Bioinformatics, 38(15), 2022, 3725–3733
https://doi.org/10.1093/bioinformatics/btac396
Advance Access Publication Date: 17 June 2022
Original Paper

Phylogenetics

NetRAX: accurate and fast maximum likelihood phylogenetic network inference

Sarah Lutteropp [1,*], Céline Scornavacca[2], Alexey M. Kozlov [1], Benoit Morel [1,3] and Alexandros Stamatakis [1,3]

[1]Computational Molecular Evolution Group, Heidelberg Institute for Theoretical Studies, Heidelberg 69118, Germany, [2]Institut des Sciences de l'Évolution Université de Montpellier, CNRS, IRD, EPHE Place Eugène Bataillon, 34095 Montpellier Cedex 05, France and [3]Institute for Theoretical Informatics, Karlsruhe Institute of Technology, Karlsruhe 76128, Germany

*To whom correspondence should be addressed.

Associate Editor: Russell Schwartz

Received on December 16, 2021; revised on May 11, 2022; editorial decision on May 30, 2022; accepted on June 14, 2022

Abstract

Motivation: Phylogenetic networks can represent non-treelike evolutionary scenarios. Current, actively developed approaches for phylogenetic network inference jointly account for non-treelike evolution and incomplete lineage sorting (ILS). Unfortunately, this induces a very high computational complexity and current tools can only analyze small datasets.

Results: We present NetRAX, a tool for maximum likelihood (ML) inference of phylogenetic networks in the absence of ILS. Our tool leverages state-of-the-art methods for efficiently computing the phylogenetic likelihood function on trees, and extends them to phylogenetic networks via the notion of 'displayed trees'. NetRAX can infer ML phylogenetic networks from partitioned multiple sequence alignments and returns the inferred networks in Extended Newick format. On simulated data, our results show a very low relative difference in Bayesian Information Criterion (BIC) score and a near-zero unrooted softwired cluster distance to the true, simulated networks. With NetRAX, a network inference on a partitioned alignment with 8000 sites, 30 taxa and 3 reticulations completes within a few minutes on a standard laptop.

Availability and implementation: Our implementation is available under the GNU General Public License v3.0 at https://github.com/lutteropp/NetRAX.

Contact: sarah.lutteropp@h-its.org

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

It is broadly accepted that one cannot always describe evolution via a phylogenetic tree. Events such as horizontal gene transfer, hybridization or recombination induce non-treelike evolutionary relationships among taxa. In such cases, a (rooted) phylogenetic network better describes the evolutionary relationships. A rooted phylogenetic network differs from a rooted phylogenetic tree as it also comprises nodes with two parents, so-called *reticulations*, in addition to regular tree nodes with one parent. A reticulation represents a non-treelike event.

Initially, phylogenetic network inference methods based on maximum likelihood (ML) such as NEPAL (NEPAL, 2006) and PhyloDAG (Nguyen and Roos, 2015) did not account for incomplete lineage sorting (ILS). Very recently, a new ILS-unaware ML network inference tool, PhyLiNC, has been proposed (Allen-Savietta, 2020). The tool is available as part of the PhyloNetworks (Solís-Lemus *et al.*, 2017) package. However, it is deactivated by default and the authors emphasize that it is not ready for use yet (Ané, 2021). Unfortunately, NEPAL and PhyloDAG do also not appear to be able to reconstruct phylogenetic networks from genomic data, as we elaborate in Section 3.4.

In recent years, the focus shifted toward developing methods for ML network inference that also account for ILS. While models accounting for ILS are expected to yield more accurate networks as they incorporate additional mechanisms to explain non-treelike evolution, they face substantial computational challenges. For example, the ILS-aware ML method implemented in PhyloNET (Wen *et al.*, 2018) can only be applied to extremely small datasets with 10 taxa and up to 4 reticulations (Solís-Lemus and Ané, 2016). Consequently, faster-to-compute pseudolikelihood models accounting for ILS, such as implemented in SNaQ (Solís-Lemus and Ané, 2016), were developed. These pseudolikelihood models first compute ILS-aware likelihoods of 4-taxon subtrees (quartets) on gene trees, and subsequently compute a pseudolikelihood for the entire network from these quartet likelihoods. More recent versions of the aforementioned PhyloNET tool also deploy pseudolikelihoods (Wen *et al.*, 2018), but still face scalability challenges (Cao *et al.*, 2019). Apart from ML-based network inference tools, there also exist tools that rely on maximum parsimony (e.g. Nakhleh *et al.*, 2005), which often perform poorly (Hejase and Liu, 2016), or Bayesian inference (e.g. Zhang *et al.*, 2018), which also face substantial scalability challenges.

Here, we present NetRAX, a tool for ML inference of phylogenetic networks that does not account for ILS. By leveraging state-of-the-art methods for efficiently computing the phylogenetic likelihood function on trees and extending them to networks, our tool permits to run—on a standard laptop and within minutes—phylogenetic analyses on partitioned alignments for evolutionary scenarios comprising up to four reticulations. Based on its performance on simulated data and on an empirical dataset, NetRAX allows to routinely infer non-treelike evolutionary histories. The remainder of this article is structured as follows. We start by introducing the NetRAX likelihood model (Section 2.1) and outline its implementation (Section 2.2). We then describe the optimization of branch lengths and model parameters that are not associated with the network topology (Sections 2.3 and 2.4). Next, we outline the available topology-rearrangement moves (Section 2.5) and present the search algorithm for finding the best-scoring network (Section 2.6). To compare network topologies, we implement normalized versions of common network distance measures (Section 3.1). We then describe the synthetic data generation (Section 3.2) and our experimental setup (Section 3.3). The NetRAX performance is assessed on synthetic data (Section 3.4) and on empirical data (Section 4). We conclude and discuss future work in Section 5.

2 Materials and methods

In the following, we provide a general, abstract outline of NetRAX. Additional technical details are provided in the Supplementary Material.

2.1 Phylogenetic network likelihood model

A rooted binary phylogenetic network N is a single-source, directed, acyclic graph without parallel edges. We call its source node the *root* node of N. Apart from the root, there are three types of nodes: (i) internal tree nodes with one incoming edge and two outgoing edges, (ii) reticulation nodes with two incoming edges and one outgoing edge and (iii) leaf nodes with one incoming edge and no outgoing edges. Each leaf is associated with a distinct taxon. Each edge e in a phylogenetic network has a branch length and a probability $P(e)$. The incoming edges of a reticulation node (called *reticulation edges*) are assigned inheritance probabilities that must sum to one. The probability of observing a tree (i.e. non-reticulation) edge is always one.

Figure 1 shows a simple example network.

Phylogenetic analyses are typically conducted on multiple sequence alignments (MSA) $\mathcal{A}$ subdivided into multiple blocks $A_1, \ldots, A_p$. A block consists of a set of sites that are likely to have evolved together (e.g. sites of a single gene), following the same evolutionary process. With NetRAX, our goal is to infer a phylogenetic network N from a partitioned MSA $\mathcal{A}$ that maximizes a network likelihood $L(N|\mathcal{A})$.

