Karlsruhe Institute of Technology

Approximating Phylogenetic Tree Distributions with Distance-Based Methods

Master's Thesis of

Noah Wahl

At the Department of Informatics
Institute of Theoretical Informatics (ITI)

Reviewer:	Prof. Dr. Alexandros Stamatakis
Second reviewer:	Prof. Dr. Peter Sanders
Advisor:	Dr. Benoit Morel

01. June 2023 – 01. December 2023

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I hereby declare that this document has been composed by myself and describes my own work, unless otherwise acknowledged in the text. I also declare that I have read and observed the *Satzung zur Sicherung guter wissenschaftlicher Praxis am Karlsruher Institut für Technologie.*

Karlsruhe, 01. December 2023

.....

(Noah Wahl)

Abstract

Accurately reconstructing the evolutionary history of a group of organism is a complex task. Current state-of-the-art tools produce phylogenetic tree distributions with Markov-chain Monte-Carlo (MCMC) methods by sampling the posterior tree distribution under a given model to reflect uncertainties in the underlying models and data. While these distributions offer very good insight into the phylogenetic history, they are very compute-intensive. In this thesis we present and evaluate multiple heuristics to approximate these distributions with distance-based methods. To judge the quality of our heuristics, we compare our distribution against a reference MCMC-based distribution with split- and frequency-based metrics. We show that our method works well for some types of data, but not all, compared to other tools, and that further information about the data needs to be incorporated to make this viable in practice. Our most successful method is characterized by the use of *pair-wise distance distributions* to apply likelihood-supported perturbation to the input distances for the Neighbor Joining algorithm. Because this ignores the interdependencies between distances, we need to add *parsimony filtering* as a post-processing step to eliminate unlikely trees from our distributions, which significantly improves the results. Finally, we also discuss the shortcomings and future potential of our heuristics to more accurately estimate pair-wise distances and their interdependencies, which should lead to more competitive results.

Zusammenfassung

Die genaue Rekonstruktion der Evolutionsgeschichte einer Gruppe von Organismen ist eine komplexe Aufgabe. Die derzeit modernsten Tools erzeugen phylogenetische Baumverteilungen mit einem Markov-Ketten Monte-Carlo Algorithmus, der Stichproben aus der posteriore Verteilung der Bäume unter einem gegebenen Model zieht und somit die Unsicherheiten in den zugrunde liegenden Modellen und Daten widerspiegeln. Diese Verteilungen bieten zwar einen sehr guten Einblick in die phylogenetische Geschichte, sind aber gleichzeitig sehr rechenintensiv. In dieser Arbeit werden mehrere Heuristiken vorgestellt und evaluiert, um diese Verteilungen mit distanzbasierten Methoden zu approximieren. Um die Qualität unserer Heuristiken zu beurteilen, vergleichen wir unsere Verteilung mit einer Referenzverteilung mit split- und frequenzbasierten Metriken. Wir zeigen, dass unsere Methode für gewisse Daten gut funktioniert, aber nicht für alle, im Vergleich mit anderen Tools, und dass weitere Informationen über die Daten einbezogen werden müssen, um um sie in der Praxis anwendbar zu machen. Unsere Methode nutzt *paarweise Distanz-Verteilungen* um Likelihood-basierte Perturbationen für die Eingabe-Distanzen des Neighbor Joining Algorithmus zu erzeugen. Da dies jedoch die Interdependenzen der Distanzen nicht berücksichtigt, benötigen wir *Parsimony-Filterung* als einen Nachbereitungsschritt um unwahrscheinliche Bäume aus unseren Verteilungen zu eliminieren, was die Ergebnisse erheblich verbessert. Schließlich diskutieren wir auch die Mängel und das zukünftige Potenzial unserer Heuristiken. Wir erhoffen uns dabei eine genauere Schätzung der paarweisen Abstände und ihrer gegenseitigen Abhängigkeiten, was zu wettbewerbsfähigeren Ergebnissen führen sollte.

Contents

List of Figures

List of Tables

1. Introduction

1.1. Motivation

Phylogenetic tree inference is used to reconstruct the evolutionary relationships among a group of organisms based on genetic, morphological, or other biological data, using computational methods to depict their common ancestry and divergence. Because evolutionary models are complex and there is no ground-truth available, verifying whether the "true" tree has been found is nigh impossible. Therefore, it becomes desirable to not only generate a single tree, but a whole set of trees or a distribution where plausible trees are more likely. For gene tree reconciliation these distributions are of great value because large uncertainties in the gene tree topologies are very common. The current tools use either Bayesian inference with Markov-chain Monte-Carlo methods or ultra-fast bootstrapping. While the former produces a statistically meaningful posterior distribution under a given model, it has no runtime guarantees and suffers from missing parallelism. The latter is substantially faster, but the resulting tree sets are not strictly distributions, but only contain highly likely trees under the inferred model. We want to use Neighbor Joining, another algorithm to construct phylogenetic trees, as an alternative way to (efficiently) generate sets of trees that closely approximate the distributions of Bayesian inference tools.

1.2. Related Work

While they both are very relevant to this thesis, we will mention *MrBayes* [Ron+12] and *IQ-TREE* [Min+20] only shortly in this section. Their underlying methods and other details will be addressed separately in Sections 2.2.5 and 2.2.6, respectively. In the remainder of this section we will list other methods that use Neighbor Joining (NJ) to infer phylogenetic trees.

There are several high-performance implementations of NJ available, including *BioNJ* [Gas97] and *RapidNJ* [SMP08] which use branch-and-bound to speed up the selection of nodes to join, as well as *FastTree* [PDA10] and *FastME* [LDG15] which use a heuristic approach coupled with local search based on Nearest Neighbor Interchange (NNI) and Subtree Pruning and Regrafting (SPR). Other methods try to address the uncertainty in phylogenetic distances in other ways. *Multifurcating NJ* [FSS23] extends traditional NJ by allowing polytomies in addition to dichotomies at inner nodes if there are multiple, equally suited candidates to join. This not only incorporates uncertainties, but also eliminates inconsistencies in the resulting trees from the order of the taxa in the input. Finally, *KDETREES* [Wey+14] seeks to identify interesting outliers in phylogenetic tree distributions, especially for gene histories.

As of writing this thesis there is no literature on approximating phylogenetic tree distributions from MCMC methods with distance-based methods.

1.3. Contribution

The goal of this thesis is to explore distance-based heuristics to approximate the posterior phylogenetic tree distributions of Bayesian inference tools in less time while outperforming the ultra-fast bootstrap method in terms of quality. While we motivate the approximation of these distributions with their application in gene tree reconciliation, which is an active field of research in computational biology [BS20 | MWSS23], the methods in this thesis are developed and evaluated for general DNA and amino acid sequences. We use the Neighbor Joining (NJ) algorithm, normally used to construct a single tree from pair-wise distances between the sequences, as the basis of our methods. We propose a set of representative metrics to compare sets of phylogenetic trees and explore perturbation strategies to add variance to the trees generated by repeated NJ execution. While we fall short of achieving results that are consistently competitive in quality, we do propose methods that should be considered for a more accurate approximation in the future. We analyze and discuss the current shortcomings and point out potential solutions. The most promising avenue of application for our method currently seems to be in generating trees for challenging datasets. The experimental evaluation of our strategies is done in our own NJ implementation *neighbo-rs*[1], allowing for in-depth customization of perturbation of the input, filtering the output and modifying other steps in the NJ algorithm. Part of *neighbo-rs* is a metrics-module that offers fast computation of similarity and distance functions for comparing phylogenetic tree sets similar to well-known Python tools, but significantly faster through multithreading and focusing on split representations of the tree sets.

1.4. Outline

Chapter 2 establishes some key concepts relevant to this thesis. We provide information about the input data in Section 2.2.1, as well as models in Section 2.2.3 and metrics to assess the quality of single trees in Sections 2.2.2 and 2.2.4, and whole distributions of trees in Section 2.3. We also quickly touch on the underlying mechanisms of the tools we compare ourselves against in Sections 2.2.5 and 2.2.6. In the end of this chapter we lay out our evaluation strategy, list key metrics in Section 2.4.1 for our datasets and explain how the data will be visualized in Section 2.5.

Chapter 3 introduces our methods for inferring different trees with NJ and contains preliminary evaluation of these strategies. In Section 3.1 we present 2 approaches to slightly change the distances of the input and compare them on our parameter tuning dataset. In Section 3.2 we do the same for our 3 randomized NJ strategies. Finally, in Section 3.3 we evaluate a post-processing step using parsimony scores to improve our results.

In Chapter 4 we analyze the results of our best method more thoroughly by comparing solution quality within difficulty classes in Section 4.1, finding the best possible results by individually optimizing each Multiple Sequence Alignment (MSA) of the parameter tuning set on its own in Section 4.2, as well as comparing ourselves against the other tools on the full dataset and analyzing the runtime in Section 4.4.

Chapter 5 concludes the thesis by summarizing our results, discussing the impact and listing future work.

[1] https://github.com/noahares/neighbo-rs

2. Preliminaries

2.1. Phylogenetic Trees

A *phylogenetic tree* $\mathcal{T}$ describes the evolutionary history between a set of *taxa*, e.g. organisms, genes, languages and other entities that follow evolutionary principles. Each taxon is assigned to a leaf of $\mathcal{T}$ and edges are *weighted* by relative evolutionary distances. In the context of this thesis phylogenetic trees will be *unrooted* and *bifurcating* unless stated otherwise, because we restrain from making assumptions about ancestry relationships as we do not have information about out-groups or other root-inducing methods for our data. All of our algorithms and metrics are uninfluenced by whether the tree is rooted.

To evaluate our methods, we extensively use the *split representation* of trees. Let therefore $T = \{\tau_1, \ldots, \tau_n\}$ be a set of taxa and $\mathcal{T}$ a *phylogenetic tree* on these taxa. A *split* of $\mathcal{T}$ is a bi-partition $(T_1 \mid T_2)$ of T, s.t. $T_1 \cup T_2 = T, T_1 \cap T_2 = \emptyset$ and $\mathcal{T}(T_1), \mathcal{T}(T_2) \subset \mathcal{T}$ with the latter meaning that the induced trees of T_1 and T_2 are subtrees of $\mathcal{T}$. The set of all such valid splits uniquely identifies the tree $\mathcal{T}$.

2.2. Phylogenetic Inference

The process of reconstructing the evolutionary history for a group of organisms is called *phylogenetic inference*. There exist many ways to approach this problem, all with their strengths and weaknesses. In this section we give a short overview of the most common scoring functions, parsimony and likelihood, before moving on to the methods that will be compared in this thesis.

2.2.1. Input Data

The raw input for *phylogenetic inference* are sequences of Deoxyribonucleic Acid (DNA) or Amino Acids (AA) (in the scope of this thesis). DNA encodes hereditary information about the species and comes in 4 different forms: Adenine, Cytosine, Guanine and Thymine. Sequences are encoded as strings from the alphabet $\Sigma = \{A, C, G, T\}$ representing the aforementioned bases. The encoding of Ribonucleic Acid (RNA) is closely related to DNA, substituting Thymine for Uracil (U). Triplets of RNA form so-called *codons*, which encode for the 20 different AAs present in all organisms.

The preprocessed input for phylogenetic inference is a *Multiple Sequence Alignment* (MSA), consisting of many DNA or AA sequences with gaps (–) inserted such that they all have the same length and some metric between the resulting sequences is minimized. A column of an MSA is called a *site*.

The result of phylogenetic inference is a single tree or set of trees that explain the evolutionary ancestry between the input sequences the best. This optimization can be done under a wide variety of optimality criteria, like *Maximum Likelihood* (ML) and *Maximum Parsimony* (MP). Finding the optimal tree for both of these is NP-hard [CT05 | DJS86].

2.2.2. Parsimony

The *parsimony score* [Fit77] is one of the simplest methods to score a phylogenetic tree. It optimizes for the *Minimum Evolution* (ME) criterion by counting the number of mutations across the tree.

It represents a quantitative measure of the evolutionary changes necessary to transform an ancestral state into the observed character states in the data. In the context of DNA sequence data, this typically involves the number of nucleotide substitutions, insertions, and deletions required to explain the observed sequences on a given phylogenetic tree. With AA sequence data the number of nucleotide substitutions is replaced by whole AA substitutions. The tree with the lowest parsimony score is considered the most parsimonious, suggesting that it is the most likely representation of the true evolutionary history. The runtime to score one tree is $O(sm)$ where s is the number of sequences and m the length of one sequence. In practice, the hidden constant is very small.

MP uses the parsimony score to infer phylogenetic trees. It seeks to identify the tree that minimizes the total parsimony score across all characters and branches. To do this, the MP method explores different tree topologies, evaluates potential character state changes along each branch, and calculates the parsimony score for each tree. The tree with the lowest overall parsimony score is selected as the preferred phylogenetic hypothesis. The MP criterion is particularly valuable when dealing with limited data where model parameters for ML (see Section 2.2.4) are hard to estimate. Because the tree search space is vast, various algorithms, such as branch-and-bound or heuristic search algorithms, are employed to find the optimal or near-optimal solutions efficiently. This approach provides a valuable perspective on phylogenetic relationships, particularly when simplicity is required.

