Combining Networks Using Cherry Picking Sequences

Janssen, Remie; Jones, Mark; Murakami, Yukihiro

DOI
10.1007/978-3-030-42266-0_7

Publication date
2020

Document Version
Accepted author manuscript

Published in
Algorithms for Computational Biology

Combining Networks using Cherry Picking Sequences*

Remie Janssen[0000−0002−5192−1470]1, Mark Jones[0000−0002−4091−7089]2, and Yukihiro Murakami[0000−0003−1355−5884]1

[1] Delft Institute of Applied Mathematics, Delft University of Technology, Van Mourik Broekmanweg 6, 2628 XE Delft, The Netherlands {R.Janssen-2, Y.Murakami}@tudelft.nl
[2] Centrum Wiskunde & Informatica, P.O.Box 94079, 1090 GB Amsterdam, The Netherlands markelliotlloyd@gmail.com

Abstract. Phylogenetic networks are important for the study of evolution. The number of methods to find such networks is increasing, but most such methods can only reconstruct small networks. To find bigger networks, one can attempt to combine small networks. In this paper, we study the Network Hybridization problem, a problem of combining networks into another network with low complexity. We characterize this complexity via a restricted problem, Tree-child Network Hybridization, and we present an FPT algorithm to efficiently solve this restricted problem.

Keywords: Phylogenetic networks, Network hybridization, Tree-child networks, FPT algorithms

1 Introduction

Evolutionary histories are often represented by phylogenetic trees, and more recently, by phylogenetic networks. Knowing the evolutionary history of a species is vital for understanding their biology. Therefore, it is important to have methods for finding phylogenetic networks that accurately represent the true evolutionary histories. Many methods exist to find evolutionary histories; some are purely combinatorial, others have a likelihood component as well. Here, we focus mainly on the purely combinatorial problems.

One classic combinatorial method is to solve Hybridization: given a set of trees, find the simplest network that displays these trees [1]. Unfortunately, the problem is NP-hard, even on inputs of two trees [2]. For this problem, it is assumed we can construct accurate phylogenetic trees for small parts of the genomes of the studied taxa. When the input consists of only two or three trees, it can be solved relatively efficiently—EPT time [17, 8]—even though the problem is already NP-hard in that case. For an input consisting of three trees or more, there is still an FPT algorithm [9], but it is not practical. In these cases, it is useful to limit the search space to networks with a restricted structure, such as tree-child networks [7], or temporal networks [6].

Another combinatorial approach for finding phylogenetic networks is to combine smaller networks. The smaller networks are often assumed to have certain properties. For example, it may be assumed that we are given all strict subnetworks containing the full set of leaves. In that case, it is possible to reconstruct a level-k tree-child network from all its level-$(k-1)$ subnetworks [15]. Another assumption could be that the input consists of all subnetworks obtained by removing exactly one leaf [11]. Instead of having almost all leaves, the subnetworks can also be allowed to have only few leaves. For example, low level networks can be reconstructed from their full set of binets [5, 12], trinets [16, 10] or quarnets [4].

In practice, it may not be easy to find *all* subnetworks. This renders many of the previously mentioned methods useless. Furthermore, these methods typically only work for low level networks. This means that they cannot be used when the phylogenetic signal comes from a complicated evolutionary history, or if there is some randomness in the data, complicating the data as well.

In this paper, we combine networks that all contain the full set of leaves, but we do not assume we have all the subnetworks of the network we want to find. The problem we solve is analogous to Hybridization, but with networks as the input, Network Hybridization: Given a set of

* Research funded by the Netherlands Organization for Scientific Research (NWO), with the Vidi grant 639.072.602.

networks with taxa X, find a network N with minimal reticulation number, that displays all input networks. Since this is a generalization of the Hybridization problem, the problem remains NP-hard in general, even for inputs of two networks. We show that for the restricted problem on tree-child (topologically restricted class of networks; defined formally in Section 2) inputs and output, we can use tree-child sequences to obtain an FPT algorithm. This FPT algorithm is an extension of the one introduced in [7] in which they considered tree inputs; the tree-child sequence approach was first introduced in [14]. We also comment briefly on how some measure of an optimal solution to the Network Hybridization problem can be characterized by solving this restricted problem.

Structure of the paper We start with a quick introduction of relevant concepts from mathematical phylogenetics in Section 2. Then, in Section 3, we formally introduce Tree-child Network Hybridization and prove its relation to tree-child sequences. This section also relates this problem to the more general Network Hybridization. In Section 4.1, we lay the theoretical foundation to extend the algorithm in [7] that takes inputs of trees to also work for inputs of networks. As a last theoretical section in the paper, we present an FPT algorithm that solves Tree-Child Network Hybridization (Section 4.2). We conclude the paper with a discussion, including some open questions (Section 5).

2 Preliminaries

The main objects of study for this paper are phylogenetic networks. These graphs are used in biology to represent evolutionary scenarios for a given set of species.

Definition 1. *A* (rooted phylogenetic) network *on a finite set of* taxa X *is a directed acyclic graph with*

- *one node of indegree-0 and outdegree-1, the* root*;*
- *nodes of indegree-1 and outdegree-0 labelled bijectively with X, the* leaves*;*
- *nodes of indegree-1 and outdegree-2, the* tree nodes*;*
- *nodes of indegree greater than 1 and outdegree-1, the* reticulations*.*

If all the reticulation nodes have indegree-2, the network is called *binary*. An edge uv is called a *tree edge* if v is a tree node or leaf, and a *reticulation edge* if v is a reticulation. The vertex u is the *parent* of v, and v is the *child* of u. The *reticulation number* $r(N)$ of a network N is the total number of reticulation edges minus the total number of reticulations.