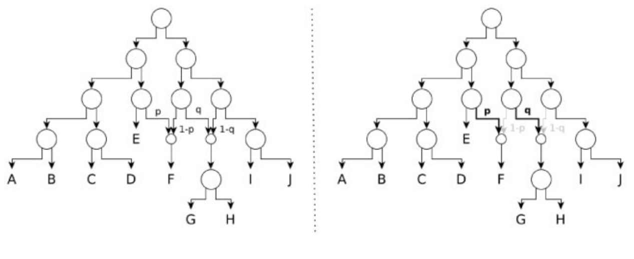


Fig. 1. Left: A phylogenetic network with two reticulation nodes. Right: A displayed tree of the phylogenetic network on the left. The probability of displaying the highlighted tree is the product $p * q$ over the respective reticulation probabilities

Here, we assume that neither ILS nor recombination occurs among MSA sites. Given these assumptions, we can model reticulate evolution in a network via its set of induced *displayed trees* (Jin *et al.*, 2006). We obtain a displayed tree from a network by choosing one parent per reticulation node (disabling the incoming edges belonging to parents that are not chosen). We can convert it into a phylogenetic tree by suppressing unlabeled leaves as well as single-child paths. Figure 1 shows a displayed tree in a phylogenetic network.

Let $N = (V, E)$ be a phylogenetic network with a set of displayed trees $\mathcal{T}(N)$. To compute the probability of a displayed tree T in $\mathcal{T}(N)$, we proceed as follows. Let E_r be the set of reticulation edges that need to be taken in order to generate T. The probability of T in N is then:

$$P(T|N) = \prod_{e \in E_r} P(e).$$

In cases where we encounter the exact same phylogenetic tree more than once, we compute the sum over the respective log likelihoods, which are weighted by the corresponding reticulation probabilities.

To define $L(N|\mathcal{A})$, let $\mathcal{A}$ be a partitioned MSA with blocks $A_1, \ldots, A_p$, that is, every MSA site is assigned to exactly one block A_i. Let $\vartheta = (\vartheta_1, \ldots, \vartheta_p)$ be the parameter vector, comprising the per-block network branch lengths and likelihood model parameters. We consider each block as being independent. Thus, we define the likelihood of a phylogenetic network given ϑ as the product over the per-block likelihoods: $L(N|\mathcal{A}, \vartheta) = \prod_{i=1}^{p} L(N|A_i, \vartheta_i)$.

To avoid numerical underflow, we take the logarithm. We assess and implement two versions for computing the log likelihood (lnL) on networks on partitioned MSAs. They both aggregate over the lnL of the trees displayed by the network (Jin *et al.*, 2006):

$$\ln L(N|A_i, \vartheta_i) = \ln\Big(\sum_{T \in \mathcal{T}(N)} L(T|A_i, \vartheta_i) * P(T|N) \Big), \tag{1}$$

$$\ln L(N|A_i, \vartheta_i) = \ln\Big(\max_{T \in \mathcal{T}(N)} L(T|A_i, \vartheta_i) * P(T|N) \Big), \tag{2}$$

where $L(T|A_i, \vartheta_i)$ is the standard phylogenetic likelihood function for a tree T, given an alignment A_i and the parameter vector ϑ_i. Equation (1) presents the Weighted Average version (`LhModel.AVERAGE`), and Equation (2) the Best Tree version (`LhModel.BEST`). In the former, the likelihood of a network for a given block is the weighted average over the displayed tree likelihoods. We use the sum here, because the probability of event A or B to occur corresponds to the sum over the probability of observing A and the probability of observing B. The weighted average can thus be interpreted as the expected value, if we treat each displayed tree as a statistical event.

Note that both definitions face the same identifiability issue: they cannot distinguish between different rooted network topologies that display the same set of trees. Thus, the networks reconstructed by NetRAX should be considered as semi-rooted (Solís-Lemus and Ané, 2016) and unzipped (Pardi and Scornavacca, 2015).

Because we need to compute the logarithm of a sum over very small numbers and we need to exponentiate the displayed tree lnLs, we use arbitrary-precision arithmetic [using MPFR C++ library (Holoborodko, 2010)] to compute $L(N|\mathcal{A}_i)$ from the per-block displayed tree likelihoods. As in the RAxML-NG (Kozlov *et al.*, 2019) tool for ML phylogenetic tree inference, we use the libpll (Flouri, 2015b) and pll-modules (Darriba, 2016) libraries to compute displayed tree lnLs via the standard Felsenstein pruning algorithm (Felsenstein, 1981).

Currently, NetRAX supports two branch length models. Under the linked branches model, we share the same set of branch lengths among *all* blocks. Under the unlinked branches model, each block has its own, independent set of branch lengths. The choice of the model has an effect on the type of reticulations we can recover.

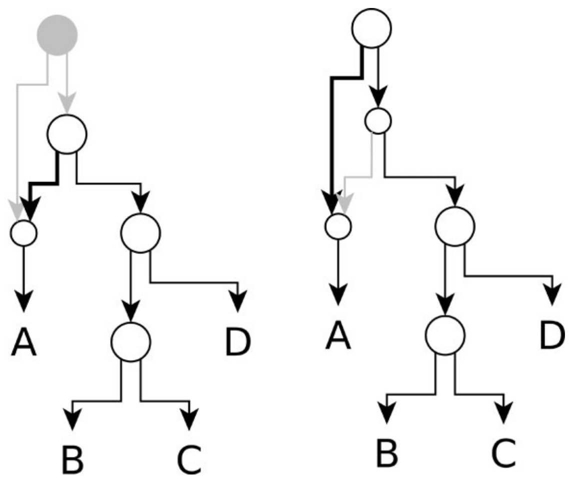


Fig. 2. Two displayed trees in a phylogenetic network. Both displayed trees induce the same topology after collapsing single-child nodes. They only differ in some branch lengths. For example, in the left tree, the branch length between the root node and leaf A is $b_3 + b_5$, and in the right tree it is $b_1 + b_5$. Under the unlinked branches model, NetRAX would simply return a tree. For phylogenetic tree likelihood computation, simple paths are collapsed and the tree is transformed into an unrooted tree. Therefore, b_2 is not considered in the left displayed tree

Figure 2 shows an example network with a reticulation that cannot be recovered under the unlinked branches model. By default, NetRAX uses the linked branch model.

2.2 Computing the likelihood of a phylogenetic network

In order to compute the lnL of a network N using the above formulas, we initially need to compute the per-block likelihoods $L(T|A_i, \vartheta_i)_{i\in\{1,\dots,p\}}$ and the probabilities $P(T|N)$ of all trees $T \in \mathcal{T}(N)$ displayed by N.

We use the libpll library to compute $\ln L(T|\mathcal{A}_i, \vartheta_i)$. To compute the per-block lnLs of a phylogenetic tree, libpll uses an internal per-node data structure, called *conditional likelihood vector* (CLV, Flouri, 2015a) first introduced by Felsenstein (1981). A CLV for a node v stores the per-site likelihoods for the subtree rooted at v. The libpll library computes the per-node CLVs via a post-order traversal of the tree using Felsenstein's pruning algorithm (Felsenstein, 1981). It computes the CLV of a given node based on the CLVs of its respective children. The libpll library also provides incremental likelihood computations: it only updates (re-computes) those CLVs affected by a topological rearrangement move or branch length change and re-uses unaffected CLVs that are still valid.