2.2.3. Substitution Models

One big downside of the parsimony score is that it treats all substitutions the same. However, some substitutions are much more likely to happen than others. To consider this information in evaluating phylogenetic trees, a multitude of *substitution models* have been proposed. In the scope of this thesis, we mainly focus on the Generalized Time Reversible (GTR) model [TM86] for DNA data and the LG model [LLG08] for AA data.

A *substitution model* M for a set of traits (in our case DNA or AA characters) $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ is a tuple $(\pi, R)_\Sigma$ where π is a vector containing the initial frequencies for each $\sigma \in \Sigma$. The frequencies in π are called *equilibrium frequencies* and $\sum_\pi = 1$. R is the *substitution rate matrix* and R_{ij} contains the rate at which σ_i is substituted for σ_j. Evolution is modeled as a continuous time Markov-chain and therefore the rows of R must all sum to 0, and we assume that evolution is time-reversible, so $\pi_i R_{ij} = \pi_j R_{ji}\ \forall i, j \in [n]$. Additionally, R is normalized by a factor, such that $-\sum_{i \in [n]} \pi_i R_{ii} = 1$. Therefore, M can

be described by at most $\frac{n(n-1)}{2} - 1$ parameters for the rates of R and $n-1$ parameters for π. For both parts of the model, we can fix one parameter to be constant (1) because of their interdependencies.

GTR uses the maximum amount of parameters, while LG has no parameters. The parameters for GTR need to be estimated from the input DNA sequences. While this can lead to over-parameterization and higher runtime, it is widely used in phylogenetic analysis. For AA data, using many free parameters is computationally infeasible for most applications, so the most commonly used models have very few or no parameters at all. The substitution rates for the LG model come from empirical analysis of large datasets and are fixed.

Studying the speed at which different alignment sites change has shown that modeling among-site rate-heterogeneity is an important part of selecting a substitution model [Yan96]. These variable rates are assumed to follow a Gamma distribution. A model M is then expanded by a rate-vector $\Gamma = \{\gamma_1, \ldots, \gamma_c\}$ of size c (often 4) and likelihood calculations (Section 2.2.4) are then performed with c different rate matrices $\gamma_1 R, \ldots, \gamma_c R$ that are weighted equally. The name of a model containing Gamma-rates is extended by +G (e.g. GTR+G). For simplicity, we will not use Gamma-rates for inferring trees in this thesis.

2.2.4. Likelihood

Another way of scoring trees is to compute the *likelihood* $L(\Theta|D)$ of observing the tree parameters $\Theta = (T, b, M)$, given the MSA D [Yan14]. The likelihood score considers the topology T of the tree, the substitution model M, as well as the branch lengths b. Because the inner states of the tree are usually unknown, and to not have to iterate all possible inner state combinations, conditional likelihoods are computed via Felsenstein pruning [Fel81 | Fel73], a dynamic programming algorithm that allows summation over all possible inner states without having to enumerate them explicitly. This can be done for each site of the sequences individually without dependencies, which leads to great parallelization opportunities. Because per-site likelihoods become very small, *log*-likelihoods are used in practice to prevent numerical problems.

Let Σ be the 4-letter DNA alphabet or the 20-letter AA alphabet, respectively. Let k be an inner node of the tree with children i and j and respective branch lengths b_i and b_j. Furthermore, let $P(t) = exp(Rt)$ be the instantaneous rate matrix of the substitution model $M = (\pi, R)_\Sigma$ with branch length t. The recursive formula for the conditional likelihood for the inner state $a \in \Sigma$ at an inner node k for a site c is shown in Equation 2.1.

$$L_a^k(c) = \left(\sum_{s \in \Sigma} P_{as}(b_i) L_s^i(c)\right)\left(\sum_{s \in \Sigma} P_{as}(b_j) L_s^j(c)\right) \tag{2.1}$$

The overall likelihood of site c can then be computed at the root r as $L(c) = \sum_{s \in \Sigma} \pi_i L_s^r(c)$. As with parsimony (see Section 2.2.2) the runtime is $O(sm)$, but the hidden constant is much higher.

ML explores the tree space similar to MP to find a tree that maximizes the likelihood.

2.2.5. Markov-Chain Monte-Carlo Methods

Phylogenetic inference often involves estimating parameters such as branch lengths, substitution models, and tree topologies that explain the observed sequence data. These parameters are essential for understanding the evolutionary history of the species under study. However, estimating these parameters requires a lot of (high quality) data, which often is not available. If the parameters are wrong, the tree inferred under e.g. ML might not be the correct one. This is where the Markov-Chain Monte-Carlo (MCMC) technique comes into play. Instead of producing one single tree, it generates a distribution which potentially contains better trees even if the parameters are chosen suboptimally. We use *MrBayes* [Ron+12] to generate phylogenetic tree distributions with MCMC.

MCMC allows sampling from the posterior distribution of phylogenetic parameters. The fundamental idea behind MCMC is to construct a Markov-chain in which each step proposes a new set of parameter values based on the current state. These proposals are accepted or rejected based on their posterior probabilities, with a higher probability of acceptance for more probable parameter values. Over many iterations, MCMC explores the posterior distribution, and the samples collected converge to a representation of the true posterior distribution.

MrBayes employs a specific variant of MCMC known as Metropolis-Coupled Markov-Chain Monte-Carlo (MC^3). This approach enhances exploration of the parameter space by running multiple chains in parallel, occasionally allowing them to swap states. Each chain in *MrBayes* represents a phylogenetic tree and associated parameter values, and the swapping of states helps overcome local optima and explore the parameter space more effectively.

MrBayes allows users to specify various aspects of the phylogenetic model, including substitution models, priors, and the number of chains to run. Users can also specify the number of generations, burn-in period, and sampling frequency. The program then performs MCMC iterations to estimate the posterior distribution of phylogenetic parameters. Posterior samples are collected during the stationary phase of the MCMC run, typically after the burn-in period.

2.2.6. Bootstrapping

In phylogenetic inference, the *bootstrap analysis* is often used to assess the reliability and robustness of inferred phylogenetic trees. This process involves resampling the original MSA with replacement to create multiple bootstrap datasets. Each bootstrap dataset is subjected to the same phylogenetic inference procedure as the primary dataset. The result is a collection of trees, each representing a different sample of the data. These bootstrap trees are then used to construct a consensus tree that quantifies the level of support for various clades within the phylogeny, allowing to draw conclusions about the stability of the inferred tree topologies.

We use IQ-TREE [Min+20] to infer these bootstrap trees and treat them as a representative sample of a pseudo-distribution for our analysis. IQ-TREE uses a variation of bootstrapping which is called ultra-fast bootstrapping. This is significantly faster than the traditional bootstrap by reusing trees and speeding up likelihood calculations.

2.2.7. Neighbor Joining

Algorithm 2.1: Neighbor-Joining Algorithm

Input: $n \times n$ Distance matrix D
Output: Phylogenetic tree T

```
1  T ← set of n trees, initially one for each sequence
2  while |T| > 1 do
3      d ← Array of size |T|
4      for each i ∈ T do
5          d[i] ← Σ_{k=1}^{n} D[i][k]
6      for each (i, j) ∈ T × T do
7          Q(i, j) ← (|T| − 2)D[i][j] − d[i] − d[j]
8      (i, j) ← argmin(Q)
9      t: Tree ← mergeNodes(i, j)
10     T.removeTree(i); T.removeTree(j)
11     D.removeSequence(i); D.removeSequence(j)
12     D.addSequence(t)
13     for each i ∈ T do
14         D.updateDistance(i, t)
15     T.addTree(t)
16 return T
```

Neighbor Joining (NJ) is a distance-based, greedy heuristic for the *balanced* ME criterion by Saitou and Nei [SN87] to infer the shortest phylogenetic tree for n sequences from a pair-wise distance matrix D. Starting from a star tree with all sequences (Line 1), the NJ algorithm iteratively combines two nodes with the smallest value in the so-called Q-Matrix into a new node (Lines 8-12). An entry $Q(i, j)$ is calculated from the respective distance in D and the sum over all distances from the nodes i and j (Line 4-7). Then, the distances from all remaining nodes to this new node are computed from the old distances and a new Q-Matrix is calculated (Lines 13-14). The algorithm is formalized in Algorithm 2.1.

Because the Q-Matrix has to be recalculated in each step and the minimum has to be found, the runtime adds up to $O(n^3)$. Some implementations like *RapidNJ* achieve expected $O(n^2)$ runtime [SMP08].

Implementation Details Explicitly materializing the Q-Matrix in each step of the NJ algorithm introduces many unnecessary write-operations and also wastes space. To circumvent this, we use a closure that takes 2 indices of the distance matrix and computes the Q-Matrix entries (Algorithm 2.1, Line 6-7) on demand from the pre-computed row-sums vector d (Algorithm 2.1, Line 3-5) and the distance matrix D. We keep a list of active indices to keep track of which rows and columns in D represent nodes that still need to be joined and reuse D between NJ steps. Combined with the aforementioned representation of the Q-Matrix, this allows for efficiently finding the minimum without materializing large vectors explicitly. Thanks to Rust's excellent support for functional programming paradigms, we can iterate the active indices, create a lexicographically ordered Cartesian product and find the minimum by passing the

Q-Matrix closure to `Iterator::min_by_key()`. *Perf*[1] reports that finding the minimum of the Q-Matrix makes up 46.7% of the NJ runtime, the remainder is mostly spent computing d (39.6%).

To save space, the distance matrix is represented as a 1-dimensional vector for the upper triangle matrix (D is symmetrical) with an appropriate index function. The overhead from this slightly more complicated index function is not significant. For the inner loops we make sure that to access entry $D[i][j]$, $i < j$ holds, simplifying the index calculation. The newly joined node t takes the place of $min(i, j)$ in the distance matrix and in the collection of tree nodes (Algorithm 2.1, Line 9-15).

2.3. Metrics

Deciding whether a distribution of trees is "good" often does not have a straightforward answer. Especially with empirical data, the ground-truth is often missing and even with simulated data it requires those simulations to reflect real-world evolution to be meaningful. Because of these challenges, we decided to always look at relative comparisons. In this case, this means generating a tree distribution with a long-running MCMC chain and compare other distributions on a wide array of metrics. Since there is no established one-beats-all metric to compare phylogenetic tree distributions, we consider many metrics in our evaluation with the goal of shining a light from different angles onto the problem. Broadly speaking, these metrics fall into two categories: split-based and likelihood metrics.

The first set of split-based metrics are dealing with the frequencies of splits. Let P, Q be two phylogenetic tree distributions represented by relative frequencies of their splits, so $P = \{p_i \mid i \in [n]\}$, $Q = \{q_i \mid i \in [n]\}$ with $\sum_P = 1, \sum_Q = 1$. Further, let S_P, S_Q be the sets of splits of P and Q, respectively, so $n = |S_P \cup S_Q|$. All the metrics are normalized to the range $[0, 1]$ with a value of 0 representing identical distributions and 1 representing completely disjoint distributions to easier visualize them against each other.

The most known distance metric we use is the *Average Standard Deviation of Split Frequencies* [Lak+08] (*ASDSF*, Equation 2.2), which amplifies large differences in split frequencies. The *Hellinger Distance* [Hel09] (Equation 2.3) and *Simple Distance* (Equation 2.4) are similar in concept, but weight the mismatches in frequencies differently. The *Hellinger Distance* increases the contribution of infrequent splits, while the *Simple Distance* does not have any such effect. Throughout our evaluation, the distributions rank consistently regardless of which of these three metrics are used. Nonetheless, all three are interesting because comparing their values yields insight on which frequency-magnitudes match the reference the best. Finally, we also use the *Pearson Correlation Coefficient* [Pea94] (*PCC*, Equation 2.5) to measure the linear correlation between distributions, where $\text{cov}(P, Q)$ is the covariance and σ refers to the standard deviation of P and Q respectively.

[1] https://perf.wiki.kernel.org/index.php/Main_Page

$$ASDSF(P,Q) = \sqrt{\frac{1}{\sum_{p_i \in P} p_i^2 + \sum_{q_i \in Q} q_i^2} \sum_{i=1}^{n} (p_i - q_i)^2} \tag{2.2}$$

$$Hellinger(P,Q) = \frac{1}{\sqrt{2}} \sqrt{\sum_{i=1}^{n} (\sqrt{p_i} - \sqrt{q_i})^2} \tag{2.3}$$

$$Simple(P,Q) = \frac{1}{2} \sum_{i=1}^{n} \mid p_i - q_i \mid \tag{2.4}$$

$$PCC(P,Q) = \frac{\mathrm{cov}(P,Q)}{\sigma_P \sigma_Q} \tag{2.5}$$

Another mode of analyzing splits is to look at them as sets. This is very similar to the classic *RF-Distance* [RF81], but for tree distributions instead of single trees. We look at the *Unique Split Ratio* (USR Equation 2.6) which is the normalized symmetric difference of all splits occurring in the distributions. The *One-sided Unique Split Ratios* (USR_R and USR_T Equation 2.7) help with identifying whether a distribution hits most of the reference splits (USR_R) or if it has many splits that are absent in the reference (USR_T).

$$USR(P,Q) = \frac{\mid (S_P \cup S_Q) \setminus (S_P \cap S_Q) \mid}{\mid S_P \cup S_Q \mid} \tag{2.6}$$

$$USR_{R/T}(P,Q) = \frac{\mid S_P \setminus S_Q \mid}{\mid S_P \cup S_Q \mid} \tag{2.7}$$

Implementation Details Initially, we used existing python tools (*Dendropy*[2], *ETE Toolkit*[3] and *Biopython*[4]) to compute the metrics they supported. However, because these tools offer much more functionality than we require, they create internal representations of the trees that increase the parsing time and memory requirements significantly. This made the evaluation a time-consuming process and limited the amount of distributions we could generate with *neighbo-rs* while exploring the parameter space. Therefore, *neighbo-rs* now contains a sub-crate where we compute the metrics ourselves. The main way in which we speed up calculating our results is by parsing *newick* trees straight into their split representation (see Section 2.1). An unrooted tree on n taxa can be represented by $n-3$ bit-vectors representing the splits. These bit-vectors have length n (one bit for each taxon) and indicate on which side of the split a taxon is placed. To prevent ambiguity (a bit-vector being identical to its complement), the bit-vectors are normalized such that the first bit is always set to 0 and therefore, a 0 at index i indicates that taxon i is on the same side of the split as taxon 0. Similarly, a 1 at index j means that taxon j is on the other side of the split relative to taxon 0.

