Reconstructing Phylogenetic Networks via Cherry Picking and Machine Learning

Giulia Bernardini
University of Trieste, Italy
CWI, Amsterdam, The Netherlands

Leo van Iersel
Delft Institute of Applied Mathematics, Delft University of Technology, The Netherlands

Esther Julien
Delft Institute of Applied Mathematics, Delft University of Technology, The Netherlands

Leen Stougie
CWI and Vrije Universiteit, Amsterdam, The Netherlands
Erable, France

Abstract

Combining a set of phylogenetic trees into a single phylogenetic network that explains all of them is a fundamental challenge in evolutionary studies. In this paper, we apply the recently-introduced theoretical framework of cherry picking to design a class of heuristics that are guaranteed to produce a network containing each of the input trees, for practical-size datasets. The main contribution of this paper is the design and training of a machine learning model that captures essential information on the structure of the input trees and guides the algorithms towards better solutions. This is one of the first applications of machine learning to phylogenetic studies, and we show its promise with a proof-of-concept experimental study conducted on both simulated and real data consisting of binary trees with no missing taxa.

2012 ACM Subject Classification Applied computing → Computational biology

Keywords and phrases Phylogenetics, Hybridization, Cherry Picking, Machine Learning, Heuristic

Digital Object Identifier 10.4230/LIPIcs.WABI.2022.16

Supplementary Material *Software (Source Code)*: `https://github.com/estherjulien/HybridML`, archived at `swh:1:dir:da2304df7fc11a746b630fcbe73fbbe3e2ea655d`

Funding This paper received funding from the Netherlands Organisation for Scientific Research (NWO) under project OCENW.GROOT.2019.015 "Optimization for and with Machine Learning (OPTIMAL)".
Giulia Bernardini: MUR-FSE REACT EU – PON R&I 2014-2020
Leen Stougie: The Netherlands Organisation for Scientific Research (NWO) under Gravitation-grant NETWORKS-024.002.003

Acknowledgements The authors thank Remie Janssen for providing ideas and preliminary code for the randomised heuristics, and Yukihiro Murakami for the inspiring discussions.

1 Introduction

Phylogenetic networks describe the evolutionary relationships between, for example, genes, genomes, or species. One of the first and most natural approaches to construct phylogenetic networks is to build a network from a set of gene trees. In the absence of incomplete lineage sorting, the constructed network is naturally required to "display", or embed, each of the gene trees. In addition, following the parsimony principle, a network assuming a minimum

number of reticulate evolutionary events (like hybridization or lateral gene transfer) is often sought. Unfortunately, the associated computational problem, called Hybridization, is NP-hard even for two binary input trees with equal leaf sets [4].

For a long time, research on this topic was mostly restricted to inputs consisting of two trees. Proposed algorithms for multiple trees were either completely impractical or ran in reasonable time only for very small numbers of input trees. This situation changed drastically with the introduction of so-called cherry-picking sequences [10]. This theoretical set-up opened the door to solving instances consisting of many input trees, like most practical datasets have. Indeed, a recent paper showed that this technique can be used to solve instances with up to 100 input trees to optimality [19], although it was restricted to binary trees all having the same leaf set and to so-called "tree-child" networks. Moreover, its running time has a (strong) exponential dependence on the number of reticulate events.

In this paper, we show significant progress towards a fully practical method by developing heuristics that are based on cherry picking and machine learning. Admittedly, our method is not yet widely applicable since it is still restricted to binary trees. However, our set-up is made in such a way that it can be extended to general trees, and we plan to do so in future work. Furthermore, although our method can theoretically be applied to trees with different leaf sets, in this paper we only conduct experiments on input trees with equal leaf sets. Still, we see our current method already as a breakthrough as it scales well with number of trees, number of taxa and number of reticulations. In fact, we experimentally show that it can easily handle sets of 100 trees in reasonable time (13 minutes on average for sets consisting of trees with 100 leaves each). We also have a faster version of the heuristic that already finds feasible solutions in 8 seconds for the same instances. As the running time depends at most quadratically on the input size, and linearly on the output number of reticulations, we expect it to be able to solve much larger instances still in a reasonable amount of time. In addition, our method is not restricted to tree-child networks.

The method we present is one of the first applications of machine learning in phylogenetics, and shows its promise. In particular, for the networks generated in our simulation study, it shows that features of interest of the networks can be identified with very high test accuracy (99.8%) purely based on the trees displayed by the networks. It is important to note at this point that no method is able to reconstruct any specific network from displayed trees as networks are, in general, not uniquely determined by the trees they display [12].

We focus on *orchard* networks (also called *cherry picking* networks), which are precisely those networks that can be drawn as a tree with additional horizontal arcs [17]. Such horizontal arcs can for example correspond to lateral gene transfer events (LGT). Nevertheless, orchard networks are applicable more broadly because also networks in which reticulations represent hybridization or recombination can be orchard networks. In particular, the orchard networks class is much bigger than the class of tree-child networks.

Related work. Previous practical algorithms for Hybridization include PIRN [21], PIRNs [11] and Hybroscale [1], exact methods that are only applicable to (very) small numbers of trees and/or to trees that can be combined into a network with a (very) small reticulation number.

The theoretical framework of cherry picking was introduced in [7] (for the restricted class of temporal networks) and [10] (for the class of tree-child networks) and was turned into algorithms for reconstructing tree-child [19] and temporal [5] networks. These methods can handle instances containing many trees but do not scale well with the number of reticulations, due to an exponential dependence. Other methods such as PhyloNet [20] and

PhyloNetworks [16] also construct networks from trees but have different premises and use completely different models. The class of orchard networks, which is based on cherry picking, was introduced in [15] and independently (as cherry-picking networks) in [8], although their practical relevance, as trees with added horizontal edges, was only discovered later [17].

The applicability of machine-learning techniques to Phylogenetics has not yet been fully explored, and to the best of our knowledge existing work is mainly limited to phylogenetic trees inference [2, 22] and to testing evolutionary hypotheses [9].

Our contributions. We introduce the CPH class of heuristics to combine a set of binary phylogenetic trees into a single binary phylogenetic network based on cherry picking. Specifically, we address the Hybridization problem, asking to combine a set of trees in a single network with the minimum possible reticulation number.

We define and analyse several heuristics in the CPH class, all of which are guaranteed to produce feasible solutions to Hybridization and all of which can handle instances of practical size. Two of the methods we propose are simple randomised heuristics that showed to be extremely fast and to produce good solutions when run multiple times.

The main contribution of this paper consists in a machine-learning model that potentially captures essential information about the structure of the input set of trees. We trained the model over an extensive set of synthetically generated data, and applied it to guide our algorithms towards better solutions. In proof-of-concept experiments we show that the two machine-learned heuristics we design yield satisfactory results when applied to both synthetically generated and real data.

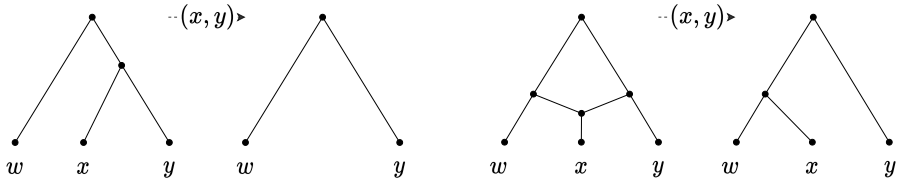
2 Preliminaries

A *phylogenetic network* $N = (V, E, X)$ on a set of taxa X is a directed acyclic graph (V, E) with a single *root* with in-degree 0 and out-degree 1, and the other nodes with either (i) in-degree 1 and out-degree $k > 1$ (*tree nodes*); (ii) in-degree $k > 1$ and out-degree 1 (*reticulations*); or (iii) in-degree 1 and out-degree 0 (*leaves*). The leaves of N are biunivocally labelled by X. A surjective map $\ell : E \to \mathbb{R}^{\geq 0}$ may assign a nonnegative *branch length* to each edge of N. We will denote by $[1, n]$ the set of integers $\{1, 2, ..., n\}$.

Throughout this paper, we will only consider binary networks (with $k = 2$), and we will identify the leaves with their labels. We will also often drop the term "phylogenetic", as all the networks considered in this paper are phylogenetic networks. The *reticulation number* $r(N)$ of a network N is $\sum_{v \in V} \max(0, d^-(v) - 1)$, where $d^-(v)$ is the in-degree of v. A network T with $r(T) = 0$ is a *phylogenetic tree*. It is easy to verify that binary networks with $r(N)$ reticulations have $|X| + r(N) - 1$ tree nodes.

Cherry-picking. We denote by $\mathcal{N}$ a set of networks and by $\mathcal{T}$ a set of trees. An ordered pair of leaves (x, y), $x \neq y$, is a *cherry* in a network if x and y have the same parent; (x, y) is a *reticulated cherry* if the parent $p(x)$ of x is a reticulation, and $p(y)$ is a tree node and a parent of $p(x)$ (see Figure 1). A pair is *reducible* if it is either a cherry or a reticulated cherry.