A network is *stack-free* if every reticulation has a child that is a tree node or a leaf. A network is *tree-child* if it is stack-free and every tree node has a child that is a tree node or a leaf. We now define some relevant notation for local structures in phylogenetic networks.

Definition 2. *Let N be a network on X and $x, y \in X$ two leaves. Then we say N has a* cherry $\{x, y\}$ *if the parent of x is the parent of y; we say that N has a* reticulated cherry (x, y) *if the parent of x is a reticulation, and the parent of y is a parent of this reticulation. If (x, y) is a cherry or a reticulated cherry in N, then it is called a* reducible pair*.*

Tree-child sequences are built on the notion of *reducing* cherries and reticulated cherries from networks.

Definition 3. *Let N be a network on X, and (x, y) a pair of leaves. Let p_x and p_y denote the parents of x and y in N, respectively Then* reducing (x, y) *in N results in a network $N(x, y)$ obtained as follows:*

- *If $\{x, y\}$ is a cherry in N, remove x and the pendant edge $p_x x$, and suppress p_x if it has become a degree-2 node;*
- *If (x, y) is a reticulated cherry in N, remove the reticulation edge $p_y p_x$ and suppress p_x or p_y if it has become a degree-2 node.*
- *Otherwise, $N(x, y) := N$.*

The reversal of reducing cherries and reticulated cherries can be done by *adding* ordered pairs of leaves to the network.

Definition 4. *Let N be a network and let (x, y) be reducible pair. Then we may* construct *N from $N(x, y)$—also called* add *(x, y) to $N(x, y)$—by applying the following.*

1. *If x is a leaf in $N(x, y)$ (i.e., if (x, y) is a reticulated cherry in N), and*
   (a) *if p, the parent of x in $N(x, y)$, is a reticulation then add a node q directly above y, and add an edge qp.*
   (b) *otherwise, add nodes p and q directly above x and y respectively, and add an edge qp.*
2. *If x is not a leaf in $N(x, y)$ (i.e., if (x, y) is a cherry in N) then add a labelled node x, insert a node q directly above y, and add an edge qx.*

The above notion of adding an ordered pair of leaves (x, y) to a network N is well-defined if y is already a leaf in N. If this is indeed the case, we may obtain a network from a sequence of ordered pairs by iteratively adding ordered pairs to an existing network. To do so, we impose the following condition on our sequence of ordered pairs: *The second coordinate of each pair has to occur as a first coordinate in the remainder of the sequence, or as the second coordinate of the last pair.* Then, the following procedure constructs a network from a sequence.

Procedure ConstructNetworkFromSequence(S)

Input: A sequence of ordered pairs $S = (x_1, y_1) \cdots (x_n, y_n)$;
Output: The network that can be constructed from S;

```
1  Set N to be the tree on one leaf y_n;
2  for i = n, ..., 1 do
3      if x_i is a leaf of N then
4          if the parent of x_i is a reticulation then
5              Let p_x denote the parent of x_i;
6          else
7              Subdivide the incoming edge of x_i with a node p_x;
8          Subdivide the incoming edge of y_i with a node p_y;
9          Add the edge p_y p_x to N;
10     else
11         Subdivide the incoming edge of y_i with a node p_y;
12         Add a new node labelled x_i to N;
13         Add the edge p_y x_i to N;
14 return N;
```

Note that because we only add reticulation edges to existing reticulation nodes wherever possible (Line 4), the network obtained by using the above procedure is always stack-free. Imposing another condition: *no first coordinate leaf is used as a second coordinate in the remainder of the sequence* on the sequence ensures that the network we obtain is tree-child. With this in mind, we formally define a tree-child sequence.

Definition 5. *A* tree-child sequence (TCS) *is a sequence of ordered pairs of two leaves such that the following conditions hold:*

- *the second coordinate of each pair has to occur as a first coordinate in the remainder of the sequence or as the second coordinate of the last pair;*
- *no first coordinate leaf is used as a second coordinate in the rest of the sequence.*

Let S be a TCS, that involves the leaves X. Then, the *weight* of S is $w(S) = |S| - |X| + 1$. Given a sequence of ordered pairs $S = S_1 S_2 \cdots S_{|S|}$, we let NS denote the network

$$NS := (\cdots((NS_1)S_2)\cdots S_{|S|-1})S_{|S|} = NS_1S_2\cdots S_{|S|}.$$

We introduce some notation for subsequences of a sequence S. For $i \in [|S|]$, we use the following notation for subsequence of S. The ith ordered pair of S is $S_i = (x_i, y_i)$. The first i ordered pairs in S is denoted by $S_{[:i]} = (x_1, y_1), \ldots, (x_i, y_i)$. The subsequence of S without the first i ordered pairs is denoted by $S_{[i+1:]} = (x_{i+1}, y_{i+1}), (x_{i+2}, y_{i+2}), \ldots, (x_n, y_n)$. We say that the leaves $x_1, \ldots, x_i$ are *forbidden* for $S_{[:i]}$. Forbidden leaves do not appear as a second coordinate leaf in a TCS (by the second condition of TCSs).

We say S *reduces* N *to the leaf* x if NS is the tree with the single leaf x. Similarly, let $\mathcal{N}$ be a set of networks, then we denote by $\mathcal{N}S$ the set of reduced networks $\{NS : N \in \mathcal{N}\}$, and we say S *reduces* $\mathcal{N}$ *to* x if NS is the one leaf tree x for all $N \in \mathcal{N}$.