[2] https://dendropy.org
[3] http://etetoolkit.org
[4] https://biopython.org

We store the resulting splits of each distribution in a set to calculate the set-based metrics (*USR*, USR_R and USR_T). To compute the frequency-based metrics we compute the relative frequencies of all splits and store them in a matrix where the rows represent the individual distributions and the columns the splits. This makes computing pairwise metrics very straightforward by picking the respective rows and iterating them simultaneously to sum up the frequency-differences according to the desired metric function.

This potentially leads to increased runtime and wasted space if the tree distributions are vastly different from each other because many frequency values will be 0. We only encountered problems with the weighted selection strategy (Section 3.2.1), because there are many tree distributions to evaluate (f adds another dimension to the parameter space) and the strategy produces many unique trees. In the future this could be optimized by hashing the frequencies by the split they belong to.

2.3.1. Difficulty

A problem of ML heuristics is the lack of global knowledge about the shape of the likelihoods of trees across the search space. Therefore, these heuristics may converge on a local instead of a global maximum. To mitigate this, several independent tree searches are performed and if they produce very topologically similar trees it is likely that this is indeed a global maximum. Otherwise, we must assume that there are multiple local maxima. This is called the *difficulty* of the MSA and was quantified by Haag et al. [HHBS22]. The extremes of this score are 0.0 for very easy MSAs and 1.0 for hopelessly difficult MSAs. Their tool infers several ML trees, evaluates their statistical significance and rates the difficulty of the MSA based on how many of the inferred trees are plausible [Mor+21]. The more trees are plausible, the easier the MSA is rated.

2.4. Experimental Setup

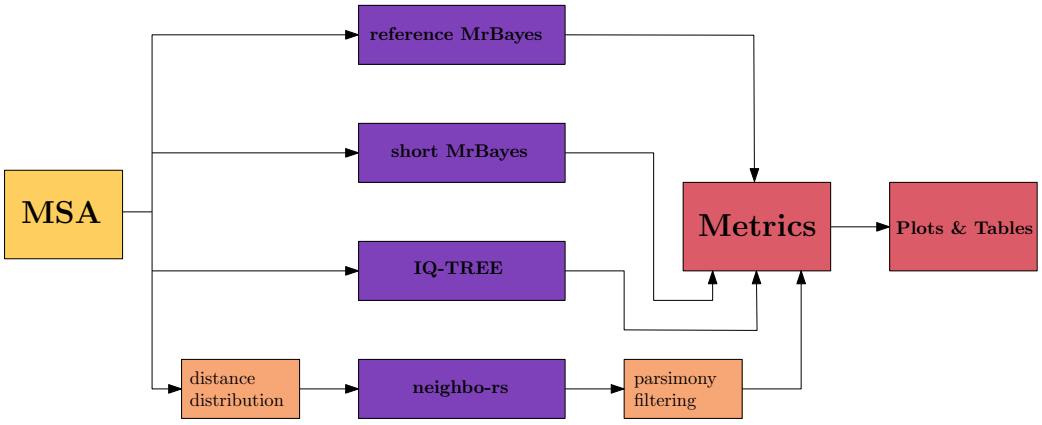


Figure 2.1.: The pipeline for the experiments consists of the input MSA (yellow), the tools to compare (purple), optional pre- and post-processing steps for *neighbo-rs* (orange) as well as the metrics module and plotting scripts (red).

2.4.1. Input Data

We evaluate our methods on both, simulated and empirical data. We use *SimPhy* [MOP16] to generate 100 MSAs for DNA and AA sequences. *SimPhy* simulates evolution of genetic sequences along a phylogenetic tree and generates sequences based on a set of provided parameters. For both, DNA and AA, we generate 100 MSAs with 50 taxa and 200 sites each. We use Gamma distributions to add among-site heterogeneity for the simulated alignments to get more realistic data. The α-parameter of the Gamma distributions is sampled from an $Exp(2)$ distribution for the GTR+G (DNA) and LG+G (AA) models, respectively. The detailed configuration can be found in Appendix A.1. To generate the sequences, *SimPhy* uses *INDELible* [FY09]. Its configuration can also be found in appendix A.2. An overview for the difficulties of the MSAs can be found in Tables A.1 and A.2 in Appendix A.4

Our empirical data comes from *TreeBASE* [PDS00]. We filtered for MSAs similar in size to our simulated data. Again, a short overview with the most important metrics can be found in Tables A.3 and A.4 in Appendix A.5.

2.4.1.1. Parameter Tuning Set

Because evaluating all parameter combinations for the different strategies in Chapter 3 would be very compute-intense, we created a parameter tuning set consisting of 10 MSAs from each of the 4 datasets with varying difficulty (see Section 2.3.1) and size. We use this subset of the data to identify which strategies and parameter configurations are worth including in a more detailed evaluation in Chapter 4. The most important metrics are listed in Tables 2.1b and 2.1a for empirical and simulated DNA data, as well as Tables 2.1d and 2.1c for empirical and simulated AA data, sorted by difficulty.

2.4.2. Tool Parameters

We use a long reference run of *MrBayes* as the distribution we want to approximate. This reference run has 4 independent chains with 550,000 generations, a sample frequency of 500 and a burn-in of 500. The tools we compare against our implementation (*neighbo-rs*) are a shorter *MrBayes* run with 2 chains, 100,000 generations, a sample frequency of 100 and a burn-in of 100, as well as *IQ-TREE* with 1000 bootstrap trees including branch lengths (`iqtree2 -B 1000 -wbtl`). The configuration for all *MrBayes* runs can be found in Appendix A.3.

Due to implementation limitations, *neighbo-rs* does not support any model optimization, neither estimating GTR parameters, nor Gamma-rates. Therefore, we compute the GTR parameters with *raxml-ng* and run all tools without among-site heterogeneity.

We list the key values for the most common metrics used throughout Chapter 3 for *MrBayes* and *IQ-TREE* in Table 2.2, because we reuse the data for these tools in the comparison figures. We will reference them if appropriate, but their contribution to the figures should rather be seen as visual aid when accessing the quality of the *neighbo-rs* strategies.

MSA	Difficulty
dataset_004	0.39
dataset_006	0.39
dataset_008	0.42
dataset_009	0.47
dataset_007	0.48
dataset_005	0.49
dataset_002	0.5
dataset_010	0.51
dataset_001	0.52
dataset_003	0.54

(a) Simulated DNA. All MSAs contain 50 taxa and 200 sites.

MSA	Difficulty	Taxa	Sites
alignment_10148_2	0.34	73	138
alignment_15827_1	0.36	30	117
alignment_20312_4	0.43	33	133
alignment_16855_3	0.54	51	277
alignment_604_0	0.54	39	145
alignment_12828_0	0.76	49	150
alignment_26579_5	0.81	49	280
alignment_15621_0	0.83	65	278
alignment_16634_1	0.83	52	197
alignment_27596_24	0.86	65	187

(b) Empirical DNA.

MSA	Difficulty
dataset_002	0.16
dataset_008	0.25
dataset_001	0.26
dataset_009	0.28
dataset_005	0.3
dataset_003	0.33
dataset_006	0.35
dataset_004	0.43
dataset_007	0.57
dataset_010	0.78

(c) Simulated AA. All MSAs contain 50 taxa and 200 sites.

MSA	Difficulty	Taxa	Sites
alignment_13985_9	0.05	46	149
alignment_10068_0	0.11	32	291
alignment_25084_2	0.12	55	219
alignment_15931_0	0.13	52	119
alignment_18551_1	0.13	78	104
alignment_14979_1	0.16	88	235
alignment_15021_2	0.21	53	285
alignment_21817_1	0.27	81	247
alignment_16190_3	0.55	76	115
alignment_23593_0	0.76	87	268

(d) Empirical AA.

Table 2.1.: MSA properties for the parameter tuning set

Dataset	Tool	PCC				ASDSF				USR				USR_R				USR_T			
		Mean	Std	Min	Max	Mean	Std	Min	Max	Mean	Std	Min	Max	Mean	Std	Min	Max	Mean	Std	Min	Max
Simulated DNA	MrBayes	0.996	0.001	0.992	0.998	0.053	0.012	0.033	0.083	0.310	0.110	0.124	0.501	0.232	0.098	0.068	0.417	0.135	0.060	0.061	0.225
	IQ-TREE	0.916	0.021	0.876	0.941	0.278	0.039	0.228	0.353	0.598	0.107	0.418	0.721	0.506	0.182	0.197	0.693	0.275	0.047	0.191	0.358
Empirical DNA	MrBayes	0.987	0.013	0.961	0.999	0.097	0.058	0.028	0.193	0.614	0.261	0.244	0.932	0.529	0.266	0.168	0.886	0.442	0.291	0.107	0.857
	IQ-TREE	0.752	0.218	0.307	0.985	0.454	0.256	0.114	0.863	0.793	0.219	0.472	0.990	0.729	0.312	0.2	0.980	0.487	0.301	0.150	0.982
Simulated AA	MrBayes	0.995	0.007	0.976	0.999	0.047	0.039	0.015	0.146	0.239	0.137	0.097	0.567	0.188	0.133	0.071	0.527	0.083	0.043	0.027	0.162
	IQ-TREE	0.952	0.015	0.925	0.978	0.198	0.039	0.124	0.260	0.493	0.072	0.377	0.629	0.325	0.108	0.164	0.481	0.329	0.047	0.275	0.434
Empirical AA	MrBayes	0.942	0.115	0.620	0.999	0.155	0.178	0.005	0.608	0.347	0.181	0.155	0.671	0.283	0.160	0.109	0.560	0.148	0.123	0.041	0.434
	IQ-TREE	0.942	0.095	0.674	0.989	0.189	0.139	0.094	0.562	0.540	0.153	0.363	0.924	0.294	0.283	0.0	0.918	0.397	0.082	0.261	0.493
Combined	MrBayes	0.980	0.060	0.620	0.999	0.088	0.102	0.005	0.608	0.378	0.226	0.097	0.932	0.308	0.215	0.068	0.886	0.202	0.211	0.027	0.857
	IQ-TREE	0.890	0.141	0.307	0.989	0.280	0.179	0.094	0.863	0.606	0.184	0.363	0.990	0.463	0.286	0.0	0.980	0.372	0.173	0.150	0.982

Table 2.2.: Comparing *MrBayes* and *IQ-TREE* on mean, standard deviation, minimum and maximum values for the metrics *PCC*, *ASDSF*, *USR*, USR_R and USR_T grouped by the different parameter tuning datasets.

2.4.3. Hardware and Software

All experiments were conducted on an AMD Ryzen 5 2600 with 6 cores with hyper-threading @3.4GHz with 2×8GB of DDR4 RAM @2133MT/s. We use *MrBayes* version 3.2, *IQ-TREE* version 2.1.2, *SimPhy* version 1.0.2, *INDELible* version 1.03 and *raxml-ng* version 1.2.0. *neighbo-rs* and its metrics and distance-estimator sub-crates were written in Rust 2021 Edition and compiled using *rustc* version 1.70 with `RUSTFLAGS="C target-cpu=native"` and a custom *cargo* profile based on the default `release` profile, but with link time optimization enabled to boost performance. The commit of *neighbo-rs* is `70c9b7ec636da03bc85eba79757f4a88de0398c2` for all generated data. The repository contains further information about the versions of libraries used in *neighbo-rs*. They are omitted here for sake of simplicity.

MrBayes and *IQ-TREE* were run sequentially while *neighbo-rs* was executed sequentially and in parallel with 12 threads, because all steps can be trivially parallelized. The pre-computation of the distance distributions can be parallelized over the pairs of sequences, while generating a tree with NJ and evaluating the parsimony score can be done in parallel for multiple trees at once. We go further into the impact of these steps and the effect on runtime in Section 4.4. In all other experiments, we focus on solution quality where runtime has no effect and there are no timeouts.