Reducing (or *picking*) a cherry (x, y) in a network N is the action of deleting x and replacing the two edges $(p(p(x)), p(x))$ and $(p(x), y)$ with a single edge $(p(p(x)), y)$ (see Figure 1a). If N has branch lengths, the length of the new edge is $\ell(p(p(x)), y) = \ell(p(p(x)), p(x)) + \ell(p(x), y)$. A reticulated cherry (x, y) is reduced (picked) by deleting the edge $(p(y), p(x))$ and replacing the other edge $(z, p(x))$ incoming to $p(x)$, and the consecutive edge $(p(x), x)$, with a single



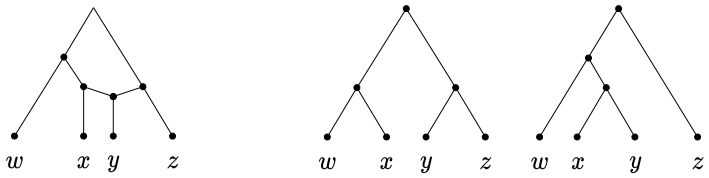
(a) Cherry picking. **(b)** Reticulated cherry picking.

Figure 1 Cherry (x, y) is picked in two different networks. In Fig. 1a (x, y) is a cherry, and in Fig. 1b (x, y) is a reticulated cherry. After picking, degree-two nodes are replaced by a single edge.

edge (z, x). The length of the new edge is $\ell(z, x) = \ell(z, p(x)) + \ell(p(x), x)$ (if N has branch lengths). Reducing a non-reducible pair has no effect on N. In all cases, the resulting network is denoted by $N_{(x,y)}$: we say that (x, y) affects N if $N \neq N_{(x,y)}$.

Any sequence $S = (x_1, y_1), \ldots, (x_n, y_n)$ of ordered leaf pairs, with $x_i \neq y_i$ for all i, is a *partial cherry-picking sequence*; S is a cherry-picking sequence (CPS) if, for each $i < n$, $y_i \in \{x_{i+1}, \ldots, x_n, y_n\}$. Given a network N and a (partial) CPS S, we denote by N_S the network obtained by reducing in N each element of S, in order. We say that S fully reduces N if N_S consists of the root with a single leaf. N is an *orchard network* (ON) if there exists a CPS that fully reduces it. If S fully reduces all $N \in \mathcal{N}$, we say that S fully reduces $\mathcal{N}$. In particular in this paper we will be interested in CPS that fully reduce a set of trees.

Hybridization. The Hybridization problem can be thought of as the computational problem of combining a set of phylogenetic trees into a network with the smallest possible reticulation number, that is, to find a network that displays each of the input trees in the sense specified by Definition 1. See Figure 2 for an example. The definition describes not only what it means to display a tree but also to display another network, which will be useful later.



(a) Network N. **(b)** Input tree set $\mathcal{T}$.

Figure 2 The two trees in Fig. 2b are displayed in the network of Fig. 2a.

▶ **Definition 1.** *Let $N = (V, E, X)$ and $N' = (V', E', X')$ be networks on the sets of taxa X and $X' \subseteq X$, respectively. The network N' is* displayed *in N if there is an* embedding *of N' in N: an injective map of the nodes of N' to the nodes of N, and of the edges of N' to edge-disjoint paths of N, such that the mapping of the edges respects the mapping of the nodes, and the mapping of the nodes respects the labelling of the leaves.*

We call *exhaustive* a tree displayed in $N = (V, E, X)$ with the whole X as leaf set. Note that Definition 1 only involves the topologies of the networks, disregarding possible branch lengths. In the following problem definition, the input trees may or may not have branch lengths, and the output is a network without branch lengths. We allow branch lengths for the input because they will be useful for the machine-learned heuristics of Section 4.

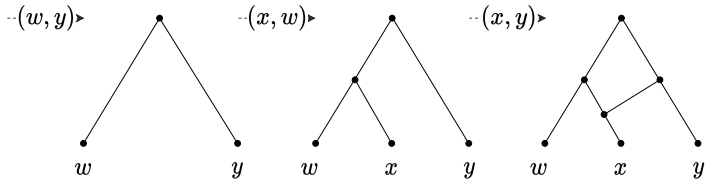


Figure 3 The ON reconstructed from the sequence $S = (x, y), (x, w), (w, y)$. The pairs are added to the network in reverse order: if the first element of a pair is not yet in the network, it is added as a cherry with the second element (see the pair (x, w)). Otherwise, a reticulation is added above the first element with an incoming edge from a new parent of the second element (see the pair (x, y)).

Hybridization
Input: A set of phylogenetic trees $\mathcal{T}$ on a set of taxa X.
Output: A network displaying $\mathcal{T}$ with minimum possible reticulation number.

3 Solving the Hybridization Problem via Cherry-Picking Sequences

We will develop heuristics for the Hybridization problem using cherry-picking sequences that fully reduce the input trees, leveraging the following result by Janssen and Murakami.

▶ **Theorem 2** ([8], Theorem 3). *Let N be a binary orchard network, and N' a (not necessarily binary) orchard network on sets of taxa X and $X' \subseteq X$, respectively. If a minimum-length CPS S that fully reduces N also fully reduces N', then N' is displayed in N.*

For binary ON, the following lemma, which is a special case of [8, Lemma 1], also holds.

▶ **Lemma 3.** *Let N be a binary orchard network, and let (x, y) be a reducible pair of N. Then reducing (x, y) and then adding it back to $N_{(x,y)}$ results in N.*

Theorem 2 and Lemma 3 provide the following approach for finding a feasible solution to Hybridization: find a CPS S that fully reduces all the input trees, and then reconstruct the unique binary orchard network N for which S is a minimum-length CPS. N can be reconstructed from S using one of the methods underlying Lemma 3 proposed in the literature, e.g., in [8] (illustrated in Figure 3) or in [19]. In doing so, the number of reticulations of the resulting N is $r(N) = |S| - |X| + 1$ [18]. In the next section we focus on the first part of the heuristic: producing a CPS that fully reduces a given set of phylogenetic trees.

3.1 Randomised Heuristics

We define a class of randomised heuristics that construct a CPS by picking one reducible pair of the input set $\mathcal{T}$ at a time and by appending this pair to a growing partial sequence, as described in Algorithm 1 (the two subroutines PickNext and CompleteSeq will be described in details below). We call this class CPH (for Cherry-Picking Heuristics). Recall that $\mathcal{T}_S$ denotes the set of trees $\mathcal{T}$ after reducing all trees with a (partial) CPS S.

Algorithm 1 CPH.

INPUT: A set $\mathcal{T}$ of phylogenetic trees
OUTPUT: A CPS reducing $\mathcal{T}$.

1: $S \leftarrow \emptyset$;
2: **while** there is a reducible pair in $\mathcal{T}_S$ **do**
3: $(x, y) \leftarrow \mathsf{PickNext}(\mathcal{T}_S)$;
4: $S \leftarrow S \circ (x, y)$;
5: Reduce (x, y) in all trees of $\mathcal{T}_S$;
6: $S \leftarrow \mathsf{CompleteSeq}(S)$;
7: **return** S;

The while loop at lines 2-5 produces, in general, a partial CPS S, as shown in Example 4. To make it into a CPS, the subroutine CompleteSeq at line 6 appends at the end of S a sequence S' of pairs such that each second element in a pair of $S \circ S'$ is a first element in a later pair (except for the last one), as required by the definition of CPS. These additional pairs do not affect the trees in $\mathcal{T}$, that are already fully reduced by S. Algorithm 2 in Appendix A describes a procedure CompleteSeq that runs in time linear in the length of S.

▶ **Example 4.** Let $\mathcal{T}$ consist of the two 2-leaf trees (x, y) and (w, z). A partial CPS at the end of the while loop in Algorithm 1 could be, e.g., $S = (x, y), (w, z)$. The two trees are both reduced to one leaf, so there are no more reducible pairs, but S is not a CPS. To make it into a CPS it suffices to append either pair (y, z) or pair (z, y): e.g., $S \circ (y, z) = (x, y), (w, z), (y, z)$ is a CPS, and it still fully reduces the two input trees.

The class of heuristics described in Algorithm 1 is concretised in different heuristics depending on the function PickNext at line 3, which is used to choose a reducible pair at each iteration. To formulate them below we need to introduce the notions of height pair and trivial pair. Let N be a network with branch lengths and let (x, y) be a reducible pair in N. The *height pair* of (x, y) in N is a pair $(h_x^N, h_y^N) \in \mathbb{R}^2_{\geq 0}$, where $h_x^N = \ell(p(x), x)$ and $h_y^N = \ell(p(y), y)$ if (x, y) is a cherry, and $h_x^N = \ell(p(y), p(x)) + \ell(p(x), x)$ and $h_y^N = \ell(p(y), y)$ if (x, y) is a reticulated cherry. The *height* $h_{(x,y)}^N$ of (x, y) is the average $(h_x^N + h_y^N)/2$ of h_x^N and h_y^N.

Let $\mathcal{T}$ be a set of trees whose leaf sets are subsets of a set of taxa X. An ordered leaf pair (x, y) is a *trivial pair* of $\mathcal{T}$ if it is reducible in each $T \in \mathcal{T}$ that contains both x and y, and there is at least one tree in which it is reducible. We define the following three heuristics in the CPH class, resulting from as many possible implementations of PickNext.