We call a sequence S' of ordered pairs a *partial TCS* if there exists a TCS S such that $S_{[:i]} = S'$ for some i.

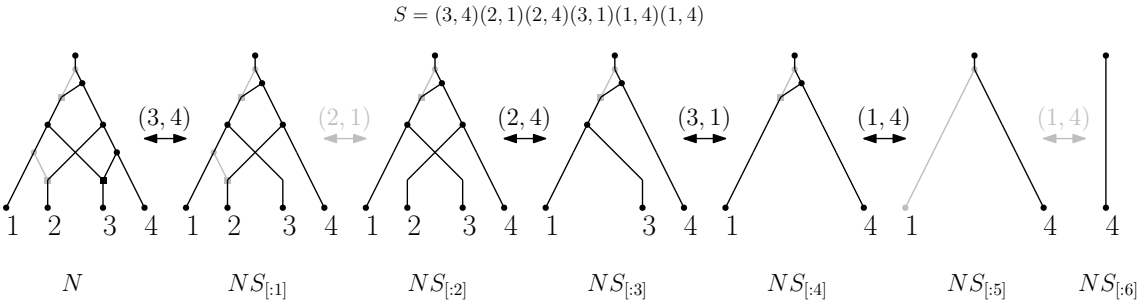

Fig. 1. A binary tree-child network N (grey and black) reduced to a leaf 4 by a tree-child sequence S. The reduction is shown as a sequence of networks $NS_{[:i]}$ for $i = 0, 1, \ldots, 6$ from left to right, in which an element of S is applied to the network successively. Each element of S reduces a pair in N. An example of a cherry $(3, 1)$ can be seen in the network $NS_{[:3]}$, and a reticulated cherry $(3, 4)$ can be seen in the network N. The subnetwork of N consisting of the black edges is also reduced by S, and the embedding can be constructed by building both networks simultaneously and keeping track of the edges added by the pairs that change the subnetwork (black pairs and arrows).

3 Network Hybridization

In this section we formally define the Tree-child Network Hybridization problem, which asks to find a tree-child network with minimal reticulation number that *displays* all input tree-child networks on the same set of taxa. We generalize the results presented in [14] (they considered inputs of trees while we consider inputs of networks) by showing how this problem relates to the more generalized problem of Network Hybridization and also to the Tree-child Weight problem. For the Tree-child Network Hybridization problem, it turns out that there is not always a solution for some given inputs; we also comment on when this is the case.

We start by defining what it means for a network to *display* another network.

Definition 6. *Let N be a network on the set of taxa, X. A network N' on $Y \subseteq X$ is a* subnetwork *of N if N' can be obtained from N by deleting reticulation edges, removing leaves not labelled by Y, and suppressing all degree-2 nodes in the resulting subgraph. If N' can be obtained from a subnetwork of N by contracting edges, then we say N* displays *N'. Given a set of networks $\mathcal{N}$ on some subsets of the taxa X, then we say that N* displays *$\mathcal{N}$ if N displays all networks in $\mathcal{N}$.*

If a network N' on X is a subnetwork of another network N on X, then an *embedding* of N' into N is the mapping of the nodes of N' to the nodes of N such that the leaves of N' are mapped to the leaves of N with the same labels, and the edges of N' are mapped to edge-disjoint paths of N. Our main focus of this paper is to solve the following problem.

Tree-child Network Hybridization
Input: A set of rooted tree-child networks $\mathcal{N}$ on X.
Output: A tree-child network that displays $\mathcal{N}$ with minimal reticulation number if it exists, NO otherwise.

Given an optimal tree-child network to the Tree-child Network Hybridization problem, one may find a TCS that reduces it. We will show that the weight of such a TCS is equal to the weight of an optimal solution to the following related problem.

Tree-child Weight
Input: A set of rooted networks $\mathcal{N}$ on X.
Output: A minimal weight TCS that reduces $\mathcal{N}$ if it exists, NO otherwise.

Let $\mathcal{N}$ be a set of networks on X. The reticulation number of an optimal solution to Tree-child Hybridization is denoted $h_{\text{tc}}(\mathcal{N})$. The weight of an optimal solution to Tree-child Weight is denoted $s_{\text{tc}}(\mathcal{N})$.

For a set of trees $\mathcal{T}$, the relation $h_{\text{tc}}(\mathcal{T}) = s_{\text{tc}}(\mathcal{T})$ holds. We will extend this result for network inputs. We first recall some key lemmas from [13]. The first lemma loosely states that each TCS reducing a set of networks $\mathcal{N}$ gives a tree-child network with corresponding reticulation number that displays $\mathcal{N}$. The second lemma gives the reverse statement: each tree-child network that displays a set of networks $\mathcal{N}$ gives a TCS of corresponding weight that reduces $\mathcal{N}$.

Lemma 1 ([13], Lemma 8). *Let N and N' be a tree-child network. Suppose there is a TCS S that reduces both N and N', such that each element of S that reduces a pair in N' also reduces a reducible pair in N. Then N' is a displayed by N.*

Lemma 2 ([13], Corollary 4). *Let N, N' be tree-child networks on X such that N' is displayed by N. If a TCS S reduces N, then S also reduces N'.*

Unlike when the input consists of only trees, a solution to Tree-child Network Hybridization does not always exist when the input may also contain networks.

Definition 7. *A set of tree-child networks $\mathcal{N}$ are* tree-child compatible *if there exists a tree-child network that displays all networks in $\mathcal{N}$.*

Our next step, is to prove that there is a strong connection between tree-child compatibility and TCSs.

