Bioinformatics
doi.10.1093/bioinformatics/xxxxxx
Advance Access Publication Date: Day Month Year
Original Paper

Phylogenetics

The NetRAX tool for Maximum-Likelihood Phylogenetic Network Inference

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

[1] Computational Molecular Evolution Group, Heidelberg Institute for Theoretical Studies, Heidelberg 69118, Germany and
[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: XXXXXXX

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Abstract

Motivation: Phylogenetic networks are required to model non-treelike evolutionary events. Current, actively developed approaches for phylogenetic network inference also account for incomplete lineage sorting (ILS) in their models but can only analyze small datasets due to their high computational complexity. There is no actively developed Maximum-Likelihood based phylogenetic network inference tool that does not model ILS.

Results: We present NetRAX, a tool for maximum-likelihood inference of phylogenetic networks via displayed trees that does not account for incomplete lineage sorting. Our tool leverages state-of-the-art methods for efficiently computing the phylogenetic likelihood function on trees, and extends them to phylogenetic networks. NetRAX can infer a maximum-likelihood phylogenetic network from a partitioned multiple sequence alignment and returns the inferred network in the Extended Newick format.

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

We can not always describe evolution via a phylogenetic tree. For example, hybridization events among plants and bacteria can lead to the acquisition of genes from bacteria or plants located in a distinct lineage. In such cases, a phylogenetic network is better suited to explain the evolutionary relationships.

Initially, methods for inferring maximum-likelihood (ML) based phylogenetic networks did not account for incomplete lineage sorting (ILS). For instance, the NEPAL (Jin *et al.*, 2006) tool does not account for ILS in its network likelihood model. It starts from a phylogenetic tree and adds reticulations, while keeping the underlying backbone-tree topology fixed. Unfortunately, the authors of NEPAL have unfortunately lost its source code (nep, 2021) and the tool segfaults when trying to execute it for unknown reasons. Cao *et al.* showed that inferring networks by fixing the underlying backbone-tree topology and only adding reticulations does not perform well (Cao *et al.*, 2019).

In the past years, focus shifted towards developing methods for ML network inference that also account for ILS. These methods infer a network from a multiple sequence alignment (MSA) and a given set of gene trees. While models accounting for ILS are expected to yield more accurate networks as they incorporate more mechanisms that explain non-treelike evolution, they face substantial computational challenges.

For example, the model originally implemented in PhyloNET (Than *et al.*, 2008) can only analyze very small datasets with 10 taxa and up to 4 reticulations (Solís-Lemus and Ané, 2016). Consequently, faster-to-compute semi-likelihood models were developed, that still account for ILS (Solís-Lemus and Ané, 2016). These semi-likelihood models are based on combining the ILS-aware likelihoods of 4-taxon subnetworks. More recent versions of PhyloNET also use semi-likelihoods (Wen *et al.*, 2018), but still face scalability challenges.

Not true anymore. Cite the new tool

I think that the writing of the introduction can be polished more. A lot of people read only abstract, intro and discussion

picture(0,0)(-35,0)(1,0)30 (0,35)(0,-1)30 picture

picture(0,0)(35,0)(-1,0)30 (0,35)(0,-1)30 picture

The PhyloDAG (Nguyen and Roos, 2015) tool implements an efficient expectation-maximization algorithm to infer maximum-likelihood networks using the mixture model of Strimmer and Moulton (Strimmer and Moulton, 2000). Sarah: I really don't understand this model used by PhyloDAG at all. Does it account for ILS or not? Unfortunately, it only returns networks as a picture and does not use the Extended Newick Format (Cardona *et al.*, 2008), rendering its results difficult to use and compare.

Apart from ML network inference tools, there also exist tools that rely on maximum parsimony (Nakhleh *et al.*, 2005) or bayesian inference (Zhang *et al.*, 2018).

2 NetRAX Outline

With NetRAX, our goal is to infer a ML phylogenetic network N (see Section **??**) from a partitioned MSA $\mathcal{A}$ and to substantially increase the scalability of network inference tools. This is, we want to find a phylogenetic network N that maximizes $L(N|\mathcal{A})$. For this, we use the set of displayed trees $\mathcal{T}(N)$ induced by N (see Section **??**). We compute the network likelihood $L(N|\mathcal{A})$ as a function of the displayed tree partition likelihoods $L(T|A)_{T\in\mathcal{T}(\mathcal{N}),A\in\mathcal{A}}$ and the probabilities $P(T|N)_{T\in\mathcal{T}(N)}$ that each induced tree is displayed by the network.

In literature, there exists a plethora of different phylogenetic network definitions. Here, we assume that ILS is not present or negligible. Thus, we focus on phylogenetic networks as characterized by a set of displayed trees.

A binary phylogenetic network N is a single-source, directed, acyclic graph. We call its source node the *root* node of N. Apart from the root, there are three types of nodes in a binary phylogenetic network: (i) Internal tree nodes with 1 incoming edge and 2 outgoing edges, (ii) reticulation nodes with 2 incoming edges and 1 outgoing edge, and (iii) leaf nodes with one incoming edge and no outgoing edges. Further, a phylogenetic network on a set of n taxa has exactly n leaves.

Each edge in a phylogenetic network has a branch length and a probability. The incoming edges of a reticulation node (called *hybridization edges*) are assigned inheritance probabilities which must sum to 1.0. The probability of observing a non-hybridization edge e is $P(e) = 1.0$.

Figure 2 shows a simple example network.

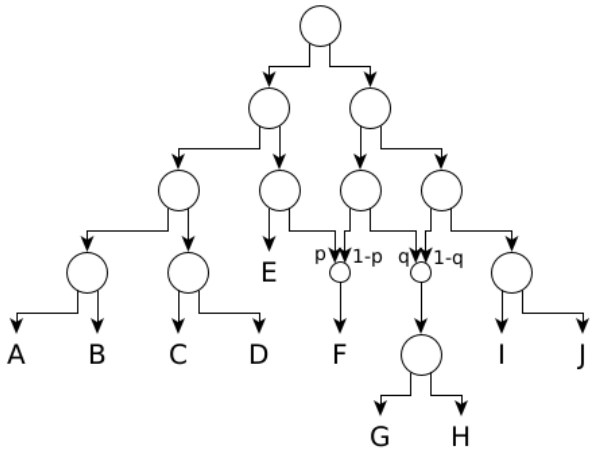

Fig. 1. A phylogenetic network with two reticulation nodes.

Celine hates my style of drawing a phylogenetic network and offers to draw nicer pictures.

In our model, we further assume that neither ILS nor recombination has occurs among the sites of the MSA. We make this assumption because it reduces the computational complexity, and ILS is not an issue if species did not diverge only recently (Maddison and Knowles, 2006). Celine: We can improve this later, let's wait for the Related Work section

Given these assumptions, we can describe reticulate evolution in a network via its set of induced *displayed trees*. We obtain a displayed tree from a network by choosing one parent per reticulation node (disabling the incoming edges belonging to non-chosen parents). Figure 2 shows an induced tree in a phylogenetic network.

We weigh each displayed tree T by the probabilities of the reticulation edges taken in order to obtain the tree. We say that the selected reticulation edges *display* the tree, and refer to them by $E_r(T)$.

The probability $P(T|N)$ of displaying a tree T in a phylogenetic network N is the product over the probabilities of the hybridization edges that display T (see Figure 2).

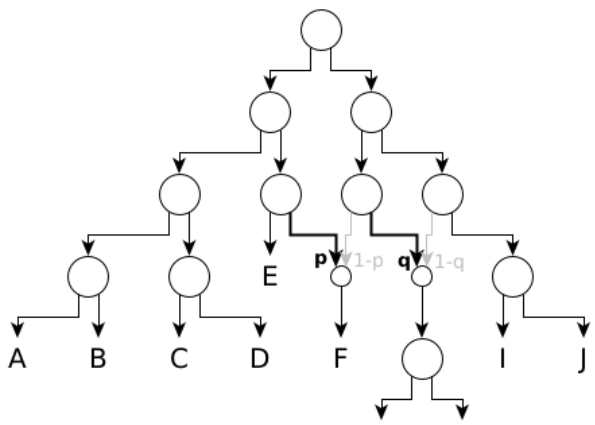

Fig. 2. A displayed tree in a phylogenetic network. The probability of the highlighted displayed tree is the product $p * q$ over the respective reticulation probabilities.

3 Phylogenetic Network Likelihood Model

Nowadays, typical phylogenetic analyses are based on a MSA $\mathcal{A}$ with multiple partitions $A_1, \ldots, A_p$. A partition 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.