Rand. Function PickNext picks uniformly at random a reducible pair of $\mathcal{T}_S$.

LowPair. Function PickNext picks a reducible pair (x, y) with the lowest average of values $h_{(x,y)}^T$ over all $T \in \mathcal{T}_S$ in which (x, y) is reducible (ties are broken randomly).

TrivialRand. Function PickNext picks a trivial pair if there exists one, and otherwise picks a reducible pair of $\mathcal{T}_S$ uniformly at random.

▶ **Theorem 5.** *Algorithm 1 computes a CPS that fully reduces $\mathcal{T}$, for any function PickNext that picks, in each iteration, a reducible pair of $\mathcal{T}_S$.*

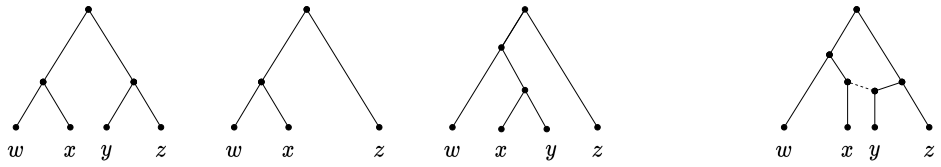
Proof. The sequence S is initiated as an empty sequence. Then, each iteration of the while loop (lines 2-5) of Algorithm 1 appends one pair to S that is reducible in at least one of the trees in $\mathcal{T}$, and reduces it in all trees. Hence, in each iteration the total size of $\mathcal{T}_S$ is reduced, so the algorithm finishes in finite time. Moreover, at the end of the while loop, each tree in $\mathcal{T}_S$ is reduced, thus the partial CPS S reduces $\mathcal{T}_S$. As CompleteSeq only appends pairs at the end of S, the result of this subroutine still reduces all trees in $\mathcal{T}_S$. ◀

In Section 5 we experimentally show that TrivialRand produces the best results among the proposed randomised heuristics. Rand is the fastest heuristic, its running time being trivially bounded by $\mathcal{O}(|\mathcal{T}|^2|X|)$ (see Lemma 9 in Appendix A). In the next section we introduce a further heuristic step for TrivialRand which improves the quality of the output even more.

3.2 Improving Heuristic TrivialRand via Tree Expansion

With respect to a trivial pair (x, y) each tree T in the set is of one of the following types: (i) (x, y) is a reducible pair of T; or (ii) neither x nor y are leaves of T; or (iii) y is a leaf of T but x is not; or (iv) x is a leaf of T but y is not.

Suppose that at some iteration of TrivialRand, the subroutine PickNext returns the trivial pair (x, y). Then, before reducing (x, y) in all trees, we do the following extra step: for each tree of type (iv), replace leaf x with cherry (x, y). We call this operation the *tree expansion* of $\mathcal{T}$: see Figure 4c. The effect of this step is that, after reducing (x, y), leaf x disappears from the set of trees, which would have not necessarily been the case before, because of trees of type (iv). The tree expansion followed by the reduction of (x, y) can, alternatively, be seen as relabelling leaf x in any tree of type (iv) by y.



(a) Input tree T. **(b)** $T_{S^1} = T_{(y,z)}$. **(c)** Expand T_{S^1} with (x, y). **(d)** Network N_T^{exp}.

Figure 4 Tree expansion of T **(a)** with the trivial cherry (x, y) of $\mathcal{T}_{(y,z)}$. **(b)** After picking cherry (y, z), leaf y is missing in T_{S^1}. **(c)** Leaf x is replaced by the cherry (x, y). After completion of the heuristic, we have $S_T = (y, z), (x, y), (y, w), (w, z)$. **(d)** The network N_T^{exp} reconstructed from S_T. Note that the input tree T is displayed in N_T^{exp} (solid edges).

To guarantee that a CPS S produced with tree expansion implies a feasible solution for Hybridization, we must show that the network N reconstructed from S displays all the trees in the input set $\mathcal{T}$. We prove that indeed this is the case with the following lemma.

▶ **Lemma 6.** *Let S be the CPS outputted by TrivialRand with input $\mathcal{T}$ and let S contain $j \geq 0$ trivial pairs. Then the network reconstructed from S displays all the trees in $\mathcal{T}$.*

Proof. We define the *network expansion* of $\mathcal{T}$ with a trivial pair, denoted by $\mathcal{T}^{exp}$, as follows. Let S^{i-1} be the partial CPS constructed in the first $i-1$ steps of TrivialRand, and let i be the first step in which we pick a trivial pair (x, y). For each $T \in \mathcal{T}$ that is reduced by S^{i-1} to a tree $T_{S^{i-1}}$ of type (iv) for (x, y), let S_T^{i-1} be the subsequence of S^{i-1} consisting only of the pairs that subsequently affect T. We use the partial CPS $S_T^{i-1} \circ (x, y)$ to reconstruct a network N_T^{exp} with a method underlying Lemma 3, starting from $T_{S^{i-1}}$: see Figure 4d. For trees of type (i)-(iii), $N_T^{exp} = T$. The set $\mathcal{T}^{exp}$ then consists of networks N_T^{exp} for all $T \in \mathcal{T}$. Note that, by construction and Lemma 3, all the elements of $\mathcal{T}^{exp}_{S^{i-1} \circ (x,y)}$ are trees.

We can generalise this notion to multiple trivial pairs: we denote by $\mathcal{T}^{exp(j)}$ the network expansion of $\mathcal{T}$ with the j-th trivial pair, (w, z) say, and suppose it is added to the partial CPS S at the k-th step. Consider a tree $T' \in \mathcal{T}^{exp(j-1)}_{S^{k-1}}$ of type (iv) for (w, z), and let $N_T^{exp(j-1)} \in \mathcal{T}^{exp(j-1)}$ be the network it originated from. Let S_T^{k-1} be the subsequence of

S^{k-1} consisting only of the pairs that subsequently affected $N_T^{exp(j-1)}$. Then $N_T^{exp(j)}$ is the network reconstructed from $S_T^{k-1} \circ (w, z)$, starting from T'. For trees of $\mathcal{T}_{S^{k-1}}^{exp(j-1)}$ that are of type (i)-(iii) for (w, z), we define $N_T^{exp(j)} = N_T^{exp(j-1)}$. The elements of $\mathcal{T}^{exp(j)}$ are all networks $N_T^{exp(j)}$. For completeness, we define $\mathcal{T}^{exp(0)} = \mathcal{T}$ and $\mathcal{T}^{exp(1)} = \mathcal{T}^{exp}$.

By construction, S fully reduces all the networks in $\mathcal{T}^{exp(j)}$, thus the network N reconstructed from S displays all of them by Theorem 2. We prove that $N_T^{exp(j)}$ displays T for all $T \in \mathcal{T}$, and thus N displays the original tree set $\mathcal{T}$ too, by induction on j.

In the base case we pick $j = 0$ trivial pairs, so the statement is true by Theorem 2. Now let $j > 0$. The induction hypothesis is that each network $N_T^{exp(j-1)} \in \mathcal{T}^{exp(j-1)}$ displays the tree T it originates from. Let (w, z) be the j-th trivial pair, added to the sequence at position k. Let $T' \in \mathcal{T}_{S^{k-1}}^{exp(j-1)}$ be a tree of type (iv) for (w, z), and let $N_T^{exp(j-1)}$ be the network it originates from. Then there are two possibilities: either z is a leaf of $N_T^{exp(j-1)}$ or it is not. In case it is not, then adding (w, z) to $N_T^{exp(j-1)}$ does not create any new reticulation, and clearly $N_T^{exp(j)}$ keeps displaying T. If z does appear in $N_T^{exp(j-1)}$, then it must have been reduced by a pair (z, v) of S^{k-1} (otherwise T' would not be of type (iv)). Then the network $N_T^{exp(j)}$ has an extra reticulation, created with the insertion of (z, v) at some point after (w, z) during the backwards reconstruction. In both cases, by [8, Lemma 10] $N_T^{exp(j-1)}$ is displayed in $N_T^{exp(j)}$, and thus by the induction hypothesis T is displayed too. ◀

3.3 Good Cherries in Theory

By Lemma 3, the binary network N reconstructed from a CPS S is such that S is of minimum length for N. By Theorem 2, if S, in turn, fully reduces $\mathcal{T}$, then N displays all the trees in $\mathcal{T}$. Depending on S, though, N is not necessarily an optimal network (that is, with minimum reticulation number) among the ones displaying $\mathcal{T}$.

Let $\mathsf{OPT}(\mathcal{T})$ denote the set of networks that display $\mathcal{T}$ with the minimum possible reticulation number. Ideally, we would like to produce a CPS fully reducing $\mathcal{T}$ that is also a minimum-length CPS fully reducing some network of $\mathsf{OPT}(\mathcal{T})$. In other words, we aim to find a CPS $\tilde{S} = (x_1, y_1), \ldots, (x_n, y_n)$ such that, for any $i \in [1, n]$, (x_i, y_i) is a reducible pair of $\tilde{N}_{\tilde{S}^{i-1}}$, where $\tilde{S}^0 = \emptyset$, $\tilde{S}^k = (x_1, y_1), \ldots, (x_k, y_k)$ for all $k \in [1, n]$, and $\tilde{N} \in \mathsf{OPT}(\mathcal{T})$.