Lemma 3. *Let $\mathcal{N}$ be a set of tree-child networks on X. Then $\mathcal{N}$ is tree-child compatible iff there exists a TCS that reduces $\mathcal{N}$. Furthermore, if a solution exists, then $h_{\text{tc}}(\mathcal{N}) = s_{\text{tc}}(\mathcal{N})$.*

Proof. Suppose that $\mathcal{N}$ is tree-child compatible. Then there exists a tree-child network N that displays $\mathcal{N}$, with minimal reticulation number. Now let S be a TCS for N. By Lemma 2, S also reduces all displayed networks of N, and hence it reduces $\mathcal{N}$. Furthermore, the weight of S is equal to the reticulation number of N by Lemma 3 from [13], (originally proved slightly less strong in [14]).

Now suppose there exists a TCS S that reduces $\mathcal{N}$. Let N be the tree-child network that can be constructed from S. Then, by Lemma 1, N displays $\mathcal{N}$. Because N is the network corresponding to S, the reticulation number of N is equal to the weight of S. □

3.1 The existence of a tree-child solution

In the previous subsection, we have found a strong connection between Tree-child Network Hybridization and Tree-child Weight for feasible solutions. Not all inputs, however, are feasible. Here, we investigate the feasibility of inputs, and how to deal with infeasible inputs.

Lemma 4. *Let N be a tree-child network with reticulated cherry (x, y), then any TCS that reduces N must contain the pair (x, y).*

Proof. Suppose S is a TCS that reduces N. The only ways to reduce the reticulated cherry (x, y) are by either reducing it directly with the pair (x, y), or by first turning it into a cherry $\{x, y\}$ and then reducing it with a pair (x, y) or (y, x). This second option, however, leads to a contradiction: To make the reticulated cherry into a cherry, we must reduce a pair of the form $(x, \cdot)$; however, any sequence that includes $(x, \cdot)$ and later (y, x) cannot be tree-child. □

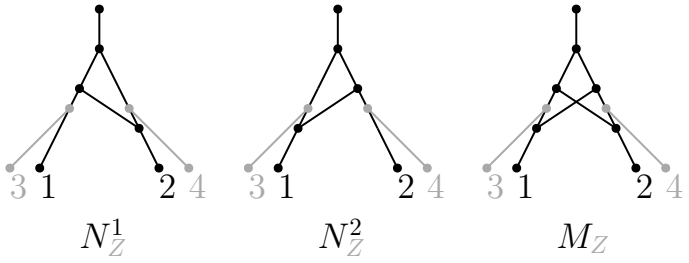

Fig. 2. Two networks $\mathcal{N} = \{N^1, N^2\}$ that are tree-child incompatible (black parts only). The network M displays $\mathcal{N}$, but it is not tree-child. By adding leaves $Z = \{3, 4\}$ to M, we get the network M_Z which is tree-child. Then, adding these leaves in the right places to N^1 and N^2, we get the set of networks $\mathcal{N}_Z \in \mathcal{N}^+$ on $X \cup Z$, that are displayed by the tree-child network M_Z.

Using the connection between tree-child compatibility and the existence of TCSs, we can prove an obstruction to tree-child compatibility of Lemma 5. This obstruction will turn out to be quite valuable in the proofs in the rest of this paper, as it allows us to quickly check whether a set of networks is tree-child compatible.

Lemma 5. *Let N_1, N_2 be tree-child networks on the same set of leaves X. For any pair of leaves x, y, if N_1 contains the reticulated cherry (x, y) and N_2 contains the reticulated cherry (y, x), then N_1 and N_2 are not tree-child compatible.*

Proof. Let N be a tree-child network that displays both N_1 and N_2. Then any TCS S for N reduces both N_1 and N_2. By Lemma 4, the sequence S must contain the pair (x, y), because N_1 has the reticulated cherry (x, y); similarly, S must contain (y, x). This means S is a TCS, but it includes both pairs (x, y) and (y, x), a contradiction. Hence we conclude that N_1 and N_2 are not tree-child compatible. □

Even if an input is infeasible, we still desire a network that displays all input networks. For this purpose, we can relax the tree-child constraint on output (and input) of the Tree-child Network Hybridization problem, giving rise to the following problem.

Network Hybridization
Input: A set of rooted networks $\mathcal{N}$ on X.
Output: A network that displays $\mathcal{N}$ with minimal reticulation number.

This problem can be viewed as the natural extension of the classic Hybridization problem for trees. Linz and Semple show that Hybridization can be solved by adding leaves in the right place to all input trees, and then solving Tree-child Hybridization [14]. This also holds for the network versions of these problems, as the solution to Network Hybridization can be made tree-child by adding leaves, and all networks displayed by a tree-child network are tree-child networks as well (Figure 2).

4 An Algorithm for Tree-child Network Hybridization

In this section, we give an FPT algorithm for Tree-child Network Combination. We extend the algorithm given in [7] by allowing for inputs to be networks, and by looking for reducible pairs within networks rather than cherries in trees. Given an input $\mathcal{N}$ of tree-child networks, we first look for *trivial* reducible pairs. We show that it is safe to reduce trivial reducible pairs as soon as we encounter one, in any order. We then branch on all possible non-trivial reducible pairs of the network, and by showing that the total number of possible reducible pairs at each branching point is at most $8k$ for the reticulation number k of the optimal solution, we show that the running time of the algorithm is $O((8k)^k \cdot \text{poly}(|X|, |\mathcal{N}|))$.