In NetRAX, we do not store each displayed tree topology separately, but use a network data structure that implicitly induces each tree. This allows us to avoid redundant CLV computations: When two displayed trees share an identical subtree, there is no need to compute the CLVs for nodes in this subtree more than once (see further below).

Finally, we also parallelize per tree lnL computations over the MSA sites by using the Message Passing Interface (MPI).

Sharing CLVs among displayed trees: Naïvely, by explicitly iterating over each displayed tree in a phylogenetic network N with n nodes and r reticulations, one would require $n * 2^r$ CLVs to calculate the lnL of the network: one CLV per node *and* displayed tree. To improve efficiency, for each node v in N, we store as many CLVs as there are different displayed subtree topologies rooted at v. By sharing CLVs among identical subtrees in multiple displayed trees, we reduce the total number of CLVs required to compute the lnL of this network. To implement this CLV sharing optimization, we update the CLVs via a bottom up traversal (using a reversed topological sort) of the nodes in the phylogenetic network. For each node v we visit, we update the CLVs for each of the distinct displayed subtree topologies rooted at v. More information can be found in the Supplementary Material.

In the following, we describe how we optimize the network topology N and its associated parameter vector ϑ to maximize $\ln L(N|\mathcal{A}, \vartheta)$.

2.3 Branch length optimization

Our goal is to optimize a branch length b in a network N, with respect to the lnL of the network. Overall, we aim at finding the branch length assignments $\hat{b}_1, \dots, \hat{b}_p$ that maximize $\ln L(N|A, \vartheta)$.

As in standard ML implementations for tree inference, we optimize b via the Newton-Raphson method. For this, we need the first and second derivatives of the network lnL with respect to b. We derive formulas for efficiently computing $(\ln L(N|A, \vartheta))'$ and $(\ln L(N|A, \vartheta))''$ from $\ln L(T|A_i, \vartheta_i)$, $(\ln L(T|A_i, \vartheta_i))'$, and $(\ln L(T|A_i, \vartheta_i))''$ in the Supplementary Material. Note that, when optimizing a branch length b, we neither need to recompute the per-block lnL nor its derivatives for displayed trees not containing b.

In the following, we describe the efficient computation of the per-block lnL and the per-block lnL derivatives for all displayed trees.

Branch-length optimization in phylogenetic trees: In order to avoid costly CLV updates when optimizing a branch (u, v) in a tree, most modern ML tree inference tools reroot the tree at node u before optimizing the branch. After rerooting the tree, the edge directions of those edges that lie on the path from the new to the old root change. We thus need to recompute the CLVs for the nodes residing on this path. When evaluating different values for the branch length (u, v) in the rerooted tree, we can simply reuse the CLVs stored at u and v. Note that the likelihood is the same regardless of the root placement under the commonly used time-reversible models of evolution.

Rerooting displayed trees in a phylogenetic network: In NetRAX, we deploy an analogous strategy for efficient branch length optimization. Before optimizing a branch (u, v), we need to reroot *all* displayed trees containing the branch at the source node u.

Recall that we do not explicitly store each displayed tree topology in order to avoid redundant CLV updates during the network lnL computation. Instead, we perform an appropriate bottom-up traversal of the nodes in the phylogenetic network and update all unique subtree CLVs shared among subsets of displayed trees. This complicates the rerooting operation for the displayed trees. The difficulty here is that with the original network root, the edge directions (and thus, the parent–child relationships needed for computing CLVs) are identical for all displayed trees. But when rerooting the displayed trees, the parent–child relationships depend on the specific displayed tree we are currently considering (see Fig. 3). These differing edge directions affect the order in which we need to process the nodes when recomputing the CLVs.

For networks, we need to recompute the CLVs (which are shared among subsets of displayed trees) that lie on *any* path between the old root and the new root (both ends included). We devised the following approach for resolving the edge direction problem: we successively process the paths between the network root and the new root. Before processing the next path, we invalidate the shared CLVs at all nodes on the current path, except for the new root node. We detect, in advance, at which nodes we need to restore the old

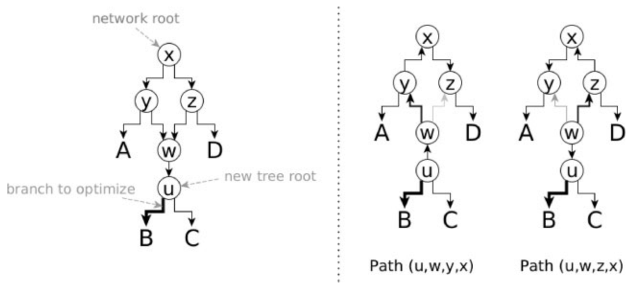


Fig. 3. We reroot the displayed trees at node u before optimizing branch (u, v). In the first rerooted displayed tree, the node y is a parent of node x and we need to recompute the CLVs on the path (u, w, y, x). In the second rerooted displayed tree, the node y is a child of node x and we need to recompute the CLVs on the path (u, w, z, x)

shared CLVs (to the values they had when using the original root), save and restore them accordingly.

After we finished optimizing a branch, we recompute the CLVs with regard to the original network root. We do this because, in networks, unlike for phylogenetic trees, the root placement *does* influence the lnL value.

2.4 Optimizing non-topology parameters

Apart from optimizing branch lengths, we also need to optimize the evolutionary model parameters [by default, we use the GTR + Γ (Tavaré *et al.*, 1986) model, although NetRAX supports all models that are supported by RAxML-NG] and reticulation probabilities. Recall that our goal is to optimize the overall network lnL. This is, we aim to find optimal parameters for the parameter vector ϑ, to maximize $\ln(L(N|A,\vartheta))$.

Likelihood model parameter optimization: We directly reuse routines from RAxML-NG for optimizing these parameters as they do not depend on an explicit tree or network topology.

Optimizing reticulation probabilities: For optimizing the first-parent probability p of a reticulation, we also reuse Brent's single-parameter optimization method as implemented in RAxML-NG. Since the first and second parent probability of a reticulation sum to 1.0, the probability for taking the second parent follows from the first parent probability.

The Brent optimization method requires recomputing the network lnL when p changes. Fortunately, per-block lnLs of the displayed trees do *not* depend on p. Thus, changing p only affects the probabilities $P(T|N), T \in \mathcal{T}(N)$ of displaying the trees. We can thus re-use the existing (already computed) per-block lnLs for displayed trees when recomputing the network lnL during the optimization of p.

2.5 Supported topology-rearrangement moves

NetRAX implements the following rooted network topology rearrangement moves as proposed by Gambette *et al.* (2017): rNNI move, rSPR move, arc insertion move, arc removal move.