Let $S = (x_1, y_1), \ldots, (x_n, y_n)$ be a CPS fully reducing $\mathcal{T}$ and let $\mathsf{OPT}_{S^k}(\mathcal{T})$ consist of all networks $N \in \mathsf{OPT}(\mathcal{T})$ such that each pair (x_i, y_i), $i \in [1, k]$, is reducible in $N_{S^{i-1}}$.

▶ **Lemma 7.** *A CPS S reducing $\mathcal{T}$ reconstructs an optimal network $\tilde{N}$ if and only if each pair (x_i, y_i) of S is reducible in $\tilde{N}_{S^{i-1}}$, for all $i \in [1, n]$.*

Proof. ($\implies$) By Lemma 3, S is a minimum-length CPS for the network $\tilde{N}$ that is reconstructed from it; and a CPS $C = (w_1, z_1), \ldots, (w_n, z_n)$ reducing a network N is of minimum length precisely if, for all $j \in [1, n]$, (w_j, z_j) is a reducible pair of $N_{C^{j-1}}$ (otherwise the pair (w_j, z_j) could be removed from C and the new sequence would still reduce N).
($\impliedby$) If all pairs of S affect some optimal network $\tilde{N}$, then S is a minimum-length CPS for $\tilde{N}$, thus $\tilde{N}$ is reconstructed from S (and it displays $\mathcal{T}$ by Theorem 2). ◀

Lemma 7 implies that if some pair (x_i, y_i) of S does not reduce any network in $\mathsf{OPT}_{S^{i-1}}(\mathcal{T})$, then the network reconstructed from S is not optimal: see Example 8.

▶ **Example 8.** Consider the set $\mathcal{T}$ of Figure 2b: $S = (y, x), (y, z), (w, x), (x, z)$ is a CPS that fully reduces $\mathcal{T}$ and consists only of pairs successively reducible in the network N of Fig. 2a, thus it reconstructs it by Lemma 3. Now consider (w, x), which is reducible in $\mathcal{T}$ but not in N, and pick it as first pair, to obtain e.g. $S' = (w, x), (y, z), (y, x), (w, x), (x, z)$. The network N' reconstructed from S', depicted in Figure 5, has $r(N') = 2$, whereas $r(N) = 1$.

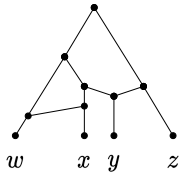


Figure 5 The network N' of Example 8.

Suppose we are incrementally constructing a CPS $S = (x_1, y_1), \ldots, (x_n, y_n)$ for $\mathcal{T}$ with some heuristic in the CPH class. If we had an oracle that at each iteration i told us if a reducible pair (x, y) of $\mathcal{T}_{S^{i-1}}$ were a reducible pair in some $N \in \mathsf{OPT}_{S^{i-1}}(\mathcal{T})$, then, by Lemma 7, we could solve Hybridization optimally. Unfortunately no such exact oracle can exist (unless $P = NP$). However, in the next section we exploit this idea to design machine-learned heuristics in the CPH framework.

4 Predicting Good Cherries via Machine Learning

In this section, we present a supervised machine-learning classifier that (imperfectly) simulates the ideal oracle described at the end of Section 3.3. The goal is to predict, based on $\mathcal{T}$, whether a given cherry of $\mathcal{T}$ is a reducible pair in a network N displaying $\mathcal{T}$ with a close-to-optimal number of reticulations, without knowing N. Based on Lemma 7, we then exploit the output of the classifier to define new functions PickNext, that in turn define new machine-learned heuristics in the class of CPH (Algorithm 1).

Specifically, we train a random forest classifier on data that encapsulates information on the cherries in the tree set. Each reducible pair in $\mathcal{T}_S$ is represented by one data point. Each data point is a pair $(\mathbf{F}, \mathbf{c})$, where $\mathbf{F}$ is an array of the features of cherry (x, y) and $\mathbf{c}$ is an array containing the probability that the cherry belongs to each of the possible classes described below. Recall that cherries are ordered pairs, so (x, y) and (y, x) give rise to two distinct data points. The classification model learns the association between $\mathbf{F}$ and $\mathbf{c}$.

The true class of a cherry (x, y) of $\mathcal{T}$ depends on whether, for the (unknown) network N that we aim to reconstruct: (class 1) (x, y) is a cherry of N; (class 2) (x, y) is a reticulated cherry of N; (class 3) (x, y) is not reducible in N, but (y, x) is a reticulated cherry; or (class 4) neither (x, y) nor (y, x) are reducible in N. Thus, for the data point of a cherry (x, y), $\mathbf{c}[i]$ contains the probability that (x, y) is in class i, and $\mathbf{c}[1] + \mathbf{c}[2]$ gives the predicted probability that (x, y) is reducible in N. We define the following two heuristics in the CPH framework.

ML. Given a threshold $t \in [0, 1)$, function PickNext picks the cherry with the highest predicted probability of being reducible in N, if this probability is at least t; or a random cherry if none of them have a probability of being reducible above the threshold.

TrivialML. Function PickNext picks a random trivial pair, if there exists one; otherwise it uses the same rules as ML.

In both cases, whenever a trivial pair is picked, we do tree expansion, as described in Section 3.2. Note that if $t = 0$, since the predicted probabilities are never exactly 0, ML is fully deterministic. In Section 5 we study the performance of ML with different thresholds.

Table 1 Features of a cherry (x, y). Features 6-12 can be computed for both branch lengths and unweighted branches. We refer to these two options as *distance* and *topological distance*, respectively.

Num	Feature name	Description
1	Cherry in tree	Ratio of trees that contain cherry (x, y)
2	Trivial	Ratio of trees with both leaves x and y that contain cherry (x, y)
3	Leaves in tree	Ratio of trees that contain both leaves x and y
4	New cherries	Number of new cherries of $\mathcal{T}$ after picking cherry (x, y)
5	Before/after	Ratio of the number of cherries of $\mathcal{T}$ before/after picking cherry (x, y)
Features measured by distance (d) and topology (t)		
$6_{d,t}$	Cherry depth	Avg over trees with (x, y) of ratios "depth of (x, y) in the tree/depth of tree"
$7_{d,t}$	Tree depth	Avg over trees with (x, y) of ratios "depth of tree/max depth of trees with (x, y)"
$8_{d,t}$	Leaf distance	Avg over trees with x and y of ratios "x-y leaf distance/depth of the tree"
$9_{d,t}$	LCA distance	Avg over trees with x and y of ratios "x-LCA(x, y) distance/y-LCA(x, y) distance"
$10_{d,t}$	Leaf depth x	Avg over trees with x and y of ratios "root-x distance/depth of tree"
$11_{d,t}$	Leaf depth y	Avg over trees with x and y of ratios "root-y distance/depth of tree"
$12_{d,t}$	Depth x/y	Avg over trees with x and y of ratios "root-x distance and root-y distance"

To assign a class to each cherry, we define 19 features, summarised in Table 1, that may capture essential information about the structure of the set of trees, and that can be efficiently computed and updated at every iteration of the heuristics.

The *depth* (resp. *topological* depth) of a node u in a tree T is the total branch length (resp. the total number of edges) on the root-to-u path; the depth of T is the maximum depth of any leaf of T; the depth of a cherry (x, y) is the depth of the common parent of x and y. The (topological) leaf distance between x and y is the total branch length of the path from the parent of x to the lowest common ancestor of x and y, denoted by LCA(x, y), plus the total length of the path from the parent of y to LCA(x, y) (resp. the total number of edges on both paths). In particular, the leaf distance between the leaves of a cherry is zero.

Features 1-5 can be initially computed for all cherries of $\mathcal{T}$ with two bottom-up traversals of all trees: one for identifying and storing all cherries of $\mathcal{T}$, the second for actually computing the features. Picking a cherry then entails recomputing features 1-2 and 4-5 for up to $|\mathcal{T}|$ data points ($|\mathcal{T}|$ denotes the number of trees), as this can result in up to one new cherry in each tree; and recomputing feature 3 for up to $|X|$ cherries, where X is the set of taxa of $\mathcal{T}$.

Features $6_{d,t}$ to $12_{d,t}$ can also be initially computed with a traversal of $\mathcal{T}$, made efficient by preprocessing each tree to allow constant-time LCA queries [6] and by storing the depth (both topological and w.r.t. branch lengths) of each node of each tree. This preprocessing also allows for efficient updates of each feature. We defer a thorough analysis of the time complexity of computing and updating the features to a future full version of this paper.

Obtaining Training Data. The high-level idea to obtain training data is to first generate a phylogenetic network N; then to extract a subset $\mathcal{T}$ of the trees displayed in N; and finally, to iteratively choose a random reducible pair (x, y) of N, to reduce it in $\mathcal{T}$ as well as in N, and to label the remaining cherries of $\mathcal{T}$ with one of the four classes defined in Section 4.