3.1 Computing the Likelihood on a Tree

Site-based Phylogenetic Tree Loglikelihood Let $T = (V, E)$ be a phylogenetic tree. Let $\mathcal{A}$ be a partitioned MSA with partitions $A_1, \ldots, A_p$. Let $\vartheta = (\vartheta_1, \ldots, \vartheta_p)$ be the parameter vector, storing per-partition branch lengths and other likelihood model parameters (e.g., substiution rates, stationary frequencies, etc.). The likelihood of T given $\mathcal{A}$ is:

$$L(T|\mathcal{A}) = \prod_{i=1}^{p} L(T|A_i, \vartheta_i) = \prod_{i=1}^{p} \prod_{s\in A_i} L(T|s, \vartheta_i) \qquad (1)$$

To avoid numerical underflow, one tyupically computes the log likelihood instead:

picture(0,0)(-35,0)(1,0)30 (0,-35)(0,1)30 picture

picture(0,0)(35,0)(-1,0)30 (0,-35)(0,1)30 picture

picture(0,0)(-35,0)(1,0)30 (0,35)(0,-1)30 picture

picture(0,0)(35,0)(-1,0)30 (0,35)(0,-1)30 picture

$$\log L(T|\mathcal{A}) = \sum_{i=1}^{p} \log L(T|A_i, \vartheta_i) = \sum_{i=1}^{p} \sum_{s \in A_i} \log L(T|s, \vartheta_i). \quad (2)$$

3.2 Computing the Likelihood on a Network

Probability of a Displayed Tree Let $N = (V, E)$ be a phylogenetic network and T be a displayed tree of N. Let E_r be the set of reticulation edges that need to be taken in order to obtain/generate T. Let $P(e)$ be the probability of an edge e. The probability of T in N is then:

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

Let $N = (V, E)$ be a phylogenetic network with a set of displayed trees $\mathcal{T}(N)$. Let $\mathcal{A}$ be a partitioned MSA with partitions $A_1, \ldots, A_p$. Let $\vartheta = (\vartheta_1, \ldots, \vartheta_p)$ be the parameter vector, storing per-partition network branch lengths and likelihood model parameters.

We consider each partition as being independent. Thus, we define the likelihood of a phylogenetic network as the product over the per-partition likelihoods:

$$L(N|\mathcal{A}, \vartheta) = \prod_{i=1}^{p} L(N|A_i, \vartheta_i).$$

To avoid numerical underflow, we take the logarithm:

$$\log L(N|\mathcal{A}, \vartheta) = \sum_{i=1}^{p} \log L(N|A_i, \vartheta_i).$$

We assess and implement two versions for computing the log likelihood on partitioned networks. They both aggregate over the log likelihood of the trees displayed by the network:

1. **Weighted Average Version**:

$$\log L(N|A_i, \vartheta_i) = \log \left(\sum_{T \in \mathcal{T}(N)} L(T|A_i, \vartheta_i) * P(T|N) \right). \quad (3)$$

2. **Best Tree Version**:

$$\log L(N|A_i, \vartheta_i) = \log \left(\max_{T \in \mathcal{T}(N)} L(T|A_i, \vartheta_i) * P(T|N) \right). \quad (4)$$

In the weighted average version, the likelihood of a network for a given partition is the weighted average over the displayed tree likelihoods. We use the sum here, because the probability of event A or B to occur is 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. To avoid numerical problems, we use arbitrary-precision arithmetics (using MPFR C++(Holoborodko, 2021)) to compute $L(N|\mathcal{A}_i)$ from the displayed tree per-partition likelihoods.

We use libpll (lib, 2021b) and pll-modules (pll, 2021) to compute displayed tree likelihoods via the standard Felsenstein pruning algorithm (Felsenstein, 1981).

3.3 Branch Length Model

Currently, NetRAX supports two branch length models: Under the linked branches model, we share the same set of branch lengths among *all* partitions. Under the unlinked branches model, each partition has its own independent set of branch lengths.

In future NetRAX versions, we will also support the scaled branches model. In the scaled branches model, we share the same set of branch lengths among all partitions, but each partition scales the branches via its own scaling factor.

The branch length model choice has an effect on which type of reticulations we can recover. Figure 3.3 shows an example network with a reticulation that is impossible to infer under the unlinked branches model. Thus by default, we use linked branch lengths in NetRAX.

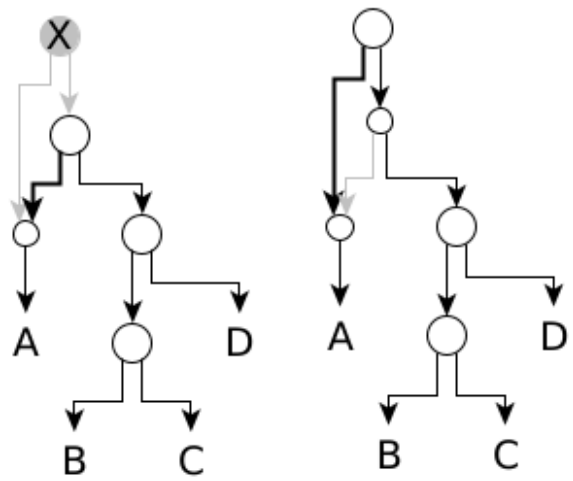

Fig. 3. Two displayed trees in a phylogenetic network. Both displayed trees induce the same topology after collapsing single-child nodes. They only differ in some branch lengths.

4 Computing the Likelihood of a Phylogenetic Network

In order to compute the log likelihood (LnL) of a network N using the formulas from Section 3.2, we first need to compute the per-partition likelihoods $L(T|A_i, \vartheta_i)_{i \in \{1,\ldots,p\}}$ and the probabilities $P(T|N)$ of all trees $T \in \mathcal{T}(N)$ displayed by N. To avoid numerical underflow, we use the MPFR C++ wrapper (Holoborodko, 2021) to compute the network loglikelihood given the displayed tree per-partition loglikelihoods and displayed tree probabilities.

We use the libpll library to compute $\log L(T|\mathcal{A}_i, \vartheta_i)$. In order to compute the per-partition LnLs of a phylogenetic tree, libpll uses an internal per-node data structure, called the *conditional likelihood vector (CLV)* (lib, 2021a). A CLV for a node v stores the per-site likelihoods for the subtree rooted at v (see Figure 4). Libpll computes the per-node CLVs via a post-order traversal of the tree, via the Felsenstein pruning algorithm (Felsenstein, 1981). It computes the CLV of a given node based on the CLVs of its respective children. Libpll also provides incremental likelihood computations: it only updates those CLVs affected by a topological rearrangement move or branch length change and re-uses unaffected CLVs that are still valid.

picture(0,0)(-35,0)(1,0)30 (0,-35)(0,1)30 picture

picture(0,0)(35,0)(-1,0)30 (0,-35)(0,1)30 picture

picture(0,0)(-35,0)(1,0)30 (0,35)(0,-1)30 picture

picture(0,0)(35,0)(-1,0)30 (0,35)(0,-1)30 picture

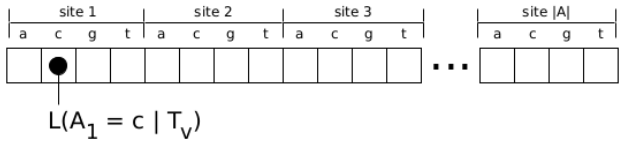

Fig. 4. A conditional likelihood vector (CLV) for a subtree T_v rooted at a node v in a phylogenetic tree T and a MSA partition A.

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 save some 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 Section 4.1).

Since libpll operates on a strictly bifurcating phylogenetic tree, it requires each inner node to have exactly two children. For each reticulation parent taken in order to display a tree in a network, we mark the reticulation edge coming from the alternative reticulation parent as inactive. This temporarily removes a child from the non-chosen reticulation parent.

We apply some adaptations to deploy libpll on networks: We need to handle single-child nodes (see Section **??**), and we need to skip dead (unused) nodes (see Section **??**).

4.1 Sharing CLVs among Displayed Trees

Figure 4.1 shows a network with $n := 19$ nodes and $r := 2$ reticulations. It displays $2^r = 4$ trees. Naïvely, by explicitly iterating over each displayed tree, one would require $n * 2^r = 76$ CLVs to calculate the likelihood of this network: one CLV per node and displayed tree.

To alleviate this, for each node v in a rooted phylogenetic network, we store as many CLVs as there are different displayed subtree topologies rooted at v. By sharing CLVs among subtrees that are shared by multiple displayed trees, we reduce the total number of CLVs to compute the LnL of this network to 32.