2.5. Data Visualization

Because all our metrics are normalized to the range $[0, 1]$ (except *PCC* which is $\in [-1, 1]$, but we never observed values < 0), they can be tightly represented in a single plot with their values on the y-axis and the specific metric on the x-axis. We chose *matplotlib's categorical boxen-plots* (also known as letter-value plots) [HKW11] to group the data points per tool for most figures (apart from runtime comparisons) to preserve information about the distribution of the metric values. We use the standard parameters, so $log_2(n) - 3$ levels for the boxen tails and the width of a sub-box represents the percentile. The largest box contains 50% of the data points with the median as a black line dividing this box, the second-largest boxes above and bellow the largest box contain 25% of the data points combined, et cetera. These plots are supplemented with the tables containing other core features of the constant tools from Section 2.4.2 as well as concrete values for e.g. mean and standard deviation throughout the text. To compare runtimes in Section 4.4, we use simple scatter-plots and stacked bar-plots to highlight individual components. Figures also contain the number of MSAs, their difficulty range and optimal parameters where applicable. For the weighted minimization of our metrics we use the following weights: *PCC*: -5.0, *ASDSF*: 5.0, UBR_R: 1.0, UBR_T: 1.0. This is based on the desire to approximate the frequencies of splits more accurately, while not completely disregarding the amount of splits that occur in only the reference or the tools. To show how the metrics change across our parameter grid, we use 3D surface plots with heatmap encoding in Section 3.1.1.

3. Methods

To arrive at a set of phylogenetic trees that approximates a distribution through repeated NJ execution, one has two obvious choices to add perturbation. The first is to change the pair-wise distances before generating a new tree, which we explore in Section 3.1, the second is to add randomness to the decision about which nodes to join in each step, in Section 3.2. In this chapter, we describe multiple variations for both approaches and discuss their strengths and weaknesses. Each section will contain a short preliminary evaluation of the respective strategy on our parameter tuning dataset. For sake of readability in our figures, we use a reduced selection of our metrics described in Section 2.3 by not showing the *Hellinger Distance* and *Simple Distance*, because the *ASDSF* offers enough insight into the mismatch in split frequencies. We will revisit the other two in Chapter 4 for a more in-depth analysis.

3.1. Input Perturbation

As a first step to add perturbation to the input distances, we simply add fixed random noise onto a fraction of the matrix entries (Section 3.1.1), because this is a very easy and fast way to arrive at different trees while controlling the width of the distribution through the amount of noise with parameters. However, this clearly is too simplistic to produce satisfying results, so we then switch to actual likelihood-based distance distributions between the sequences (Section 3.1.2).

3.1.1. Random Noise

Method We define $p, r \in [0, 1]$ to be the perturbation and rate parameters. Let further $\mathcal{D}(0, p)$ be a normal distribution with mean 0 and standard deviation p. We sample a value s from $\mathcal{D}$ and multiply a fraction r of pair-wise distances with e^s. This can be thought of as a percentage change in the distances. While this is very fast and easy to implement, it has no theoretical support and simply presents a (rather basic) heuristic to model uncertainty in the distances. Instead of coming up with a reasonable distribution of trees where trees with higher likelihood are observed more often, this approach tries to cover a wide set of trees. Our goal is that this set is diverse enough to contain many trees that would also be observed in the reference distribution, but not so broad that a large fraction of the trees become unrealistic. Through tuning the noise parameter p of the normal distribution and the fraction r of matrix entries that get perturbed, we can optimize for different metrics. Another parameter for this method is whether to apply the same percentage change to all selected entries or draw a new percentage for each one individually. We decided to use the same noise for all entries because these results were strictly better. Overall, this is a good baseline to compare the other input perturbation strategy against, because it is rather uninformed and shows how simple noise performs against the more elaborated method.

Evaluation Figure 3.1 shows the comparison of this strategy against the short *MrBayes* run and *IQ-TREE* on the parameter tuning set relative to the long *MrBayes* run. The parameters of *neighbo-rs* are optimized by minimizing the weighted sum of the values of the *PCC*, *ASDSF*, fraction of unique splits in the reference USR_R and fraction of unique splits in the respective tool USR_T (see Section 2.5).

This figure already shows the clear weakness of this strategy and what we will continue to address in this thesis: To closely approximate the reference distribution in terms of split frequencies, we need to add a lot of noise to the pair-wise distances. This leads to many of the splits in the reference being hit by *neighbo-rs* (USR_R mean: 0.259, std: 0.257), while simultaneously creating many splits that are unique to our tool (USR_T mean: 0.77, std: 0.154). Figures 3.2a and 3.2b show how the one-sided *USRs* behave across our noise parameter grid and that they are almost strictly inversely correlated. We observe this consistently across all strategies and therefore, the objective becomes to minimize these trade-offs. From figure 3.2c, and based on the optimization results in Figure 3.1, we conclude that choosing higher parameter values improves the important metrics *PCC* and *ASDSF*, so we need to find ways to prevent or filter the occurrence of trees that introduce splits that are not found in the reference.

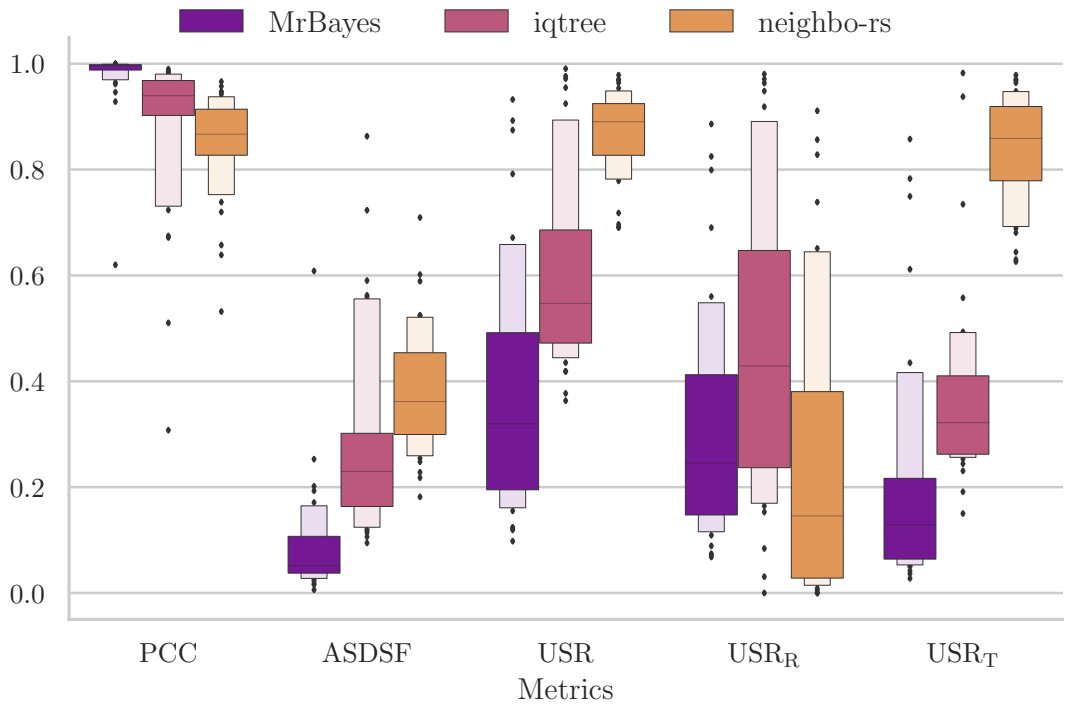
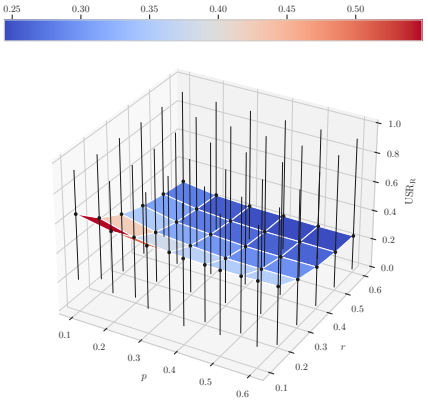
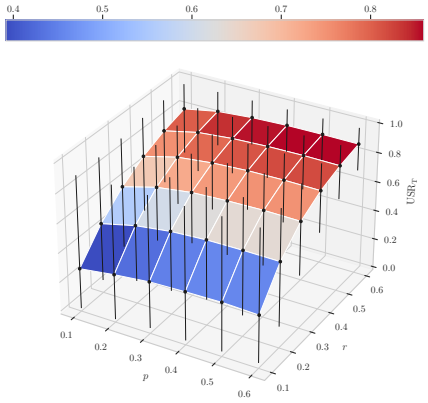


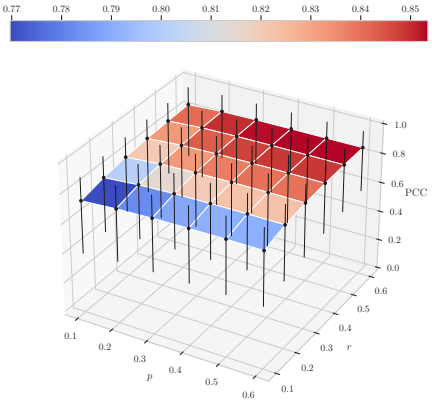
Figure 3.1.: 40 MSAs (combined parameter tuning set), difficulty: 0.05–0.86. $p = 0.5$, $r = 0.4$. The randomized noise strategy produces high USR_T values when optimized to match split frequencies. Its frequency-based metrics also lack behind the other tools.



(a) The USR_R values are naturally lower with higher noise parameters. The error bars (minimum and maximum values) are notably large, indicating that the variance is large between different MSAs.



(b) The USR_T values are almost completely inversely correlated with the USR_R values. Again, the error bars cover a large range.



(c) The PCC values are consistently high from moderately high to very high noise parameters. The ratio r is more important for better PCC values.

Figure 3.2.: 40 MSAs (combined parameter tuning set), difficulty: 0.05–0.86. Surface plots of USR_R, USR_T and PCC values across the noise parameter grid show that high noise is preferable and trade-offs are unavoidable.

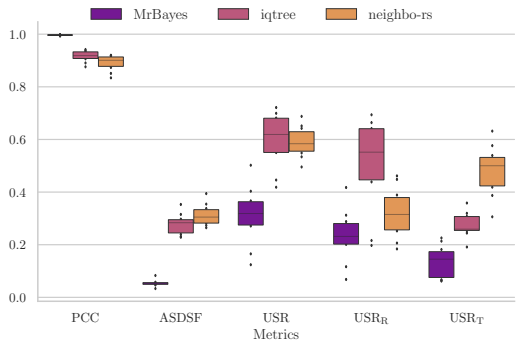
3.1.2. Pair-wise Distance Distributions

Method To get more realistic pair-wise distances between sequences, we generate a distribution of distances for each pair of sequences. To approximate these distributions we use *rejection sampling*, but this requires a good prior estimation on the distribution of distances to get a sufficient number of samples accepted quickly. There is little literature on generalizations of the distribution of phylogenetic distances. We assume they generally follow an Exponential or Gamma distribution, which is at least true for branch lengths in empirical trees [Par16]. Thus, we use a Gamma distribution with an empirically estimated mean from the input data as a proposition mechanism for the rejection sampling process to approximate a distribution of distances for each sequence pair. More formally, let $k \in \mathbb{R}_+$ be the shape parameter and $\overline{d}$ the average pair-wise *Hamming distance* of the input MSA. The scale parameter is $\theta = \frac{\overline{d}}{k} \in \mathbb{R}_+$ and $\Gamma(k, \theta)$ the Gamma distribution. We draw samples until we reach 1000 accepted distances with a burn-in of 10 as a default. The acceptance of a distance is based on a simplified likelihood calculation by Yang [Yan14] for the pair of sequences (a, b) where $L_{a,b}(t) = \sum_{i=1}^{n} \pi(a_i) P_{a_i b_i}(t)$ is the likelihood that t is the distance between a and b under the prior vector π and instantaneous rate matrix $P(t)$. Our evaluations show that this is a solid heuristic to approximate a distribution of distances and works well for most of our MSAs. The number of samples drawn until enough are accepted rarely exceeds 10,000. However, during the final large experimental runs, especially with difficult, empirical MSAs, we sometimes saw the number of required samples reach as much as 1,000,000. The offending MSAs contained many duplicate sequences, while the average pair-wise Hamming distance was relatively high. This means that accurate samples for both, very similar and very different sequences, are very unlikely to be drawn. This occurred infrequently enough to not be a concern for the validity of our results, but needs to be addressed in the future. For example, switching to a uniform distribution if the sampling reaches a certain threshold without enough accepted distances.