We generate phylogenetic networks with branch lengths and up to 9 reticulations using the LGT (lateral gene transfer) network generator of [14] for binary orchard networks. For each such network N, we then generate the set $\mathcal{T}$ consisting of all the exhaustive trees displayed in N. Although N is not guaranteed to be an optimal network for $\mathcal{T}$, we expect it to be reasonably close to an optimal one, because we remove redundant reticulations when we generate it and because the trees in $\mathcal{T}$ cover all the edges of N. In particular, $r(N)$ provides an upper bound estimate on the minimum possible number of reticulations of any network displaying $\mathcal{T}$. We will thus use it as a reference value for assessing the quality of our results on synthetic data.

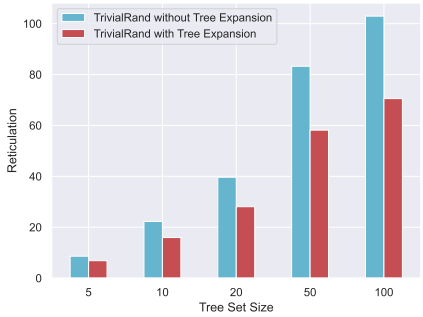


Figure 6 TrivialRand with and without tree expansion. The height of the bars is the average reticulation number over each group, obtained by selecting the best of 200 runs for each instance.

5 Experiments

The code of all our heuristics, available at `https://github.com/estherjulien/HybridML`, is written in Python. All experiments ran on an Intel Xeon Gold 6130 CPU @ 2.1 GHz with 96 GB RAM. We conducted experiments on both synthetic and real data, comparing the performance of Rand, TrivialRand, ML and TrivialML, using threshold $t = 0$. Although our methods can be applied to trees with different leaf sets, we restrict this proof-of-concept study to trees with the same set of leaves, and defer more complete experiments to future extensions. Similar to the training data, we generated two synthetic datasets by first growing a binary orchard network N using [14], and then extracting $\mathcal{T}$ as a subset of the exhaustive trees displayed in N. We provide details on each dataset in Section 5.2.

We start by analysing the usefulness of tree expansion, the heuristic rule described in Section 3.2. We synthetically generated 112 instances for each tree set size $|\mathcal{T}| \in \{5, 10, 20, 50, 100\}$ (560 in total), all consisting of trees with 20 leaves each, and grouped them by $|\mathcal{T}|$; we then ran TrivialRand 200 times (both with and without tree expansion) on each instance, selected the best output for each of them, and finally took the average of these results over each group of instances. The results are in Figure 6, showing that the use of tree expansion brought the output reticulation number down by at least 16% (for small instances) and up to 40% for the larger instances. We consistently chose to use this rule in all the heuristics that detect trivial cherries, namely, TrivialRand, TrivialML, and ML (although ML does not explicitly favour trivial cherries, it does check whether a selected cherry is trivial using feature number 2).

5.1 Prediction Model

The random forest is implemented with Python's `scikit-learn` [13] package using default settings. We evaluated the performance of our trained random forest models on different datasets in a holdout procedure: namely, we removed 10% of the data from each training dataset, trained the models on the remaining 90% and used the holdout 10% for testing. The accuracy was assessed by assigning to each test data point the class with the highest predicted probability, and comparing it with the true class. Before training the models, we balanced each dataset so that each class had the same number of representatives.

Each training dataset differed in terms of the number M of networks used for generating it and the number of leaves of the networks. For each dataset, the number L of leaves of each generated network was uniformly sampled from $[2, \max L]$, where $\max L$ is the maximum

number of leaves per network. We constructed the networks using the LGT generator of [14]. This generator has three parameters: n for the number of steps, α for the probability of lateral gene transfer events, and β for regulating the size of the biconnected components of the network (called *blobs*). The combination of these parameters determines the level (maximum number of reticulations per blob), the number of reticulations, and the number of leaves of the output network. In our experiments, α was uniformly sampled from $[0.1, 0.5]$ and $\beta = 1$. See [14] for more details.

Each generated network gave rise to a number of data points: the total number of data points per dataset is shown in Table 5 in Appendix C. Each row of Table 5 corresponds to a dataset on which the random forest can be trained, obtaining as many ML models. We tested all the models on all the synthetically generated instances: we show these results in Figure 14 in Appendix D. In Section 5.2 we will report the results obtained for the best performing model for each type of instance.

Among the advantages of using a random forest as prediction model there is the ability of computing feature importance, shown in Table 6 in Appendix C. Some of the most useful features for a cherry (x, y) seem to be "Trivial" (the ratio of the trees containing both leaves x and y in which (x, y) is a cherry) and "Cherry in tree" (the ratio of trees that contain (x, y)). This was not unexpected, as these features are well-suited to identify trivial cherries.

'Leaf distance' (the average, over all trees containing both leaves x, y, of the ratio between the x-to-y distance and the depth of the tree) and "Depth x/y" (the average, over all trees containing both leaves x, y, of the ratio between the x-to-root and the y-to-root distances) are also two important features. The rationale behind these features was to try to identify reticulated cherries. This was also the idea for the feature "Before/after", but this has, surprisingly, a very low importance score. In future extensions of this work we plan to conduct a thorough analysis on whether some of the seemingly least important features can be removed without affecting the quality of the results.

5.2 Experimental Results

We assessed the performance of our heuristics on instances of three types: Full Tree Set (FTS), Restricted Tree Set (RTS), and real data. FTS and RTS are synthetically generated. We generated the FTS instances much like we did for the training data: we first grew a network with the LGT generator and then extracted all the exhaustive trees displayed in the network. We generated FTS data for different combinations of the following parameters: $L \in \{20, 50, 100\}$ (number of leaves per tree) and $R \in \{5, 6, 7\}$ (reticulation number of the original network). Note that, for FTS instances, $|\mathcal{T}| = 2^R$.

For generating RTS instances, we also started by growing LGT networks, but we then extracted only a subset of the exhaustive trees from each of them, up to a certain amount $|\mathcal{T}| \in \{20, 50, 100\}$; the other parameter for RTS instances is the number of leaves $L \in \{20, 50, 100\}$, while the number of reticulations is uniformly sampled in the ranges $[5, 25]$, $[6, 30]$, $[7, 35]$ for $|\mathcal{T}| = 20$, $|\mathcal{T}| = 50$ and $|\mathcal{T}| = 100$, respectively. We generated 112 FTS instances and as many RTS instances for each possible combination of the parameters: by *instance group* we indicate all the (FTS or RTS) instances generated for one parameter pair.

The real-world dataset we tested our methods on consists of gene trees on homologous gene sets found in bacterial and archaeal genomes. This dataset was originally constructed in [3] and made binary in [19]. We extracted a subset of instances (Table 2) from the binary dataset, for every combination of parameters $L \in \{20, 50, 100\}$ and $|\mathcal{T}| \in \{10, 20, 50, 100\}$.

Table 2 Number of real data instances for each group (combination of parameters L and $|\mathcal{T}|$).

L	$\|\mathcal{T}\|$				Tot. Trees
	10	20	50	100	
20	50	50	50	50	1684
50	20	20	20	20	290
100	5	5	1	0	53

For the synthetically generated FTS and RTS datasets, we evaluated the performance of each heuristic in terms of the output number of reticulations, comparing it with the number of reticulations of the network N from which we extracted $\mathcal{T}$. Indeed, although N is not guaranteed to be an optimal network for $\mathcal{T}$, $r(N)$ clearly provides an estimate (from above) of the optimal value, and thus we used it as a reference value for our experimental evaluation.

For real data we used a different reference value. In the absence of the natural estimate on the optimal number of reticulations provided by the starting network, on real data we evaluated the performance of the heuristics using the best result obtained by running TrivialRand 1000 times as reference value. We also compared our results with the algorithm from [19], denoted by TreeChild, and the algorithm from [1], denoted by Hybroscale, using the same datasets that were used to test the two methods in [19], which consist of rather small instances ($|\mathcal{T}| \leq 8$) .

Full Tree Set (FTS) instances. We evaluated the output of ML with threshold $t = 0$ on all the random forest models described in Table 5. The model that gave the best results for datasets of this type is the one trained with parameters $\max L = 100, M = 500$ (see Figure 14a), so we chose to use this model for both ML and TrivialML, and compare them to Rand and TrivialRand. We ran the machine-learned heuristics once for each instance, and then averaged the results within each group. The randomised heuristics Rand and TrivialRand were run $\min\{x(I), 1000\}$ times for each instance I, where $x(I)$ is the number of runs that can be executed in the same time as one run of ML on the same instance.

In Figure 7 we summarise the results for the four heuristics. Solid bars represent the ratio of the average reticulation number to the reference value, for each instance group and for each of the four heuristics. Dashed bars represent the ratio of the average (within each group) of the best of the $\min\{x(I), 1000\}$ runs for each instance I to the reference value. The machine-learned heuristics ML and TrivialML seem to perform very similarly, the best one (on average) being one or the other depending on the instance group. The fully randomised heuristic, Rand, always performed much worse than all the others, and we omitted the results for LowPair because they were at least 44% worse, on average, than the ones for Rand.

