Contents lists available at ScienceDirect

Molecular Phylogenetics and Evolution

journal homepage: www.elsevier.com/locate/ympev

Inferring phylogenetic networks from multifurcating trees via cherry picking and machine learning

Giulia Bernardini[a,*], Leo van Iersel[b], Esther Julien[b,*], Leen Stougie[c,d,e]

[a] University of Trieste, Trieste, Italy
[b] Delft Institute of Applied Mathematics, Delft, The Netherlands
[c] CWI, Amsterdam, the Netherlands
[d] Vrije Universiteit, Amsterdam, The Netherlands
[e] INRIA-Erable, France

ARTICLE INFO

Keywords:
Hybrid phylogeny
Hybridization problem
Cherry-picking
Machine learning
Heuristic

ABSTRACT

The Hybridization problem asks to reconcile a set of conflicting phylogenetic trees into a single phylogenetic network with the smallest possible number of reticulation nodes. This problem is computationally hard and previous solutions are limited to small and/or severely restricted data sets, for example, a set of binary trees with the same taxon set or only two non-binary trees with non-equal taxon sets. Building on our previous work on binary trees, we present FHyNCH, the first algorithmic framework to heuristically solve the Hybridization problem for large sets of multifurcating trees whose sets of taxa may differ. Our heuristics combine the cherry-picking technique, recently proposed to solve the same problem for binary trees, with two carefully designed machine-learning models. We demonstrate that our methods are practical and produce qualitatively good solutions through experiments on both synthetic and real data sets.

1. Introduction

Until recently, the evolutionary history of a set of species was normally modeled as a rooted phylogenetic tree. However, the greater availability of molecular data is encouraging a paradigm shift to multilocus approaches for phylogenetic inference, which often leads to discovering relationships among the species that deviate from the simple model of a tree (Huson et al., 2010; Nakhleh, 2010; Bapteste et al., 2013). Indeed, the phylogenetic trees inferred from different loci of the genomes often have conflicting branching patterns, due to evolutionary events like recombination, hybrid speciation, introgression or lateral gene transfer (Randal Linder and Rieseberg, 2004; Mallet, 2005; Boto, 2010; Mallet et al., 2016). In the presence of such events, evolution is more accurately represented by a rooted phylogenetic network, which extends the tree model and allows representing multi-parental inheritance of genetic material as *reticulation* nodes (Randal Linder et al., 2004; Mallet et al., 2016).

A crucial problem is then to infer a single phylogenetic network from a set of conflicting trees built from different loci of the genomes in a data set. A commonly used criterion to estimate such a network, which is reasonable when discordance between trees is believed to be caused by multi-parent inheritance, is *parsimony* (Huson et al., 2010): the goal is then to construct a network that simultaneously explains all ancestral relationships encoded by the trees with the fewest number of reticulation nodes. This problem is known in the literature by the name of Hybridization and has been extensively studied.

Hybridization has been shown to be NP-hard even for two binary input trees (Bordewich and Semple, 2007). Most of the solutions proposed in the literature are limited to inputs consisting of only two binary trees with identical leaf sets. A few methods exist that waive some of these assumptions: some admit inputs consisting of several binary trees with identical (van Iersel et al., 2022) or largely overlapping (Bernardini et al., 2022; Bernardini et al., 2023) leaf sets; others are able to process a pair of multifurcating (i.e. nonbinary) trees with overlapping, but not identical, leaf sets (Huson and Linz, 2018) or several multifurcating trees with identical leaf sets (Yufeng, 2010; Mirzaei and Yufeng, 2015).

However, to the best of our knowledge, there currently exist no solutions to Hybridization for several multifurcating trees with different leaf sets, although realistic phylogenetic trees in biological studies are usually multifurcating and hardly contain exactly the same taxa. This

* Corresponding authors.
E-mail addresses: giulia.bernardini@units.it (G. Bernardini), E.A.T.Julien@tudelft.nl (E. Julien).

https://doi.org/10.1016/j.ympev.2024.108137
Received 18 November 2023; Received in revised form 19 February 2024; Accepted 29 June 2024
Available online 17 July 2024

work aims to fill this gap: we propose FHyNCH[1] (Finding Hybridization Networks via Cherry-picking Heuristics), a heuristic framework to find feasible (and qualitatively good) solutions to Hybridization for a large number of multifurcating phylogenetic trees with overlapping, but not identical, leaf sets. Our methods combine the technique of *cherry picking*, first introduced in (Linz and Semple, 2019), with a machine-learning model to guide the search in the solution space.

The high-level scheme and the theoretical foundations of the methods we propose are the same as in (Bernardini et al., 2022): however, the approach of Bernardini et al. (2022) is restricted to binary trees, while most practical data sets consist of multifurcating trees. A straightforward adaptation to multifurcating trees would lead to a time-consuming algorithm that would be impractical for large instances (see Section 2.3.2). In contrast with previous methods, our new heuristics employ two machine-learning classifiers that are used sequentially at every iteration. The main novelty resides in the design and use of the first classifier, whose crucial role is to reduce the solution space at every iteration. Furthermore, making the new machine-learned heuristics applicable to multifurcating trees with missing leaves required new, nontrivial techniques to generate training data: see Section 2.3.4.

Two things are important to notice at this point. First, since networks are not uniquely determined by the trees they contain (Pardi and Scornavacca, 2015), there may exist a large number of different optimal solutions, and our algorithm does not attempt to enumerate them all: in fact, how to summarize all equally good networks is still an open practical problem (Huson and Scornavacca, 2012; Huber et al., 2021). In particular, no method to solve Hybridization (whose goal is just to minimize the number of reticulations) can guarantee to reconstruct a specific network: all networks that display the input trees with the minimum possible number of reticulation nodes are optimal solutions. The network outputted by any algorithm that solves Hybridization, including ours, should thus be interpreted as a possible (parsimonious) evolutionary history which is consistent with all the input trees.

Second, our heuristics output networks from the broad *orchard* class, which contains all and only the networks that can be obtained from a tree by adding horizontal arcs (van Iersel et al., 2022). Such horizontal arcs can model lateral gene transfer (LGT) events, but also many networks with reticulation nodes modelling (for example) hybridization events are in the class of orchard networks. On the other hand, our methods are not suitable to be applied in the presence of incomplete lineage sorting.

The rest of the paper is organized as follows. In Section 1.1 we discuss related work; in Section 2.1 we introduce notation and basic notions; in Section 2.2 we summarize the cherry-picking framework for Hybridization, which lies at the heart of our solutions; in Section 2.3 we describe FHyNCH-MultiML, our main algorithmic scheme based on machine learning; in Section 3 we present our experimental results; finally, in Section 4 we give conclusions and future directions.

1.1. Related work

Several methods have been proposed in the literature to solve Hybridization for two binary trees with equal leaf sets, both exactly (Bordewich and Semple, 2007; Albrecht et al., 2012) and heuristically (Park et al., 2010; Park et al., 2012). The first practical methods to solve Hybridization to optimality for more than two binary trees with equal leaf sets were PIRN$_C$ (Yufeng, 2010) and Hybroscale (Albrecht, 2015), which were able to process a small number of input trees (up to 5) that could be combined into a network with a relatively small number of reticulations. More recently, heuristic methods have been proposed to process larger sets of binary trees with identical taxa (Mirzaei and Yufeng, 2015; Zhang et al., 2023).

The introduction of the so-called cherry-picking sequences (Humphries et al., 2013; Linz and Semple, 2019) was a game changer in the area: this theoretical framework allowed the design of the first methods capable of processing instances of up to 100 binary trees with identical leaf sets to optimality (van Iersel et al., 2022; Borst et al., 2022), albeit with restrictions on the class of the output network and its number of reticulations.

To the best of our knowledge, only two methods have been proposed to solve Hybridization for multifurcating trees, both limited to inputs consisting of only two trees: a simple FPT algorithm for trees with identical leaf set (Piovesan and Kelk, 2012) and the Autumn algorithm, which allows differences between the leaf sets (Huson and Linz, 2018).

The potential of machine learning in phylogenetic studies has not been extensively explored yet. A few methods have been proposed for phylogenetic tree inference (Abadi et al., 2020; Azouri et al., 2021; Zhu and Cai, 2021; Kulikov et al., 2023; Smith and Hahn, 2023; Azouri et al., 2023), testing evolutionary hypotheses (Kumar and Sharma, 2021), and distance imputation (Bhattacharjee and Bayzid, 2020); finally, in previous work by the authors of this paper, machine learning techniques have been combined with cherry picking to solve Hybridization for multiple binary trees with largely overlapping leaf sets (Bernardini et al., 2022; Bernardini et al., 2023).

2. Methods

2.1. Definitions and notation

A *rooted phylogenetic network* N on a set of taxa X is a rooted directed acyclic graph such that the nodes other than the root are either (i) *tree nodes*, with in-degree 1 and out-degree greater than 1, or (ii) *reticulations*, with in-degree greater than 1 and out-degree 1, or (iii) *leaves*, with in-degree 1 and out-degree 0. The leaves of N are bi-univocally labelled by X, and we identify the leaves with their labels. The edges of N may be assigned a nonnegative *branch length*. We denote by $[1,n]$ the set of integers $\{1,2,\ldots,n\}$. Throughout this paper, we will often drop the terms "rooted" and "phylogenetic", as all the networks we consider are rooted phylogenetic networks.

We denote the *reticulation number* of a network N by $r(N)$, which can be obtained using the following formula: $r(N) = \sum_{v\in V}\max(0, d^-(v) - 1)$, where V is the set of nodes of N and $d^-(v)$ is the in-degree of a node v. A network T with $r(T) = 0$ is a *phylogenetic tree*.