Note that rNNI and rSPR moves are a generalization of the corresponding operations on trees (see Supplementary Material). We also provide an efficient implementation of the respective reversal (undo) operations. When undoing a move, we restore the original topology and branch lengths. Doing or undoing a move also invalidates some CLVs. Evidently, there exists a trade-off between recomputing the invalidated CLVs versus storing them. In our current implementation, we simply recompute the CLVs to reduce code complexity as well as memory requirements.

Comparing networks of different complexity: NetRAX supports vertical topology-rearrangement moves that increase (arc insertion move) or decrease (arc removal move) the number of reticulations in a network. Because model complexity changes when adding or removing reticulations from a network, we cannot compare networks of different complexity directly via their respective lnLs. For this, NetRAX implements AIC, AICc and BIC scoring. By default, NetRAX uses the BIC score to compare different networks, since Park and Nakhleh (2012) showed that using BIC performs best in network searches, even though it is not a perfect solution (Blair and Ané, 2019).

The BIC score of a network N with r reticulations on a partitioned MSA $\mathcal{A}$ and parameter vector ϑ is defined as follows: $\mathrm{BIC}(N|\mathcal{A},\vartheta) = -2 * \ln L(N|\mathcal{A},\vartheta) + \#\mathrm{free\_parameters} * \ln(\mathrm{sample\_size})$. The free parameters are the substitution model parameters, the reticulation first-parent probabilities, and the branch lengths. The sample size is the product of the number of taxa and the number of MSA sites.

2.6 Network search

NetRAX uses a greedy hill climbing approach to search for network topologies. It deploys an outer search loop to iterate over different move types (see Section 2.5) and an inner search loop to search for the best-scoring network using a specific move type. We provide an overview in Figure 4.

In the outer search loop, we search in waves by repeatedly iterating over move types in the following order: arc removal, rSPR, rNNI, arc insertion. For each move type, we invoke an inner search loop. The outer search loop terminates when no move type improves the BIC score.

Start networks: NetRAX can initiate a network search from a given set of start networks provided in Extended Newick format. For example, it can be launched on a user-specified number of strictly bifurcating random and maximum parsimony trees or a best-known ML tree that can be generated, for instance, with RAxML-NG (Kozlov *et al.*, 2019) in a separate step.

Inner search loop: As already mentioned, the inner search loop searches for the best-scoring network using a single move type only.

Assembling the set of move candidates: We can reach multiple alternative network topologies by applying a single move of a given type to the current network. We call such a move a *move candidate*.

We build the set of move candidates for a specific move type by iterating over all nodes in the network. For each node, we add the move candidates induced by applying the move to the current node to the set. When assessing possible move candidates for rSPR or arc insertion moves, NetRAX uses a default search radius of 5. This is, for each node in the network, we only consider and evaluate move candidates within a radius of 5 nodes around the current node. Due to their smaller neighborhood size, we do not restrict the search radius of rNNI and arc removal moves.

Filtering move candidates: In order to determine the most promising move candidate(s) and accelerate move candidate evaluation, we apply a three-stage pre-filtering process: we filter the move candidates using the PREFILTER, RANK and CHOOSE stages. These stages differ in the number of costly branch length optimizations we conduct, before scoring each candidate:

- PREFILTER—do not optimize branch lengths. (Exception: for arc insertion moves, we do need to optimize the length of the newly introduced branch.)
- RANK—optimize branches directly affected by the move.
- CHOOSE—optimize all branches in the network.

We use the Elbow Method (see Supplementary Material) to determine the number of promising move candidates to keep after each filtering stage. The most promising move candidate is the one with the lowest (=best) BIC score after the CHOOSE stage.

Accepting a move and updating the set of move candidates: If the most promising move candidate obtained by the CHOOSE stage yields a better-scoring network, we accept the move and apply it to the current network.

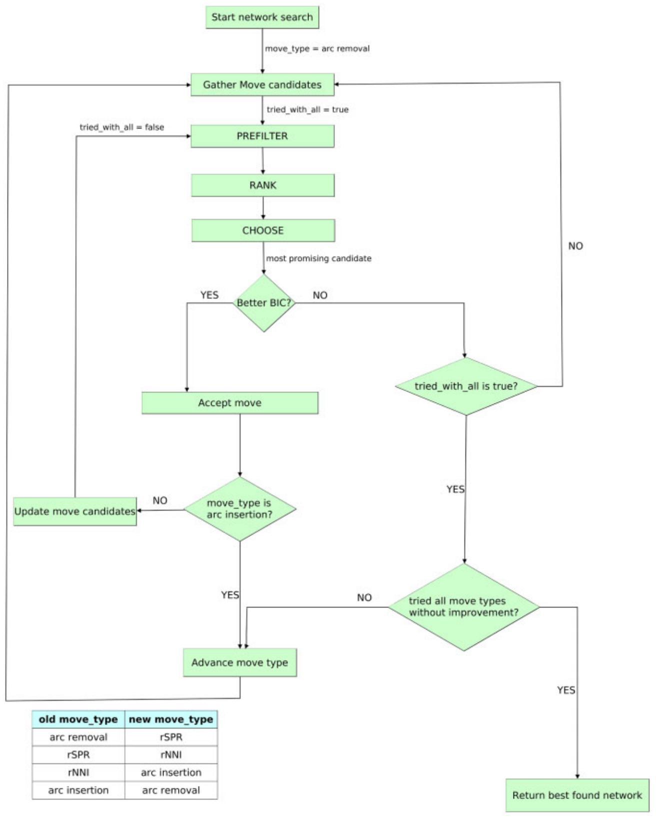
When accepting a move, we optimize all branch lengths, reticulation probabilities and remaining model parameters in this new best network.

If the inner search loop executes arc insertion moves, it terminates after accepting a move and immediately returns to the outer search loop. This is done in order to reduce the time spent optimizing a network with an excessively high reticulation count. For all other move types, after accepting a move, we continue searching for score-improving moves of the same move type until we cannot find a better-scoring network by considering other promising candidate moves from the PREFILTER phase. First, we remove previous promising moves that have become inapplicable after accepting the current move, and add new move candidates to the set, that are seeded at nodes directly affected by the accepted move. When we do not find a better-scoring network by searching this modified set of candidate moves, we again consider the complete set of move candidates. If these do also not yield a better-scoring network, the inner search loop terminates.

3 Simulation study

3.1 Topology-based evaluation

For comparing network topologies, we implement normalized versions of several distances (Huson *et al.*, 2010). Due to space constraints, we only discuss and report unrooted softwired cluster



old move_type	new move_type
arc removal	rSPR
rSPR	rNNI
rNNI	arc insertion
arc insertion	arc removal

Fig. 4. Overview of the NetRAX network search algorithm for a single start network. The table on the bottom left shows what move type is tried next when a wave for the current move type did not yield a better network. We loop through arc removal -> rNNI -> rSPR -> arc insertion move waves and repeat this as long as we find a better network. We terminate the search if the waves for arc removal, rNNI, rSPR, arc insertion do all not find a network with improved BIC score

distances (SCDs) here, and discuss the other measures in the Supplementary Material. Since the SCD is based on the topologies of displayed trees, it alleviates effects caused by network identifiability issues, where different networks can induce the same set of displayed trees (Pardi and Scornavacca, 2015). Furthermore, it resembles the Robinson–Foulds distance (Robinson and Foulds, 1981) on trees and is easy to interpret. We choose the unrooted version, as networks inferred by NetRAX should be interpreted as semi-rooted.