While just one run of ML or TrivialML gave better results than the average of multiple runs of TrivialRand (up to 64% on average), picking the best output over all the runs of TrivialRand seems to give the best results for FTS instances, TrivialRand being better than ML by up to 25% on average for instances obtained from networks with 20 leaves and 5 reticulations and yielding results only slightly better than ML and TrivialML in the rest of instance groups (ML being better by 3% on average than the best output of TrivialRand only for the instance group (100,7)).

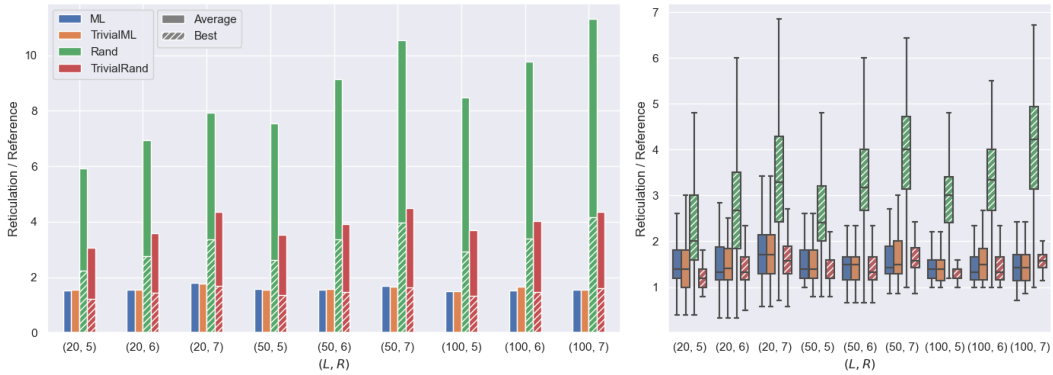


Figure 7 Results for FTS instances for each combination of the parameters (L, R) and distribution of the results per instance group (boxplots on the right, for the randomized heuristics only considering the best result over all runs on each instance).

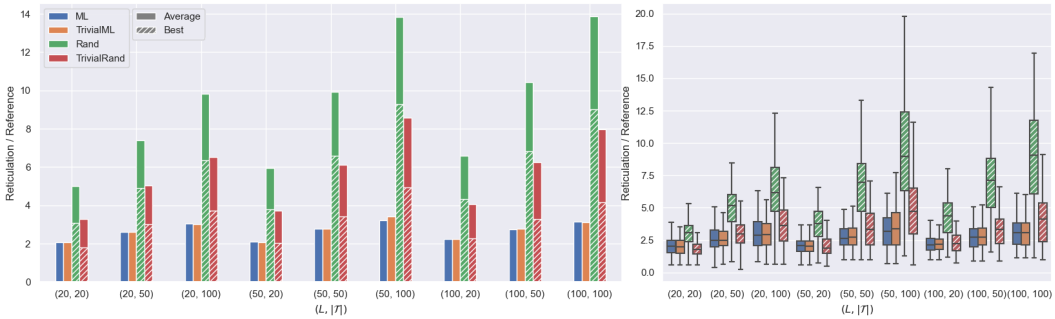


Figure 8 Results for RTS instances for each combination of parameters L and $|\mathcal{T}|$ and distribution of the results per instance group (boxplots on the right, for the randomized heuristics only considering the best result over all runs on each instance).

Restricted Tree Set (RTS) instances. The ML-model that gave the best results for RTS datasets is the one trained with parameters $\max L = 100, M = 10$ (Figure 14b in Appendix D). The results in Figure 8 were obtained with the same procedure as for FTS. Comparing Figures 7 and 8 it is clear that, for all the heuristics, it was more difficult to reconstruct N when the input trees were not the totality of the exhaustive trees displayed in N.

In particular, whereas running TrivialRand several times and picking the best output appeared to be the winning strategies for most of the FTS instances (although ML and TrivialML performed almost as good), this is not the case for RTS, where it seems that machine learning often makes up for the lack of information in the input better than the randomised strategies. For RTS instances, the result of ML was better than the best result of TrivialRand by up to 34% on average (for instances of 100 trees obtained from networks with 50 leaves), the difference being more marked for seemingly "difficult" instance groups, for which all the tested heuristics gave results diverging more from the reference value. The best result of TrivialRand was (slightly) better than the result of ML only for the instance group (20,20).

Real Data. We conducted two sets of experiments on real data, using the ML model trained on the dataset with parameters $\max L = 100, M = 10$. For sufficiently small instances, we compared the results of our heuristics with the results of two existing tools for reconstructing

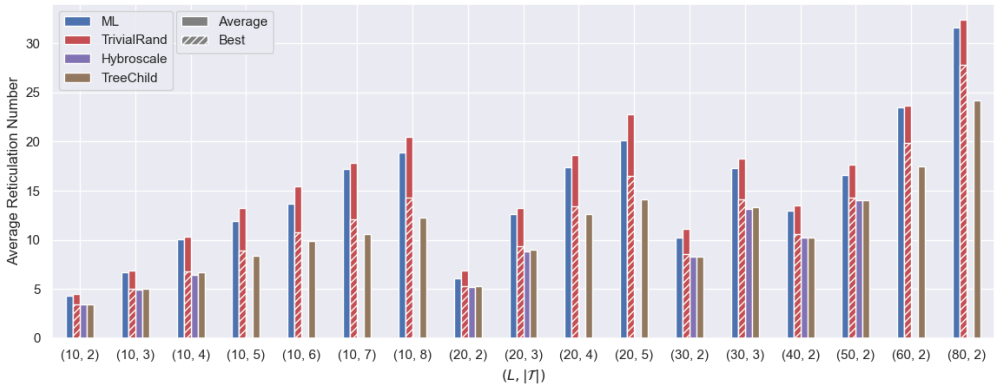


Figure 9 Comparison of ML, TrivialRand, Hybroscale, and TreeChild on real data. The distribution of the results is shown in Appendix B (Figure 12).

networks from binary trees: TreeChild [19] and Hybroscale [1]. Hybroscale is an exact method performing an exhaustive search on the networks displaying the input trees, therefore it can only handle reasonably small instances in terms of number of input trees. TreeChild is a fixed-parameter (in the number of reticulations of the output) exact algorithm that reconstructs the best *tree-child* network, a restricted class of phylogenetic networks, and due to its fast-growing computation time cannot handle large instances either.

We tested ML and TrivialRand against Hybroscale and TreeChild using the same dataset used in [19], in turn taken from [3]. The dataset consists of ten instances for each possible combination of the parameters $|\mathcal{T}| \in [2, 8]$ and $L \in \{10, 20, 30, 40, 50, 60, 80, 100, 150\}$. In Figure 9 we show results only for the instance groups for which Hybroscale or TreeChild could output a solution within 1 hour, consistent with the experiments in [19]. As a consequence of Hybroscale and TreeChild being exact methods (TreeChild only for a restricted class of networks), they performed better than both ML and TrivialRand on all instances they could solve, although the best results of TrivialRand are often close (no worse than 14%) and sometimes match the optimal value. Table 4 in the appendix summarises the running time of all the methods on all instance groups.

The main advantage of our heuristics is that they can handle much larger instances. In Figure 10 we show the results obtained for the instance groups of Table 2. Like for FTS and RTS data, Rand and TrivialRand are executed $\min\{x(I), 1000\}$ times for each instance I, $x(I)$ being the time required for executing ML once on instance I. The results are then averaged within each group and divided by the reference value that we obtained by running TrivialRand 1000 times on each instance and taking the best outputs. For this reason, in contrast with the synthetic datasets, for which we had more accurate estimates, the results of ML and TrivialML are even better (up to 10% on average) than the reference value, for some instance groups. The results essentially confirm the ones we obtained for RTS instances: the best outputs of TrivialRand are quite close to the outputs of ML (and TrivialML), being up to 15% worse and up to 9% better on average, depending on the instance group.

Effect of the Threshold on ML. We tested the effectiveness of adding a threshold $t > 0$ to ML on different types of data (FTS, RTS and real). For these experiments we used a subset of the instance groups, consisting of trees obtained using the parameters $R = 5$ for

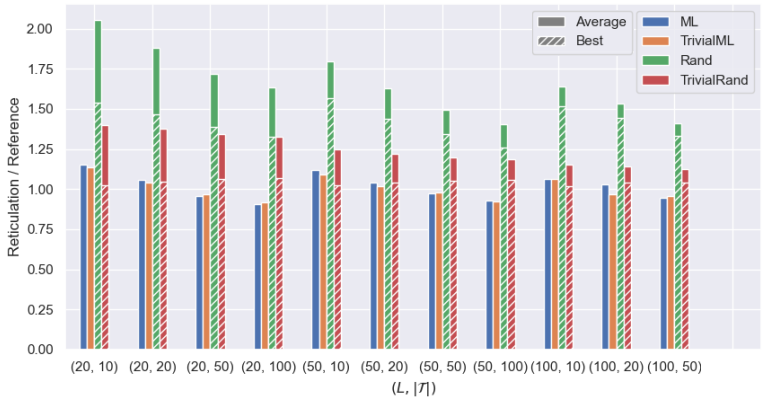


Figure 10 Results for the real instance class for each combination of parameters L and $|\mathcal{T}|$. The distribution of the results is shown in Appendix B (Figure 13.)