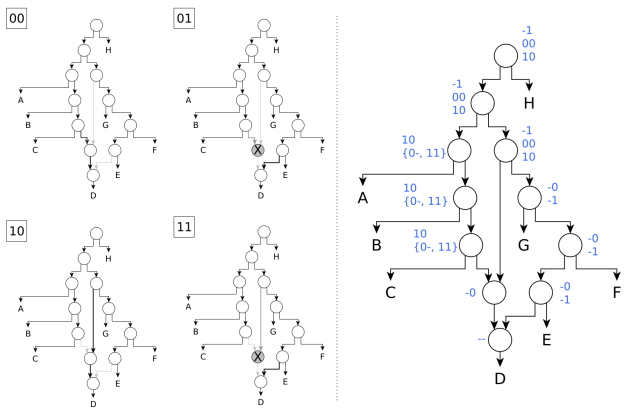

Fig. 5. A network and its displayed trees. We encode each displayed tree by a binary r-bit number. We set the i-th bit to 0 if we take the first reticulation parent for the i-th reticulation, and to 1 otherwise. On the right side, we see for each node which reticulation choices require their own, private, non-shared CLV. To reduce visual clutter, we use the character "-" (don't care) to denote multiple reticulation choices. For example, we use -1 to represent both displayed trees 01 and 11. Each line next to a node in the right image represents one required CLV. For example, the lines 10 and {0-, 11} mean that we need to store one CLV for tree 01 and one CLV for trees 00,01,11.

In order to use our share-CLVs optimization, we need to interleave the CLV update operations performed by libpll among the displayed trees. We do this via a bottom up traversal (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 present at v.

Due to the peculiarities of networks additional, predominantly technical modifications to the network traversal and displayed tree data structure are required that are described in detail in the supplement.

5 Branch-Length Optimization

We intend to optimize a branch length b in a network N, with respect to the LNL of the network. Overall, we aim to find the branch length assignments $\hat{b}_1, \ldots, \hat{b}_p$ that maximize

$\log L(N|A, \vartheta) = \sum_{i=1}^{p} \log L(N|\mathcal{A}_i, \vartheta_i)$.

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 $(\log L(N|A, \vartheta))'$ and $(\log L(N|A, \vartheta))''$ out of $\log L(T|A_i, \vartheta_i)$, $(\log L(T|A_i, \vartheta_i))'$, and $(\log L(T|A_i, \vartheta_i))''$ in the supplementary text. In the following, we describe the efficient computation of the per-partition LnL and the per-partition LnL derivatives for all displayed trees $T \in \mathcal{T}(N)$.

When optimizing a branch length, we do not need to recompute the per-partition LnL and its derivatives for trees where b is inactive. A branch is *active* in a displayed tree, if neither its source nor its target is a dead node, and it is not connecting an inactive parent to a reticulation (see Figure 5).

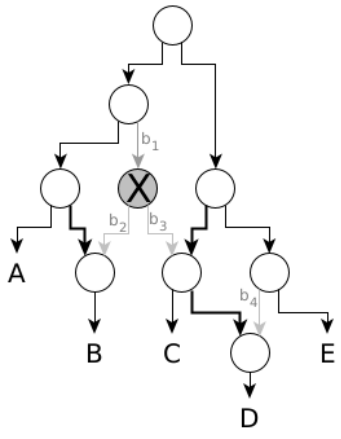

Fig. 6. A displayed tree in a phylogenetic network. The branches b_1, b_2, b_3, and b_4 are inactive in this displayed tree. When optimizing the length of any of the inactive branches, we do not need to recompute the likelihood of this displayed tree.

5.1 Branch-Length Optimization in Phylogenetic Trees

In order to avoid costly CLV updates when optimizing a branch (u, v) in a tree, all modern ML tree inference tools re-root the tree at the node u before optimizing the branch (see Figure 5.1). After re-rooting the tree, the edge directions in subtrees rooted at nodes that lie on the path between the old and the new root (both ends included) 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 re-rooted tree, we can reuse the CLVs stored at u and v without having to recompute them. For this, libpll provides functions that takes the

picture(0,0)(-35,0)(1,0)30 (0,-35)(0,1)30 picture

picture(0,0)(35,0)(-1,0)30 (0,-35)(0,1)30 picture

picture(0,0)(-35,0)(1,0)30 (0,35)(0,-1)30 picture

picture(0,0)(35,0)(-1,0)30 (0,35)(0,-1)30 picture

CLVs stored at u and v and the current length of (u, v), and return the per-partition loglikelihoods as well as the per-partition loglikelihood derivatives of the tree.

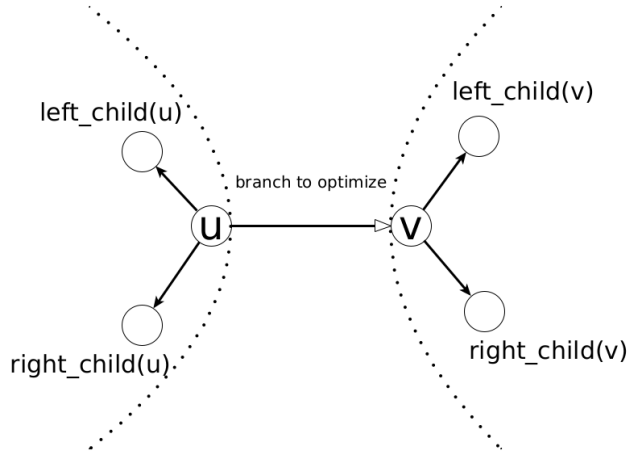

Fig. 7. When optimizing the length of a branch (u, v) in a tree, we re-root the tree at the node u, which is the source of the edge whose length we want to optimize. Some edge directions may change in this process. Note that, the phylogenetic likelihood is the same regardless of the root placement under time-reversible models as used by RAxML-NG. After the re-root operation, we need to once recompute the CLVs that lie on the path between the old root and the new root u. When changing the length of (u, v) during branch-length optimization, we can reuse the CLVs stored at u and v.

I think "reroot" is fine too.

5.2 Re-rooting Displayed Trees in a Phylogenetic Network

In NetRAX, we leverage an analogous computational saving strategy during branch length optimization as described in Section 5.1. For this, before optimizing a branch (u, v), we need to re-root *all* displayed trees (where the branch is active) 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 a custom bottom-up traversal of the nodes in the phylogenetic network, updating all unique subtree CLVs shared among subsets of displayed trees (see Section 4.1). This complicates the re-rooting 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 re-rooting the displayed trees, the parent-child relationships depend on the specific displayed tree we are currently considering (see Figure 5.2). These differing edge directions affect the order in which we need to process the nodes when recomputing the CLVs.

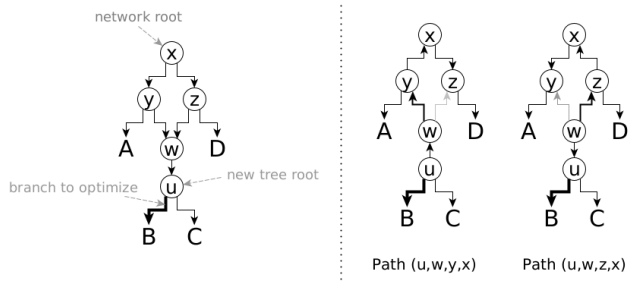

Fig. 8. Virtually re-rooting the displayed trees at node u before optimizing branch (u, v). In the re-rooted first displayed tree, the node y is a parent of node x and we need to recompute the CLVs on path (u,w,y,x). However, in the second re-rooted displayed tree, the node y is a child of node x and we need to recompute the CLVs on path (u,w,z,x).

For networks, we need to recompute the CLVs (which are shared among subsets of displayed trees) that lie on *any* path between the network root and the new tree root (both ends included). We devised the following approach for resolving the edge direction problem:

We process the paths between the network root and the new tree root successively. Before processing the next path, we invalidate the shared CLVs at all nodes on the path, except for the new tree root node. We detect, in advance, at which nodes we need to restore the old shared CLVs (from how they were when using the network root), save, and restore them accordingly.

After we finished optimizing a branch, we recompute the CLVs with regard to the original network root. This means, when successively optimizing k branches, we re-root the displayed trees in the network as follows: tree_root_1 → network_root → tree_root_2 → network_root → …tree_root_k → network_root.

6 Optimizing Non-Topology Parameters

Apart from optimizing branch lengths, we also need to optimize other standard likelihood model parameters (e.g., nucleotide substitution probabilities) 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

$$\log L(N|A, \vartheta) = \sum_{i=1}^{p} \log L(N|\mathcal{A}_i, \vartheta).$$

6.1 Likelihood Model Parameter Optimization

We reuse routines from RAxML-NG for these likelihood parameter optimizations. As these routines do not rely on an explicit tree topology, we can directly reuse them for networks.

6.2 Optimizing Reticulation Probabilities

For optimizing the first-parent probability p of a reticulation, we also reuse the Brent single-parameter optimization method of RAxML-NG. Since the first and second parent probability of a reticulation sum up to 1.0, the probability for taking the second parent follows from the first parent probability.