We denote by $\mathscr{N}$ a set of networks and by $\mathscr{T}$ a set of trees. An ordered pair of leaves (x,y), $x \neq y$, is a *cherry* in a network if x and y share the same parent. Note that cherries (x,y) and (y,x) correspond to the same nodes and edges of the tree; the reason why they are considered two distinct cherries is motivated by the definition of the cherry-picking operation given below. An ordered pair (x,y) is a *reticulated cherry* if the parent of x, denoted by $p(x)$, is a reticulation, and the parent of y is a tree node that is one of the parents of $p(x)$ (see Fig. 1 (b)). Note that, in contrast with cherries, if (x,y) is a reticulated cherry then (y,x) is not, because the reticulation is constrained to be the parent of the first element of the pair. A pair of leaves is *reducible* if it is either a cherry or a reticulated cherry. Note that trees may have cherries but no reticulated cherries.

Suppressing a node v with a single parent $p(v)$ and a single child $c(v)$ is defined as replacing the arcs $(p(v), v)$ and $(v, c(v))$ by a single arc $(p(v), c(v))$ and deleting v. If the network has branch lengths, the length of the new edge is $\ell(p(v), c(v)) = \ell(p(v), v) + \ell(v, c(v))$. *Reducing* (or *picking*) a cherry (x,y) in a network N (or in a tree) is the action of deleting x and suppressing any resulting indegree-1 outdegree-1 nodes. A reticulated cherry (x,y) is *reduced (picked)* by deleting the edge $(p(y), p(x))$ and suppressing any indegree-1 outdegree-1 nodes. See Fig. 1. Reducing a non-reducible pair does not affect N. In all cases, the resulting network is denoted by $N_{(x,y)}$: we say that (x,y) affects N if (x,y) is reducible in N, i.e., $N \neq N_{(x,y)}$.

[1] Pronounced as 'finch'. Finches are birds that love cherries and are notoriously known for picking cherries from trees in orchards.

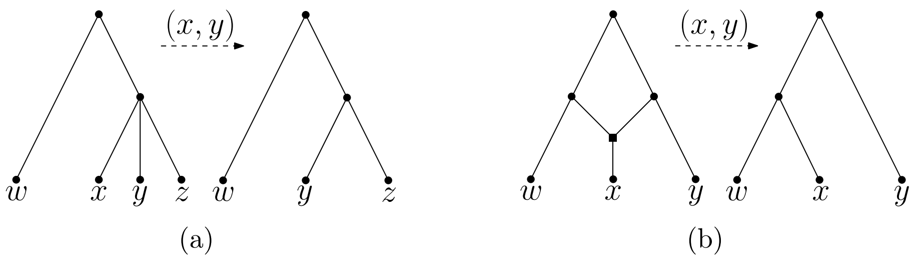

Fig. 1. The leaf pair (x,y) is picked in two different networks. In **(a)** (x,y) is a cherry, and in **(b)** (x,y) is a reticulated cherry, as well as (x,w). Note that in **(b)** the parent of x and the parent of y are suppressed after picking (x,y).

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 in addition, 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 let $S\circ(x,y)$ denote the sequence obtained by appending pair (x,y) at the end of S. We say that a CPS S *fully reduces* a network N if N_S is just a root with a single leaf; S is of minimum length for N if all pairs of S affect the network.

N is an *orchard network* (ON) if there exists a CPS that fully reduces it. If a CPS fully reduces all networks in a set $\mathscr{N}$, we say that it *fully reduces* $\mathscr{N}$. In this paper, we will consider CPSs which fully reduce a set of trees $\mathscr{T}$; $|\mathscr{T}|$ denotes the number of trees in $\mathscr{T}$.

2.1.1. The hybridization problem

The main problem considered in this paper is the following: given a set of phylogenetic trees on overlapping (but not necessarily equal) sets of taxa, infer a single network with the fewest number of reticulations that summarizes all the input trees. Definition 1 formalizes the concept of *summarizing* a set of trees: we seek a network where each of the input trees is displayed (see Fig. 2 for an example).

Definition 1. Let N be a network on a set of taxa X and let T be a tree on a set of taxa $X' \subseteq X$. Then, T is *displayed* in N if T can be obtained from N by applying a sequence of the following operations in any order:

a) Contract an edge (u,v) to a single node w: all parents of u and v except u become parents of w and all children of u and v except v become children of w.
b) Delete an edge: if the head of the edge is a leaf, delete the leaf node as well.
c) Suppress a node with in- and out-degree 1.

We now formally define the key problem of this work, called Hybridization (Baroni et al., 2005).

Input: A set $\mathscr{T} = \{T_1,T_2,\ldots,T_t\}$ of phylogenetic trees on sets of taxa $X_1,X_2,\ldots,X_t$, respectively.
Output: A binary phylogenetic network N on the set of taxa $X = \bigcup_{i=1}^{t} X_i$ which displays all the trees in $\mathscr{T}$ with the smallest possible number of reticulations.

Note that the input trees are not required to be binary nor to have

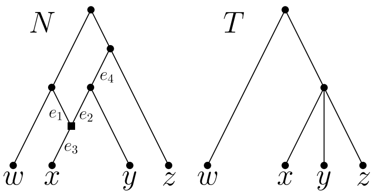

Fig. 2. Example of a multifurcating tree T that is displayed in the binary network N via the following operations: edge e_1 is deleted, then the parents of x and w are suppressed, and finally edge e_4 is contracted.

identical leaf sets: a tree $T_i \in \mathscr{T}$ on a set of taxa X_i is said to have *missing leaves* if $X_i \subsetneq X$. The input trees may or may not have branch lengths. Branch lengths do not play any role in Hybridization, as the requirements to be satisfied by a solution only affect its topological structure (and for this reason, output networks do not have branch lengths); however, when branch lengths are part of the input, our methods use them as features to train and guide the decisions of the underlying machine-learning model.

2.2. Solving hybridization via cherry picking

Our methods fall in the Cherry-Picking Heuristic (CPH) framework, first introduced in Bernardini et al. (2022) to find feasible solutions to Hybridization for *binary* input trees. In this section, we recall the main characteristics of the CPH framework; we refer the reader to Bernardini et al. (2022, Section 3) for a complete discussion.

The CPH framework relies on the following results given in Janssen and Murakami (2021): (i) if a minimum-length CPS that fully reduces a binary orchard network N on a set of taxa X also fully reduces a tree T (or another network, not necessarily binary) on a set of taxa $X' \subseteq X$, then T is displayed in N; and (ii), any CPS S can be processed in reverse order to reconstruct a unique binary orchard network for which S is a minimum-length CPS.

The main idea underlying CPH is thus to construct a CPS that fully reduces the input set of trees $\mathscr{T}$ and then to process this sequence in reverse order to obtain a network N, which is guaranteed to be a feasible solution to Hybridization by means of result (i). Any algorithm in the CPH framework constructs a CPS S in an incremental way (starting from an empty sequence) by repeating the following steps until all the input trees are fully reduced:

1. Choose a pair of leaves (x,y) that is reducible in at least one tree (i.e. a cherry of the tree set).
2. Reduce (x,y) in all trees.
3. Append (x,y) to S.

Once the input trees are fully reduced, the obtained sequence S is processed in reverse order to construct the output network N (after a last technical step to make sure S is a CPS and not just a partial sequence, see (Bernardini et al., 2022, Section 3.1) for details) using the dedicated method from Janssen and Murakami (2021). Since the latter method outputs binary networks, so do all algorithms in the CPH framework. Note that this is not a significant restriction because whenever there exists a multifurcating network displaying $\mathscr{T}$, there also exists a binary network displaying $\mathscr{T}$ with the same reticulation number. The following lemma links the number of reticulations of N with the length of the CPS from which it is reconstructed.

Lemma 1. ((*van Iersel et al., 2021*)) Let S be a CPS on a set of taxa X. The number of reticulations of the network N reconstructed from S is $r(N) = |S| - |X| + 1$.

The formula of Lemma 1 implies that the shorter the cherry-picking sequence constructed by the algorithm, the fewer the reticulations of the output network. The algorithms in the CPH framework differ from one to another for the criterion with which a reducible pair is chosen at each iteration: the goal of this study is to find a criterion that produces as short as possible sequences for input multifurcating trees with missing leaves.

Before discussing our new methods, we recall a simple, but rather effective algorithm in the CPH framework introduced in Bernardini et al. (2022) that can be easily modified to be applied to multifurcating input trees with missing leaves. In the rest of this paper, we will call FHyNCH-TrivialRand the adaptation of this strategy to multifurcating trees with missing leaves. We need the following definition (see Fig. 3 for an example).

Definition 2. An ordered leaf pair (x, y) is a *trivial cherry* (or trivial pair) of $\mathscr{T}$ if it is reducible in all $T \in \mathscr{T}$ that contain both x and y, and there is at least one tree in which it is reducible.

It has been empirically shown in Bernardini et al. (2022) that picking trivial cherries (when they exist) produces good results in terms of the number of reticulations of the output network. The criterion used by FHyNCH-TrivialRand to pick a pair at each iteration is thus to choose a trivial cherry if there is any; and to choose a pair uniformly at random among the cherries of the current tree set if no trivial cherry exists.

This randomized algorithm is so simple and fast that several runs on the same input can be computed in a reasonable time so as to select the best output as a final result: in our experiments, we will compare our new methods against this strategy.

2.3. A machine-learned algorithm for hybridization

A machine-learning model in the CPH framework for solving Hybridization on binary input trees was first proposed in Bernardini et al. (2022): although in theory this method is applicable in the presence of missing leaves (i.e. to input trees with different sets of taxa), the authors experimentally showed that the quality of the results rapidly degrades for increasing percentage of missing leaves. In principle, the model of Bernardini et al. (2022) could be straightforwardly adapted to work on multifurcating trees; however, its time complexity would get much worse, resulting in a slow algorithm that does not handle well the differences among the sets of taxa.

In this section, we propose a new, different machine-learning model specifically designed for multifurcating input trees with missing leaves.

2.3.1. Theoretical background

The theoretical foundations on which our new methods rely are the same as for the model of Bernardini et al. (2022). We report here a high-level description of this theoretical background and refer the reader to Bernardini et al. (2022, Section 3.3) for details and proofs.