4.1 Counting Cherries

Trivial pairs The algorithm in [7] reduces *trivial cherries* (a pair of leaves $\{x, y\}$ that appear as a cherry in any input tree containing x and y) whenever possible. Here, only looking at trivial cherries is not sufficient. For an input of networks, we will need to reduce *trivial reducible pairs* (trivial pairs for short) whenever possible. A trivial pair is a pair of leaves (x, y) such that all networks either only have the leaf y, or they have a reducible pair (x, y). In the following two lemmas, we prove that it is safe to reduce such a pair as soon as we encounter one.

Lemma 6 (Move to the left). *Let $\mathcal{N} = \{N_1, \ldots, N_I\}$ be a set of tree-child networks on a common set of leaves, and let $S(a, b)(x, y)S'$ be a TCS for $\mathcal{N}$. Suppose that for each $N \in \mathcal{N}S$ we have either x is not a leaf in N, or (x, y) is a reducible pair of N, and there is at least one network such that the latter holds. Then there is a TCS for $\mathcal{N}$ starting with $S(x, y)$ of length equal to that of $S(a, b)(x, y)S'$.*

Proof. Suppose $b = x$. Then there must be a network in $\mathcal{N}S$ that has both the reducible pairs (x, y) and (a, x). This can only occur if $a = y$: as x is the first as well as the second element of a reducible pair, it must form a cherry with another leaf, namely the leaf y. However, $S(y, x)(x, y)S'$ is not a TCS, which contradicts our assumption that $S(a, b)(x, y)S'$ is a TCS for $\mathcal{N}$.

Hence, for the rest of the proof, we assume $b \neq x$. In this case, $S(x, y)(a, b)S'$ is a TCS. It remains to prove that it reduces $\mathcal{N}$. This is clear if $\{x, y\} \cap \{a, b\} = \emptyset$. Observe that $a \neq y$, as otherwise $S(a, b)(x, y)S'$ would not have been a TCS to begin with. Therefore, we still need to check the cases $a = x$ and $b = y$.

If $a = x$ and a network has both reducible pairs (x, b) and (x, y), then this network has a reticulation with reticulated cherries (x, b) and (x, y). The order of reducing these pairs obviously does not matter for such networks: both options remove the reticulation edges between the parents of b and y, and the parent of x. For a network N that only has the reducible pair (x, y) after S (and not (x, b)), the network $NS(x, y)(x, b)$ is a subnetwork of $NS(x, b)(x, y) = NS(x, y)$. This means $S(x, y)(x, b)S'$ also reduces N [13]. Hence if $a = x$, the sequence $S(x, y)(a, b)S'$ is a TCS for $\mathcal{N}$.

Now suppose $b = y$. Let N be a network that has both reducible pairs (a, y) and (x, y). But all tree nodes of N are of outdegree-2; this implies that every leaf can be the second coordinate of at most one reducible pair. Therefore such a network cannot exist, and thus this case is not possible. □

Lemma 7 (Trivial pair reduction). *Let $\mathcal{N} = \{N_1, \ldots, N_I\}$ be a set of tree-child networks on a common set of leaves such that there exists a TCS SS' for $\mathcal{N}$. Suppose x, y are leaves such that for each $N \in \mathcal{N}S$ we have either x not in N, or (x, y) a reducible pair of N, and there is at least one network such that the latter holds. Then there exists a TCS $S(x, y)S''$ of length equal to SS' that reduces $\mathcal{N}$, or if y is forbidden after S and there is a sequence of the form $S(y, x)S'''$ of the same length as SS' that reduces $\mathcal{N}$.*

Proof. To reduce a network with reducible pair (x, y), the sequence S' must contain either (x, y) or (y, x). Let S'_i be the first occurrence of such a pair.

First suppose $S'_i = (x, y)$. Then for each intermediate set of networks $\mathcal{N}SS'_{[:j]}$ for $j < i$ we have that all the networks in the set either do not contain x, or have the reducible pair (x, y). Hence, by repeated application of Lemma 6, there is a sequence $S(x, y)S''$ for $\mathcal{N}$. This sequence has the same length as SS', because it is simply a reordering of the pairs.

Now suppose $S'_i = (y, x)$, then x cannot have been the first coordinate in any pair of S, so all networks in $\mathcal{N}S$ contain x. Furthermore, S' does not contain the pair (x, y), as this would violate the assumption that SS' is a TCS. Hence, each network in $\mathcal{N}S$ has a cherry or reticulated cherry on x and y, which is ultimately reduced by a pair (y, x) in S'. Suppose a network $N \in \mathcal{N}S$ does not have the cherry $\{x, y\}$. Then it has the reticulated cherry (x, y). To make this into a cherry, so that it can be reduced by (y, x), the sequence must first contain a pair of the form (x, z). However, this implies S' first contains (x, z) and then (y, x), which contradicts the fact that SS' is a TCS. Hence, we may assume that all networks in $\mathcal{N}S$ have the cherry $\{x, y\}$.

If y is not forbidden after S, we can switch the roles of x and y in the remaining part of the sequence S' to get a new TCS SS^* for $\mathcal{N}$. In S^*, the first occurrence of (x, y) or (y, x) is $S^*_i = (x, y)$,

and we are in the previous case. If y is forbidden after S, repeated application of Lemma 6 on SS' and S'_i gives a sequence $S(y,x)S'''$ for $\mathcal{N}$. □

Bounding reducible pairs in networks with all leaves In the algorithm in [7], a bound on the number of cherries after having reduced all trivial cherries was required to compute the running time. Here, we require something similar; we require a bound on the number of reducible pairs after we have reduced all the trivial pairs. [7] prove such bounds by first focusing on the case where all input trees have the same leaf set. We do the same, by first focusing on the case where all input networks have the same leaf set.

