u_k on the arc entering ℓ –associating the sequence $\sigma(y_k)$ to it– and we subdivide the arc entering y_k by a new hybrid node with second parent u_k. The tree nodes in $R^{-1}(N, \ell; S_1, (y_1, \ldots, y_{k-1}, y_k))$ that appeared in $R^{-1}(N, \ell; S_1, (y_1, \ldots, y_{k-1}))$ keep their associated sequence.

[2]Here we assume that p is a tree node. If p was a hybrid node (and this case is really exceptional because of the definition of feasible pair) then the following computation (and the one in the next item) should be adapted.

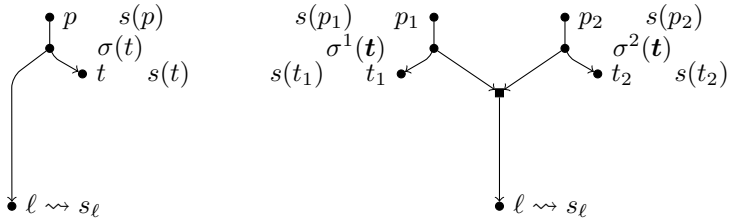

Fig 15. A depiction of the notations used in the text to define the function π to maximize for finding τ_1 (left, type T augmentation) and $\{\tau_1, \tau_2\}$ (right, type H augmentation). Although in the figure p, p_1 and p_2 are depicted as tree nodes, they can as well be hybrid nodes.

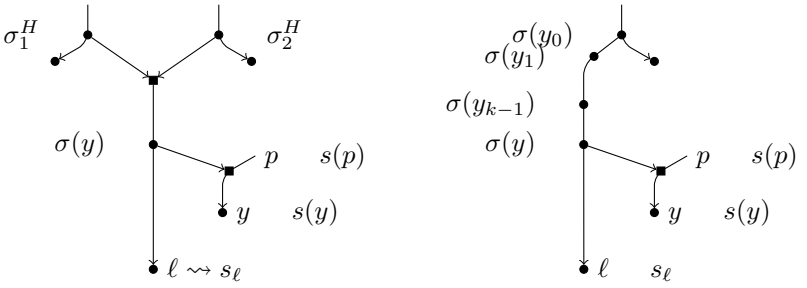

Fig 16. A depiction of the notations used in the text to define the function π to maximize for finding y_k, respectively for a type H augmentation (left) and a type T augmentation (right), assuming that $y_1, \ldots, y_{k-1}$ are already determined.

We emphasize that we do not claim that the heuristic we present here gives a global optimum. In fact, usually a sequence of optimal choices does not lead to a global optimum. The analysis, and eventually improvement, of this method of reconstruction is left as future work.

Example 1. We consider a simple model of evolution where:

- OTUs are represented by words of length 4 in the alphabet $\Sigma = \{\mathrm{A}, \mathrm{B}, \mathrm{C}\}$.
- For speciation we assume a simple Jukes-Cantor model of evolution on the characters A, B, C so that $P_S(s, s') = \mu^d(1-2\mu)^{4-d}$, where $d = d(s, s')$ is the Hamming distance between s and s' and $\mu < 1/3$ is a parameter of the model.
- In this toy example, we model hybridizations as if they were plain recombinations where half of the hybrid sequence comes from one parent and the other half from the other. So, given two sequences $s^1 = (s^1_1, s^1_2, s^1_3, s^1_4)$ and $s^2 = (s^2_1, s^2_2, s^2_3, s^2_4)$, $P_H(s^1, s^2, s') = 1/2$ if $s' = (s^1_1, s^1_2, s^2_3, s^2_4)$ or $s' = (s^2_1, s^2_2, s^1_3, s^1_4)$, and $P_H(s^1, s^2, s') = 0$ otherwise.

We consider three species, with sequences

$$s(\alpha) = \mathrm{AAAC}, \qquad s(\beta) = \mathrm{BBCC}, \qquad s(\gamma) = \mathrm{BBBB}.$$

The network N (which is in fact a tree) that fits these extant OTUs best, together with an optimal assignment of sequences to all nodes is shown in Fig 17(left).