Every edge in a tree induces a *bipartition*, since its removal splits the set of taxa into two subsets. Bipartitions induced by edges leading to leaf nodes are *trivial*, as they are present in any tree on the given set of taxa.

(Normalized) unrooted softwired cluster distance: For a given network N, let $\mathcal{T}(N)$ be the displayed trees of N and let $B(N)$ be the set of all nontrivial bipartitions of the unrooted trees in $\mathcal{T}(N)$. The unrooted SCD between two networks N_1 and N_2 is:

$$\frac{|B(N_1) \triangle B(N_2)|}{|B(N_1) \cup B(N_2)|}.$$

3.2 Simulation of phylogenetic networks and sequences

Our simulator is included in the NetRAX GitHub repository. We simulate networks under the birth-reticulation process of Zhang *et al.* (2018) with the following parameter set $\lambda \sim$ U(0,20) + 5, $\nu = \lambda^*$ 0.003, τ_0 ~exp(20) + 0.1, where λ is the speciation rate, ν the reticulation rate, and τ_0 is the overall time the process is run. The parameters have been chosen to obtain a reasonable amount of reticulations with respect to the taxon number range we assess. Unless stated otherwise, we set the reticulation probabilities to 0.5 for all reticulations in the network.

We repeat our simulations until we obtain a network with the desired number of taxa and number of reticulations. Because our simulator generates ultrametric networks, we have to discard networks that contain unrecoverable reticulations, that is, networks for which at least two of its displayed trees have the same topology. Note that branch lengths are linked in our simulations.

Subsequently, we simulate sequences for each displayed tree of the simulated network, and concatenate them into a partitioned MSA (using one block per displayed tree), which is the input of NetRAX. We simulate the sequences using Seq-Gen-1.3.4 (Rambaut and Grass, 1997) with the following parameters: `-mHKY -t3.0 -f0.3,0.2,0.2,0.3`. Although NetRAX supports the GTR model, we simulated under HKY85 as some of the competing tools only support HKY85.

The block length for each displayed tree is proportional to the probability of displaying the tree. By default, we simulate $2^r * 1000$ MSA sites in total, where r is the number of reticulations in the network. We do not draw the number of MSA sites from some distribution to keep the datasets more comparable and the results easier to interpret.

3.3 Experimental setup

We conducted extensive experiments on simulated data. For each experiment, we report (i) the number of reticulations in the inferred network, (ii) the relative BIC-score difference between the true and the inferred network, (iii) the unrooted SCD between the true and inferred network and (iv) the total inference time. In the Supplementary Material, we provide additional plots and tables for relative AIC/cAIC/lnL differences, as well as further topology-based evaluation distances.

We evaluated NetRAX with `LhModel.AVERAGE` and `LhModel.BEST`, under the linked branch lengths model on phylogenetic networks and MSAs simulated as described in Section 3.2, under different settings:

A1: Multiple starting trees: We simulated 50 networks each for (i) 10 taxa and 1 reticulation, (ii) 20 taxa and 2 reticulations and (iii) 30 taxa and 3 reticulations.

A2: 40 Taxa: We simulated 50 networks each for (i) 40 taxa and 1 reticulation, (ii) 40 taxa and 2 reticulations, (iii) 40 taxa and 3 reticulations and (iv) 40 taxa and 4 reticulations.

B: Unpartitioned data: We simulated 50 datasets with 20 taxa and 1 reticulation. In addition to normal inference, we started a second inference where we merged all simulated blocks into a single block before running the inference.

C: Scrambled blocks: We simulated a single dataset with 30 taxa and 3 reticulations. Before executing NetRAX with a RAxML-NG ML starting tree, we randomly scrambled the blocks such that $p \in \{0\%, 10\%, 20\%, 30\%, 40\%, 50\%, 60\%, 70\%, 80\%, 90\%, 100\%\}$ of the sites from each block were randomly reassigned to other blocks. Our model assumes that all sites belonging to a block evolved together. Hence, we are violating this assumption to assess the stability of NetRAX under model violations.

D: Different alignment size: We simulated a single network with 30 taxa and 3 reticulations. For this network we then simulated {50, 100, 500, 1000, 5000, 10 000, 50 000, 100 000} sites per block.

E: Comparison with other tools: We simulated a partitioned 10-taxa 1 reticulation dataset with reticulation probability 0.5 and 2000 MSA sites. We inferred an ML tree, as well as two block trees (one for each block) with RAxML-NG. We also generated a set of 14 unique tree topologies out of 10 random and 10 maximum parsimony RAxML-NG trees, which we used for invoking NetRAX with a set of multiple starting trees.

For each simulated dataset, we initiated the NetRAX inference from a RAxML-NG ML tree. In addition, for the datasets in A1, we also started another NetRAX inference using three random and three maximum parsimony starting trees. We inferred networks using NetRAX, PhyLiNC, PhyloDAG, SNaQ, PlyloNET MPL (maximum pseudo-likelihood), PhyloNET MP (maximum parsimony) and PhyloNet ML. We ran PhyLiNC, SNaQ, and PhyloNET using both 1 and 2 as the maximum number of reticulations. In addition, we also compared the number of inferred reticulations with NetRAX, PhyloNET MP and PhyloDAG (as these were the fastest tools and performed well on the smaller dataset) on a 20 taxon 2 reticulations dataset with reticulation probabilities 0.5, and 4000 MSA sites. We also attempted an inference with NEPAL. However, the tool segfaulted and its authors unfortunately have lost its source code (NEPAL, 2006). Note that PhyLiNC is available on GitHub but not ready for use yet (Ané, 2021).

PhyLiNC, and PhyloDAG operate on unpartitioned data, whereas NetRAX requires a partitioned MSA. SNaQ operates on a set of quartets that can be inferred from gene trees, and a given starting topology (we used the best RAxML-NG ML tree). Both PhyloNet MPL and PhyloNet ML operate on a set of gene trees. We inferred the respective gene trees via RAxML-NG, one gene tree per MSA block. Note that SNaQ and PhyloNET account for ILS, while the remaining tools ignore ILS.

Details on the hardware used for the experiments and results on the parallel scalability of NetRAX are given in the Supplementary Material. The data underlying this article are available at https://cme.h-its.org/exelixis/material/netrax_data.zip.

3.4 Results and discussion

We only discuss representative results here, and refer to the Supplementary Material for comprehensive results (including percentiles and standard deviation) for all experiments. Here, we report the unrooted SCD to assess topological distances to the true simulated network. To quantify the NetRAX search algorithm quality, we compare the BIC scores of the true and the inferred networks. As we optimize for BIC, a worse inferred BIC indicates that the search algorithm got stuck in a local optimum. A better inferred BIC can be encountered because of the finite number of simulated MSA sites as ML is consistent on MSAs with infinite sites.