The main idea is the following. Let $\mathsf{OPT}(\mathscr{T})$ denote the set of networks that display the input trees $\mathscr{T}$ with the minimum possible reticulation number (note that, in general, $\mathsf{OPT}(\mathscr{T})$ contains more than one network (Pardi and Scornavacca, 2015)). Ideally, we aim at finding a CPS fully reducing $\mathscr{T}$ that is also a minimum-length CPS that fully reduces some network of $\mathsf{OPT}(\mathscr{T})$. This is because any method in the CPH framework outputs a network for which the produced CPS is a

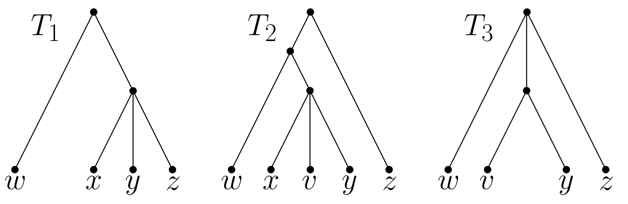

Fig. 3. Example of a tree set with a *trivial cherry* (x,y): in trees T_1 and T_2, x and y form a cherry, and x is not in T_3. In contrast, (x,z) is not a trivial cherry: it is a cherry in T_1, but both x and z are in T_2 without forming a cherry.

minimum-length sequence.

Our goal is to design a machine-learned oracle to predict, at each iteration of the method, which pairs of $\mathscr{T}$ are reducible in some optimal network. Using this prediction, at every iteration the algorithm chooses a pair that most probably leads to an optimal solution.

2.3.2. Machine-learning models

To predict whether a given cherry of the tree set is a reducible pair in some optimal network N for $\mathscr{T}$, we train two random forest classifiers: one using features that carry information on the leaves of the trees, another using features about their cherries. The main novelty of this approach compared to those proposed in Bernardini et al. (2023) is in the design and use of the first classifier, whose crucial role is to reduce the solution space at every iteration: without its introduction, the method would be infeasible for non-binary input data sets of practical size. This is because it may require computing features for a quadratic number of cherries at every iteration, in contrast with the binary case, in which the number of cherries is always linear in the number of taxa. The accuracy of the simple random forest models for our problem was so good that we did not find any advantage in applying deep learning instead.

In the first classifier, a data point is a pair $(\mathbf{F}_1, c_1)$, where $\mathbf{F}_1$ is an array containing 8 features, listed in Table 1, of a leaf x, and c_1 is a binary label modelling whether or not x belongs to a reducible pair (either a cherry or a reticulated cherry) of the unknown target optimal network N. The second classifier is similar to the one proposed in Bernardini et al. (2022): here, a data point is a pair $(\mathbf{F}_2, c_2)$, where $\mathbf{F}_2$ is an array containing 21 features, listed in Table 2, of a cherry (y,z), and c_2 is a binary label modelling whether or not (y,z) is a reducible pair of N. The two classifiers receive in input the arrays of features, learn the association between $\mathbf{F}_1$ and c_1 and between $\mathbf{F}_2$ and c_2, respectively, and output a label for each data point together with a confidence score modelling the probability that the predicted label is correct.

The general scheme of our strategy is as follows. First, the algorithm computes the array $\mathbf{F}_1$ of features for each $x \in X$, thus creating a data point for the first classifier for each leaf of the initial tree set, and initializes an empty cherry-picking sequence S. It then repeats the following steps until all the trees are fully reduced.

1. Select a subset C of k leaves from the current tree set, based on the predictions of the first classifier.
2. Compute $\mathbf{F}_2$ for each cherry in the current tree set *that contains one leaf from* C, thus creating data points for the second classifier only for this subset of cherries.
3. Choose a cherry (x,y) based on the predictions of the second classifier, append it to S, and reduce (x,y) in all trees.
4. Update $\mathbf{F}_1$ for all the data points for the first classifier.

We name this algorithmic scheme FHyNCH-MultiML. The constant $k \geqslant 1$ determining the size of the subset of leaves selected in Step 1 at each

Table 1
Features of a leaf x for the first classifier.

Num	Feature name	Description
1	Leaf pickable	Ratio of trees in which x is part of a cherry
2	Leaf in tree	Ratio of trees that contain leaf x
3	Siblings avg.	Avg over trees with x of ratios “num. of siblings of x/number of leaves in tree”
4	Siblings std.	Standard deviation of “num. of siblings of x/number of leaves in tree”
Features measured by distance (d) and topology (t)		
$5_{b,t}$	Leaf depth x avg.	Avg over the trees that contain x of ratios “depth of x/leaf-depth of the tree”
$6_{b,t}$	Leaf depth x std.	Standard deviation of “depth of x/leaf-depth of the tree”

Table 2
Features of a cherry (x, y) for the second classifier. These features are the same as those for the classifier used in Bernardini et al., 2022; however, the latter classified cherries into four classes instead of only two.

Num	Feature name	Description
1	Cherry in tree	Ratio of trees that contain cherry (x,y)
2	New cherries	Number of new cherries of $\mathscr{T}$ after picking cherry (x,y)
3	Before/after	Ratio of the number of cherries of $\mathscr{T}$ before/after picking cherry (x,y)
4	Trivial	Ratio of trees with both leaves x and y that contain cherry (x,y)
5	Leaves in tree	Ratio of trees that contain both leaves x and y
Features measured by distance (d) and topology (t)		
$6_{b,t}$	Tree depth	Avg over trees with (x,y) of ratios “cherry-depth of the tree/max cherry-depth over all trees”
$7_{b,t}$	Cherry depth	Avg over trees with (x,y) of ratios “depth of (x,y) /cherry-depth of the tree”
$8_{b,t}$	Leaf distance	Avg over trees with x and y of ratios “x-y leaf distance/cherry-depth of the tree”
$9_{b,t}$	Leaf depth x	Avg over trees with x and y of ratios “depth of x/cherry-depth of the tree”
$10_{b,t}$	Leaf depth y	Avg over trees with x and y of ratios “depth of y/cherry-depth of the tree”
$11_{b,t}$	LCA distance	Avg over trees with x and y of ratios “x-LCA(x,y) distance/y-LCA(x,y) distance”
$12_{b,t}$	Depth x/y	Avg over trees with x and y of ratios “depth of x/depth of y”
$13_{b,t}$	LCA depth	Avg over trees with (x,y) of ratios “depth of LCA(x,y)/cherry-depth of the tree”

iteration is a parameter of the algorithm: in Section 3.1 we report experiments about the impact of k on the running time and the quality of the results of our algorithm. The simplest way to implement Step 1 is to select the k leaves that are predicted by the first classifier to be part of a reducible pair of an optimal network with the highest probability; other possible strategies are supported by our method, e.g., to fix a threshold $\lambda \in (0, 1)$ and to select the k leaves uniformly at random among the ones whose probability to be part of a reducible pair of an optimal network is at least λ. A similar argument can be made for the choice of the cherry in Step 3.

The number of data points for the first classifier is always bounded by $|X|$, the number of taxa. The array $\mathbf{F}_1$ of features is efficiently updated in Step 4 at each iteration for each data point, that is, for each leaf of the current tree set. In contrast, arrays $\mathbf{F}_2$ are computed from scratch in Step 2 at every iteration because the subset of cherries for which a data point is created changes across different iterations (it depends on the leaves chosen at Step 1).

The main role of the first classifier is in fact to reduce the number of cherries for which $\mathbf{F}_2$ must be computed: this is needed because the total number of cherries in the tree set could be superlinear (up to quadratic in the number $|X|$ of taxa), which could make it impractical to compute $\mathbf{F}_2$ for every cherry at every iteration. Using the first classifier beforehand guarantees that $\mathbf{F}_2$ must be computed only for a linear number $\mathscr{O}(k|X|)$ of cherries at each iteration, resulting in a much faster and more practical algorithm.

Features 5–6 for the first classifier and 6–13 for the second classifier can be computed for both branch lengths and unweighted branches. We refer to these two options as *branch distance* and *topological distance*, respectively. The *branch 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 path from the root to u; the *leaf-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; and the *cherry-depth* of T is the maximum depth of any cherry of T. The leaf distance between x and y is the total 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). In particular, the leaf distance between the leaves of a cherry is zero as their LCA is their common parent. All of the above quantities can be defined both using branch distance and topological distance.

2.3.3. *Tree expansion*

We now briefly describe a heuristic improvement to our methods, called *tree expansion*, that was already introduced in Bernardini et al. (2022) for binary trees, and can be applied as-is to multifurcating trees. Tree expansion is applied whenever a trivial cherry (x, y) is chosen to be reduced at some iteration. By Definition 2, each tree T in the current tree set belongs to one of the following classes with respect to the trivial cherry (x,y): (i), (x, y) is a cherry of T; (ii), neither x nor y are leaves of T; (iii), T has leaf y but not x; and (iv), T has leaf x but not y.

Without tree expansion, after reducing (x, y) in the tree set, leaf x is removed from the trees in class (i), but not from the trees in class (iv), thus it still may be present in the tree set. The goal of tree expansion is to make x disappear from the whole tree set, as this empirically reduces the length of the produced sequence and thus the number of reticulations in the output network. Tree expansion consists of the following operation:

Rule 1. (*Tree expansion*) Before reducing a trivial cherry (x, y) in the tree set, add leaf y to form a cherry with x in all the trees in class (iv).

After tree expansion, picking (x, y) will make x disappear from the set. Another way of viewing this operation is as a *relabeling* of x by y in all the trees in class (iv). It was proved in Bernardini et al. (2022, Lemma 6) that this move does not affect the feasibility of the output: in other words, the network produced using tree expansion still displays the input set of trees. The same proof applies to the case of multifurcating trees.

Example 1. To illustrate the workings of FHyNCH-MultiML, we applied it to two phylogenetic trees for the Lamprologini tribe (one representing the mitochondrial phylogeny, the other the nuclear phylogeny), studied in Koblmüller et al. (2007). The same data were later used to test the Autumn algorithm (Huson and Linz, 2018). In Huson and Linz (2018), the phylogenetic trees were preprocessed to contract edges that had a bootstrap support of 50% or less. We applied FHyNCH-MultiML to these preprocessed trees, after deleting a few species that were misspelt in one of the two trees of Huson and Linz (2018) and were mistakenly considered two different species in the two phylogenies[2]. Note that removing these species does not affect the number of reticulations needed because each of the variants was in only one of the trees in Huson and Linz (2018). The trees are multifurcating and have different taxa sets.