Now, we wish to extend N in order to add a new OTU with sequence

$$s(\delta) = \mathrm{AACC}.$$

We thus proceed as discussed in the previous pages:

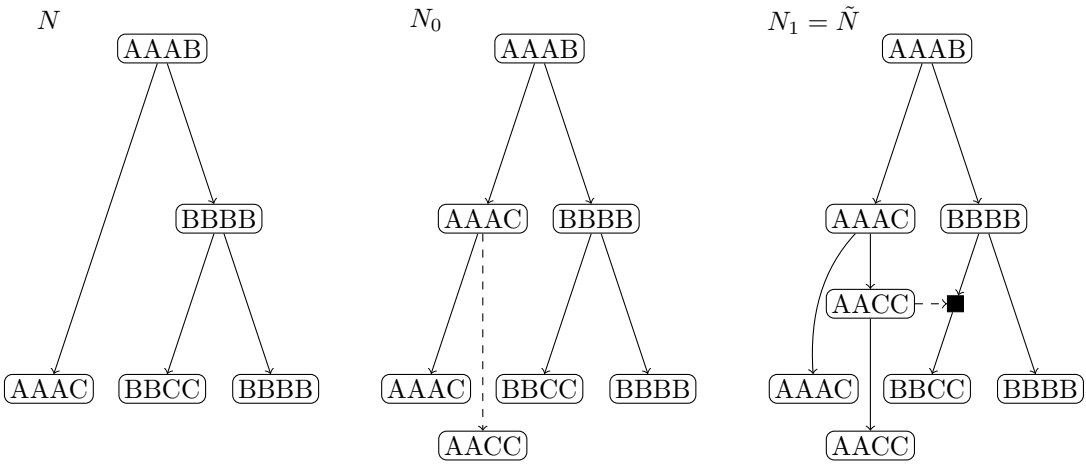

Fig 17. The networks considered in Example 1. We start with a tree on three leaves (left), chose a type T augmentation and find the best choice for τ_1 (middle) and the best choice for the vector $(y_1, \ldots, y_k)$ (right, here $k = 1$).

1. If we assume an augmentation of type T, it is not difficult to see that an optimal choice for τ_1 and its corresponding sequence σ^T are respectively α and AAAC, with $\pi(\alpha) = \mu^2(1-2\mu)^{10}$.

2. If we explore the different possibilities for an augmentation of type H, we find that the best choice is $\{\tau_1, \tau_2\} = \{\alpha, \beta\}$ with sequences AAAB and BBCC and with value $\pi(\{\tau_1, \tau_2\}) = \mu^3(1-2\mu)^{13}/2$.

3. Since $\mu < 1/3$, we get that $\pi(\alpha) > \pi(\{\alpha, \beta\})$ and we opt for the augmentation $N_0 = T^{-1}(N, \delta; \{\alpha\}, \emptyset)$ shown in Fig 17(middle).

4. Now we want to find the right choice for y_1, if any. All nodes except the least common ancestor (in N_0) of α and δ are in C. Taking any node $y \in C$ different from β we get that $\kappa(y) < 0$ and hence they are not good candidates. If we take $y = \beta$, taking into account that $\mu < 1/3$, we find an optimal sequence $\sigma(\beta) = \text{AACC}$. The value of $\pi(\beta)$ corresponds to three different evolution processes: two mutations, from AAAC to AACC and from AACC to AACC, and a hybridization of the sequences AACC and BBBB to BBCC, and hence we get $\pi(\beta) = (1-2\mu)^7\mu/2$. Now, this value must be compared with the probability of evolution without this hybridization, i.e. the probability of speciations from BBBB to BBCC and from AAAC to AACC which is $\mu^3(1-2\mu)^5$. Since $(1-2\mu)^7\mu/2 > \mu^3(1-2\mu)^5$ (assuming that $\mu < 0.2928$, which is a reasonable assumption) we conclude that the network $N_1 = T^{-1}(N, \delta; \{\alpha\}, (\beta))$, depicted in Fig 17(right), is a better explanation than N_0. Hence, we let $y_1 = \beta$.

5. If we repeat the procedure in the previous step, we find that no hybridization improves the probability of the sequences, giving as final result the network $\tilde{N} = T^{-1}(N, \delta; \{\alpha\}, (\beta))$ in Fig 17(right).

Computational Experiments

The algorithms in this paper have been implemented in python using the python library PhyloNetworks [28]. This implementation, together with the sources for the experiments that we comment in this subsection can be downloaded from `https://github.com/bielcardona/TCGenerators`.

Exhaustive and sequential construction of networks in $\mathcal{BTC}_n$. We have implemented both the exhaustive and sequential construction of BTC networks with n leaves. The number of such networks increases very rapidly, and hence the exhaustive construction is not feasible for $n \geq 8$. For $n \leq 7$ we generated all the networks in $\mathcal{BTC}_n$; see Table 1 for the number of such networks. For $n = 8$ we could not compute them all, since there are around twelve trillion of such networks: We took uniform samples of networks in $\mathcal{BTC}_7$ and computed their respective offspring, and repeated this procedure until the average number of offsprings per network stabilized up to 4 digits; this allowed us to give the estimate for $|\mathcal{BTC}_8|$.

n	$\lvert\mathcal{BTC}_n\rvert$	upper bound
1	1	1
2	3	3
3	66	85
4	4,059	7,442
5	496,710	1,317,098
6	101,833,875	387,405,870
7	31,538,916,360	169,781,857,790
8	$\simeq$ 12,000,000,000,000	103,409,407,515,286
9	?	83,400,205,845,281,275
10	?	85,947,517,732,640,544,027

Table 1. Exact number of BTC networks over $[n]$ for $n = 1, \ldots, 7$, an estimate for $n = 8$, and their upper bounds for $n \leq 10$.