Multiple starting trees, 30 taxa, 3 reticulations: In Table 1 and Figure 5, we observe that running NetRAX inferences from multiple starting trees yields more accurate networks than running NetRAX from a single ML tree. This is because a single NetRAX inference can become stuck in local optima. However, initiating multiple independent NetRAX searches evidently results in a higher accumulated runtime.

Table 1. Summary statistics 30 taxa and 3 reticulations

A_30_3_random3_parsimony3	LhModel.AVERAGE	LhModel.BEST
Inferred BIC better or equal	6 (13.04%)	5 (10.87%)
Inferred BIC worse	40 (86.96%)	41 (89.13%)
Inferred n_reticulations less	6 (13.04%)	5 (10.87%)
Inferred n_reticulations equal	39 (84.78%)	40 (86.96%)
Inferred n_reticulations more	1 (2.17%)	1 (2.17%)
Unrooted SCD zero	17 (36.96%)	18 (39.13%)
A_30_3_ml1	**LhModel.AVERAGE**	**LhModel.BEST**
Inferred BIC better or equal	2 (4.35%)	3 (6.52%)
Inferred BIC worse	44 (95.65%)	43 (93.48%)
Inferred n_reticulations less	9 (19.57%)	9 (19.57%)
Inferred n_reticulations equal	34 (73.91%)	37 (80.43%)
Inferred n_reticulations more	3 (6.52%)	0 (0.00%)
Unrooted SCD zero	14 (30.43%)	17 (36.96%)

Note: Top: Starting from three maximum parsimony and three random trees. Bottom: Starting from the RAxML-NG ML tree. Networks that contain unrecoverable reticulations have been discarded, resulting in 46 networks.

`LhModel.BEST` performs slightly better than `LhModel.AVERAGE`, but the difference is not statistically significant.

Start from ML tree, 40 taxa, 4 reticulations: In Table 2, we see that NetRAX achieves slightly better inference results with `LhModel.AVERAGE` than `LhModel.BEST`. However, these differences are small, as we can see in Figure 6.

Unpartitioned data: In all NetRAX inference runs with an unpartitioned MSA, NetRAX inferred a bifurcating tree, under both `LhModel.AVERAGE` and `LhModel.BEST`. Thus, NetRAX is not able to infer reticulations on unpartitioned data.

Scrambled blocks: In Table 3, we observe that NetRAX is still able to infer a 'good' network if at most 20% of all MSA sites are perturbed among blocks. We observe no substantial difference in result quality between `LhModel.BEST` and `LhModel.AVERAGE`. The more we scramble (i.e. assign sites to the wrong partitions), the more similar the trees best explaining the MSA partitions become. This reduces the overall signal supporting different trees and induces a decrease in inferred reticulations.

Varying alignment size: In Table 4, we observe that for small MSAs, NetRAX under LhModel.BEST is faster than under LhModel.AVERAGE. But for larger MSAs, inferences under LhModel.AVERAGE are faste. The quality of the inferred network is similar among both likelihood types. Based on our empirical observations this surprising speed difference is because (i) a single network lnL computation under `LhModel.BEST` is faster than under `LhModel.AVERAGE`. (ii) for larger MSAs, we infer the best network with less overall moves, when using `LhModel.AVERAGE`. Note that we used datasets with very few, equally-sized blocks here and only one block per displayed tree. Note that with fewer MSA sites the signal present in the MSA also decreases, yielding it harder to obtain sufficient support in the data for introducing reticulations.

Comparison with other tools: For the simulated 10 taxa 1 reticulation dataset with 2000 MSA sites, we report the total runtime as well as unrooted SCD for all tools in Table 5.

NetRAX starting from the RAxML-NG ML tree, SNaQ, and PhyloNET MP with the maximum number of reticulations set to 1 showed perfect inference accuracy, with the unrooted SCD being zero. Only PhyloNET MP was faster than NetRAX, but the tool does not optimize branch lengths. PhyloNET MP with the maximum number of reticulations set to 2 inferred 2 reticulations.

On the simulated 20 taxa 2 reticulations dataset, PhyloDAG inferred a network with 14 reticulations. PhyloNET MP with the

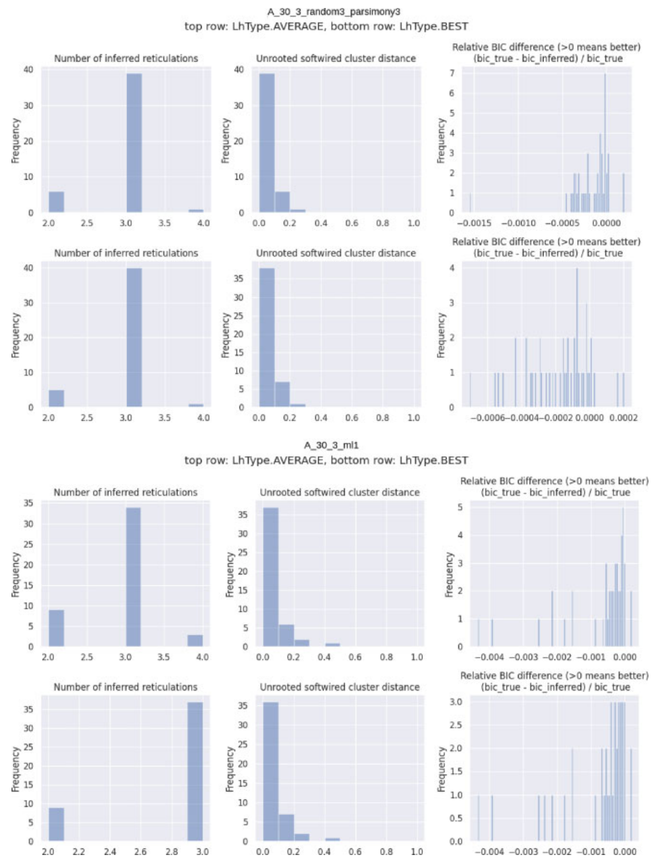


Fig. 5. Number of inferred reticulations, unrooted SCD and relative BIC difference for 50 datasets with 30 taxa and 3 reticulations each. Top: starting from 3 random and 3 maximum parsimony trees. Bottom: starting from a RAxML-NG ML tree

Table 2. Summary statistics for 40 taxa, 4 reticulations, starting from the RAxML-NG ML tree

A_nonrandom_40_4_nonrandom	LhModel.AVERAGE	LhModel.BEST
Inferred BIC better or equal	0 (0.00%)	0 (0.00%)
Inferred BIC worse	49 (100.00%)	49 (100.00%)
Inferred n_reticulations less	5 (10.20%)	10 (20.41%)
Inferred n_reticulations equal	43 (87.76%)	38 (77.55%)
Inferred n_reticulations more	1 (2.04%)	1 (2.04%)
Unrooted SCD zero	23 (46.94%)	19 (38.78%)

Note: Networks that contain unrecoverable reticulations have been discarded, resulting in 49 networks.

maximum number of reticulations set to 2 only inferred a single reticulation (with unrooted SCD 0.28). NetRAX correctly inferred 2 reticulations under all settings returning a better BIC than that of the true network and an unrooted SCD ranging between 0.19 and 0.22.