FTS, $|\mathcal{T}| = 20$ for RTS, $|\mathcal{T}| = 10$ for the real instances, and $L \in \{20, 50, 100\}$ for all three data types. We ran ML ten times for each threshold $t \in \{0, 0.1, 0.3, 0.5, 0.7\}$ on each instance, took the lowest reticulation number, and averaged these results within each instance group.

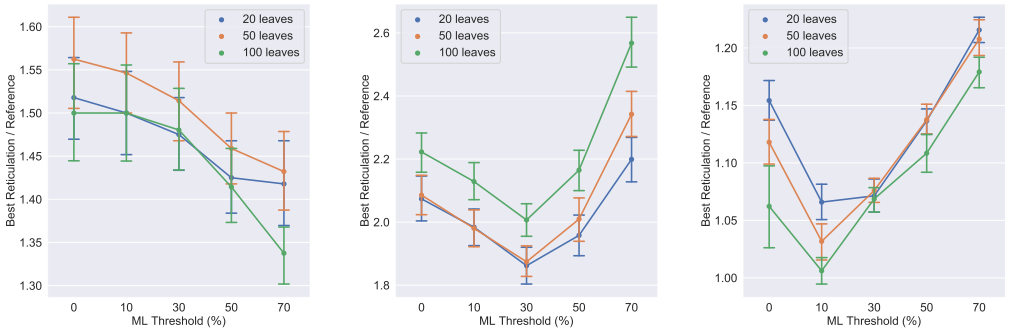
The results are shown in Figure 11. For all types of data a low threshold $t = 0.1$ is beneficial, intuitively indicating that when the probability of a pair being reducible is close to zero it gives no meaningful indication, and thus random choices among these pairs are more suited. For all but one instance group (5 real data instances each consisting of 10 trees with 100 leaves) this is true up to $t = 0.3$, and for all the synthetic datasets even up to $t = 0.5$. The seemingly best value for the threshold, though, is different for each type of instances.

The FTS instances seem to benefit from quite high values of t, the best being the highest tested value, $t = 0.7$. This is consistent with what we observed before: some randomness in the methods seems to be very effective. For the synthetic but less informative RTS instances, the best threshold seems to be around $t = 0.3$, while very high values ($t = 0.7$) are counterproductive. This is also coherent with the fact that the randomised heuristics are less effective on this types of instances. For real data, the best option seems to be around $t = 0.1$.

These experiments should be seen as an indication that the use of an appropriate threshold may improve the performance of the machine-learned heuristics, at the price of running them multiple times. We defer a more thorough analysis to future extensions of this work.

6 Discussion

Our contributions are twofold: first, we presented the first methods that allow reconstructing a phylogenetic network from a large set of large binary phylogenetic trees. Second, we show the promise of the use of machine learning in this context. Our experimental studies indicate that repeated runs of simple and fast randomised cherry-picking heuristics are quite effective in most of the cases, but machine-learned strategies bring better results in the most difficult instances. Furthermore, preliminary experiments indicate that their performance can even be improved by introducing appropriate thresholds, in fact mediating between random choices and predictions. In addition, our current implementations are in Python and hence not optimised for speed, but faster implementations could make machine-learned heuristics with nonzero thresholds even more effective.



(a) FTS Instances. **(b)** RTS Instances. **(c)** Real Instances.

Figure 11 The reticulation number when running ML with different thresholds for the three instances classes. Each instance (in total 112) is run 10 times, where the lowest reticulation value of these runs is selected. At each point, a 68% confidence interval is also shown.

For future work we would like to evaluate if there is any relationship between the accuracy of the machine-learned models and the probability of the heuristic being able to reconstruct an optimal network. We will also investigate if the use of other parameters to introduce some randomness in strategies using machine learning can further improve the results. Finally, we plan to test our methods on trees with different leaf sets and to extend them to nonbinary trees.

References

1 Benjamin Albrecht. Computing all hybridization networks for multiple binary phylogenetic input trees. *BMC bioinformatics*, 16(1):1–15, 2015.

2 Dana Azouri, Shiran Abadi, Yishay Mansour, Itay Mayrose, and Tal Pupko. Harnessing machine learning to guide phylogenetic-tree search algorithms. *Nature communications*, 12(1):1–9, 2021.

3 Robert G Beiko. Telling the whole story in a 10,000-genome world. *Biology Direct*, 6(1):1–36, 2011.

4 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.

5 Sander Borst, Leo van Iersel, Mark Jones, and Steven Kelk. New FPT algorithms for finding the temporal hybridization number for sets of phylogenetic trees. *Algorithmica*, 2022.

6 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984. doi:10.1137/0213024.

7 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.

8 Remie Janssen and Yukihiro Murakami. On cherry-picking and network containment. *Theoretical Computer Science*, 856:121–150, 2021.

9 Sudhir Kumar and Sudip Sharma. Evolutionary sparse learning for phylogenomics. *Molecular Biology and Evolution*, 38(11):4674–4682, 2021.

10 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.

11 Sajad Mirzaei and Yufeng Wu. Fast construction of near parsimonious hybridization networks for multiple phylogenetic trees. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 13(3):565–570, 2015.

12 Fabio Pardi and Celine Scornavacca. Reconstructible phylogenetic networks: do not distinguish the indistinguishable. *PLoS computational biology*, 11(4):e1004135, 2015.

13 F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

14 Joan Carles Pons, Celine Scornavacca, and Gabriel Cardona. Generation of level-k LGT networks. *IEEE/ACM transactions on computational biology and bioinformatics*, 17(1):158–164, 2019.

15 Charles Semple and Gerry Toft. Trinets encode orchard phylogenetic networks. *Journal of Mathematical Biology*, 83(3):1–20, 2021.

16 Claudia Solís-Lemus, Paul Bastide, and Cécile Ané. Phylonetworks: a package for phylogenetic networks. *Molecular biology and evolution*, 34(12):3292–3298, 2017.

17 Leo van Iersel, Remie Janssen, Mark Jones, and Yukihiro Murakami. Orchard networks are trees with additional horizontal arcs. *Bulletin of Mathematical Biology*, 2022. to appear.

18 Leo van Iersel, Remie Janssen, Mark Jones, Yukihiro Murakami, and Norbert Zeh. A unifying characterization of tree-based networks and orchard networks using cherry covers. *Advances in Applied Mathematics*, 129:102222, 2021. doi:10.1016/j.aam.2021.102222.

19 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. *Algorithmica*, 84:917–960, 2022.

20 Dingqiao Wen, Yun Yu, Jiafan Zhu, and Luay Nakhleh. Inferring phylogenetic networks using phylonet. *Systematic biology*, 67(4):735–740, 2018.

21 Yufeng Wu. Close lower and upper bounds for the minimum reticulate network of multiple phylogenetic trees. *Bioinformatics*, 26(12):i140–i148, 2010.

22 Tujin Zhu and Yunpeng Cai. Applying neural network to reconstruction of phylogenetic tree. In *ICMLC 2021: 13th International Conference on Machine Learning and Computing, Shenzhen China, 26 February, 2021- 1 March, 2021*, pages 146–152. ACM, 2021. doi:10.1145/3457682.3457704.

A Omitted Pseudocode and Proofs

Algorithm 2 summarises a procedure to complete a partial CPS S to obtain a CPS S'.

Algorithm 2 CompleteSeq.

INPUT: A partial CPS $S = (x_1, y_1), \ldots, (x_n, y_n)$ that reduces $\mathcal{T}$
OUTPUT: A CPS S' for $\mathcal{T}$.

$C \leftarrow \emptyset$; $P \leftarrow \emptyset$;
for $i = n, \ldots, 1$ **do**
  if $y_i \notin C$ **then**
    $P \leftarrow P \cup \{y_i\}$;
  $C \leftarrow C \cup \{x_i, y_i\}$;
$S' \leftarrow S$;
while $|P| > 1$ **do**
  Let r_1 and r_2 be two arbitrary elements of P;
  $S' \leftarrow S' \circ (r_1, r_2)$;
  $P \leftarrow P \setminus \{r_1\}$;
return S'

▶ **Lemma 9.** *The running time of a naive implementation of* Rand *is* $\mathcal{O}(|\mathcal{T}|^2|X|)$.

Proof. An upper bound for the length of the sequence is $(|X|-1)|\mathcal{T}|$ as each tree can individually be fully reduced using at most $|X|-1$ pairs. Hence, the while loop of Algorithm 1 is executed at most $(|X|-1)|\mathcal{T}|$ times. Furthermore, choosing a random reducible pair takes $\mathcal{O}(1)$ time, and reducing a pair in all trees in $\mathcal{T}$ takes $\mathcal{O}(|\mathcal{T}|)$ time, as it takes constant time within each tree. Combining this with the fact that CompleteSeq takes $\mathcal{O}(|S|) = \mathcal{O}(|X||\mathcal{T}|)$ time, we conclude that Rand takes no more than $\mathcal{O}(|\mathcal{T}|^2|X|)$ time. ◀

B Runtimes and Omitted Plots

Table 3 Runtimes (in seconds) of ML for instance types FTS, RTS, and real, for all instance groups and for different parameter combinations. The number of leaves per tree are given in column "L". For FTS instances, the size of the tree set is determined by R (reticulation number of the original network). For RFT and real instances, this is simply denoted by $|\mathcal{T}|$.