The Brent optimization method requires us to recompute the network LnL when p changes. Fortunately, the displayed tree per-partition LnLs do not depend on p. Changing p only affects the probabilities $P(T|N, \vartheta)_{T \in \mathcal{T}(N)}$ of displaying the trees. We can thus re-use the old

picture(0,0)(-35,0)(1,0)30 (0,-35)(0,1)30 picture

picture(0,0)(35,0)(-1,0)30 (0,-35)(0,1)30 picture

picture(0,0)(-35,0)(1,0)30 (0,35)(0,-1)30 picture

picture(0,0)(35,0)(-1,0)30 (0,35)(0,-1)30 picture

per-partition displayed tree LnL when recomputing the network LnL (see Section 3.2) during the optimization of p.

7 Supported Topology-Rearrangement Moves

NetRAX implements the following rooted network topology rearragement moves proposed by Gambette *et al.* (Gambette *et al.*, 2017), as well as reversal (undo) operations for them: rNNI move (see Figure 9), rSPR move (see Figure 10), arc insertion move (see Figure 11), and arc removal move (see Figure 11). When undoing a move, we restore the original topology and branch lengths. Doing or undoing a move also invalidates some CLVs which need to be recomputed.

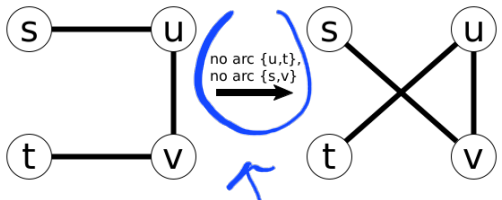

Fig. 9. Generalized visualization of a rNNI move. In a rNNI move, the nodes u and v exchange their neighbors. The resulting network needs to be acyclic, the edge directions between u, v, s, t may change after the move.

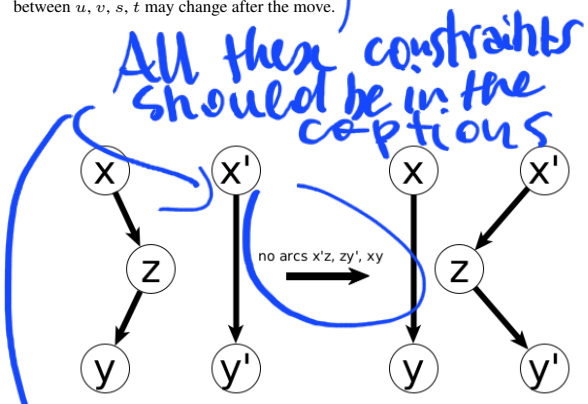

Fig. 10. Generalized visualization of a rSPR move. The resulting network needs to be acyclic. The arcs xy, yz are the donor arcs and the arc $x'y'$ is the recipient arc.

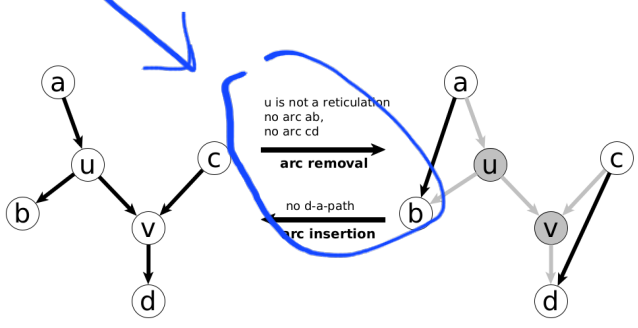

Fig. 11. Arc removal and arc insertion move. An arc removal move removes an arc uv from a reticulation v. When performing an arc removal move, we remove 1 bifurcation, remove 1 reticulation, remove 5 arcs, and add 2 new arcs. An arc insertion move chooses two distinct arcs ab and cd, with cd not being ancestral to ab. When performing an arc insertion move, we add 1 bifurcation, add 1 reticulation, remove 2 arcs, and add 5 new arcs. The resulting network needs to be acyclic.

7.1 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 LnL. For this, NetRAX implements AIC, AICc, and BIC scoring. By default, NetRAX uses the BIC score to compare different networks. Park and Nakleh (Park and Nakhleh, 2012) showed that using BIC performs best in network searches.

The BIC score of a network N with r reticulations on a partitioned MSA $\mathcal{A}$ and parameter vector ϑ is:

$$\text{BIC}(N|\mathcal{A},\vartheta) = -2{*}\log L(N|\mathcal{A},\vartheta){+}\#\text{ free\_parameters}{*}\log(\text{sample\_size}).$$

The free parameters are the substitution model parameters, reticulation first-parent probabilities, and branch lengths. The sample size is the product of the number of taxa and the number of MSA sites.

8 Network Search

NetRAX uses a greedy hillclimbing approach for improving the network topology. NetRAX has an outer search loop iterating over different move types and an inner search loop searching for the best-scoring network using a single move type. We provide pseudocode for them in the supplementary text.

In the outer search loop, we search in waves, repeatedly iterating over move types in the following order: arc removal move, rSPRMove, rNNIMove, arc insertion move. For each move type, we call the inner search loop (Section 8.2). The outer search loop ends when all move types did not yield network with a better BIC score.

TODO: Sarah: I think an overview picture would be really nice here. Some flowchart diagram.

8.1 Start Networks

NetRAX can start the network inference from either a given start network (provided in Extended Newick format), or a user-specified number of random and maximum-parsimony trees (generated with RAxML-NG (Kozlov *et al.*, 2019)).

So far, NetRAX cannot directly call a RAxML-NG (Kozlov *et al.*, 2019) tree search in order to initiate the network search from the best tree found by RAxML-NG. In order to obtain this behavior, the user has to do the RAxML-NG tree search first and then pass the maximum-likelihood tree inferred by RAxML-NG to NetRAX. We recommend using NetRAX with a maximum likelihood tree inferred by a preceding RAxML-NG run.

8.2 Inner Search Loop

In the inner search loop, we search for the best-scoring network using only move candidates of a single move type.

8.2.1 Assembling the set of Move Candidates

For a given move type, we have a set of candidate move operations (e.g., there are multiple resulting networks we can reach by doing a single move of a given type on the network – each of these possible moves is a candidate move). Searching and evaluating all move candidates varies in speed across move types, because for some move types there are more possible candidates than for others.

When gathering possible move candidates for a given move type, NetRAX by default uses a search radius of 5. This is, for each node in the network, we only consider rSPR moves and arc insertion moves within radius of 5 nodes around the current node. The search radius parameter does not affect our choice of rNNI or arc removal moves.

picture(0,0)(-35,0)(1,0)30 (0,-35)(0,1)30 picture

picture(0,0)(35,0)(-1,0)30 (0,-35)(0,1)30 picture

picture(0,0)(-35,0)(1,0)30 (0,35)(0,-1)30 picture

picture(0,0)(35,0)(-1,0)30 (0,35)(0,-1)30 picture

8.2.2 Filtering Move Candidates

In order to determine the most promising move candidate, we perform a 3-stage filtering process: We filter the candidates using the stages PREFILTER, RANK, and CHOOSE. These stages differ in the number of branches we optimize before scoring each candidate:

- PREFILTER – Do not optimize branch lengths. (Exception: For prefiltering an arc insertion, we need to optimize the length of the newly introduced branch, because we do not have a useful initial guess for it.)
- RANK – Optimize branches directly affected by the move.
- CHOOSE – Optimize all branches in the network.

We use the Elbow Method (see Figure 12) to decide on the number of candidates to keep after each filtering stage. The most promising move candidate is the candidate that has the lowest (=best) BIC score after the CHOOSE filtering stage.

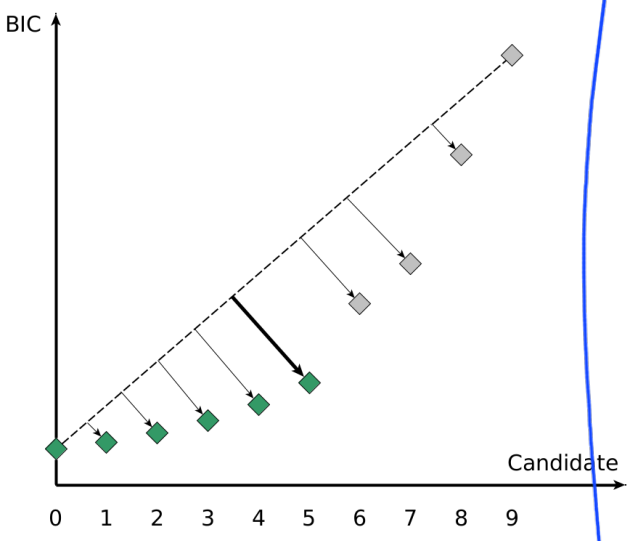

Fig. 12. In the elbow method (Thorndike, 1953), (Perera, 2021), we sort the list of pre-scored move candidates by increasing BIC score. We need to find the point with the largest distance to the line from the first to the last candidate; this point corresponds to our chosen cutoff value. We keep all candidates with BIC score smaller-or-equal than this cutoff score value. In this example, we keep the first six candidates, candidates 0 to 5.