The input trees and the network outputted by FHyNCH-MultiML are shown in Fig. 4. Notably, although FHyNCH-MultiML is a heuristic specifically designed for multiple trees, in this case, it returns a network with the same number of reticulations as in the output of the exact Autumn algorithm (Huson and Linz, 2018), thus an optimal result. We also observe that the Autumn algorithm has several practical advantages: it returns multiple optimal networks and it returns nonbinary networks. In comparison, the networks produced by FHyNCH-MultiML could be more resolved than necessary to display the input trees.

Let us now have a closer look at the first iterations of FHyNCH-MultiML. The two input trees contain several trivial cherries: e.g. (Hybrid1.2, Neolamprologus_calliurus) is a cherry in the mitochondrial tree, and the label ‘Hybrid1.2’ does not appear in the nuclear tree, thus the cherry is trivial as per Definition 2; another trivial pair is (Telmatochromis_vittatus, Julidochromis_ornatus), which is a cherry both in

[2] E.g., *Neolamprologus wauthioni* is mistakenly spelt as *Naolamprologus wauthioni* in the mitochondrial tree used in Huson and Linz (2018); the two names (correct and misspelt) labelled two different leaves of the output networks. Similar typos occurred for another two species.

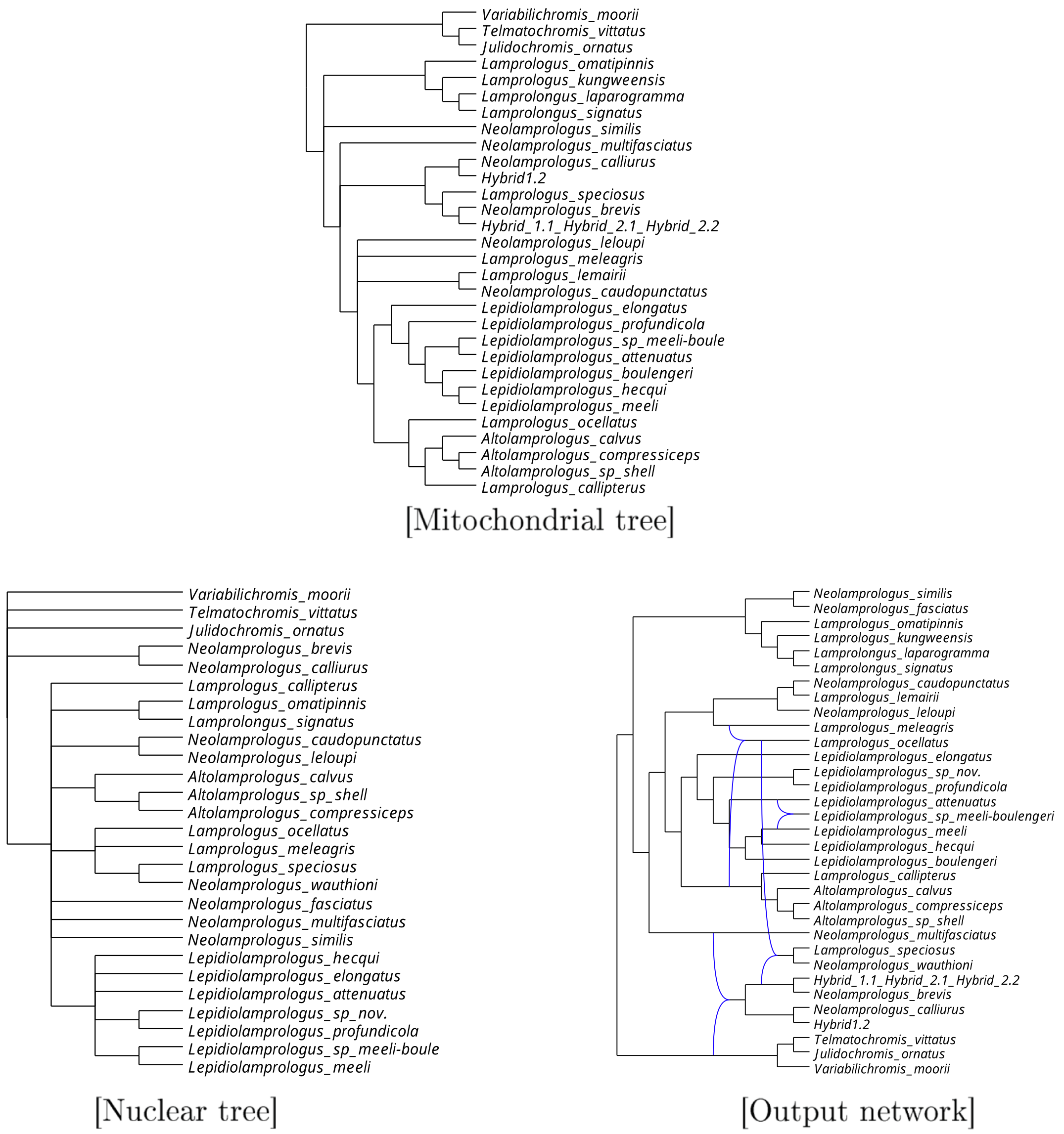

Fig. 4. Mitochondrial (a) and nuclear (b) phylogenies for the Lamprologini tribe, preprocessed as in Huson and Linz, 2018. The network outputted by FHyNCH-MultiML (c) has the optimal number of 4 reticulations.

the mitochondrial and in the nuclear tree; and many more (16 in total). The first iterations are devoted to picking all such trivial cherries, which are also cherries of the output network of Fig. 4 (c). After picking all the initial trivial cherries, the two input trees were reduced to the two trees shown in Fig. 5.

At this point, the first classifier computed the features of Table 1 for all the 15 leaves remained in the trees of Fig. 5 and returned 'Lepidiolamprologus_sp_meeli-boulengeri' (abbreviated as 'Lep_sp_meeli-boule' in the rest of the example) as the top-scoring leaf, with a score of 0.98. This leaf formed a cherry with 'Lepidiolamprologus_attenuatus' in the Mitochondrial tree (abbreviated as 'Lep_att') and with 'Lepidiolamprologus_meeli' ('Lep_meeli') in the nuclear tree. The second classifier thus computed the features of Table 2 for the four cherries (Lep_sp_meeli-boule, Lep_att), (Lep_att, Lep_sp_meeli-boule), (Lep_sp_meeli-boule, Lep_meeli), (Lep_meeli,Lep_sp_meeli-boule), and returned (Lep_sp_meeli-boule, Lep_att) as the top-scoring. This cherry was thus picked from the mitochondrial tree.

After this iteration, the cherry (Lep_sp_meeli-boule, Lep_meeli) became trivial (as 'Lep_sp_meeli-boule' was no longer present in the mitochondrial tree) and was thus picked from the nuclear tree. In the end, FHyNCH-MultiML produced a cherry-picking sequence of length 36; since the total number of taxa labelling the input trees was 33, the output reticulation number was 4.

2.3.4. Obtaining training data

Generating data to train our classifiers is nontrivial because of the lack, in general, of ground truth: no existing algorithm is able to find an optimal solution – let alone all optimal solutions – for sufficiently large instances. We thus rely on the following procedure. We first generate a binary network N on a set of taxa X using the LGT (lateral gene transfer) network generator of Pons et al. (2019) and extract the set $\mathscr{T}$ of all trees that are displayed in N and have the whole X as leaf set. We then contract and delete some edges (see Definition 1) from each of these trees using the following criteria. We set up two thresholds $\mathsf{Ml}, \mathsf{Me} \in (0,1)$; for each tree $T \in \mathscr{T}$, we choose a value $p_l^T \in (0, \mathsf{Ml})$ and a value $p_e^T \in (0, \mathsf{Me})$ uniformly at random, contract each edge of T with probability p_e^T and delete each leaf (by deleting the edge that connects it to its parent) with probability p_l^T.

The thresholds Ml and Me thus model the maximum probability with which a leaf is deleted and an edge is contracted, respectively, in any of the trees of $\mathscr{T}$; and we apply these operations with a different

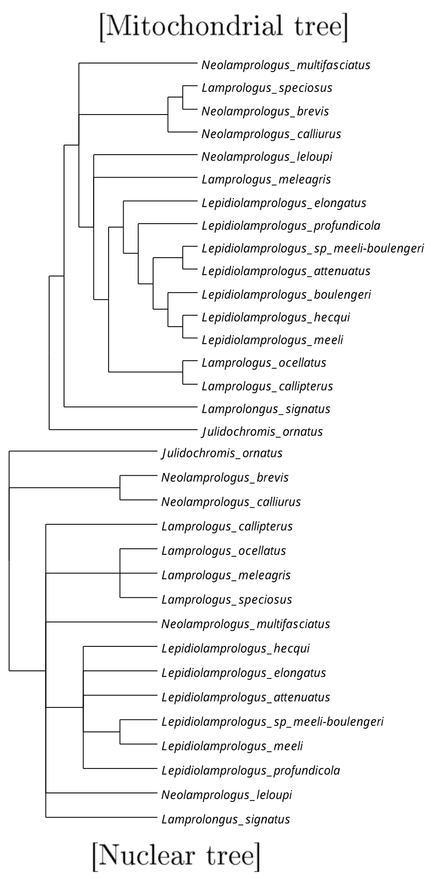

Fig. 5. The trees of Fig. 4 after reducing all their trivial cherries.

probability for each tree. The resulting tree set $\mathscr{T}$ consists of multifurcating trees (as a result of edge contractions) with missing leaves (as a result of leaf deletions).

Once we have generated the set $\mathscr{T}$, we create a data point for the first classifier for each leaf of $\mathscr{T}$, labelling it according to whether it is in a reducible pair of N or not; and similarly, we create a data point for the second classifier for each cherry of $\mathscr{T}$, labelling it according to whether it is reducible in N or not. We then iteratively choose a reducible pair from N, reduce it both in N and in $\mathscr{T}$, and update the data points and labels of each classifier. We terminate this procedure when N is fully reduced.