Let $\mathcal{N}$ be a set of networks. Then the *set of displayed trees of* $\mathcal{N}$ is the set of all trees that are displayed by the networks of $\mathcal{N}$.

Lemma 8. *Let $\mathcal{N} = \{N_1, \ldots, N_I\}$ be a set of tree-child networks on a common set of leaves such that there exists a TCS S for $\mathcal{N}$. If $\mathcal{N}$ does not contain any trivial pairs, then the set of displayed trees of $\mathcal{N}$ has no trivial cherries.*

Lemma 9 ([7] Lemma 10). *Let $\mathcal{T}$ be a set of phylogenetic trees with leaf set X such that there is a tree-child network N with k reticulations that displays $\mathcal{T}$. If $\mathcal{T}$ has no trivial cherries, then the total number of cherries of the trees in $\mathcal{T}$ is at most $4k$.*

Lemmas 8 and 9 gives the bound on the number of reducible pairs for networks with common leaf sets.

Lemma 10. *Let $\mathcal{N}$ be a set of tree-child networks with leaf set X such that there is a tree-child network N with k reticulations that displays $\mathcal{N}$. If $\mathcal{N}$ has no trivial pairs, then the total number of reducible pairs of the networks in $\mathcal{N}$ is at most $8k$.*

Proof. Each reducible pair of a network is a cherry in one of its displayed trees, and the set of displayed trees is displayed by the solution network N as well. Hence, by Lemma 9, there are at most $8k$ reducible pairs in the trees, and therefore at most $8k$ reducible pairs in the networks. □

Bounding reducible pairs in general Recall that the algorithm will build a TCS by successively appending reducible pairs; it terminates upon finding the shortest possible sequence that reduces all the input networks. In the process, it branches on all possible non-trivial pairs that the input network may have. Depending on the sequence that is being built, it is possible that leaves that exist in some of the input networks (after reduction by the existing sequence) may have already been deleted from others. Here, we show that even for these instances, it is still the case that the number of possible reducible pairs that we can branch on is bounded by $8k$. This result follows directly from Lemma 7 of [7]: we change the wording of the statement slightly to accommodate for network inputs.

Lemma 11. *Let $\mathcal{N}$ be a set of tree-child networks on X, and let $S = (x_1, y_1), (x_2, y_2), \ldots, (x_r, y_r)$ be a TCS for $\mathcal{N}$ with weight k. For any $j \in [r] \cup \{0\}$, either there exists a trivial pair of $\mathcal{N}S_{[:j]}$, or $\mathcal{N}S_{[:j]}$ has at most $8k$ reducible pairs.*

The idea of the proof is as follows. Let j be such that $\mathcal{N}S_{[:j]}$ has no trivial pairs. Then we find a set of tree-child networks $\hat{\mathcal{N}}_j$ on X with the same set of reducible pairs as $\mathcal{N}S_{[:j]}$ and tree-child hybridization number at most k. By Lemma 10, this shows that $\mathcal{N}S_{[:j]}$ has at most $8k$ reducible pairs.

The set of networks is constructed by adding back each missing leaf to each network in $\mathcal{N}S_{[:j]}$ at the root. The order in which they are placed at the root is the same as the order in which these leaves appear as first element in $S_{[:j]}$. Now, we may construct a TCS of the same weight as S that reduces this set of networks. By first reducing the part that corresponds to the part in $\mathcal{N}S_{[:j]}$, and then the leaves placed by the root, we have a TCS that reduces $\hat{\mathcal{N}}_j$ of weight at most k:

$$(x_{j+1}, y_{j+1}), (x_{j+2}, y_{j+2}), \ldots, (x_r, y_r), (x_1, y_r), (x_2, y_r), \ldots, (x_j, y_r).$$

An example of the corresponding networks and their embeddings can be found in Figure 3.

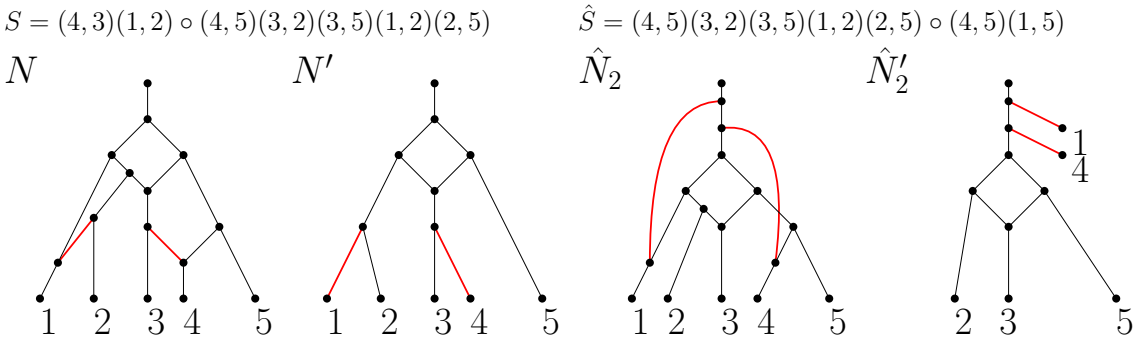

Fig. 3. A set $\mathcal{N} = \{N, N'\}$ of tree-child networks on the set of taxa $\{1,2,3,4,5\}$, with a TCS S that reduces it. A set $\hat{\mathcal{N}}_2 = \{\hat{N}_2, \hat{N}_2'\}$ of tree-child networks obtained by reducing the first two elements of S from $\mathcal{N}$, and reattaching the tail of the deleted edges (red edge) to the root edge, in the order that they were deleted in (as explained in the sketch proof of Lemma 11). The sequence $\hat{S}$ is a TCS of the same weight as S, obtained from S by deleting the first two elements and appending these two elements to the end of the sequence, for which we replace the second coordinate of the elements by 5 (the leaf that appears as the second coordinate element in the last element of S.