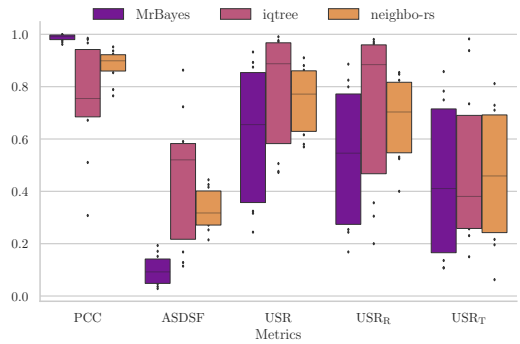
To now compute the trees, we start with the maximum likelihood distances for the first tree and sample a variable fraction r of distances before each new tree as before. One important thing to note here is that this assumes the distances between pairs of sequences to be independent of each other, which they are not as soon as the paths connecting the pairs in the resulting tree have at least one common edge. This is not a problem we can solve with the methods presented in this thesis, as it is outside the scope of this project. We try to mitigate it by filtering the resulting trees by their parsimony score as a proxy for likelihood (see Section 3.3) to discard highly unlikely trees, and distance re-sampling during NJ steps (see Section 3.2.3). We discuss more ideas on how to overcome these dependencies in Section 5.1.1. Another way to reduce the construction of unlikely distance combinations is through limiting the variance of the individual distributions of distances. This means that we approximate the samples we gather with normal distributions and scale their standard deviation by a factor $\alpha \in (0, 1]$ to create a bias towards higher likelihood distances and in extend reduce the expected amount of distinct tree topologies.

Evaluation To properly analyze this strategy, the plots are split by the type of data. For both simulated datasets (Figures 3.3a and 3.3c), the results look promising. *neighbo-rs* consistently hits more splits from the reference than *IQ-TREE* and the number of excess splits USR_T is moderate, leading to both tools having very similar *USR* scores. However, *neighbo-rs* lacks behind the other tools in the frequency metrics *PCC* and

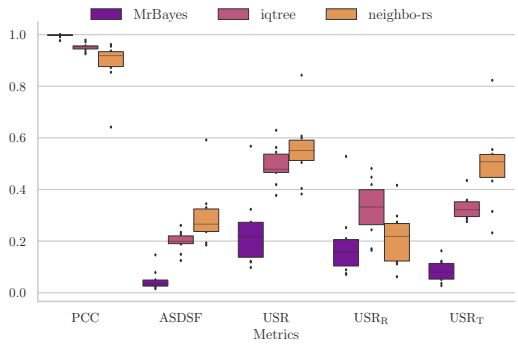
ASDSF, indicating that, while we achieve comparable split coverage, the frequencies do not line up. On the other hand, the empirical datasets (Figures 3.3b and 3.3d) show increased uncertainty in the datasets even though they have comparable difficulty (but also contain generally more MSAs of higher difficulty). Most notable, even the short *MrBayes* run struggles to cover the splits of the reference distribution, especially for the empirical DNA data. Nonetheless, the *ASDSF* and *PCC* remain relatively stable, indicating that the splits in the symmetric difference have low frequencies in both *MrBayes* runs. Further evaluation showed that the variation in performance for *IQ-TREE* and *neighbo-rs* can be attributed to the difficulty of the MSAs in the respective datasets. This will be addressed in more detail in Section 4.1.



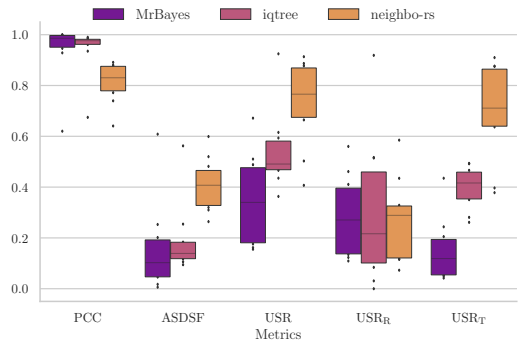
(a) 10 MSAs (simulated DNA parameter tuning set), difficulty: 0.39–0.54. $p = 0.5$, $r = 1.0$. *neighbo-rs* is very comparable in quality to *IQ-TREE* on simulated DNA data. There are slight deficits in the frequency-metrics.



(b) 10 MSAs (empirical DNA parameter tuning set), difficulty: 0.34–0.86. $p = 0.3$, $r = 1.0$. Empirical DNA data shows increased uncertainty for all tools. *USR* is high across all tools. The short *MrBayes* run still has good results for the frequency-metrics. *neighbo-rs* outperforms *IQ-TREE* with more consistent results across the dataset.



(c) 10 MSAs (simulated AA parameter tuning set), difficulty: 0.16–0.78. $p = 0.5$, $r = 0.7$. As with DNA, all tools perform well on simulated AA data, with *neighbo-rs* lacking behind in all metrics besides USR_R.



(d) 10 MSAs (empirical AA parameter tuning set), difficulty: 0.05–0.76. $p = 0.7$, $r = 1.0$. Empirical AA data is the most difficult one for *neighbo-rs*. Distance estimations are inaccurate and lead to significantly worse results.

Figure 3.3.: Metrics for the distance distribution strategy on the parameter tuning sets shows that simulated data are more favorable for *neighbo-rs*.

3.1.3. Comparison

A direct comparison of the two input perturbation strategies favors distance distributions with significantly lower miss-matching splits *USR* (0.66 mean compared to 0.8 for random noise) and a slight advantage for the *ASDSF* (0.323 versus 0.368 on average) and *PCC* (0.878 versus 0.851 on average). We will therefore only consider the distance distribution strategy in the more detailed evaluation in Chapter 4.

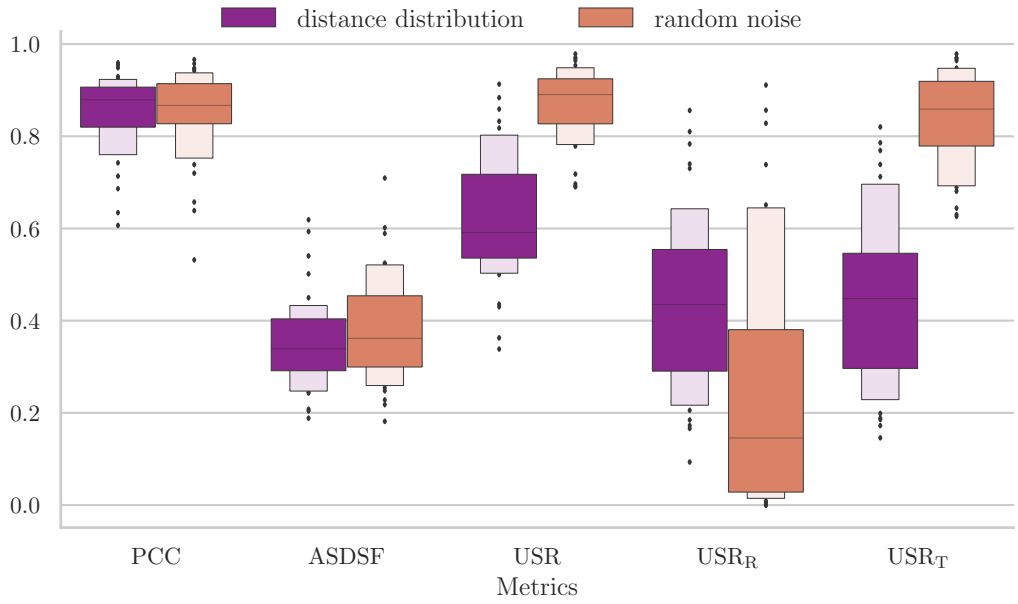


Figure 3.4.: 40 MSAs (combined parameter tuning set), difficulty: 0.05–0.86.
Random noise: $p = 0.5$, $r = 0.4$, Distance distribution $p = 0.4$, $r = 0.9$.
Optimizing across all parameter tuning sets leads to very high USR_T values for the random noise strategies which also leads to worse frequency-metrics compared to using distance distributions.

3.2. Randomized Neighbor Joining

As an alternative to changing the input before constructing the next tree, we can also keep the input fixed and look to add randomness to the NJ algorithm. This is much more likely to produce very suboptimal results, because it breaks the balanced minimum evolution criterion (with respect to the input). We show that changing the selection criterion even slightly is not a good idea, but also that combining this idea with the distance distributions from Section 3.1.2 leads to promising results.

3.2.1. Weighted Selection

Method By changing the selection criterion for the next nodes to join during the NJ algorithm (Algorithm 2.1) to a weighted selection with weight function $w\colon q \to e^{-q},\ q \in Q$ instead of the strict minimum of the Q-Matrix, we hope to get similar trees with small topological changes from preferring pairs that would be joined soon anyway. However, this is not the case. Rather, highly unlikely trees with a largely disjoint set of splits compared to reference distribution are the result. Even further weighting small values in the Q-Matrix through adjusting the weight function to $w\colon q \to e^{-q},\ q \in Q$ did not lead to acceptable results, leading us to believe that joining even slightly suboptimal pairs of nodes have a cascading downstream effect.

Evaluation Figure 3.5 shows just how far off this strategy is and highlights that even small changes to the selection criterion can have a large impact on the constructed trees. Both, *USR* and USR_T, are beyond 0.8 for all 40 parameter tuning MSAs and as a result the mean *ASDSF* is 0.656. This means this method produces widely different trees than the reference. We refrain from further analyzing this approach and deem it hopeless to pursue in the future.

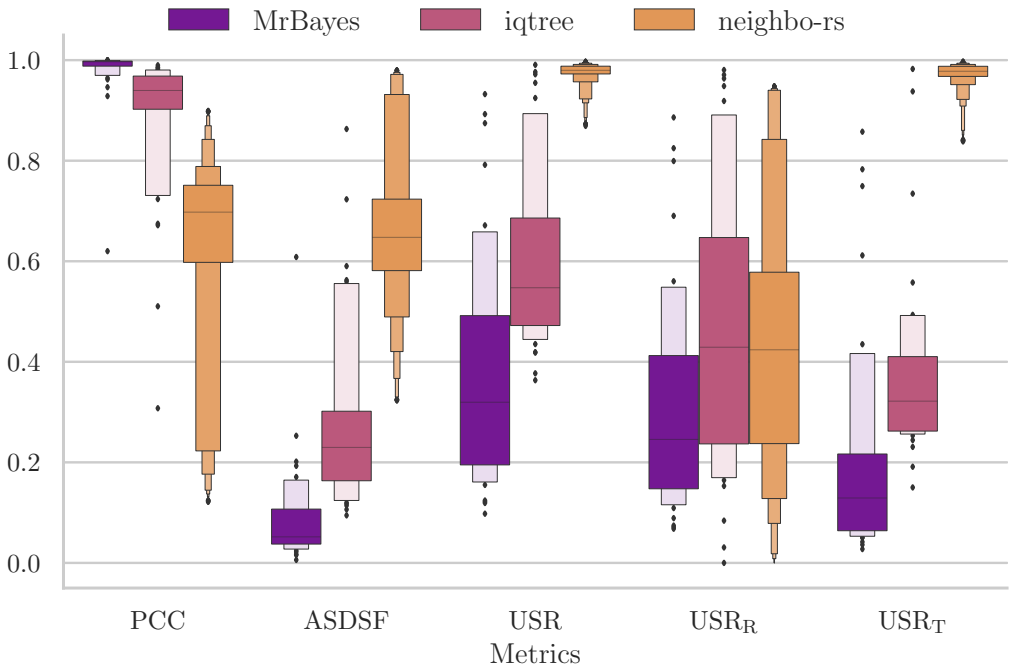


Figure 3.5.: 40 MSAs (combined parameter tuning set), difficulty: 0.05–0.86. $p = 0.0,\ r = 0.0$. Even without any noise applied to the input, very high *USR* values for all MSAs make weighted selection an infeasible strategy to infer reasonable trees.

3.2.2. Minimum Sampling

Method Another alternative to using the strict minimum of the Q-Matrix is to take a fraction f of samples from the matrix and choose the minimum of these samples. This also reduces the runtime by limiting the number of Q-Matrix entries that need to be evaluated. Similar to weighted selection (Section 3.2.1) this method adds noise to the selection process, but should still choose relatively small values from the Q-Matrix. However, after the seeing the results for weighted selection the expectation is that this strategy would suffer from the same problem, at least for small f.

Evaluation We evaluated this strategy with and without noise parameters, meaning that without noise parameters, the method uses the ML distances for each new tree and the only variable is the randomness of selecting the samples from the Q-Matrix. As with weighted selection this strategy suffers from downstream effects of choosing suboptimal pairs, as can be seen in figure 3.6. It is also to note that while this performs better than weighted selection, the parameter optimization chooses high f-values and noise parameters similar to the plain distance distributions from Section 3.1.2, so most of the time the potential runtime improvements from ignoring large parts of the Q-Matrix are void. We saw further deterioration of the results when evaluating with lower percentile values or lower noise parameters. Therefore, we conclude this to be another failed approach not worth considering in the future.