8.2.3 Accepting a Move and updating the set of Move Candidates

If the most promising move candidate obtained by the CHOOSING stage leads to 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 model parameters in the new best network.

For each move type except for arc insertion move, we continue applying moves of the given move type until we do not find a better-scoring network anymore. In order to reduce time spent optimizing a network with too high reticulation count, we directly continue with the other move types after accepting an arc insertion move.

If the inner search loop was called with the arc insertion move type, the inner search loop ends after accepting a move and we return to the outer search loop. Otherwise, after we accept a move in the search, we want to reuse other promising candidate moves gathered in the PREFILTERING phase. Because of index remappings performed by our move implementation (libpll requires CLV indices to be consecutive), we need to also update index references stored in these old promising move candidates. We remove old promising moves that have become inapplicable after accepting our chosen move, and add new move candidates to the set, seeded at nodes directly affected by the accepted move. Only after we did not find a better scoring network by continuing the search from this reduced set of candidate moves, we again consider the complete set of move candidates as obtained in Section 8.2.1. If these also do not lead to a better-scoring network, the inner search loop ends.

9 Implementation

Key ideas:

- Modular software architecture
- MPI parallelization of loglikelihood computation and computation of loglikelihood derivatives over sites in the MSA

9.1 Modular Architecture

TODO: Picture showing how we split up NetRAX into separate modules

9.2 Parallelization

We parallelized both displayed-tree loglikelihoood and displayed tree loglikelihood derivatives computation in NetRAX over the MSA sites by using MPI. For this, we reuse the `ParallelContext` class and load balancing solutions provided by RAxML-NG.

In order to avoid redundant computations and making best use of per-tree MPI parallelization, we first iterate over nodes in a network. Then, we go over all displayed trees stored in a node. We precompute data (`prob_invar` and `diagpable` in libpll) that remains unchanged among computations for multiple displayed trees.

10 Simulation of Phylogenetic Networks and Sequences

TODO: Describe the simulator (I found some old text from Celine and pasted it here)

The simulator simulates an ultrametric networks and extracts displayed trees out of the simulated network, one for partition. Then simulates sequences on the output displayed trees.

Our simulator uses the linked branch lengths model (this is, all partitions share the same set of branch lengths).

picture(0,0)(-35,0)(1,0)30 (0,-35)(0,1)30 picture

picture(0,0)(35,0)(-1,0)30 (0,-35)(0,1)30 picture

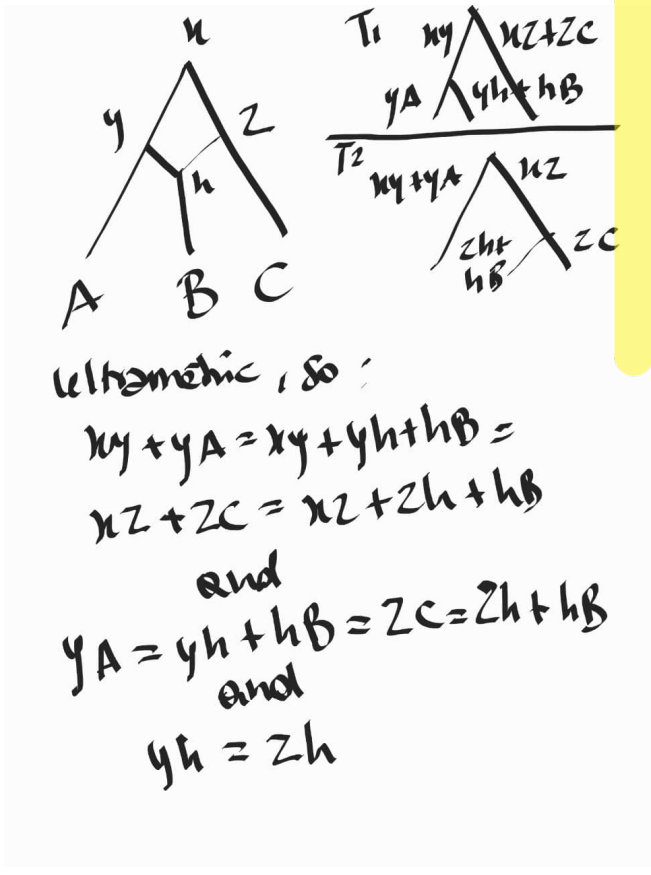

Fig. 13. Ultrametric trees from the simulator.

In more details

We are going to simulate our networks under the birth-hybridization process of Zhang *et al.* (Zhang *et al.*, 2018). In this process, we have a speciation rate l (lambda), a hybridisation rate v (nu) and the time we run the process t_0.

From their paper: "The simulator starts from the time of origin (t_0) with one species. A species splits into two (speciation) with rate l, and two species merge into one (hybridization) with rate v. At the moment of k branches, the total rate of change is r_tot = k l + choose (k 2) * v. We generate a waiting time $\sim$ exp(r_tot) and a random variable u $\sim$ U(0,1), If u < kl=r_tot, we randomly select a branch to split; otherwise, we randomly select two branches to join, and generate an inheritance probability c $\sim$ U(0,1). The simulator stops at time t_0."

No transfer from now, because NetRAX cannot recover branches with no length.

Given such generated network, we extract a random tree, using the inheritance probabilities at the reticulated nodes: for each reticulated node x, we draw a number y $\sim$ U(0,1) and we choose the first parent p_x if y < inheritanceProb of the edge p_x,x. If we do this,we have atree but not a binary one. Then we clean the tree from the dead nodes and 1-indegree 1-outdegree nodes, as Sarah explained in her document, and we obtain a binary tree. On this binary tree, we simulate N sequences with Seq-Gen-1.3.4 (currently under this model -mHKY -t3.0 -f0.3,0.2,0.2,0.3 we can change this). We do this K times, obtaining an alignment of length K*N in which each group of N sites has its tree.

In their paper, they use this parameters in the simulations

$\lambda - \nu \sim \exp(0.1)$ with mean 10 $\tau_0 \sim \exp(10)$ with mean 0.1 $\nu/\lambda \sim$ Beta distribution with alpha = 1 and beta = 1 (same as uniform distribution between 0 and 1) $\gamma \sim$ Uniform distribution (between 0 and 1)

As I said in a slack message, this process gives Yule-type networks, that act as the Yule-type trees, see example below for trees.

Once they start having a certain amount of nodes, things get crazy and they start speciating and hybridising a lot. It is not the focus of our paper to correctly simulate networks (we are not in a Bayesian framework, we do not need priors) and this is the best we can for now (for the next paper, we will do better, working on it). So we can simply throw the networks we do not like as done in line 143-147 of this simulator, which only simulate networks and not trees and sequences.

We use seq-gen for simulating sequences on the displayed trees with the following parameters: TODO

I am not sure of which networks you finally excluded, so I cannot comment on this

10.1 Excluding Weird/Unrecoverable Networks

Since our simulator simulates ultrametric networks,

TODO: Explain how we exclude weird networks. With number of pairs and number of equal pairs... This network should have a weirdness of 1.0, because all its displayed trees induce the same bipartitions.

There is still another problem though... now that I fixed the network weirdness computation, it turns out that we have a lot of "weird" simulated networks (meaning, displayed trees inducing the same bipartition sets). These networks have reticulations which we cannot infer with NetRAX (or any other tool) because the simulated networks are ultrametric, making the data indistinguishable from not having the problematic reticulations.

Simply excluding these networks sounds better to me though, since the BIC stats on the weird networks are obvious ("MSA 100 percent supports a tree? Well, let's infer a tree then.")

It still makes sense to have them in the simulator though, since extinctions etc in biology can lead to such "weird" networks. I just don't see a good reason for wasting more compute hours running NetRAX on them, if their induced MSA is indistinguishable from having a tree...

ignore "undetectable"/"weird" reticulations weird networks (e.g., displayed trees having the same bipartitions)

= I stopped here

11 Experimental Setup

TODO: Mention indistinguishability problems when evaluating topological distances (Pardi and Scornavacca, 2015).

We do not include a comparison with other tools here, as we are not aware of any working tool which uses the same non-ILS likelihood model as NetRAX and outputs the network in a text file for further analysis. We decided against comparing NetRAX with PhyloNet, because PhyloNet is only feasible for small datasets and explicitly models ILS in its likelihood function. The NEPAL tool did not work, our inference attempts resulted in a segmentation fault. PhyloDAG only outputs a picture of the inferred network.