We remark that N is not necessarily an optimal network for the generated trees (Pardi and Scornavacca, 2015). However, its number $r(N)$ of reticulations provides an upper-bound estimate of the number of reticulations of an unknown optimal network, and in Section 3.1 we will use $r(N)$ as a reference value to evaluate the quality of our results for the synthetically generated data sets.

3. Results

The code of all our heuristics and for generating data is written in Python and is available at `https://github.com/estherjulien/FHyNCH`. All experiments ran on a computing cluster with an AMD Genoa 9654 CPU, of which 16 cores were used. We conducted experiments on both synthetic and real data. Scikit-learn (Pedregosa et al., 2011) with default settings was used for the random forest model.

3.1. Synthetic data

Similar to the training data, we generated each of the synthetic datasets by first growing a binary network N on a set of taxa X using the LGT network generator of Pons et al. (2019), extracting some of the trees that are displayed in N and have the whole X as leaf set, and finally deleting some leaves and contracting some edges from each of the extracted trees as described in Section 2.3.4.

We generated several instances for different combinations of the following parameters: the number $R \in \{10, 20, 30\}$ of reticulations of the generating network N; the number $L \in \{20, 50, 100\}$ of leaves of the generating network N (i.e. the size of the set of taxa X); the number $|\mathscr{T}| \in \{20, 50, 100\}$ of trees extracted from N; and the thresholds $\mathsf{Ml}, \mathsf{Me} \in \{0, 0.1, 0.2\}$ for leaf and edge deletion, respectively (see Section 2.3.4). For each combination of the parameters R, L, Ml and Me we generated 20 networks for each value of $|\mathscr{T}| \in \{20, 50, 100\}$ and as many instances for Hybridization. The 60 instances generated for a specific combination of values for $L, R, \mathsf{Ml}, \mathsf{Me}$ constitute an *instance group*.

We run all our experiments setting the parameter $k = 1$, as the experiment summarized in Fig. 6 indicates that larger values of k increase the running time of the algorithm without improving the quality of the results.

Since no exact method can be applied to these instances, we compared FHyNCH-MultiML with FHyNCH-TrivialRand, a randomized heuristic proposed in Bernardini et al. (2022) (and here briefly summarized in Section 2.2) that can be straightforwardly modified to be applied to nonbinary trees with missing leaves. For each instance I, we ran FHyNCH-MultiML once, while FHyNCH-TrivialRand was run $\min\{x(I), 1000\}$ times, where $x(I)$ is the number of runs that can be executed in the same time as one run of FHyNCH-MultiML on the same instance; we then selected the best output over all such runs, and considered this value the result of FHyNCH-TrivialRand for instance I.

To evaluate the quality of the methods, within each instance group we used the number R of reticulations of the generating networks as a reference value and divided the number of reticulations output by each method by this value. The results are summarized in Fig. 7.

It is immediately apparent that the results of FHyNCH-TrivialRand rapidly degrade for increasing instance size and increasing percentages of missing leaves and multifurcating nodes in the input trees, while the performance of FHyNCH-MultiML is much more stable. Moreover, for the same number of leaves in the generating network N (parameter L) the results of both methods become worse for increasing number of reticulations of N (parameter R), the deterioration being much more marked for FHyNCH-TrivialRand than for FHyNCH-MultiML.

With few exceptions (including e.g. the instance group with $L = 20$, $R = 10$, $\mathsf{Ml} = \mathsf{Me} = 0.2$), the performance of FHyNCH-MultiML and FHyNCH-TrivialRand on the smaller instances ($L = 20$) do not seem to be significantly different, although both the median and the variance of the results of FHyNCH-MultiML are consistently smaller than those of FHyNCH-TrivialRand. In all the other instance groups, FHyNCH-MultiML substantially outperforms FHyNCH-TrivialRand, the difference being more pronounced in groups with higher percentages of missing leaves and contracted edges.

Unlike what happens for FHyNCH-TrivialRand, the quality of the results of FHyNCH-MultiML is only marginally affected by increasing percentages of contracted edges, and it is also mildly affected by increasing percentages of missing leaves. For example, the median for the results of FHyNCH-MultiML in the instance group with $L = 100, R = 30$ and no contracted edges nor missing leaves is 1.47, the results for 75% of the instances being within a factor 1.7 from the reference value; these values become 1.78 and 2.18, respectively, in the instance group with $L = 100, R = 30$, no missing leaves and $\mathsf{Me} = 0.2$. Increasing the percentage of missing leaves, the median of the results of FHyNCH-MultiML within the group with $L = 100, R = 30, \mathsf{Ml} = \mathsf{Me} = 0.2$ is 2.85, the results for 75% of the instances being within a factor of 3.71 from the reference.

In comparison, for the same instance groups the results of FHyNCH-TrivialRand are as follows: in the group with $L = 100, R = 30$, no missing leaves nor contracted edges, the median is 4.02, the results for 50% of the instances being within a factor in the range of 3.2 to 5.01 from the reference value; in the group with $L = 100, R = 30$, no missing leaves

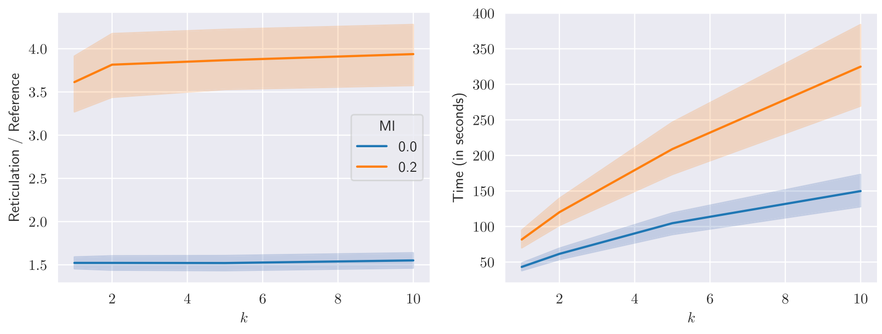

Fig. 6. Results (left) and running time in seconds (right) for synthetic instances with $L = 100, R = 30, |\mathcal{T}| \in \{20, 50, 100\}, Me = 0$ and $Ml \in \{0, 0.2\}$ for varying $k \in \{1, 2, 5, 10\}$ (k is the number of leaves chosen in Step 1 of FHyNCH-MultiML: see Section 2.3.2).

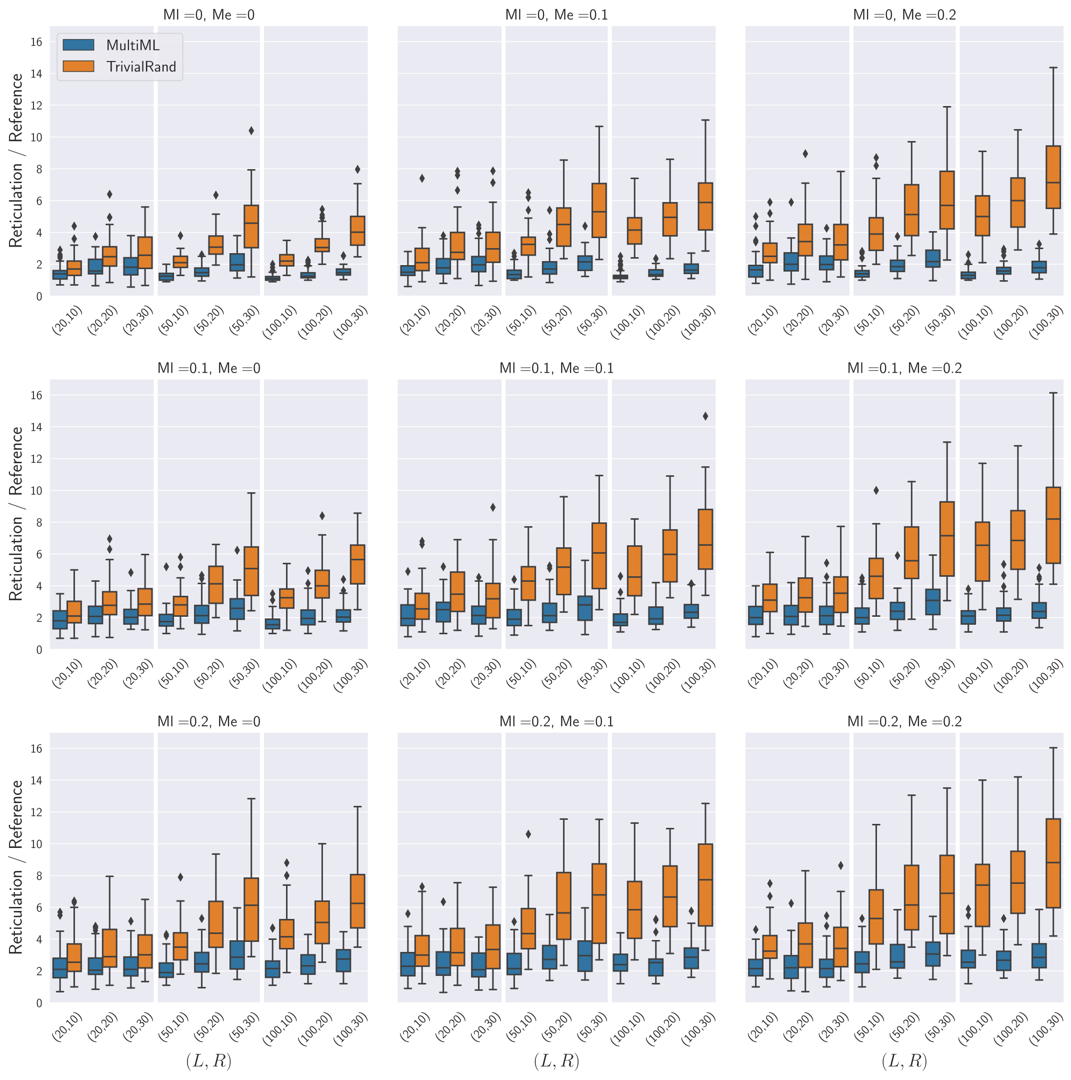

Fig. 7. *Synthetic* instance results for different values of Ml, Me, L, and R. The reference reticulation number value per instance is the network the trees were extracted from.

and Me $= 0.2$, the median is 7.13, the results for 50% of the instances being within a factor in the range of 5.52 to 9.43 from the reference value; and finally, in the group with $L = 100, R = 30$ and Me $=$ Ml $= 0.2$ the median is 8.82, the results for 50% of the instances being within a factor in the range of 5.98 to 11.56 from the reference value.