4 Empirical data

We executed a NetRAX inference on an empirical snake genomes dataset (Burbrink and Gehara, 2018; Chen *et al.*, 2017). We downloaded the individual gene alignments from `https://datadryad.org/stash/dataset/doi:10.5061/dryad.4qs50` and merged them into a partitioned MSA, treating each per gene MSA as one block. The merged dataset comprises 23 species, with one individual per species. There are 6737 distinct MSA site patterns in the merged MSA, and 304 blocks.

We inferred an ML tree for the complete dataset with RAxML-NG using its default GTR+GAMMA substitution model. We then used the ML tree inferred by RAxML-NG as starting network for NetRAX under the `LhModel.BEST` and `LhModel.AVERAGE` model, with linked branch lengths. We started additional NetRAX inferences using the 14 unique tree topologies contained in a set of 10 parsimony and 10 random starting trees.

We compared our NetRAX results with the 1-reticulation network inferred by SNaQ from the Burbrink and Gebara paper (see Supplementary Text for the detailed results). In all cases, NetRAX inferred a better BIC score than the published network (which can be expected, since the reported BIC is based on the network likelihood definition used by NetRAX) and in most cases recovered a 1-reticulation network highly similar to the one inferred by SNaQ. Nonetheless, the 2-reticulation network recovered by one NetRAX run also appears to be biologically plausible.

Another NetRAX inference has been running for several weeks (at the time of submission) on an empirical wheat-genomes dataset (Glémin *et al.*, 2019) with 47 individuals from 17 species, and 1387, 815 MSA patterns subdivided into 8738 blocks. The analysis has recovered between 5 and 6 reticulations thus far (see Supplementary Material for details). This analysis showcases that NetRAX *can* analyze such large empirical datasets, but also that additional work is required to further improve its runtime.

The main limiting factor is the number of reticulations, as there exist up to 2^r displayed trees for r reticulations.

5 Conclusion and future work

We have presented NetRAX, which, to the best of our knowledge, is the only efficient and scalable ILS-unaware ML tool for phylogenetic network inference. We also show that NetRAX recovers accurate networks.

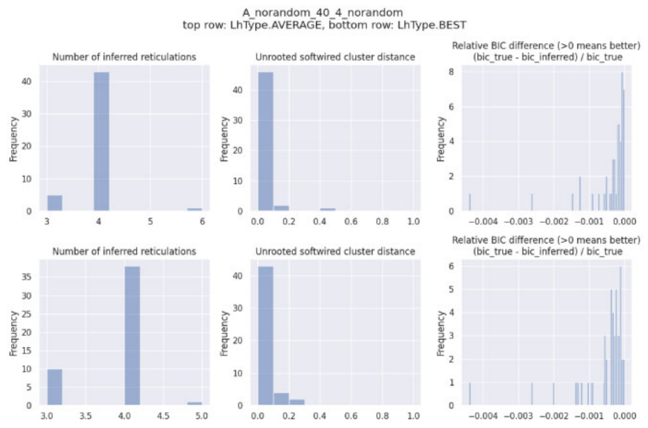


Fig. 6. Number of inferred reticulations, unrooted SCD and relative BIC difference for 50 simulated datasets with 40 taxa and 4 reticulations each, starting from the RAxML-NG ML tree

Table 3. Results for 30 taxa, 3 reticulations, with scrambled blocks, starting from the RAxML-NG ML tree

scrambling factor	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Inferred BIC	−	−	−	+	+	+	+	+	+	+	+
Inferred n_reticulations	3	3	5	3	3	3	3	1	0	0	1
Unrooted SCD	0	0	0.09	0.17	0.21	0.21	0.21	0.35	0.38	0.38	0.36
Runtime RAxML (seconds)	120	125	123	123	127	124	125	133	125	128	126
Runtime NetRAX (seconds)	380	314	2274	313	230	314	448	72	10	8	55
Inferred BIC	−	−	−	+	+	+	+	+	+	+	+
Inferred n_reticulations	3	3	5	3	3	3	2	1	0	0	1
Unrooted SCD	0	0	0.09	0.17	0.21	0.23	0.21	0.36	0.38	0.38	0.37
Runtime RAxML (seconds)	120	125	123	123	127	124	125	133	125	128	126
Runtime NetRAX (seconds)	277	219	2062	216	165	156	134	48	6	5	21

Note: Top: `LhModel.AVERAGE`, Bottom: `LhModel.BEST`. We use + for better-or-equal BIC, and − for worse BIC.

Table 4. Results for 30 taxa, 3 reticulations, different MSA size, starting from the RAxML-NG ML tree

msa patterns	397	794	3882	7634	36 264	69 919	316 379	597 921
Inferred BIC	+	−	−	−	−	−	−	−
Inferred n_reticulations	2	1	3	3	3	3	3	3
Unrooted SCD	0.08	0.19	0.03	0	0	0	0.03	0.03
Runtime RAxML (seconds)	5	9	25	34	139	340	2081	4340
Runtime NetRAX (seconds)	101	37	304	379	643	880	4544	10 452
Inferred BIC	−	−	−	−	−	−	−	−
Inferred n_reticulations	1	1	3	3	3	3	3	3
Unrooted SCD	0.21	0.21	0.03	0	0	0	0.03	0.03
Runtime RAxML (seconds)	5	9	25	34	139	340	2081	4340
Runtime NetRAX (seconds)	14	10	86	151	444	1165	5083	11 650

Note: Top: `LhModel.AVERAGE`, Bottom: `LhModel.BEST`. We use + for better-or-equal BIC, and − for worse BIC. A MSA pattern is a unique column in the MSA that may occur multiple times in the MSA.

More specifically, we demonstrated that NetRAX can infer ML networks with up to 40 taxa and up to 4 reticulations on datasets with thousands of MSA sites in less than a day. Our experimental results on simulated datasets also show that NetRAX infers high-quality ML networks with very low unrooted SCDs and very low relative BIC differences compared to the true, simulated network. Further, the MPI-based parallelization of NetRAX exhibits 'good' parallel efficiency (see Supplementary Material). We also show that NetRAX yields biologically plausible results on a well-studied empirical dataset of snakes and that it can analyze huge empirical datasets, though with currently still prohibitive runtimes.

Starting the network inference from multiple starting trees tends to yield more accurate results. For large datasets, we nonetheless recommend using NetRAX with a single ML starting tree to keep inference times within acceptable limits. Our experiments show that NetRAX can infer highly accurate ML networks, even on a single ML starting tree.

Future work will mainly focus on implementing bootstrapping, improving the scalability of NetRAX and refining its model. NetRAX already supports starting the search from a network. We thus intend to evaluate the performance NetRAX starting from a network, once computationally efficient methods for obtaining a reasonable starting network become available.