11.1 Simulated Data

We simulated phylogenetic networks using our simulator from Section 10. Our simulator uses the linked branch lengths model (this is, all partitions share the same set of branch lengths). For each displayed tree in the network, we simulated a partition with 1000 MSA sites.

In all experiments on simulated data, we compared our inferred network with the true simulated network using BIC, AIC, AICc, and the network topology distances discussed in Section 11.3.1.

We ran experiments A,B on a single in-house cluster compute node with Intel CPUs (E5-2630v3 with 20 MB cache, running at 2.40GHz). The node contains 2 CPUs with 8 physical cores each. The node has 64 GB RAM.

We used the same cluster for experiment F, using up to 4 compute nodes in the first, and up to 8 compute nodes in the second experiment.

I modified this part.

picture(0,0)(-35,0)(1,0)30 (0,35)(0,-1)30 picture

picture(0,0)(35,0)(-1,0)30 (0,35)(0,-1)30 picture

We ran experiments C,D,E on a Lenovo Thinkpad T460p laptop. The laptop has 16 GB RAM and contains a single Intel CPU (i7-6700HQ with 6 MB cache, running at 2.60 GHz) with 4 physical cores.

For each simulated dataset, we inferred a Maximum-Likelihood tree with RAxML-NG. We then did the following experiments, using NetRAX with both LikelihoodModel.AVERAGE and LikelihoodModel.BEST, under the linked branch lengths model:

11.1.1 A: Standard

In our standard experiment setting, we assessed NetRAX inference quality with varying number of taxa and number of reticulations. We simulated

- 50 networks with 10 taxa and 1 reticulation,
- 50 networks with 20 taxa and 2 reticulations,
- 50 networks with 30 taxa and 3 reticulations,
- 50 networks with 40 taxa and 1 reticulation,
- 50 networks with 40 taxa and 2 reticulations, and
- 50 networks with 40 taxa and 3 reticulations.

We used probability 0.5 for each reticulation. For each dataset, we started the NetRAX inference from the RAxML-NG maximum-likelihood tree. In addition, for the datasets with less than 40 taxa, we also started another NetRAX inference using 3 random and 3 maximum parsimony starting trees.

11.1.2 B: Reticulation Probability

In our reticulation probability experiment setting, we simulated a 50 datasets with 20 taxa and 1 reticulation. We varied the first parent probability of the reticulation to be in $\{0.1, 0.2, 0.3, 0.4, 0.5\}$. We started each NetRAX inference from a RAxML-NG maximum-likelihood tree.

11.1.3 C: Unpartitioned Data

In our unpartitioned data setting, we simulated 50 datasets with 20 taxa and 1 reticulation. For each dataset, we started the NetRAX inference from the RAxML-NG maximum-likelihood tree. In addition, we started a second inference where we merged all simulated partitions into a single partition before running tree or network inference.

11.1.4 D: Scrambled Partitions

In our scrambled partitions experiment setting, we simulated a single dataset with 30 taxa and 3 reticulations, using probability 0.5 for each reticulation. Before running NetRAX inference (using the RAxML-NG maximum-likelihood tree as starting tree), we randomly scrambled the partitions such that $p \in \{0\%, 10\%, 20\%, 30\%, 40\%, 50\%, 60\%, 70\%, 80\%, 90\%, 100\%\}$ of the sites from each partition were randomly reassigned to other partitions.

11.1.5 E: Different Alignment Size

In this experiment setting, we simulated a single network with 30 taxa and 3 reticulations, using probability 0.5 for each reticulation. We simulated $\{100, 500, 1000, 5000, 10000\}$ sites per partition. We started the NetRAX inferences from the RAxML-NG maximum-likelihood trees.

11.1.6 F: Scalability

In this experiment setting, we simulated a 10 networks with 20 taxa and 3 reticulations each, using probability 0.5 for each reticulation. We simulated $80,000$ MSA sites. We started the NetRAX inferences from the RAxML-NG maximum-likelihood trees, using $\{1, 2, 4, 8, 16, 32, 64\}$ MPI processes. For each iteration we measured the total runtime for the NetRAX network inference under both LikelihoodModel.BEST and LikelihoodModel.AVERAGE.

In addition, we also simulated 1 network with 20 taxa and 3 reticulations each, using probability 0.5 for each reticulation. Here, we simulated $800,000$ MSA sites. We started the NetRAX inferences from the RAxML-NG maximum-likelihood trees, using $\{16, 32, 48, 64, 80, 96, 112, 128\}$ MPI processes. we again measured the total runtime for the NetRAX network inference under both LikelihoodModel.BEST and LikelihoodModel.AVERAGE.

11.2 Empirical Data

We started a NetRAX inference on an empirical dataset consisting of wheat genomes (from (Glémin *et al.*, 2019)).

We downloaded the individual gene alignments from `https://bioweb.supagro.inra.fr/WheatRelativeHistory/index.php?menu=download` and merged them into a partitioned MSA. We treated each gene MSA as one partition.

The merged dataset comprises of 47 individuals over 17 species. There are 1387815 patterns in the merged MSA, and 8738 partitions. We also created a subsampled dataset on the 17 species, using the majority-consensus rule (randomly resolving ties) to obtain the per-species sequences.

We inferred a maximum-likelihood tree for both the complete and the subsampled dataset with RAxML-NG using its default GTR+GAMMA substitution model. We then used the maximum-likelihood trees inferred by RAxML-NG as starting networks for NetRAX.

Figures 11.2 and 11.2 show the trees inferred by RAxML-NG.

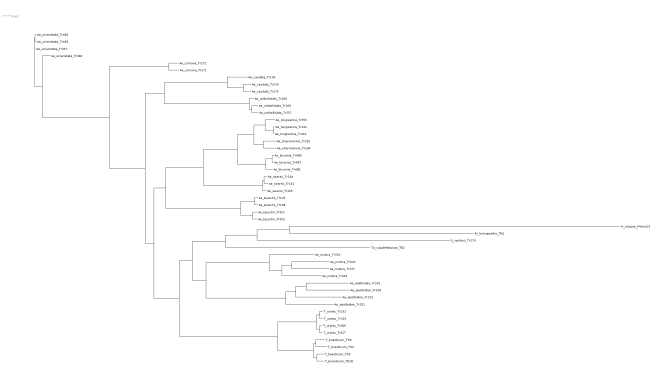

Fig. 14. The maximum likelihood tree inferred by RAxML-NG for the complete empirical dataset, which we used as start network for NetRAX.

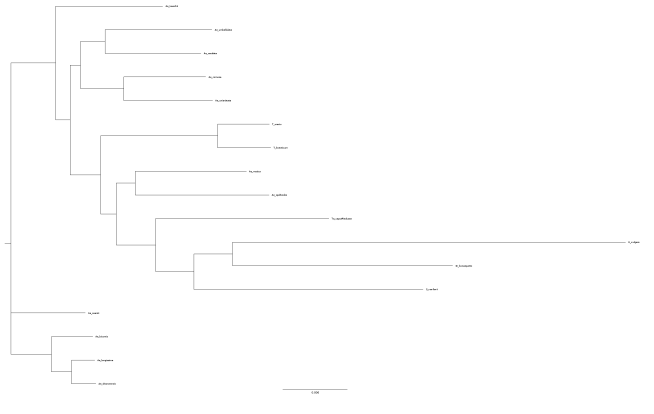

Fig. 15. The maximum likelihood tree inferred by RAxML-NG for the subsampled empirical dataset, which we used as start network for NetRAX.

picture(0,0)(-35,0)(1,0)30 (0,-35)(0,1)30 picture

picture(0,0)(35,0)(-1,0)30 (0,-35)(0,1)30 picture

picture(0,0)(-35,0)(1,0)30 (0,35)(0,-1)30 picture

picture(0,0)(35,0)(-1,0)30 (0,35)(0,-1)30 picture

11.3 Evaluation Metrics

We use the following for evaluating quality of results:

- Number of reticulations in the network
- Likelihood-based evaluation metrics
  - Loglikelihood of the network
  - BIC score of the network
  - AIC score of the network
  - cAIC score of the network
- Topology-based evaluation metrics
  - hardwired network distance
  - softwired network distance
  - displayed trees network distance
  - tripartition network distance
  - nested labels network distance
  - path multiplicity network distance

11.3.1 Topology-based evaluation metrics

See the book Phylogenetic Networks: Concepts, Algorithms and Applications

I have just figured out that we can easily plot relative distance versions (in range [0.0, 1.0]) of all topological network distances. When looking at the definitions in @celines network book, they all are of the form: (|symmetric difference between A and B|) divided by 2. -> We just need to change them to be (|symmetric difference between A and B|) divided by (|A union B|) and there we go. Then, we will get relative distances. These will make nicer plots.

So we need to diverge from the distance definitions in Celines network book: We will explicitly discard the trivial bipartitions/clusters/whatever in our own distance implementations.