The poor performance of FHyNCH-TrivialRand on instances consisting of trees with many leaves, especially when they are nonbinary, is due to its randomized nature: the more the leaves, the more the cherries in the tree set, thus the smaller the probability of picking a good pair at every iteration. When the trees are nonbinary, the number of cherries increases even more, making the issue more serious; and the same holds for missing leaves - when a leaf is missing from a tree, this may originate a cherry that would not have been there in the complete tree. In contrast, the more leaves in the trees the more data are available for the machine-learned models to make good decisions.

In the next section, we show that this trend is conserved when the two methods are applied to real datasets: when the instances are small enough, FHyNCH-TrivialRand often outperforms FHyNCH-MultiML, while on larger instances the results of FHyNCH-MultiML are significantly better.

3.2. Real data

Evaluating the performance of FHyNCH-MultiML on real data is a nontrivial task because, since no exact method exists for solving Hybridization for more than two multifurcating trees with missing leaves, we do not have a baseline to compare against. We thus adopted two different strategies depending on the instance size.

For small enough instances, consisting of up to 6 trees, we apply a procedure - described in the paragraph devoted to small instances - to make the trees binary and with equal leaf sets, to be able to apply the exact TreeChild method from van Iersel et al. (2022) and use its result as a reference value. Although this is not necessarily the true optimum, both because by making the trees binary and adding missing leaves we could introduce spurious constraints that might originate unnecessary reticulations and because TreeChild is exact only for a special class of networks, this value is expected to be reasonably close to the real optimum.

Larger instances cannot be processed by TreeChild nor other exact methods, thus we simply compared the performance of FHyNCH-MultiML and FHyNCH-TrivialRand and reported the *relative error* of one compared to the other, i.e. the difference between the two results divided by the best (thus the smallest) one. More details will be provided in the paragraph devoted to large instances.

Data sets We extracted several instances of Hybridization from the publicly available data set used in Beiko (2011), consisting of phylogenetic trees for 159, 905 distinct homologous gene sets from 1173 sequenced bacterial and archaeal genomes. The trees are multifurcating and have missing leaves.

For different sizes of the tree set $|\mathscr{T}| \in \{2,4,6,10,20,30,40,50,60\}$, we extracted instances. For each instance, we also fixed an approximate number of leaves $L \in \{10,20,50,100,150\}$ and the maximum fraction of missing leaves (from the union of leaves from all trees in the set) $\mathsf{MI} \in \{0.1,0.2,0.3\}$.

To generate an instance, we sampled one tree at a time from the full data set uniformly at random, and depending on whether it was consistent with the fixed values for parameters L and MI we added it to the instance or discarded it and sampled another tree, until we reached the predetermined number $|\mathscr{T}|$ of trees.

More in detail, for every sampled tree T we checked whether the number of its leaves was between $(1-\mathsf{MI})L$ and L, whether the union of its leaves and the leaves of the trees already selected for that instance was of size at most L and the difference between the leaf set of T and such union was at most $100 \cdot \mathsf{MI}\%$. If an instance could not be completed within a timeframe of 10 min, we aborted the search and started generating a new instance with the same parameters from scratch. We aimed at generating 10 instances for each combination of parameters $|\mathscr{T}|$, L and MI, however, for some combinations we did not find enough trees with the desired properties to generate as many instances. Table 3 reports the number of distinct instances that we were able to extract from the data set for each parameter combination. We consider *small* the instances with $|\mathscr{T}| \in \{2,4,6\}$ and *large* the rest.

3.2.1. Experiments on small instances

We assess the performance of FHyNCH-MultiML against FHyNCH-TrivialRand on small instances using a baseline obtained by applying TreeChild (available at https://github.com/nzeh/tree_child_code) to a modified version of the same instance. The modification is needed because TreeChild can only be applied to a tree set of binary trees with the same leaf sets. To generate these modified instances, the first step is to add as many missing leaves as possible as follows.

Let $\mathscr{T} = \{T_1, T_2, \ldots, T_n\}$ be the set of input trees and let R be the set of cherries of $\mathscr{T}$. For each T_i and each leaf ℓ missing from T_i, we compute the set $R_i^\ell = \{(\ell, y) \in R \mid y \in T_i\}$ of cherries of $\mathscr{T}$ such that one element is ℓ and the other is a label present in T_i. We then find the cherry $(\ell, z) \in R_i^\ell$ that occurs the most in $\mathscr{T}$ (ties are broken randomly), add ℓ as another child of the parent of z in T_i and remove ℓ from M_i. If after applying this procedure for all missing leaves of T_i some leaves are still missing, we add them randomly.

In an effort to minimize the bias introduced in the results because of this randomized step, we generated 10 different modified instances for each of the original instances, ran TreeChild on each of them and used the result for the best of these modified instances as a reference value for the original instance. Before doing so, however, we make all the trees of all the modified instances binary by using the dedicated method that can be found in the TreeChild repository https://github.com/nzeh/tree_child_code.

We remark that the trees of the modified instances display those of the original instances by construction (they can be obtained reverting the procedure, i.e contracting edges and deleting leaves) thus the solutions obtained by applying TreeChild to the modified instances are always feasible for the original ones, although we cannot guarantee their optimality.

For each of the original instances I, we thus ran FHyNCH-MultiML once, selected the best of the $\min\{1000, x(I)\}$ runs of FHyNCH-TrivialRand (recall that $x(I)$ denotes the number of runs of FHyNCH-TrivialRand that can be completed in the time required for a single run of FHyNCH-MultiML) and divided these results by the best value returned by TreeChild among the 10 modified instances obtained from I. We summarize the results in Fig. 8.

From the experiments on synthetic data, it was already clear that the performance of FHyNCH-TrivialRand is close to that of FHyNCH-MultiML

Table 3
Number of instances extracted from the *Bacterial and Archaeal Genomes* data set, for different combinations of parameters $|\mathscr{T}|, L$, and MI.

	$L=10$			$L=20$			$L=50$			$L=100$			$L=150$		
MI / $\lvert\mathcal{T}\rvert$	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3
2	10	10	10	10	10	10	6	10	10	10	10	10	9	10	10
4	10	10	10	10	10	10	0	7	10	10	10	10	0	7	10
6	10	10	10	10	10	10	0	3	10	10	10	10	0	0	9
10	10	10	10	10	10	10	0	0	6	10	9	10	0	0	3
20	5	10	10	7	7	10	0	0	0	4	9	10	0	0	0
30	6	10	10	7	6	10	0	0	0	1	6	8	0	0	0
40	2	10	9	6	7	5	0	0	0	1	3	8	0	0	0
50	3	9	8	6	5	8	0	0	0	0	3	8	0	0	0
60	1	6	8	5	6	6	0	0	0	0	1	7	0	0	0

Table 4
Number of (modified) small instances extracted from the *Bacterial and Archaeal Genomes* data set TreeChild was able to solve within a time limit of 2 h using 16 cores.

	$L=10$			$L=20$			$L=50$			$L=100$			$L=150$		
MI	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3
$\|\mathcal{T}\|$															
2	10	9	10	10	10	10	6	7	1	1	0	0	0	0	0
4	10	10	10	8	5	4	0	0	0	0	0	0	0	0	0
6	10	9	10	0	1	0	0	0	0	0	0	0	0	0	0

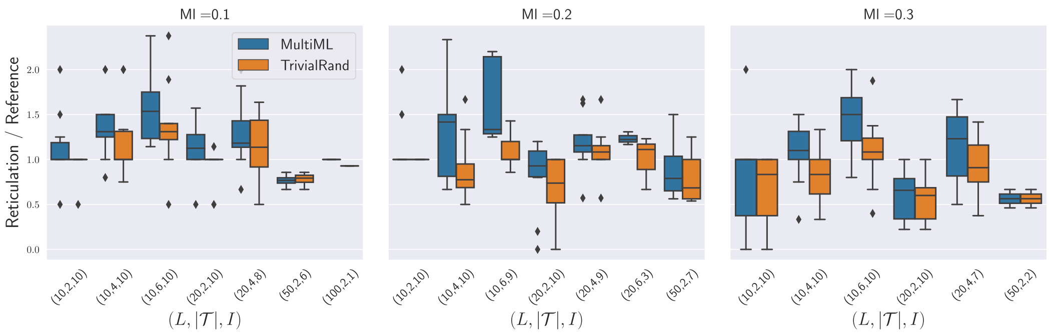

Fig. 8. Results for the small instances extracted from the *Bacterial and Archaeal Genomes* data set for different values of MI. The reference value for each instance is the best output of TreeChild among the corresponding 10 modified instances described in Section 3.2.1. We do not show results for instances for which TreeChild could not provide a solution within 2 h, using 16 cores, for any of the modified instances, which was the case for at least one instance within 31 out of 40 small instance groups (see Table 4 and compare it with Table 3). Parameters L and $|\mathcal{T}|$ are as described in Section 3.2.1; I denotes the number of instances that TreeChild was able to solve for each instance group. Values of the ratio results/reference smaller than 1 indicate that TreeChild did not return the true optimum for some instances because of the (unavoidable) artificial constraint introduced in the modified instances.

on small enough instances. The smallest instances of the synthetic data set, which consist of 20, 50, or 100 trees with 20 leaves each, are much larger than the small instances of the real data set, which consist of only 2, 4, or 6 trees each: for the latter, the good performance of FHyNCH-TrivialRand becomes more pronounced, its results being always comparable or better than those of FHyNCH-MultiML. This is because when the size of the leaf sets and the number of trees are not too large, multiple runs of FHyNCH-TrivialRand can explore a significant part of the solution space and thus return a good enough solution.