4.2 Adapting the algorithm

Our algorithms are the same as those presented in [7], except for the following changes.

- The input set of trees $\mathcal{T}$ is changed into an input set of tree-child networks $\mathcal{N}$;
- trivial cherries are now trivial pairs;
- In line 4, the stop condition of a non-pickable reticulated cherry is added;

The first change is necessary for the algorithm to take an input consisting of networks. The second change is necessary as not all reducible pairs are cherries anymore, when the input consists of networks. The while-loop that reduces all the trivial pairs is still correct in the algorithm, because there is an optimal sequence that first reduces all trivial pairs (Lemma 7). The last change makes sure we stop when the reduced input $\mathcal{N}S$ cannot be fully reduced using a TCS that can be appended after the prefix S.

Otherwise, the algorithm is still correct. Indeed, the algorithm branches over all non-trivial pairs, to find a shortest sequence that reduces all input networks; and this shortest sequence corresponds to a network with minimal reticulation number that displays all input networks. Furthermore, the running time follows as each branch-out is over at most $8k$ pairs, and the search depth is at most k.

Theorem 1. *Let $\mathcal{N}$ be a set of tree-child networks on a set of taxa X. If there exists a tree-child network with at most k reticulations that displays $\mathcal{N}$, then it can be found in $O((8k)^k \cdot \text{poly}(|X|, |\mathcal{N}|))$ time using* TreeChildNetwork$(\mathcal{N}, k)$.

5 Discussion

In this paper, we have introduced Network Hybridization, the problem of finding a network with minimal reticulation number that displays a set of networks. We showed that the Tree-child Network Hybridization problem, in which we restrict our inputs and output to be tree-child networks, can be solved by making slight adjustments to the FPT algorithm presented in [7].

In practice, our algorithm can be sped up using the heuristic improvement that was introduced in [7]. We may consider branch reduction, in which we ignore parts of the search tree where no better solution can be found.

For this problem, FPT is essentially the best we can do, because solving the Network Hybridization problem for an input set of tree-child networks is NP-hard. This follows from the fact that it is already NP-hard for an input set of trees. It has recently been shown that if all level-$(k-1)$ subnetworks of a level-k tree-child networks are given, this network can be constructed

Procedure TreeChildSequence($\mathcal{N}, S, k$)

Input: A collection $\mathcal{N}$ of tree-child networks, a partial TCS S, an integer k;
Output: An optimal TCS SS' of weight at most k for $\mathcal{N}$ if such a sequence exists; FAIL otherwise;

```
while There exists a trivial pair (x, y) in NS with y not forbidden by S do
    Set S = S(x, y);
Set N' = NS;
if some network in N' has a cherry (x, y) with x, y forbidden or a reticulated cherry (x, y) with y forbidden then
    return FAIL;
else
    Set n' = |{x ∈ X : x is a leaf in N'}|;
    Set k' = |S| − |X| + n';
    Set C = {(x, y) | (x, y) is a reducible pair of some network in N'};
    if |C| == 0 then
        return S;
    else if |C| > 8k or k' ≥ k then
        return FAIL;
    else
        Set S_opt = FAIL;
        foreach (x, y) ∈ C with y not forbidden by S do
            Set S_temp = TreeChildSequence(N, S(x, y), k);
            if S_temp ≠ FAIL and (S_opt = FAIL or (S_opt ≠ FAIL and w(S_temp) < w(S_opt))) then
                Set S_opt = S_temp;
        return S_opt;
```

Procedure TreeChildNetwork($\mathcal{N}, k$)

Input: A collection $\mathcal{N}$ of tree-child networks, an integer k;
Output: A tree-child phylogenetic network N on X that displays $\mathcal{N}$ with reticulation number at most k, if such a network exists; otherwise NONE;

```
Set S = TreeChildSequence(N, ∅, k);
if S == FAIL then
    return NONE;
else
    Set N = ConstructNetworkFromSequence(S);
    return N;
```

in polynomial time [15]. In other words, the TREE-CHILD NETWORK HYBRIDIZATION problem is easy to solve when we are given all level-$(k-1)$ subnetworks of a level-k network. This suggests that the problem becomes easy if the difference in reticulation number between the inputs and the output network is bounded. We wonder if this is still true for networks that are not tree-child, and therefore it would be interesting to see whether the HYBRIDIZATION problem is FPT with this difference in reticulation number as parameter. And, if this is the case, whether the current algorithm can be proven to have this running time.

Recall that a TCS is a sequence of ordered pairs with two conditions imposed on them: the first condition ensures that we obtain a network from the sequence upon using the CONSTRUCT-NETWORKFROMSEQUENCE algorithm; the second condition ensures that the network we obtain is tree-child. Upon removing this second condition from sequences of ordered pairs, we obtain what is called a *cherry-picking sequence* [13]. Networks that can be reduced by a cherry-picking sequence are called *orchard* networks [13, 3]. A natural extension of the results we have presented in this paper would be to consider the following problem.

Orchard Network Hybridization
Input: A set of orchard networks $\mathcal{N}$ on X.
Output: An orchard network that displays $\mathcal{N}$ with minimal reticulation number.