Totally ok for discarding trivial bipartitions and use the denominator you suggested and not 2

Cluster A *cluster* in a rooted phylogenetic network is a non-empty proper subset of the set of taxa present in the network. The cluster induced by an edge in the network is the set of taxa "below" the edge. This is, the set of taxa that are descendants of the edge's target node.

- Two clusters are *compatible*, if they are disjoint or one cluster contains the other.
- For obtaining the set of *hardwired* clusters, we collect the clusters induced by every edge in the network. Here, we have all reticulation edges activated at once. With the hardwired interpretation, one edge induces a single cluster.
- For obtaining the set of *softwired* clusters, we go through each tree displayed by the network and add the set of clusters induced by the displayed tree to the total set. Here, we toggle active and inactive reticulation edges when going through the displayed trees. With the softwired interpretation, one edge induces a set of clusters (up to one per displayed tree).

- A *cluster* induced by an edge is the set of taxa descending from the target node of the edge.
- For *hardwired* clusters, we keep all reticulation edges activated at once, and each edge induces a single cluster this way.
- For *softwired* clusters, we go through the displayed trees one-by-one, switching reticulation edges on and off during the process. Here, each edge induces a set of clusters (up to one per displayed tree).

Unrooted Softwired Distance For a given network N, $T(N)$ are the displayed trees of N and $B(N)$ are the set of all bipartitions of the trees in $T(N)$. Then, the softwired unrooted distance between two networks N_1 and N_2 is

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

12 Results and Discussion

Detailed experimental results are in the supplementary text.

- Good result: BIC better-or-equal or unrooted softwired distance zero
- Okay result: BIC worse, unrooted softwired distance >0, but correct number of reticulations
- Bad result: BIC worse, unrooted softwired distance >0, wrong number of reticulations

12.1 A: Standard, 10 taxa, 1 reticulation

12.2 A: Standard, 20 taxa, 2 reticulations

12.3 A: Standard norandom, 40 taxa, 1 reticulation

12.4 A: Standard norandom, 40 taxa, 2 reticulations

12.5 A: Standard norandom, 40 taxa, 3 reticulations

12.6 A: Standard norandom, 40 taxa, 4 reticulations

12.7 B: Reticulation Probability

12.7.1 Reticulation prob 0.1

12.7.2 Reticulation prob 0.2

12.7.3 Reticulation prob 0.3

12.7.4 Reticulation prob 0.4

12.7.5 Reticulation prob 0.5

12.8 C: Unpartitioned Data

12.9 D: Scrambled Partitions

We use + for better-or-equal BIC/AIC/AICc/loglh, and - for worse BIC/AIC/AICc/loglh.

We use + for good result, o for okay result, - for bad result.

12.10 E: Different Alignment Size

partition size	100	500	1000	5000	10000
Inferred BIC	+	-	-	-	-
Inferred AIC	-	-	-	-	-
Inferred AICc	-	-	-	-	-
Inferred logl	-	-	-	-	-
Inferred n_reticulations	2	**3**	**3**	**3**	**3**
Unrooted softwired distance	0.08	0.14	0	0	0
Result	+	o	+	+	+
Runtime RAxML	25	89	182	1219	2440
Runtime NetRAX	73	370	529	1988	4884

Table 3. Results for experiment E, LikelihoodType.AVERAGE, starting from RAxML-NG best tree.

picture(0,0)(-35,0)(1,0)30 (0,-35)(0,1)30 picture

picture(0,0)(35,0)(-1,0)30 (0,-35)(0,1)30 picture

picture(0,0)(-35,0)(1,0)30 (0,35)(0,-1)30 picture

picture(0,0)(35,0)(-1,0)30 (0,35)(0,-1)30 picture

scrambling factor	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Inferred BIC	-	-	-	+	+	+	+	+	+	+	+
Inferred AIC	-	-	-	+	+	+	+	+	+	+	+
Inferred AICc	-	-	-	+	+	+	+	+	+	+	+
Inferred logl	-	-	-	+	+	+	+	+	+	+	-
Inferred n_reticulations	**3**	**3**	5	**3**	**3**	**3**	**3**	1	0	0	1
Unrooted softwired distance	**0**	**0**	0.09	0.17	0.21	0.21	0.21	0.35	0.38	0.38	0.36
Result	+	+	-	+	+	+	+	+	+	+	+
Runtime RAxML	120	125	123	123	127	124	125	133	125	128	126
Runtime NetRAX	380	314	2274	313	230	314	448	72	10	8	55

Table 1. Results for experiment D, LikelihoodType.AVERAGE, starting from RAxML-NG best tree.

scrambling factor	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Inferred BIC	-	-	-	+	+	+	+	+	+	+	+
Inferred AIC	-	-	-	+	+	+	+	+	+	+	+
Inferred AICc	-	-	-	+	+	+	+	+	+	+	+
Inferred logl	-	-	-	+	+	+	+	+	+	+	+
Inferred n_reticulations	**3**	**3**	5	**3**	**3**	**3**	2	1	0	0	1
Unrooted softwired distance	**0**	**0**	0.09	0.17	0.21	0.23	0.21	0.36	0.38	0.38	0.37
Result	+	+	-	+	+	+	+	+	+	+	+
Runtime RAxML	120	125	123	123	127	124	125	133	125	128	126
Runtime NetRAX	277	219	2062	216	165	156	134	48	6	5	21

Table 2. Results for experiment D, LikelihoodType.BEST, starting from RAxML-NG best tree.

partition size	100	500	1000	5000	10000
Inferred BIC	+	-	-	-	-
Inferred AIC	-	-	-	-	-
Inferred AICc	-	-	-	-	-
Inferred logl	-	-	-	-	-
Inferred n_reticulations	2	**3**	**3**	**3**	**3**
Unrooted softwired distance	0.08	0.14	0.14	0	0
Result	+	o	o	+	+
Runtime RAxML	25	89	182	1219	2440
Runtime NetRAX	28	243	534	1848	4661

Table 4. Results for experiment E, LikelihoodType.BEST, starting from RAxML-NG best tree.

12.11 F: Scalability

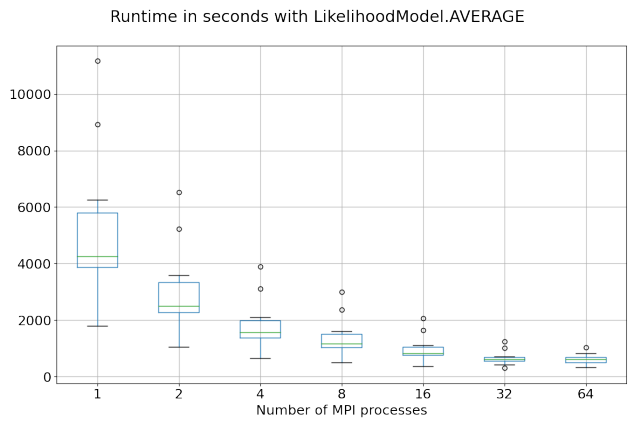

Fig. 16. Runtime in seconds for LikelihoodModel.AVERAGE, over 10 different datasets.

picture(0,0)(-35,0)(1,0)30 (0,-35)(0,1)30 picture

picture(0,0)(35,0)(-1,0)30 (0,-35)(0,1)30 picture

picture(0,0)(-35,0)(1,0)30 (0,35)(0,-1)30 picture

picture(0,0)(35,0)(-1,0)30 (0,35)(0,-1)30 picture

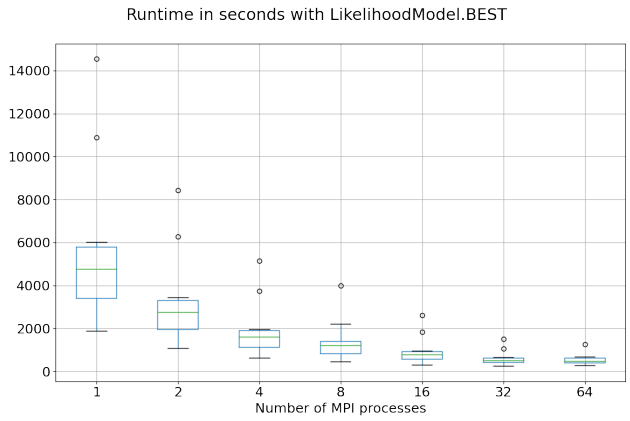

Fig. 17. Runtime in seconds for LikelihoodModel.BEST, over 10 different datasets.