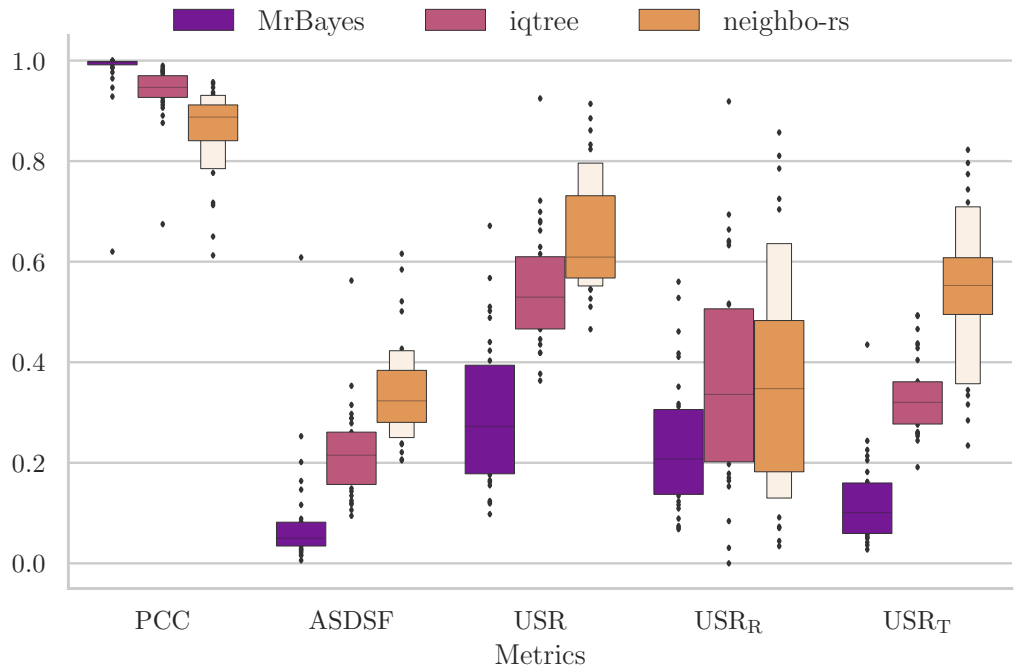


Figure 3.6.: 40 MSAs (combined parameter tuning set), difficulty: 0.05–0.86. $p = 0.5$, $r = 1.0$, $f = 0.75$. The optimal fraction f of Q-Matrix entries is very high. Lowering this fraction worsens the results significantly, showing that the potential benefit of this strategy over plain distance distributions does not apply.

3.2.3. Distance Re-sampling

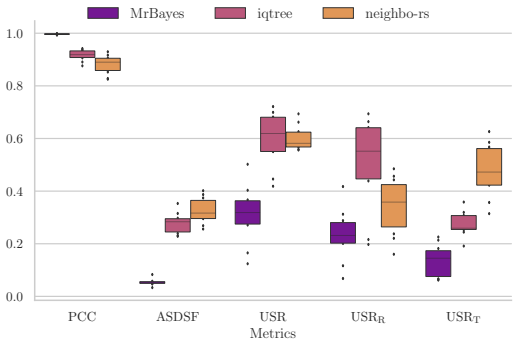
Method Instead of re-sampling the matrix before constructing a new tree, the re-sampling can also be done during execution of the NJ algorithm (Algorithm 2.1). We use the same distributions of distances as in 3.1.2 but now start with the maximum likelihood distances and re-sample a fraction of distances each time two nodes are joined. When two nodes are joined, we fix their distance as the one currently present in the distance matrix and update the distance distribution to the remaining nodes by using samples of their pair-wise distance distributions to be used for the standard NJ update of distances (Algorithm 2.1, Line 14). Afterward, such a distribution is represented by a set of samples, rather than a normal distribution. Ideally, we would want to really combine pairs of distributions in this process, respecting covariance and other dependencies. This, again, is outside the scope of this thesis, but the approach is discussed in Section 5.1.1.

Evaluation Because this shows similar potential to plain distance distributions (Section 3.1.2), we split the analysis by the type of data. It performs significantly better on simulated datasets (figures 3.7c and 3.7a) than empirical data (figures 3.7d and 3.7b) on the optimized metrics *PCC* (0.89 and 0.886 compared to 0.822 and 0.843 for empirical data on average) and *ASDSF* (0.297 and 0.319 compared to 0.4 and 0.373 for empirical data on average) with performance being very poor on empirical AA data where the *USR* is 0.757 on average. As with the other NJ randomization strategies, trying to add small variations to the selection criterion seems to do more harm than good. Another downside to this approach is that updating the distributions adds significant runtime. Therefore, a trade-off has to be made between closely approximating the distributions by drawing many samples and having reasonable runtime, because this adds a lot of work to the innermost loop of the program (updating distances in the NJ algorithm).

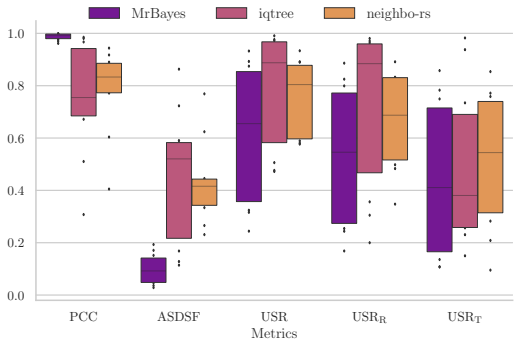
However, we believe that there is potential for this to produce viable results with the aforementioned changes to the updating of distance distributions. This would also eliminate the increased runtime from more distance samples (at the cost of increased runtime from combining distributions). Although beyond the scope of this thesis, this is an interesting strategy to consider when trying to create variations in NJ results.

3.2.4. Comparison

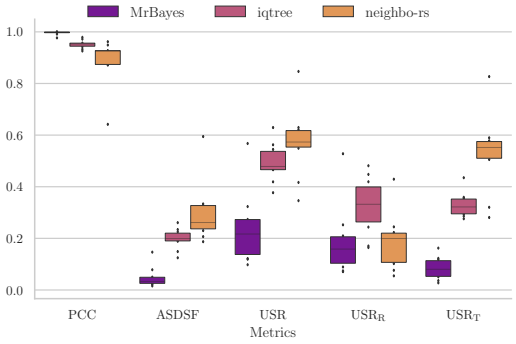
Because all randomization strategies apart from distance re-sampling (Section 3.2.3) have clear negative results, we conclude this section with a short comparison of distance re-sampling and plain distance distributions (Section 3.1.2) in figure 3.8. Looking at the combined datasets shows that distance re-sampling performs worse than distance distributions in all metrics with more negative outliers in total when optimizing for the combined parameter tuning MSAs. Exact values can be found in Table 3.1. Finally, distance re-sampling is approximately 7 times slower than plain distance distributions, while also having an order of magnitude fewer samples per distance (100 compared to 1000 for distance distribution).



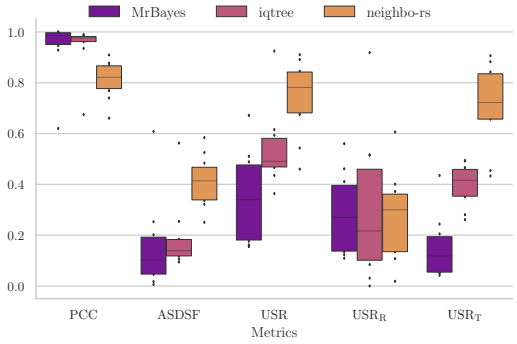
(a) 10 MSAs (simulated DNA parameter tuning set), difficulty: 0.39–0.54. $p = 0.5$, $r = 0.8$. Similar to plain distance distributions, distance re-sampling produces good results on simulated DNA data.



(b) 10 MSAs (empirical DNA parameter tuning set), difficulty: 0.34–0.86. $p = 0.4$, $r = 1.0$. The empirical DNA is difficult to analyze with distance re-sampling. Again, frequency-metrics remain relatively stable for *neighbo-rs* while the *USR* is very high.



(c) 10 MSAs (simulated AA parameter tuning set), difficulty: 0.16–0.78. $p = 0.5$, $r = 0.9$. As with its DNA counterpart, inferring trees for simulated AA MSAs works well. *neighbo-rs* with distance re-sampling misses about as many splits as the short *MrBayes* run (see USR_R), but the frequencies are mismatched by large USR_T values.



(d) 10 MSAs (empirical AA parameter tuning set), difficulty: 0.05–0.76. $p = 0.8$, $r = 1.0$. Distance re-sampling does not work very well for empirical AA data. Especially the frequency-metrics are highly suboptimal compared to the other datasets.

Figure 3.7.: Metrics for the distance re-sampling strategy on the parameter tuning sets show that simulated data is again more favorable for *neighbo-rs*.

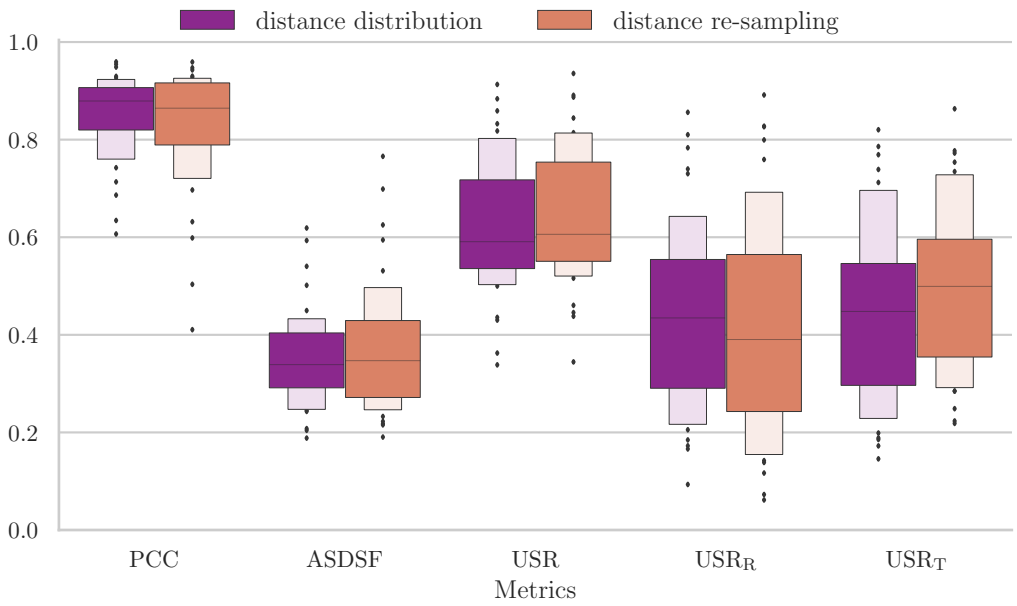


Figure 3.8.: 40 MSAs (combined parameter tuning set), difficulty: 0.05–0.86. Distance distribution: $p = 0.4$, $r = 0.9$, Distance re-sampling: $p = 0.4$, $r = 0.8$. Distance distribution has a slight edge over distance re-sampling with less deviation and better means for all metrics besides USR_R.

Strategy	PCC				ASDSF				USR				USR$_R$				USR$_T$			
	Mean	Std	Min	Max	Mean	Std	Min	Max	Mean	Std	Min	Max	Mean	Std	Min	Max	Mean	Std	Min	Max
Distance distribution	0.878	0.072	0.645	0.962	0.323	0.097	0.184	0.592	0.660	0.151	0.313	0.914	0.319	0.215	0.054	0.862	0.580	0.180	0.206	0.909
Distance re-sampling	0.860	0.087	0.560	0.961	0.347	0.113	0.187	0.669	0.676	0.157	0.345	0.936	0.319	0.230	0.050	0.899	0.607	0.178	0.267	0.922

Table 3.1.: Comparing distance distribution and distance re-sampling on mean, standard deviation, minimum and maximum values for the metrics PCC, $ASDSF$, USR, USR_R and USR_T shows that distance distributions are the preferable strategy.

3.3. Parsimony Filtering

Method Because the previous methods of this chapter do not use any information about the overall likelihood of the generated trees, we end up with an over-proportional number of trees that are highly unlikely, and also fewer trees with high likelihood when we use parameters that increase the amount of noise applied to the distance matrices. To mitigate this effect, and still cover most of the reference distribution, we would ideally want to filter some of the less likely trees. Calculating the parsimony and likelihood scores has the same asymptotic runtime, but likelihood has significantly higher constants due to many floating-point operations and matrix multiplications. Therefore, we propose to use parsimony as a proxy for likelihood in deciding which trees to keep. Let n be the desired number of trees to generate. To have a reasonable comparison we generate kn trees with *neighbo-rs* and then discard $(k-1)n$ of them through filtering. For our experiments we choose $k = 5$. For the filtering we explored three different strategies: *strict maximum* where only the best n trees are kept, *rejection sampling* where the distribution is approximated by drawing samples until we have n accepted trees, as well as *weighted selection* of n trees based on their parsimony scores. *Strict maximum* is superior throughout our evaluation, and therefore we omit figures for the other two here. We show that not only are parsimony scores a good approximation of likelihood relations, but also that this filtering process significantly improves our results.

Evaluation Figure 3.9 shows improvements across all metrics except USR_R compared to distance distributions without parsimony filtering on the combined parameter tuning data set. We assume this to be a direct result of the reference tree distributions containing some trees with lower likelihoods which we filter with this method. However, even other strategies like *weighted selection* filtering did not improve this, leading us to believe that while the unlikely trees we generate contain some of the fringe-splits, others are still missing. We believe missing these splits is not detrimental to the quality of the filtered distribution because we simultaneously significantly improve the *ASDSF* and *PCC*, so the missed splits have very small frequencies in the reference distribution. In fact, even though the 50^{th} percentile of the USR_R lies significantly higher with parsimony filtering, the average is below that without parsimony filtering. Only for empirical DNA MSAs the mean USR_R is higher with parsimony filtering. Exact numbers can be found in Table 3.2. Overall, the improvements are not very significant on average except for the USR_T which shows large drops in magnitude. Considering we started with the intention of reducing this as much as possible while using high noise parameters to cover the reference splits, this strategy already presents a success. We look further into which splits we miss in Section 4.2 by looking at the other frequency metrics.