Ideally, in Algorithm TreeChildSequence, we would simply remove the tree-child condition to obtain an algorithm which works for orchard networks as well. However, simply doing so could potentially result in a much higher running time, as we do not have a bound on the number of reducible pairs for orchard networks (see Figure 4). Nevertheless, solving Orchard Network Hybridization could lead to better upper bounds for the network hybridization number, and the algorithm could still be efficient in practice. In this light, this paper has taken the first step towards finding good solutions for Network Hybridization.

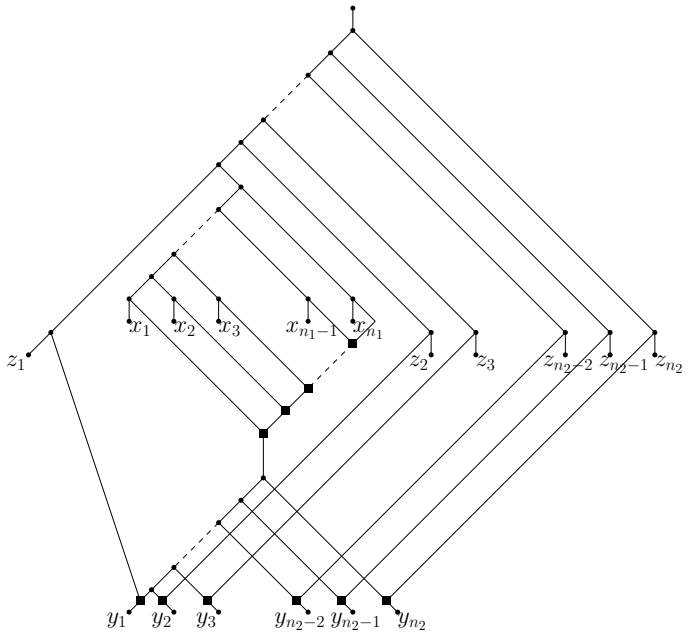

Fig. 4. An orchard network with $n_1 + n_2 - 1$ reticulations such that the set of displayed trees have at least $n_1 n_2$ cherries.

Acknowledgements The final authenticated version is available online at `https://doi.org/10.1007/978-3-030-42266-0_7`.

References

1. Mihaela Baroni, Charles Semple, and Mike Steel. A framework for representing reticulate evolution. *Annals of Combinatorics*, 8(4):391–408, 2005.
2. Magnus Bordewich and Charles Semple. Computing the minimum number of hybridization events for a consistent evolutionary history. *Discrete Applied Mathematics*, 155(8):914–928, 2007.
3. Péter L Erdős, Charles Semple, and Mike Steel. A class of phylogenetic networks reconstructable from ancestral profiles. *Mathematical biosciences*, 313:33–40, 2019.
4. Katharina T Huber, Vincent Moulton, Charles Semple, and Taoyang Wu. Quarnet inference rules for level-1 networks. *Bulletin of mathematical biology*, 80(8):2137–2153, 2018.
5. Katharina T Huber, Leo Van Iersel, Vincent Moulton, Celine Scornavacca, and Taoyang Wu. Reconstructing phylogenetic level-1 networks from nondense binet and trinet sets. *Algorithmica*, 77(1):173–200, 2017.
6. Peter J Humphries, Simone Linz, and Charles Semple. Cherry picking: a characterization of the temporal hybridization number for a set of phylogenies. *Bulletin of mathematical biology*, 75(10):1879–1890, 2013.

7. Leo van Iersel, Remie Janssen, Mark Jones, Yukihiro Murakami, and Norbert Zeh. A practical fixed-parameter algorithm for constructing tree-child networks from multiple binary trees. *arXiv preprint arXiv:1907.08474*, 2019.
8. Leo van Iersel, Steven Kelk, Nela Lekic, Chris Whidden, and Norbert Zeh. Hybridization number on three rooted binary trees is ept. *SIAM Journal on Discrete Mathematics*, 30(3):1607–1631, 2016.
9. Leo van Iersel and Simone Linz. A quadratic kernel for computing the hybridization number of multiple trees. *Information Processing Letters*, 113(9):318–323, 2013.
10. Leo van Iersel and Vincent Moulton. Trinets encode tree-child and level-2 phylogenetic networks. *Journal of mathematical biology*, 68(7):1707–1729, 2014.
11. Leo van Iersel and Vincent Moulton. Leaf-reconstructibility of phylogenetic networks. *SIAM Journal on Discrete Mathematics*, 32(3):2047–2066, 2018.
12. Leo van Iersel, Vincent Moulton, Eveline de Swart, and Taoyang Wu. Binets: fundamental building blocks for phylogenetic networks. *Bulletin of mathematical biology*, 79(5):1135–1154, 2017.
13. Remie Janssen and Yukihiro Murakami. Solving phylogenetic network containment problems using cherry-picking sequences. *arXiv preprint arXiv:1812.08065*, 2018.
14. Simone Linz and Charles Semple. Attaching leaves and picking cherries to characterise the hybridisation number for a set of phylogenies. *Advances in Applied Mathematics*, 105:102–129, 2019.
15. Yukihiro Murakami, Leo van Iersel, Remie Janssen, Mark Jones, and Vincent Moulton. Reconstructing tree-child networks from reticulate-edge-deleted subnetworks. *Bulletin of mathematical biology*, 81(10):3823–3863, 2019.
16. James Oldman, Taoyang Wu, Leo Van Iersel, and Vincent Moulton. Trilonet: piecing together small networks to reconstruct reticulate evolutionary histories. *Molecular biology and evolution*, 33(8):2151–2162, 2016.
17. Chris Whidden, Robert G Beiko, and Norbert Zeh. Fixed-parameter algorithms for maximum agreement forests. *SIAM Journal on Computing*, 42(4):1431–1466, 2013.