Running time Table 5 reports the average and standard deviation of the running times of FHyNCH-MultiML and TreeChild for each of the small instance groups of Table 3. Following van Iersel et al. (2022), we imposed a time limit of 2 h for the execution of TreeChild. We used 16 cores to run TreeChild and only 1 core to run FHyNCH-MultiML, which is single-threaded. Note that for the smallest instances consisting of 2 trees with $L\leqslant 20$ or 4 trees with $L=10$ TreeChild is on average faster than FHyNCH-MultiML.

However, the opposite becomes true as soon as the number of trees and leaves are increased: already for instances of 6 trees with $L=10$, TreeChild is slower than FHyNCH-MultiML on average by one or two orders of magnitude; and it is slower by three orders of magnitude or exceeds the time limit for instances with at least 4 trees with $L\geqslant 20$ or just

Table 5
Running times of FHyNCH-MultiML (MML) and TreeChild (TC) for the small instance groups. The first value in each pair is the average time in seconds within the group, the second value is the standard deviation. For each instance group, the average time required by the fastest method is highlighted in bold. A dash indicates an empty instance group; "> t.l." means that the time limit of 2 h was exceeded. Only 1 core was used to run FHyNCH-MultiML on each instance; 16 cores were used to run TreeChild on each instance.

L	MI	2 trees		4 trees		6 trees	
		MML	TC	MML	TC	MML	TC
10	0.1	(5.6, 0.2)	(**0.1**, 0.1)	(1.6, 0.3)	(**0.5**, 1.0)	(**2.4**, 0.9)	(325.0, 731.9)
	0.2	(1.0, 0.1)	(**0.2**, 0.2)	(1.6, 0.3)	(**0.2**, 0.2)	(**2.4**, 0.9)	(720.5, 2159.8)
	0.3	(1.0, 0.1)	(**0.2**, 0.2)	(2.7, 1.9)	(**0.2**, 0.3)	(**2.2**, 0.6)	(15.7, 27.6)
20	0.1	(4.7, 2.0)	(**0.1**, 0.1)	(**3.1**, 0.9)	(1835.9, 2828.8)	(**6.1**, 0.9)	> t.l.
	0.2	(4.5, 1.1)	(**0.1**, 0.1)	(**3.1**, 0.6)	(1449.3, 2174.8)	(**5.8**, 1.6)	(5408.3, 2891.1)
	0.3	(1.4, 0.1)	(**0.5**, 0.7)	(**7.2**, 0.9)	(3533.1, 3310.1)	(**4.8**, 1.6)	> t.l.
50	0.1	(**3.5**, 0.2)	(82.1, 176.5)	-	-	-	-
	0.2	(**5.4**, 1.4)	(3321.8, 3021.2)	(**7.6**, 0.9)	> t.l.	(**13.7**, 0.5)	> t.l.
	0.3	(**3.2**, 0.6)	(5846.6, 2712.4)	(**8.6**, 2.4)	> t.l.	(**11.8**, 3.9)	> t.l.
100	0.1	(**8.6**, 0.8)	(6480.3, 2159.1)	(**20.4**, 1.5)	> t.l.	(**39.6**, 3.4)	> t.l.
	0.2	(**8.8**, 1.1)	> t.l.	(**21.0**, 1.6)	> t.l.	(**40.4**, 3.9)	> t.l.
	0.3	(**7.5**, 1.0)	> t.l.	(**20.8**, 4.0)	> t.l.	(**36.2**, 3.9)	> t.l.
150	0.1	(**13.7**, 0.3)	> t.l.	-	-	-	-
	0.2	(**13.1**, 1.5)	> t.l.	(**30.2**, 3.1)	> t.l.	-	-
	0.3	(**11.5**, 0.8)	> t.l.	(**26.9**, 2.4)	> t.l.	(**47.2**, 6.5)	> t.l.

2 trees with $L \geqslant 50$. FHyNCH-MultiML requires, on average, less than a minute for all these instance groups.

3.2.2. *Experiments on large instances*

In contrast with the small instances, no exact methods could solve any of the larger instances, not even when made binary and with equal leaf sets. It is thus not possible to compute any reference value for these instances. We thus simply compare the performance of FHyNCH-MultiML against FHyNCH-TrivialRand computing their *relative errors*, defined as follows. Let $r_{ML}(I)$ and $r_{TR}(I)$ be the number of reticulations output by FHyNCH-MultiML and FHyNCH-TrivialRand, respectively, for the same instance I, and let $m = \min\{r_{ML}(I), r_{TR}(I)\}$. The relative error of FHyNCH-MultiML against FHyNCH-TrivialRand for instance I is given by $\frac{r_{ML}(I)-m}{m}$; likewise, the relative error of FHyNCH-TrivialRand against FHyNCH-MultiML is $\frac{r_{TR}(I)-m}{m}$. The relative error of one method against the other is 0 whenever the method is the best-performing one. We computed these values for each instance, averaged them over all instances within each instance group, and rescaled them to express them as a percentage. The results are shown in Table 6.

Note that when the mean relative error is 0.0 for some method in some instance group, by definition, that method is the best-performing one for all the instances within the group. It is thus immediately evident that FHyNCH-MultiML is systematically the best method for any instance with a number of leaves $L \geqslant 100$ and any number of trees and missing leaves. For these instance groups, the mean relative error for FHyNCH-TrivialRand ranged between 12.2% and 33.5%.

Once again confirming the behavior observed for synthetic data, FHyNCH-TrivialRand performs the best on small instances, its results getting worse with increasing values of parameters L and $|\mathscr{T}|$. In particular, FHyNCH-TrivialRand is the best-performing method, on average, for all instance groups with $|\mathscr{T}| = 10$ and $L \leqslant 50$. Increasing L and $|\mathscr{T}|$, FHyNCH-MultiML outperforms FHyNCH-TrivialRand in more and more instance groups: in particular, it is the best-performing method for all groups with $L \geqslant 20$ and $|\mathscr{T}| \geqslant 50$, and it is the best one for 7 out of 9 instance groups with $L = 20$ and $|\mathscr{T}| \in \{20, 30, 40\}$.

Running time In Table 7, we report the average running time of FHyNCH-MultiML within each of the large instance groups of Table 3. Noticeably, the average running time for the group with the largest instances (60 trees with up to 100 leaves and 30% missing leaves) is under 15 min.

4. Conclusions

We presented FHyNCH-MultiML, the first heuristic scheme specifically designed to solve the hybridization problem for large sets of multifurcating phylogenetic trees with missing leaves. FHyNCH-MultiML combines the use of two suitably designed machine-learning models with the technique of cherry-picking.

Experiments on synthetically generated data sets suggest that the results obtained with our method are qualitatively good, given the hardness of the problem: the number of reticulations in the generated networks is always within a small constant factor from the number of reticulations of the network the trees were sampled from. These results are particularly impressive in the case of large inputs consisting of 100 multifurcating trees on a set of 100 taxa with missing leaves. Although it is hard to evaluate the performance of the method on real data because of the lack of reference values (since, before this work, no method existed for this problem) we show that on large enough instances FHyNCH-MultiML is systematically better than repeating a randomized heuristic many times and choosing the best solution.

This work shows the potential for combining machine learning with cherry picking for phylogenetic reconstruction. The major advantage of this approach is its versatility. Indeed, the method presented here can be applied to an arbitrary phylogenetic tree data set. This is an important step forward in the field of phylogenetic networks since all previous methods were limited to restricted types of data. Hence, an important next step is to train the model on very large amounts of data to further improve its performance. Also, the use of more complex machine learning models, such as graph neural networks, could be investigated. In addition, although our method uses branch lengths of input trees within the algorithm to predict which cherries to pick, it does not yet use them to predict the branch lengths of the output network. Finally, in this paper, we have only evaluated the method in terms of the number of reticulations of the constructed network. In future work, it is important

Table 7
Running times for the large instances extracted from the *Bacterial and Archaeal Genomes* data set. For each instance group, we give the average running time in seconds. Dashes indicate empty instance groups.

L	MI	10 trees	20 trees	30 trees	40 trees	50 trees	60 trees
10	0.1	7.5	9.6	9.6	12.4	18.0	17.0
	0.2	3.2	5.9	7.7	12.7	21.2	16.6
	0.3	3.5	5.0	9.3	12.6	18.6	16.1
20	0.1	11.2	24.3	39.8	64.5	79.8	124.4
	0.2	9.7	19.1	32.2	57.8	69.0	106.9
	0.3	9.4	21.1	27.1	49.2	57.5	75.6
50	0.3	23.0	-	-	-	-	-
100	0.1	75.6	200.4	348.2	458.6	-	-
	0.2	79.5	195.4	346.5	511.4	683.8	856.7
	0.3	76.6	185.4	329.7	481.6	661.8	833.3
150	0.3	106.9	-	-	-	-	-

Table 6
Results for the experiments on the large instances extracted from the *Bacterial and Archaeal Genomes* data set. FHyNCH-MultiML and FHyNCH-TrivialRand are denoted by MML and TR, respectively; the results in columns labeled MML report the mean relative error (in %) of the results of FHyNCH-MultiML against FHyNCH-TrivialRand; and symmetrically for the columns labeled TR. We highlight in bold the smallest of the two errors for each instance group, identifying the best-performing method for each group. Dashes denote empty instance groups (see also Table 3).