To evaluate how good parsimony scores are as a proxy for likelihoods, we plot the likelihoods for the tree distributions with boxen-plots in figures 3.10a, 3.10b, 3.10c and 3.10d for one example MSA per parameter tuning dataset. To eliminate effects of minor branch length discrepancies in the distributions, we re-estimated the branch lengths with *raxml-ng* [Koz+19] and then compute the likelihoods with it as well. The blue horizontal line is the ML value computed with *raxml-ng* in search-mode. While *neighbo-rs* lacks behind relative to the other tools in some MSAs, the effect of parsimony filtering is clearly positive. We attribute the discrepancy in likelihoods compared to the

other tools to our distance estimation from Section 3.1.2. Especially for empirical data we believe that the distributions of distances are not accurate enough. We observed substantial improvements in likelihoods for most MSAs, as can be seen throughout the figures. Even in cases where there was little improvement, the filtering removed the "tail-end" of the distributions, eliminating trees with very low likelihood scores.

In terms of runtime, parsimony filtering increases the runtime by a factor of k to compute the additional trees. Computing the parsimony scores further adds $kn * O(sm)$ (see Section 2.2.2, s = #sequences, m = #sites), but can be parallelized by iterating sites (or trees) in parallel, similar to computing the likelihood (see Section 2.2.4). The filtering is in $O(sort(kn))$. Computing the scores dominates the runtime of *neighbo-rs*, but from the observed results we conclude that this is acceptable (see Section 4.4).

Dataset	Strategy	PCC				ASDSF				USR				USR_R				USR_T			
		Mean	Std	Min	Max	Mean	Std	Min	Max	Mean	Std	Min	Max	Mean	Std	Min	Max	Mean	Std	Min	Max
Simulated DNA	neighbo-rs	0.894	0.023	0.848	0.919	0.309	0.036	0.266	0.382	0.594	0.060	0.495	0.684	0.300	0.101	0.092	0.465	0.494	0.108	0.306	0.629
	neighbo-rs parsimony-filtered	0.914	0.027	0.858	0.945	0.275	0.046	0.217	0.369	0.552	0.056	0.441	0.634	0.291	0.115	0.08	0.467	0.431	0.111	0.247	0.567
Empirical DNA	neighbo-rs	0.908	0.045	0.828	0.954	0.291	0.073	0.207	0.402	0.751	0.128	0.520	0.914	0.543	0.277	0.145	0.862	0.607	0.183	0.216	0.816
	neighbo-rs parsimony-filtered	0.914	0.053	0.790	0.965	0.278	0.078	0.182	0.443	0.721	0.160	0.375	0.910	0.553	0.257	0.192	0.855	0.565	0.197	0.259	0.808
Simulated AA	neighbo-rs	0.892	0.090	0.649	0.961	0.291	0.117	0.184	0.592	0.567	0.136	0.404	0.867	0.181	0.096	0.054	0.395	0.524	0.156	0.291	0.855
	neighbo-rs parsimony-filtered	0.905	0.090	0.661	0.972	0.263	0.116	0.149	0.562	0.480	0.158	0.267	0.737	0.177	0.113	0.054	0.451	0.414	0.183	0.160	0.669
Empirical AA	neighbo-rs	0.817	0.074	0.663	0.898	0.402	0.105	0.247	0.587	0.732	0.184	0.281	0.913	0.256	0.150	0.115	0.573	0.702	0.214	0.163	0.909
	neighbo-rs parsimony-filtered	0.825	0.073	0.669	0.904	0.388	0.107	0.227	0.580	0.689	0.199	0.258	0.882	0.254	0.150	0.092	0.576	0.641	0.253	0.065	0.876
Combined	neighbo-rs	0.878	0.071	0.649	0.961	0.323	0.097	0.184	0.592	0.661	0.153	0.281	0.914	0.320	0.215	0.054	0.862	0.582	0.182	0.163	0.909
	neighbo-rs parsimony-filtered	0.890	0.073	0.661	0.972	0.301	0.101	0.149	0.580	0.610	0.177	0.258	0.910	0.319	0.216	0.054	0.855	0.513	0.208	0.065	0.876

Table 3.2.: Comparing distance distribution with and without parsimony filtering on mean, standard deviation, minimum and maximum values for the metrics *PCC*, *ASDSF*, *USR*, USR_R and USR_T grouped by the parameter tuning sets.

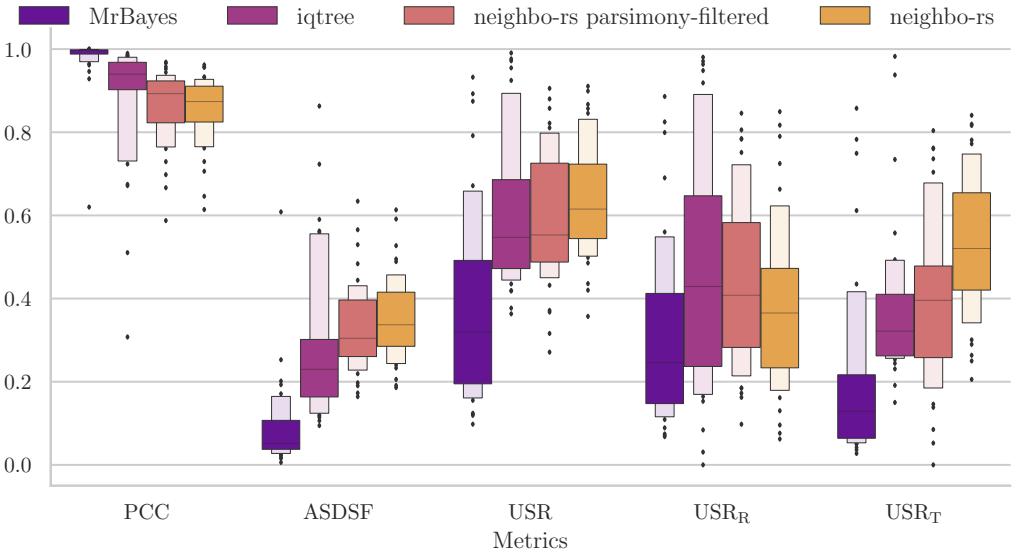
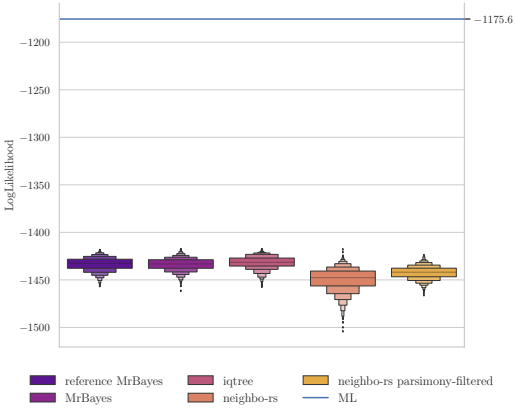
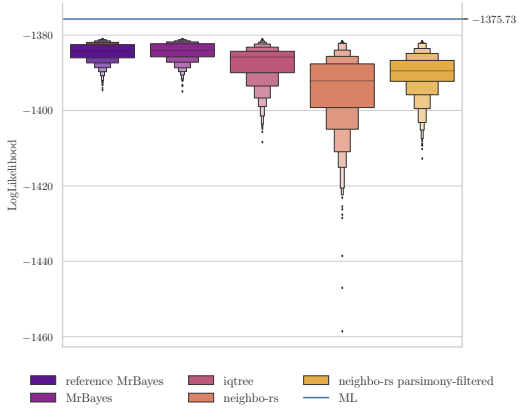


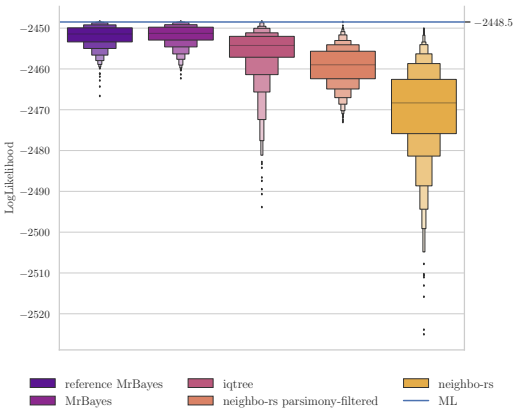
Figure 3.9.: 40 MSAs (combined parameter tuning set), difficulty: 0.05–0.86. $p = 0.5$, $r = 0.8$. Parsimony filtering significantly reduces our original problem of high USR_T values, while also improving the frequency-metrics.



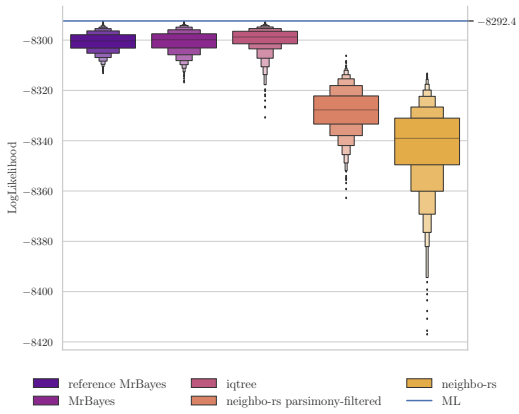
(a) Simulated DNA MSA `dataset_003`, difficulty: 0.54



(b) Empirical DNA MSA `alignment_15827_1`, difficulty: 0.22



(c) Simulated AA MSA `dataset_001`, difficulty: 0.26



(d) Empirical AA MSA `alignment_18551_1`, difficulty: 0.12

Figure 3.10.: Across all data types parsimony filtering significantly improves the likelihoods of the trees generated with *neighbo-rs* by eliminating highly unlikely trees.

4. Detailed Evaluation

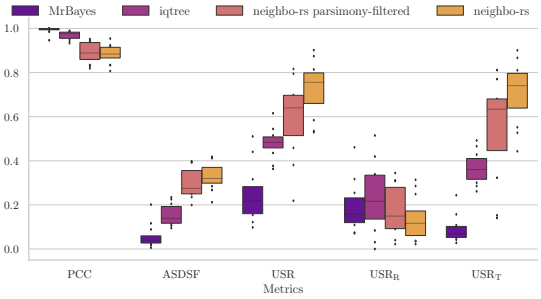
In this chapter we present a more detailed analysis of the tree distributions provided by distance distributions with parsimony filtering from Section 3.3, our best *neighbo-rs* configuration. We look at the parameter tuning dataset split by difficulty ranges in Section 4.1 to assess whether *neighbo-rs* provides consistent results for more difficult MSAs. In Section 4.2 we show the best possible results by optimizing the noise parameters for each MSA individually. Section 4.3 evaluates how the optimal parameters determined for the parameter tuning sets perform for the full datasets. Finally, Section 4.4 compares the running time of the evaluated tools, as well as the impact of distance estimations and parsimony filtering on *neighbo-rs*.

One more thing to look at is the quality of the reference *MrBayes* distribution. We will not address this in great detail in this thesis, because more data would be required to make definitive claims. However, in Figure A.1, we show how the reference run compares against the other tools. We look at the minimum, mean, maximum *RF-distance* as well as the USR_T of the trees from the respective distributions to the "true" tree generated by *SimPhy* for the simulated parameter tuning set and full dataset. Overall, these all are very similar, indicating that all tools perform well, and our other metrics do a good job at highlighting the small variations of the distributions.

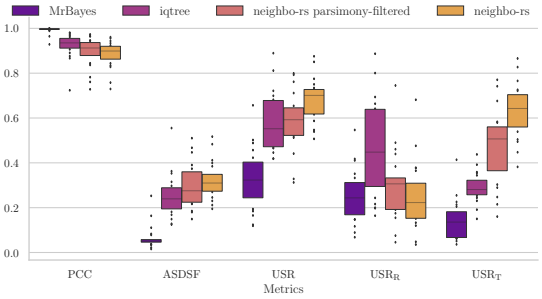
4.1. Performance by Difficulty

The per-dataset evaluations throughout Chapter 3 showed large differences between data types (AA versus DNA, simulated versus empirical). Because all datasets (and therefore the parameter tuning sets as well) contain different distributions of difficulties, we are interested in evaluating the effect of difficulty in our results. Throughout this section we will refer to several classes of difficulty. These are *easy* (difficulty $\in [0.0, 0.3]$), *medium* (difficulty $\in [0.3, 0.7]$) and *hard* (difficulty $\in [0.7, 1.0]$). We expect that *neighbo-rs parsimony-filtered* performs best on easy datasets because there should be fewer plausible trees and therefore the distances between sequences are expected to have a clear signal. Figures 4.1a and 4.1b show that *neighbo-rs parsimony-filtered* performs very consistent throughout easy and medium MSAs, but contrary to our expectations lacks behind *IQ-TREE* in the frequency-metrics *PCC* (0.895 compared to 0.968 for *IQ-TREE* on average for easy MSAs and 0.899 to 0.924 for medium MSAs) and *ASDSF* (0.289 compared to 0.15 for *IQ-TREE* on average for easy MSAs and 0.29 to 0.25 for medium MSAs). However, already for medium MSAs *IQ-TREE* begins to struggle with hitting the reference splits reliably and for hard MSAs (Figure 4.1c) *IQ-TREE* is unable to find topologically similar trees to the reference at all, reaching 90% splits in the symmetric difference on average. In fact, even the short *MrBayes* run deteriorates on hard MSAs, although its frequency-based metric values remain relatively stable, indicating that most splits in the symmetric difference occur very infrequently. This means both *MrBayes* runs generally agree on the tree topologies. Again, we see the positive effect of parsimony filtering, as the *PCC* (mean: 0.853) and *ASDSF* (mean:

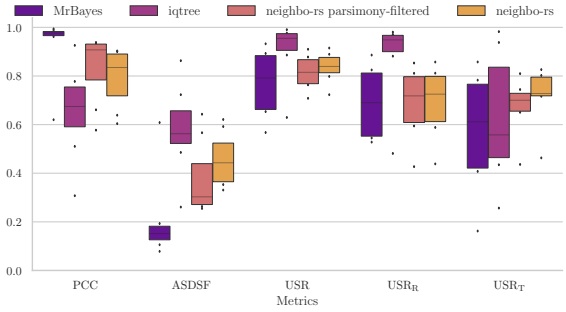
0.356) only slightly worsen compared to easy and medium MSAs. We do not know the cause for this positive outcome, but in trying to understand it in the future, we hope to improve the results for easy and medium MSAs, as well as make *neighbo-rs* a valid alternative for fast inference of phylogenetic tree distributions on hard MSAs.