L	FTS – R			RTS – $\lvert\mathcal{T}\rvert$			Real – $\lvert\mathcal{T}\rvert$			
	5	6	7	20	50	100	10	20	50	100
20	18.86	32.80	63.23	21.60	49.10	87.97	55.24	96.49	216.15	419.84
50	62.28	115.78	227.26	55.74	125.80	251.10	65.80	120.33	303.40	649.32
100	204.20	375.73	744.92	146.45	338.48	661.54	98.84	172.26	407.26	-

Table 4 Runtimes of CPH (same for ML and TrivialRand), Hybroscale, and TreeChild in seconds, for combinations of parameters L (leaves per tree) and $|\mathcal{T}|$ (number of trees).

L	$\lvert\mathcal{T}\rvert$	CPH	Hybroscale	TreeChild
10	2	3.404	0.211	0.009
	3	3.668	6.206	0.008
	4	4.844	251.880	0.073
	5	5.308	-	0.556
	6	5.851	-	4.084
	7	6.762	-	12.870
	8	7.350	-	249.799
20	2	5.591	0.248	0.008
	3	7.194	7.991	0.021
	4	8.897	-	0.943
	5	9.604	-	0.858
30	2	8.373	0.393	0.025
	3	10.348	114.656	0.173
40	2	9.276	0.365	0.016
50	2	11.794	0.729	0.038
60	2	15.043	-	463.171
80	2	19.926	-	1372.930

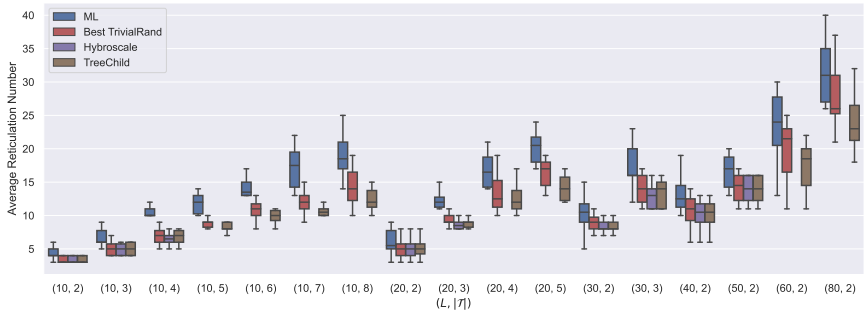


Figure 12 Boxplots for the distribution of the results of Figure 9.

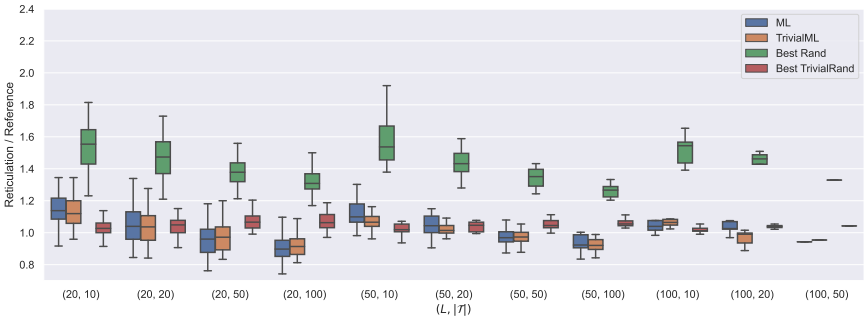


Figure 13 Boxplot for the distribution of the results of Figure 10.

C Random Forest Models

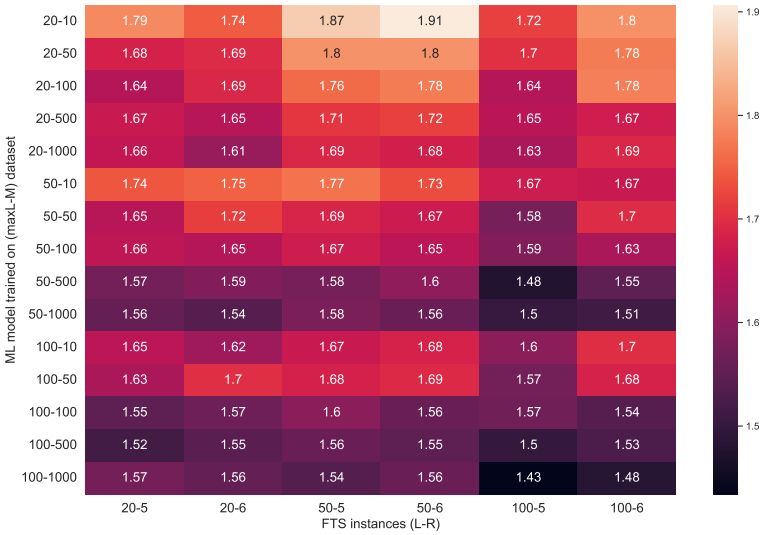
Table 5 Trained random forest models on different datasets for different combinations of max L (maximum number of leaves per network) and M (number of total networks). Each row in the table represents one model. For each model, the testing accuracy is given under "Accuracy", the total number of data points retrieved from all M networks is given under "Num. data". Each dataset is split for training and testing ($90\% - 10\%$). The training duration for the random forest is given in column "Train time" and the time needed to generate the training data is given in column "Datagen time", in hours per core (we used 16 cores in total).

max L	M	Accuracy	Num. data	Train time (min)	Datagen time (hour/core)
20	10	0.968	1,896	00:00	00:00:48
	50	0.984	6,312	00:01	00:02:23
	100	0.983	12,060	00:02	00:04:13
	500	0.983	59,236	00:13	00:23:33
	1000	0.979	114,268	00:27	00:45:19
50	10	0.986	3,556	00:00	00:03:04
	50	0.995	21,228	00:02	00:17:11
	100	0.992	44,136	00:05	00:37:52
	500	0.995	218,256	00:49	02:26:56
	1000	0.995	417,624	02:36	04:29:13
100	10	0.997	14,224	00:02	00:19:19
	50	0.997	42,628	00:07	00:55:42
	100	0.998	105,540	00:22	01:58:03
	500	0.998	529,708	03:59	08:35:33
	1000	0.998	1,098,280	08:09	16:48:45

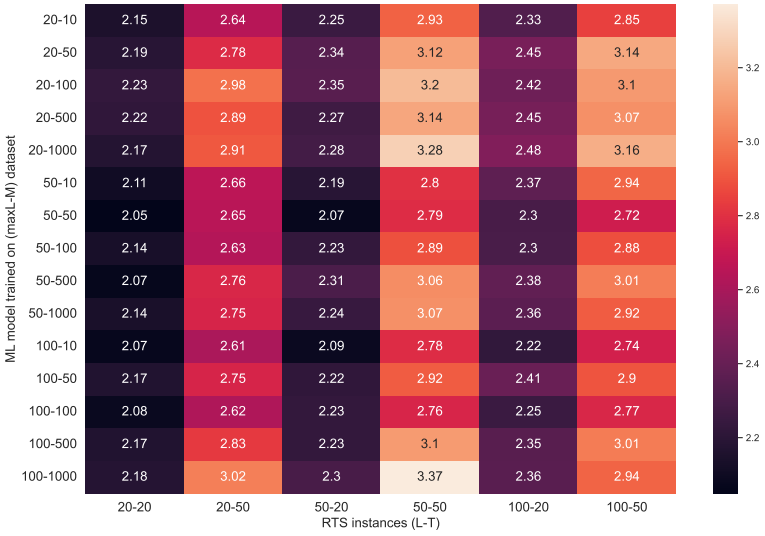
Table 6 Feature importances of random forest trained on the biggest dataset ($M = 1000$ and max $L = 100$). A higher "Importance" corresponds to that feature having more effect on the trained model. The values sum up to one. The descriptions of the features are given in Table 1.

Features		Importance
Trivial		0.183
Leaf distance	(t)	0.158
Cherry in tree		0.152
Leaf distance	(d)	0.127
Depth x/y	(d)	0.095
LCA distance	(t)	0.050
Depth x/y	(t)	0.049
LCA distance	(d)	0.036
Cherry depth	(t)	0.030
Leaf depth y	(t)	0.018
Leaf depth x	(t)	0.017
Leaf depth y	(d)	0.015
Leaf depth x	(d)	0.015
Cherry depth	(d)	0.014
New cherries		0.013
Tree depth	(d)	0.011
Tree depth	(t)	0.009
Before/after		0.005
Leaves in tree		0.003

D Heuristic Performance of ML Models



(a) FTS Instances.



(b) RTS Instances.

Figure 14 Results for ML with the random forest model trained on each of the datasets given in Table 5. For each training dataset, identified by the parameter pair max L-M, the value shown in the heatmap is the average, within each instance group, of the reticulation number found by ML divided by the reference value. We used a group of 112 instances for each combination of parameters $L \in \{20, 50, 100\}$ and $R \in \{5, 6\}$ (for FTS), and $L \in \{20, 50, 100\}$ and $|\mathcal{T}| \in \{20, 50\}$ (for RTS).