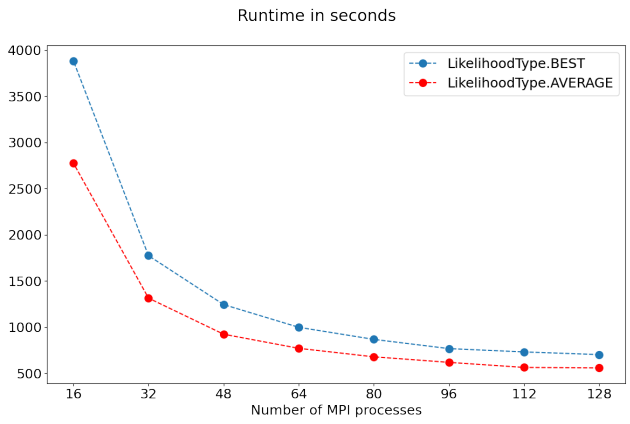

Fig. 18. Runtime in seconds for a simulated dataset with 20 taxa and 3 reticulations, with 189212 MSA patterns in total. The inference with LikelihoodModel.BEST took RSPRMove: 4, RNNIMove: 6, ArcRemovalMove: 2, ArcInsertionMove: 5. The inference with LikelihoodModel.AVERAGE took RSPRMove: 4, RNNIMove: 3, ArcRemovalMove: 1, ArcInsertionMove: 4

TODO.

Spoiler: If you look at the unrooted softwired network distance, we are getting great results starting from only RAxML-NG best tree. Often even relative distance zero, even with 40 taxa and 4 reticulations!

I also noticed that LikelihoodModel.AVERAGE always performed better-or-equal (and yes, sometimes slightly better!) than LikelihoodModel.BEST in our simulated datasets. Which is a surprising result because LikelihoodModel.BEST should be fine for our simulated data (since we simulated each partition on a single displayed tree). My explanation attempt is that the NetRAX network search gets stuck in local optima. Also, non-surprising as it requires less computations, LikelihoodModel.BEST was always the faster one.

Some more initial spoiler informations I see from closely looking at the CSV file (the one I posted above) from the standard experiment round: The relative unrooted softwired network distance is damn good, overall There were two cases where that distance was a bit higher: In Case 1, we found a different network with slightly better BIC score than the true network In Case 2, there were two near-zero branches in the simulated network In one of the setting (with 10 taxa and 1 reticulation), BIC preferred a tree. In all other settings, we always inferred the correct number of reticulations when using LikelihoodModel.AVERAGE. In the rare cases where LikelihoodModel.AVERAGE performed better than LikelihoodModel.BEST, it was because LikelihoodModel.BEST inferred 1 reticulation less than LikelihoodModel.AVERAGE. (edited) 9:25 I hereby conclude that the slower-to-evaluate LikelihoodModel.AVERAGE is the better choice regarding quality of inference results. This is because it seems to perform slightly better when it comes to avoiding local optima in the search. It is not an inherent advantage of the model per se, but happens when interacting with the currently implemented network search algorithm.

now it's prefiltering for 7 reticulations :exploding_head: ... (the theoretical maximum we could end up with here is 16, as there are 16 partitions in the dataset) I need to abort this experiment and run it with way less taxa and reticulations to start with! It already gets very clear that LikelihoodModel.BEST is garbage if the passed partitions are dirty. Still interesting to see what will happen with LikelihoodModel.AVERAGE, that one should work out just fine. (edited) 8:48 I am 99% sure that the error we will get is an out of memory error. Because now with prefiltering 7 reticulations (meaning we have 14 displayed trees to keep track of, and our current implementation keeps all CLVs for all displayed trees in memory), NetRAX already uses 10 out of 16 GB RAM on the PhD laptop.

essentially what is happening here is garbage in, garbage out. Because when having a partition, we assume all sites belonging to the partition evolved together. With the scrambling, we are violating this assumption.

Interesting. With 30 taxa 3 reticulations, we did not have the rapid reticulation growth in the scramble partitions experiment. Instead, both likelihood models performed equally bad the more messy the partitions got, LikelhoodModel.AVERAGE was only slightly better, but comparable to LikelihoodModel.BEST. (edited) 4:05 both LikelihoodModel.BEST and LikelihoodModel.AVERAGE had the overestimating number of reticulations issue as soon as 30% of the sites were scrambled among partitions. They just overestimated the number of reticulations less (by just 1) than before.

13 Future Work

We describe future work ideas for improving runtime performance, improving inference result quality, and providing more features in the supplementary text.

14 Conclusion

TODO (I am waiting for the experiments to finish before writing the conclusion)

Acknowledgement

Part of this work was funded by the Klaus Tschira foundation.

References

(2021a). Computing the likelihood of a tree. `https://github.com/xflouris/libpll/wiki/Computing-the-likelihood-of-a-tree`. Website. Accessed July 28, 2021.

(2021b). libpll-2. `https://github.com/xflouris/libpll-2.git`. Website. Accessed July 28, 2021.

(2021). Nepal – phylogenetic networks parsimony and likelihood toolkit. `http://old-bioinfo.cs.rice.edu/nepal/`. Website. Accessed July 28, 2021.

(2021). pll-modules. `https://github.com/ddarriba/pll-modules`. Website. Accessed July 28, 2021.

Cao, Z., Zhu, J., and Nakhleh, L. (2019). Empirical performance of tree-based inference of phylogenetic networks. In *19th International Workshop on*

picture(0,0)(-35,0)(1,0)30 (0,-35)(0,1)30 picture

picture(0,0)(35,0)(-1,0)30 (0,-35)(0,1)30 picture

picture(0,0)(-35,0)(1,0)30 (0,35)(0,-1)30 picture

picture(0,0)(35,0)(-1,0)30 (0,35)(0,-1)30 picture

Algorithms in Bioinformatics (WABI 2019). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Cardona, G., Rosselló, F., and Valiente, G. (2008). Extended newick: it is time for a standard representation of phylogenetic networks. *BMC bioinformatics*, **9**(1), 1–8.

Felsenstein, J. (1981). Evolutionary trees from dna sequences: a maximum likelihood approach. *Journal of molecular evolution*, **17**(6), 368–376.

Gambette, P., Van Iersel, L., Jones, M., Lafond, M., Pardi, F., and Scornavacca, C. (2017). Rearrangement moves on rooted phylogenetic networks. *PLoS computational biology*, **13**(8), e1005611.

Glémin, S., Scornavacca, C., Dainat, J., Burgarella, C., Viader, V., Ardisson, M., Sarah, G., Santoni, S., David, J., and Ranwez, V. (2019). Pervasive hybridizations in the history of wheat relatives. *Science advances*, **5**(5), eaav9188.

Holoborodko, P. (2021). Mpfr c++. `http://www.holoborodko.com/pavel/mpfr/`. Website. Accessed July 28, 2021.

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

Kozlov, A. M., Darriba, D., Flouri, T., Morel, B., and Stamatakis, A. (2019). Raxml-ng: a fast, scalable and user-friendly tool for maximum likelihood phylogenetic inference. *Bioinformatics*, **35**(21), 4453–4455.

Maddison, W. P. and Knowles, L. L. (2006). Inferring phylogeny despite incomplete lineage sorting. *Systematic biology*, **55**(1), 21–30.

Nakhleh, L., Jin, G., Zhao, F., and Mellor-Crummey, J. (2005). Reconstructing phylogenetic networks using maximum parsimony. In *2005 IEEE Computational Systems Bioinformatics Conference (CSB'05)*, pages 93–102. IEEE.

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*, pages 126–140. Springer.

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

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

Perera, A. (2021). Finding the optimal number of clusters for k-means through elbow method using a mathematical approach compared to graphical approach. `https://www.linkedin.com/pulse/finding-optimal-number-clusters-k-means-through-elbow-asanka-perera/`. Website. Accessed July 28, 2021.

Solís-Lemus, C. and Ané, C. (2016). Inferring phylogenetic networks with maximum pseudolikelihood under incomplete lineage sorting. *PLoS genetics*, **12**(3), e1005896.

Strimmer, K. and Moulton, V. (2000). Likelihood analysis of phylogenetic networks using directed graphical models. *Molecular biology and evolution*, **17**(6), 875–881.

Than, C., Ruths, D., and Nakhleh, L. (2008). Phylonet: a software package for analyzing and reconstructing reticulate evolutionary relationships. *BMC bioinformatics*, **9**(1), 1–16.

Thorndike, R. L. (1953). Who belongs in the family? *Psychometrika*, **18**(4), 267–276.

Wen, D., Yu, Y., Zhu, J., and Nakhleh, L. (2018). Inferring phylogenetic networks using phylonet. *Systematic biology*, **67**(4), 735–740.

Zhang, C., Ogilvie, H. A., Drummond, A. J., and Stadler, T. (2018). Bayesian inference of species networks from multilocus sequence data. *Molecular biology and evolution*, **35**(2), 504–517.

picture(0,0)(-35,0)(1,0)30 (0,-35)(0,1)30 picture

picture(0,0)(35,0)(-1,0)30 (0,-35)(0,1)30 picture