Random construction of networks in $\mathcal{BTC}_n$. We have implemented the following construction, that does not generate networks uniformly, but is the closest we could get to it. We start with the network N_1 with a single node labeled by 1. At each stage $i = 1, \ldots, n-1$, we explicitly find all feasible pairs inside N_i and choose at random and uniformly one of them to generate the network N_{i+1}. This procedure generates all possible networks in $\mathcal{BTC}_n$, but not uniformly, since different networks over the same set of taxa may have different numbers of feasible pairs.

Computation of bounds for $|\mathcal{BTC}_n|$. Finally, we have implemented the recursive computation for the upper bounds of $|\mathcal{BTC}_n|$ using the bounds for the offsprings of BTC networks found previously. The results for n up to 10 are given in Table 1, where it is observed that, at least for small values of n, the true number of networks and the upper bounds have the same order of magnitude.

Conclusion

The main result of this paper is a systematic way of recursively generating, with unicity, all BTC networks with a given number of leaves. This procedure relies on a pair of reduction/augmentation operations that generalize analogous operations for phylogenetic trees. Indeed, given a (rooted, binary) phylogenetic tree over $[n]$, we can obtain a phylogenetic tree over $[n-1]$ by deleting the leaf labeled by n and removing the elementary node that this deletion generates. Conversely, given a tree T over $[n-1]$ and one of its nodes u, we can construct a tree over $[n]$ by simply hanging a pendant leaf labeled by n to the single incoming arc of u. Since different choices for T and u give different trees over $[n]$, this gives a recursive procedure to generate, with unicity, all binary rooted phylogenetic trees over a given set of taxa: we start with the leaf labeled

by 1, then we add the leaf labeled by 2, then the leaf labeled by 3 in all possible ways, and so on. Biologically, we can think of this procedure as follows: Once the evolutionary history of a given set of OTUs is correctly established[3] and modeled by a phylogenetic tree, extending this evolutionary history to consider a "new" OTU n consists in finding where to place n in the tree, i.e. finding the speciation event that leads to the diversification of n.

Unfortunately, when working with classes of phylogenetic networks, the removal of a single leaf (and of all elementary nodes created by this removal) does not necessarily give a phylogenetic network within the same class. In the case of BTC networks, we were able to find the minimal set of nodes that one must remove so that, after their deletion and that of all elementary nodes created by this removal, one gets a BTC network with one leaf less. As in the case of trees, given a BTC network over $[n-1]$ and some set of nodes with certain restrictions (i.e. the feasible pairs S_1 and S_2) we can construct a BTC network over $[n]$ leaves, in such a way that different choices for the BTC network or for the feasible pair give different BTC networks over $[n]$. Hence, we find a procedure to recursively generate all BTC networks over a given set of taxa. Biologically, we can think of this procedure as an extension of what can happen when adding a new OTU n to a phylogenetic tree: here the diversification of n can involve a reticulated event (when n is added as hybrid node) and the ancestors of n participate to $|S_2|$ reticulated events, which were impossible to detect before the introduction of n.

Acknowledgments

The authors would like to thank the anonymous reviewers and the editors for their comments, that greatly improved the quality of the manuscript.

References

1. Gambette P, van Iersel L, Jones M, Lafond M, Pardi F, Scornavacca, Celine Rearrangement Moves on Rooted Phylogenetic Networks. PLoS Computational Biology 2017;8(13):e1005611.

2. Maddison WP. Gene Trees in Species Trees. Systematic Biology. 1997;46(3):523–536.

3. Scornavacca C, Huson DH. A Survey of Combinatorial Methods for Phylogenetic Networks. Genome Biology and Evolution. 2010;3:23–35. doi:10.1093/gbe/evq077.

4. Semple C, Baroni M, Steel M. Hybrids in Real Time. Systematic Biology. 2006;55(1):46–56. doi:10.1080/10635150500431197.

5. Baroni M, Semple C, Steel M. A Framework for Representing Reticulate Evolution. Annals of Combinatorics. 2005;8(4):391–408. doi:10.1007/s00026-004-0228-0.

6. Erdos PL, Semple C, Steel M. A class of phylogenetic networks reconstructable from ancestral profiles; 2019. Available from: `https://arxiv.org/abs/1901.04064v1`.