L	MI	10 trees		20 trees		30 trees		40 trees		50 trees		60 trees	
		MML	TR	MML	TR	MML	TR	MML	TR	MML	TR	MML	TR
10	0.1	**7.8**	8.0	9.0	**2.9**	**1.7**	10.4	28.6	**0.0**	7.8	**0.0**	**0.0**	6.7
	0.2	7.0	**1.4**	**11.5**	12.2	17.5	**2.3**	**8.0**	9.9	9.1	**7.3**	9.5	**7.2**
	0.3	13.1	**0.8**	7.5	**4.7**	12.0	**9.1**	9.1	**2.2**	**1.6**	9.1	8.7	**8.2**
20	0.1	7.9	**3.6**	**2.7**	7.5	**2.6**	6.0	**0.1**	11.2	**2.3**	9.2	**0.1**	11.3
	0.2	4.6	**1.8**	4.5	**3.8**	**2.0**	5.8	**0.0**	9.4	**0.2**	8.4	**0.0**	9.2
	0.3	8.4	**0.6**	**2.1**	9.2	**6.7**	9.8	3.8	**3.5**	**2.4**	19.9	**2.7**	7.8
50	0.3	5.1	**1.0**	-	-	-	-	-	-	-	-	-	-
100	0.1	**0.0**	15.9	**0.0**	19.1	**0.0**	20.4	**0.0**	16.9	-	-	-	-
	0.2	**0.0**	19.4	**0.0**	19.5	**0.0**	18.4	**0.0**	17.1	**0.0**	17.6	**0.0**	33.5
	0.3	**0.0**	20.3	**0.0**	18.7	**0.0**	19.7	**0.0**	24.3	**0.0**	22.0	**0.0**	24.0
150	0.3	**0.0**	12.2	-	-	-	-	-	-	-	-	-	-

to analyze how close the constructed networks are to the original simulated network, topologically, for example using tail-moves (Janssen et al., 2018) with edge-insertions/deletions.

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)”, from the MUR - FSE REACT EU - PON R&I 2014–2020 and from the PANGAIA and ALPACA projects that have received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreements No 872539 and 956229, respectively.

Declaration of Competing Interest

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

Data availability

Data will be made available on request.

References

Abadi, Shiran, Avram, Oren, Rosset, Saharon, Pupko, Tal, Mayrose, Itay, 2020. Modelteller: model selection for optimal phylogenetic reconstruction using machine learning. Mol. Biol. Evol. 37 (11), 3338–3352.

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

Albrecht, Benjamin, Scornavacca, Céline, Cenci, Alberto, Huson, Daniel H., 2012. Fast computation of minimum hybridization networks. Bioinform. 28 (2), 191–197.

Azouri, Dana, Abadi, Shiran, Mansour, Yishay, Mayrose, Itay, Pupko, Tal, 2021. Harnessing machine learning to guide phylogenetic-tree search algorithms. Nat. Commun. 12 (1), 1–9.

Dana Azouri, Oz Granit, Michael Alburquerque, Yishay Mansour, Tal Pupko, Itay Mayrose, 2023. The tree reconstruction game: phylogenetic reconstruction using reinforcement learning. CoRR, abs/2303.06695.

Bapteste, E., van Iersel, L., Janke, A., Kelchner, S., Kelk, S., McInerney, J.O., Morrison, D. A., Nakhleh, L., Steel, M., Stougie, L., Whitfield, J., 2013. Networks: expanding evolutionary thinking. Trends in Genetics 29 (8), 439–441.

Baroni, Mihaela, Semple, Charles, Steel, Mike, 2005. A framework for representing reticulate evolution. Ann. Comb. 8, 391–408.

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

Giulia Bernardini, Leo van Iersel, Esther Julien, and Leen Stougie, 2022. Reconstructing phylogenetic networks via cherry picking and machine learning. In 22nd International Workshop on Algorithms in Bioinformatics (WABI), volume 242 of LIPIcs, pages 16:1–16:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Bernardini, Giulia, van Iersel, Leo, Julien, Esther, Stougie, Leen, 2023. Constructing phylogenetic networks via cherry picking and machine learning. Algorithms Mol. Biol. 18 (13).

Bhattacharjee, Ananya, Bayzid, Md Shamsuzzoha, 2020. Machine learning based imputation techniques for estimating phylogenetic trees from incomplete distance matrices. BMC Genom. 21, 1–14.

Bordewich, Magnus, Semple, Charles, 2007. Computing the hybridization number of two phylogenetic trees is fixed-parameter tractable. IEEE/ACM Trans. Comput. Biol. Bioinf. 4 (3), 458–466.

Bordewich, Magnus, Semple, Charles, 2007. Computing the minimum number of hybridization events for a consistent evolutionary history. Discr. Appl. Math. 155 (8), 914–928.

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

Boto, Luis, 2010. Horizontal gene transfer in evolution: facts and challenges. Proc. Roy. Soc. B: Biol. Sci. 277 (1683), 819–827.

Huber, Katharina T., Moulton, Vincent, Spillner, Andreas, 2021. Phylogenetic consensus networks: Computing a consensus of 1-nested phylogenetic networks. arXiv preprint arXiv:2107.09696.

Humphries, Peter J., Linz, Simone, Semple, Charles, 2013. Cherry picking: a characterization of the temporal hybridization number for a set of phylogenies. Bull. Math. Biol. 75 (10), 1879–1890.

Huson, Daniel H., Linz, Simone, 2018. Autumn algorithm - computation of hybridization networks for realistic phylogenetic trees. IEEE ACM Trans. Comput. Biol. Bioinform. 15 (2), 398–410.

Huson, Daniel H., Rupp, Regula, Scornavacca, Celine, 2010. Phylogenetic networks: concepts, algorithms and applications. Cambridge University Press.

Huson, Daniel H., Scornavacca, Celine, 2012. Dendroscope 3: an interactive tool for rooted phylogenetic trees and networks. Syst. Biol. 61 (6), 1061–1067.

Janssen, Remie, Jones, Mark, Erdős, Péter L, 2018. Leo Van Iersel, and Celine Scornavacca. Exploring the tiers of rooted phylogenetic network space using tail moves. Bull. Math. Biol. 80, 2177–2208.

Janssen, Remie, Murakami, Yukihiro, 2021. On cherry-picking and network containment. Theoret. Comput. Sci. 856, 121–150.

Koblmüller, S., Duftner, N., Sefc, K.M., Aibara, M., Stipacek, M., Blanc, M., Egger, B., Sturmbauer, C., 2007. Reticulate phylogeny of gastropod-shell-breeding cichlids from Lake Tanganyika–the result of repeated introgressive hybridization. BMC Evolutionary Biology 7, 1–13.

Kulikov, N., Derakhshandeh, F., Mayer, C., 2023. Machine learning can be as good as maximum likelihood when reconstructing phylogenetic trees and determining the best evolutionary model on four taxon alignments. bioRxiv 2023-07.

Kumar, Sudhir, Sharma, Sudip, 2021. Evolutionary sparse learning for phylogenomics. Mol. Biol. Evol. 38 (11), 4674–4682.

Randal Linder, C., Moret, Bernard M.E., Nakhleh, Luay, Warnow, Tandy, 2004. Network (reticulate) evolution: biology, models, and algorithms. In: The Ninth Pacific Symposium on Biocomputing (PSB).

Randal Linder, C., Rieseberg, Loren H., 2004. Reconstructing patterns of reticulate evolution in plants. Am. J. Botany 91 (10), 1700–1708.

Linz, Simone, Semple, Charles, 2019. Attaching leaves and picking cherries to characterise the hybridisation number for a set of phylogenies. Adv. Appl. Math. 105, 102–129.

Mallet, J., 2005. Hybridization as an invasion of the genome. Trends in Ecol. Evol. 20 (5), 229–237.

Mallet, James, Besansky, Nora, Hahn, Matthew W., 2016. How reticulated are species? BioEssays 38 (2), 140–149.

Mirzaei, Sajad, Yufeng, Wu., 2015. Fast construction of near parsimonious hybridization networks for multiple phylogenetic trees. IEEE/ACM Trans. Comput. Biol. Bioinf. 13 (3), 565–570.

Nakhleh, Luay, 2010. Evolutionary phylogenetic networks: models and issues. In: Problem solving handbook in computational biology and bioinformatics. Springer, pp. 125–158.

Pardi, Fabio, Scornavacca, Celine, 2015. Reconstructible phylogenetic networks: do not distinguish the indistinguishable. PLoS Comput. Biol. 11 (4), e1004135.

Park, H., Jin, G., Nakhleh, L., 2010. Algorithmic strategies for estimating the amount of reticulation from a collection of gene trees. In: Proceedings of the 9th Annual International Conference on Computational Systems Biology. Citeseer, pp. 114–123.

Park, H.J., Nakhleh, L., 2012. Inference of reticulate evolutionary histories by maximum likelihood: the performance of information criteria. In: BMC bioinformatics, Vol. 13. BioMed Central, pp. 1–10.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: machine learning in python. J. Mach. Learn. Res. 12, 2825–2830.

Piovesan, Teresa, Kelk, Steven M, 2012. A simple fixed parameter tractable algorithm for computing the hybridization number of two (not necessarily binary) trees. IEEE/ACM Trans. Comput. Biol. Bioinf. 10 (1), 18–25.

Pons, Joan Carles, Scornavacca, Celine, Cardona, Gabriel, 2019. Generation of level-*k* LGT networks. IEEE/ACM Trans. Comput. Biol. Bioinf. 17 (1), 158–164.

Smith, Megan L., Hahn, Matthew W., 2023. Phylogenetic inference using generative adversarial networks. Bioinformatics 39 (9), btad543.

van Iersel, Leo, Janssen, Remie, Jones, Mark, Murakami, Yukihiro, 2022. Orchard networks are trees with additional horizontal arcs. Bull. Math. Biol. 84 (8), 76.

van Iersel, Leo, Janssen, Remie, Jones, Mark, Murakami, Yukihiro, Zeh, Norbert, 2021. A unifying characterization of tree-based networks and orchard networks using cherry covers. Adv. Appl. Math. 129, 102222.

van Iersel, Leo, Janssen, Remie, Jones, Mark, Murakami, Yukihiro, Zeh, Norbert, 2022. A practical fixed-parameter algorithm for constructing tree-child networks from multiple binary trees. Algorithmica 84, 917–960.

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

Zhang, Louxin, Abhari, Niloufar, Colijn, Caroline, Yufeng, Wu., 2023. A fast and scalable method for inferring phylogenetic networks from trees by aligning lineage taxon strings. Genome Res. 33 (7), 1053–1060.

Zhu, Tujin, Cai, Yunpeng, 2021. Applying neural network to reconstruction of phylogenetic tree. In: ICMLC 2021: 13th International Conference on Machine Learning and Computing. ACM, pp. 146–152.