Table 5. Runtime (in seconds) and unrooted SCD to the true network for inferences using NetRAX, PhyLiNC, PhyloDAG, SNaQ, PhyloNET MPL and PhyloNet ML

Tool	Inference runtime	avg runtime per run	Inferred reticulations	Unrooted SCD
NetRAX single `LhModel.AVERAGE`	3	3	1	0
NetRAX multi `LhModel.AVERAGE`	40	4	1	0.1
NetRAX single `LhModel.BEST`	2	2	1	0
NetRAX multi `LhModel.BEST`	28	3	1	0.1
PhyLiNC max_reticulations 1	45 919	4592	1	0.36
PhyLiNC max_reticulations 2[a]	38 365	3837	2	0.56
PhyloDAG[b]	145	145	1	0.58
SNaQ max_reticulations 1	4899	490	1	0
SNaQ max_reticulations 2[c]	7489	749	1	0
PhyloNET MPL max_reticulations 1	158	16	1	0.3
PhyloNET MPL max_reticulations 2	223	22	2	0.1
PhyloNET MP max_reticulations 1	8	2	1	0
PhyloNET MP max_reticulations 2	8	2	2	0.25
PhyloNET ML max_reticulations 1	387	78	1	0.27
PhyloNET ML max_reticulations 2	19 799	3960	2	0.18

Note: The term `single` refers to starting NetRAX from the best RAxML-NG ML tree. The term `multi` refers to starting NetRAX from the 11 unique tree topologies contained in a set of 10 random and 10 RAxML-NG maximum parsimony trees. The true network has 10 taxa and 1 reticulation. Under all configurations, the network inferred by NetRAX had a better BIC than the true network. All methods inferred a network with a displayed tree distance of 1. [a]The tool printed error messages, but nonetheless returned a network. [b]PhyloDAG only returned the network as a picture. We had to manually write the Extended Newick for it. [c]The inferred network has 2 reticulations, but one reticulation has 0/1 probability. We had to manually prune the network to remove this trivial reticulation.

Acknowledgements

The authors wish to thank Adam Leache (U. Washington) for his feedback on the discussion of the empirical snake dataset results and Cecile Ane (U. Wisconsin-Madison) for feedback on the manuscript.

Funding

Part of this work was funded by the Klaus Tschira Foundation. B.M. is funded by DFG grant STA 860/6-2. C.S. was supported by the French Agence Nationale de la Recherche program (CoCoAlSeq project, ANR-19-CE45-0012).

Conflict of Interest: none declared.

References

Allen-Savietta,C. (2020) *Estimating Phylogenetic Networks from Concatenated Sequence Alignments*. The University of Wisconsin-Madison, Madison, Wisconsin, USA.

Ané,C. (2021) Phylonetworks Users Google Group Discussion. https://groups.google.com/g/phylonetworks-users/c/KCu45cDRy_Q/m/RLpaZJajBAAJ (14 August 2021, date last accessed).

Blair,C. and Ané,C. (2019) Phylogenetic trees and networks can serve as powerful and complementary approaches for analysis of genomic data. *Syst. Biol.*, **69**, 593–601.

Burbrink,F.T. and Gehara,M. (2018) The biogeography of deep time phylogenetic reticulation. *Syst. Biol.*, **67**, 743–755.

Cao,Z. *et al.* (2019) Practical aspects of phylogenetic network analysis using phylonet. *BioRxiv*, page 746362.

Chen,X. *et al.* (2017) Using phylogenomics to understand the link between biogeographic origins and regional diversification in ratsnakes. *Mol. Phylogenet. Evol.*, **111**, 206–218.

Darriba,D. (2016) pll-modules. https://github.com/ddarriba/pll-modules (28 July 2021, date last accessed).

Felsenstein,J. (1981) Evolutionary trees from DNA sequences: a maximum likelihood approach. *J. Mol. Evol.*, **17**, 368–376.

Flouri,T. (2015a) Computing the likelihood of a tree. https://github.com/xflouris/libpll/wiki/Computing-the-likelihood-of-a-tree (28 July 2021, date last accessed).

Flouri,T. (2015b) libpll-2. https://github.com/xflouris/libpll-2.git (28 July 2021, date last accessed).

Gambette,P. *et al.* (2017) Rearrangement moves on rooted phylogenetic networks. *PLoS Comput. Biol.*, **13**, e1005611.

Glémin,S. *et al.* (2019) Pervasive hybridizations in the history of wheat relatives. *Sci. Adv.*, **5**, eaav9188.

Hejase,H.A. and Liu,K.J. (2016) A scalability study of phylogenetic network inference methods using empirical datasets and simulations involving a single reticulation. *BMC Bioinformatics*, **17**, 1–12.

Holoborodko,P. (2010) Mpfr c++. http://www.holoborodko.com/pavel/mpfr/ (28 July 2021, date last accessed).

Huson,D.H. *et al.* (2010) *Phylogenetic Networks: Concepts, Algorithms and Applications*. Cambridge University Press, Cambridge, UK.

Jin,G. *et al.* (2006) Maximum likelihood of phylogenetic networks. *Bioinformatics*, **22**, 2604–2611.

Kozlov,A.M. *et al.* (2019) RAxML-NG: a fast, scalable and user-friendly tool for maximum likelihood phylogenetic inference. *Bioinformatics*, **35**, 4453–4455.

Nakhleh,L. *et al.* (2005) Reconstructing phylogenetic networks using maximum parsimony. In: *2005 IEEE Computational Systems Bioinformatics Conference (CSB'05), Stanford, CA, USA*, pp. 93–102.

NEPAL (2006) http://old-bioinfo.cs.rice.edu/nepal/ (28 July 2021, date last accessed).

Nguyen,Q. and Roos,T. (2015) Likelihood-based inference of phylogenetic networks from sequence data by phylodag. In: *International Conference on Algorithms for Computational Biology*, *Mexico City, Mexico*, Springer, pp. 126–140.

Pardi,F. and Scornavacca,C. (2015) Reconstructible phylogenetic networks: do not distinguish the indistinguishable. *PLoS Comput. Biol.*, **11**, e1004135.

Park,H.J. and Nakhleh,L. (2012) Inference of reticulate evolutionary histories by maximum likelihood: the performance of information criteria. *BMC Bioinformatics*, **13**, 1–10.

Rambaut,A. and Grass,N.C. (1997) Seq-Gen: an application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees. *Comput. Appl. Biosci.*, **13**, 235–238.

Robinson,D.F. and Foulds,L.R. (1981) Comparison of phylogenetic trees. *Math. Biosci.*, **53**, 131–147.

Solís-Lemus,C. and Ané,C. (2016) Inferring phylogenetic networks with maximum pseudolikelihood under incomplete lineage sorting. *PLoS Genet.*, **12**, e1005896.

Solís-Lemus,C. *et al.* (2017) Phylonetworks: a package for phylogenetic networks. *Mol. Biol. Evol.*, **34**, 3292–3298.

Tavaré,S. *et al.* (1986) *Some Probabilistic and Statistical Problems in the Analysis of DNA Sequences*. Lectures on Mathematics in the Life Sciences, Vol. 17, Providence, R.I. American Mathematical Society, pp. 57–86.

Wen,D. *et al.* (2018) Inferring phylogenetic networks using phylonet. *Syst. Biol.*, **67**, 735–740.

Zhang,C. *et al.* (2018) Bayesian inference of species networks from multilocus sequence data. *Mol. Biol. Evol.*, **35**, 504–517.