7. Gusfield D, Eddhu S, Langley C. Efficient Reconstruction of Phylogenetic Networks with Constrained Recombination. In: Proceedings of the IEEE Computer Society Conference on Bioinformatics. CSB '03. Washington, DC, USA:

[3]In practice, we can never be sure that we got the correct tree, but here we suppose we do.

IEEE Computer Society; 2003. p. 363–. Available from: http://dl.acm.org/citation.cfm?id=937976.938101.

8. Huson DH, Klöpper TH. Beyond Galled Trees - Decomposition and Computation of Galled Networks. In: Speed T, Huang H, editors. Research in Computational Molecular Biology. Berlin, Heidelberg: Springer Berlin Heidelberg; 2007. p. 211–225. Available from: https://doi.org/10.1007/978-3-540-71681-5_15.

9. van Iersel L, Kelk S. Constructing the Simplest Possible Phylogenetic Network from Triplets. Algorithmica. 2011;60(2):207–235. doi:10.1007/s00453-009-9333-0.

10. Cardona G, Rosselló F, Valiente G. Comparison of Tree-Child Phylogenetic Networks. IEEE/ACM Transactions on Computational Biology and Bioinformatics. 2009;6(4):552–569. doi:10.1109/TCBB.2007.70270.

11. Francis AR, Steel M. Which phylogenetic networks are merely trees with additional arcs? Systematic biology. 2015;64(5):768–777.

12. Cardona G, Pons JC, Rosselló F. A reconstruction problem for a class of phylogenetic networks with lateral gene transfers. Algorithms for Molecular Biology. 2015;10(1):28. doi:10.1186/s13015-015-0059-z.

13. van Iersel L, Semple C, Steel M. Locating a tree in a phylogenetic network. Information Processing Letters. 2010;110(23):1037 – 1043. doi:https://doi.org/10.1016/j.ipl.2010.07.027.

14. van Iersel L, Moulton V. Trinets encode tree-child and level-2 phylogenetic networks. Journal of Mathematical Biology. 2014;68(7):1707–1729. doi:10.1007/s00285-013-0683-5.

15. Semple C. Phylogenetic Networks with Every Embedded Phylogenetic Tree a Base Tree. Bulletin of Mathematical Biology. 2016;78(1):132–137. doi:10.1007/s11538-015-0132-2.

16. Bordewich M, Semple C, Tokac N. Constructing Tree-Child Networks from Distance Matrices. Algorithmica. 2018;80(8):2240–2259. doi:10.1007/s00453-017-0320-6.

17. Pardi F, Scornavacca C. Reconstructible phylogenetic networks: do not distinguish the indistinguishable. PLoS computational biology. 2015;11(4):e1004135.

18. Gunawan AD, Rathin J, Zhang L. Counting and Enumerating Galled Networks. arXiv e-prints. 2018; p. arXiv:1812.08569.

19. McDiarmid C, Semple C, Welsh D. Counting Phylogenetic Networks. Annals of Combinatorics. 2015;19(1):205–224. doi:10.1007/s00026-015-0260-2.

20. Fuchs M, Gittenberger B, Mansouri M. Counting Phylogenetic Networks with Few Reticulation Vertices: Tree-Child and Normal Networks. arXiv e-prints. 2018; p. arXiv:1803.11325.

21. Cardona G, Llabrés M, Rosselló F, Valiente G. A distance metric for a class of tree-sibling phylogenetic networks. Bioinformatics. 2008;24(13):1481–1488. doi:10.1093/bioinformatics/btn231.

22. Cardona G, Llabrés M, Rosselló F, Valiente G. Metrics for phylogenetic networks II: Nodal and triplets metrics. IEEE/ACM Transactions on Computational Biology and Bioinformatics. 2009;6(3):454–469. doi:10.1109/TCBB.2008.127.

23. Oldman J, Wu T, Van Iersel L, Moulton V. Trilonet: piecing together small networks to reconstruct reticulate evolutionary histories. Molecular biology and evolution. 2016;33(8):2151–2162.

24. Huber KT, Van Iersel L, Moulton V, Scornavacca C, Wu T. Reconstructing phylogenetic level-1 networks from nondense binet and trinet sets. Algorithmica. 2017;77(1):173–200.

25. Jin G, Nakhleh L, Snir S, Tuller T. Maximum likelihood of phylogenetic networks. Bioinformatics. 2006;22(21):2604–2611.

26. Meng C, Kubatko LS. Detecting hybrid speciation in the presence of incomplete lineage sorting using gene tree incongruence: a model. Theoretical population biology. 2009;75(1):35–45.

27. Gusfield D. ReCombinatorics: the algorithmics of ancestral recombination graphs and explicit phylogenetic networks. MIT press; 2014.

28. Cardona G, Sánchez D. PhyloNetworks: A Python library for phylogenetic networks; 2012. Available from: `https://pypi.org/project/phylonetwork/`.