(a) 13 MSAs (mostly AA), difficulty: 0.05–0.3. $p = 0.8$, $r = 0.9$. On easy MSAs *neighbo-rs* is able to consistently have low USR_R values, but very high USR_T values inhibit good frequency-metrics. Parsimony filtering is not able to improve this compared to the other tools.



(b) 21 MSAs, difficulty: 0.3–0.57. $p = 0.6$, $r = 1.0$. *neighbo-rs* results for medium MSAs improve slightly compared to easy ones, while especially *IQ-TREE* starts to get worse.



(c) 7 MSAs (mostly DNA), difficulty: 0.76–0.86. $p = 0.4$, $r = 1.0$. On hard MSAs *neighbo-rs* performs worse than on easier MSAs, but the difference is less than for the other tools, especially *IQ-TREE*.

Figure 4.1.: Metrics for distance distribution strategy with parsimony filtering grouped by difficulty

4.2. Individual Optimization

So far we optimized *neighbo-rs* parameters for the whole dataset, because in practice one would want to have one run with fixed parameters instead of many runs. However, optimal parameters do not only depend on the type of data, but also on difficulty (see Section 4.1) and possibly other factors, so to gain insight about the best possible results, we compare the best parameters for each MSA individually to those optimized for the whole combined parameter tuning set. Unsurprisingly, we see slight improvements to *PCC*, *ASDSF* and USR_R, and increases in *USR* and USR_T in Figure 4.2. But unless

there is a way to limit the amount of reasonable parameter choices for an MSA through quick preprocessing in the future, we suggest to stick with the optimized parameters for the whole dataset, as the increased runtime from evaluating large parts of the noise parameter grid outweighs the benefits of slightly better results.

We can now also analyze the other frequency-based metrics we overlooked so far: *Simple* and *Hellinger distance*. In Section 2.3 we describe how the different distance functions weight the frequencies of the mismatching splits differently. It is important to note that we have not yet fully analyzed how quickly each of the metrics scales and therefore the conclusions we draw here should be carefully verified once their exact relationship is established. In Figure 4.3 we see that the difference in *Hellinger distance* and *Simple distance* between the tools are very similar to those of the *ASDSF*. However, the differences are slightly larger for these new metrics. Larger differences in the *Hellinger distance* indicate that *neighbo-rs* has more large mismatches for splits that are infrequent. This means that the high *usrt* values we have seen throughout our results for *neighbo-rs* come from infrequent splits, which is a good sign. However, large differences in the *Simple distance* mean that the mismatches across all splits are higher for *neighbo-rs* than for the other tools.

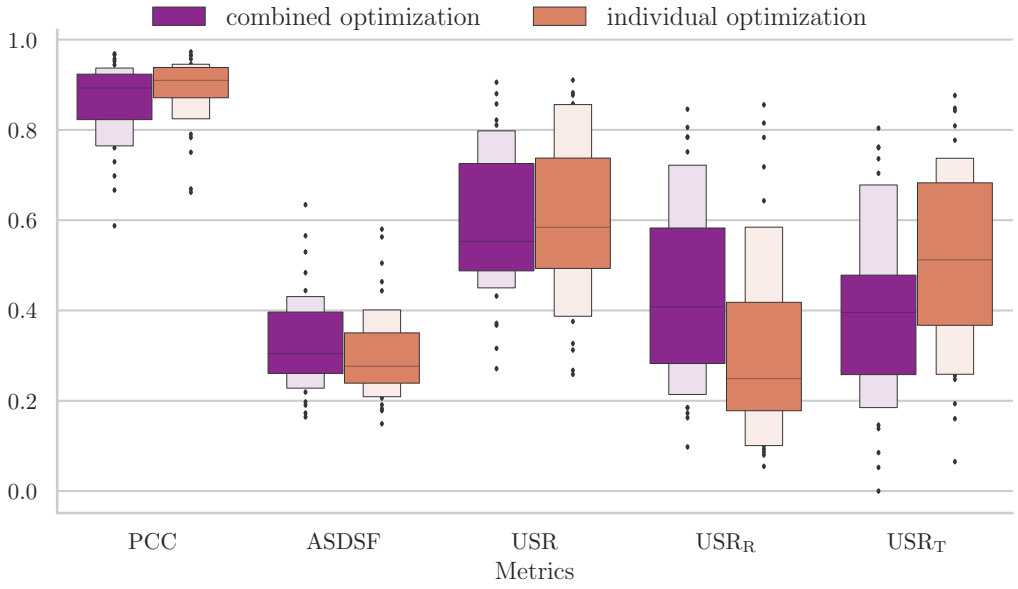


Figure 4.2.: 40 MSAs (combined parameter tuning set), difficulty: 0.05–0.86. Individual optimization slightly improves most metrics, but because we have not found a pattern regarding the choice of ideal parameters it is not realistic for large and diverse sets of MSAs.

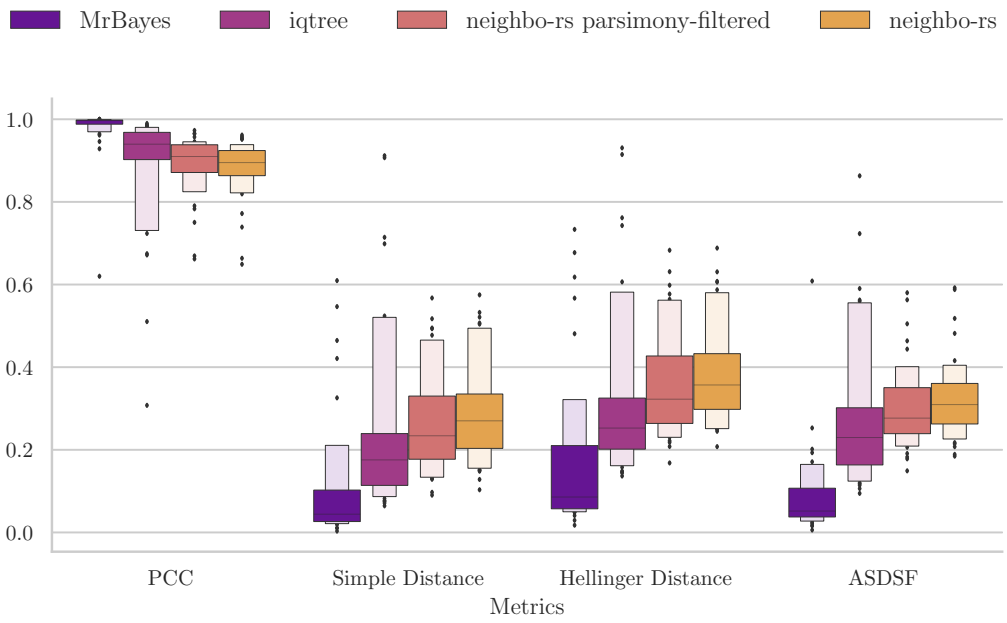
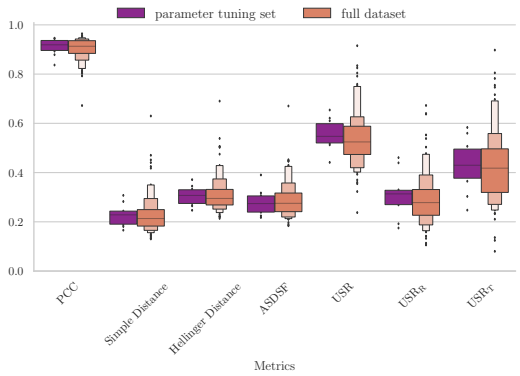


Figure 4.3.: 40 MSAs (combined parameter tuning set), difficulty: 0.05–0.86.
While the exact relation of the frequency-metrics needs to be verified in the future, the high *Hellinger distance* and low *Simple distance* indicate that mismatching frequencies have low absolute values.

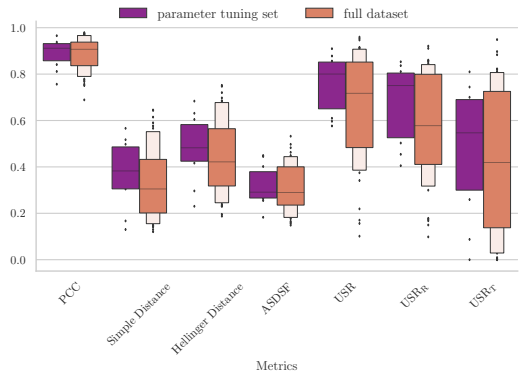
4.3. Full Dataset Evaluation

After showing in Section 4.2 that optimizing the noise parameters for the whole parameter tuning set is preferable to individual optimization, we now also want to evaluate whether these parameters produce acceptable results on the set of all 324 MSAs. Here we again split the data by type and look at different classes of data with the tuned parameters determined previously. The results can be seen in Figures 4.4b, 4.4a, 4.4d and 4.4c. Throughout all datasets we see very good results where the largest boxes (50^{th} percentile) align very well between the parameter tuning set and the full dataset. However, the proportion of outliers and the standard deviations are very large, indicating that more information or a finer division of data types is needed. The high USR and USR_T values for empirical data unfortunately remain so for most of the MSAs of the full dataset and the 50^{th} percentile boxes extend further than for the parameter tuning sets. Because these datasets are on the more difficult side of the spectrum, we assume that the reference distributions are very spread out and hitting all the plausible trees they identify may be a rather hopeless endeavor.

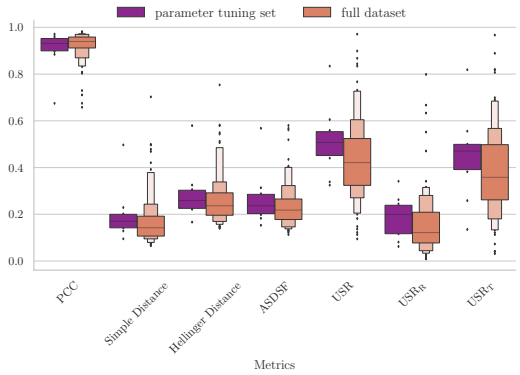
One potentially interesting next step beyond this thesis would be to look into ways to use difficulty information for better grouping and parameter optimization. To motivate this we included mean, standard deviation, min and max values for our key metrics in Table 4.1 grouped by difficulty. We also conclude that our method works better for simulated data which is much more “behaved” compared to empirical data, leading to easier distance estimations. We hope to improve the distance estimation in the future to shrink this gap between simulated and empirical data.



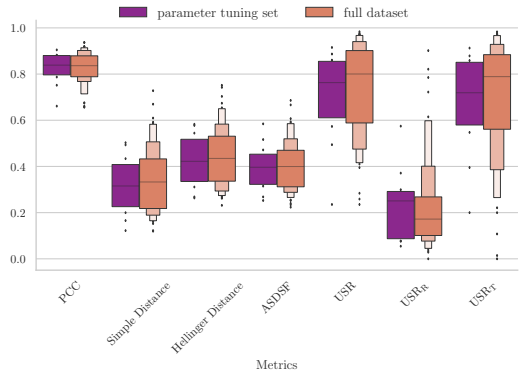
(a) 10 MSAs (simulated DNA parameter tuning set), difficulty: 0.39–0.54.
100 MSAs (simulated DNA set), difficulty: 0.33–0.84.
$p = 0.6,\ r = 1.0$



(b) 10 MSAs (empirical DNA parameter tuning set), difficulty: 0.34–0.86.
46 MSAs (empirical DNA set), difficulty: 0.22–0.87.
$p = 0.4,\ r = 1.0$



(c) 10 MSAs (simulated AA parameter tuning set), difficulty: 0.16–0.78.
100 MSAs (simulated AA set), difficulty: 0.11–0.78.
$p = 0.6,\ r = 0.8$



(d) 10 MSAs (empirical AA parameter tuning set), difficulty: 0.05–0.76.
79 MSAs (empirical AA set), difficulty: 0.0–0.83.
$p = 0.9,\ r = 0.9$

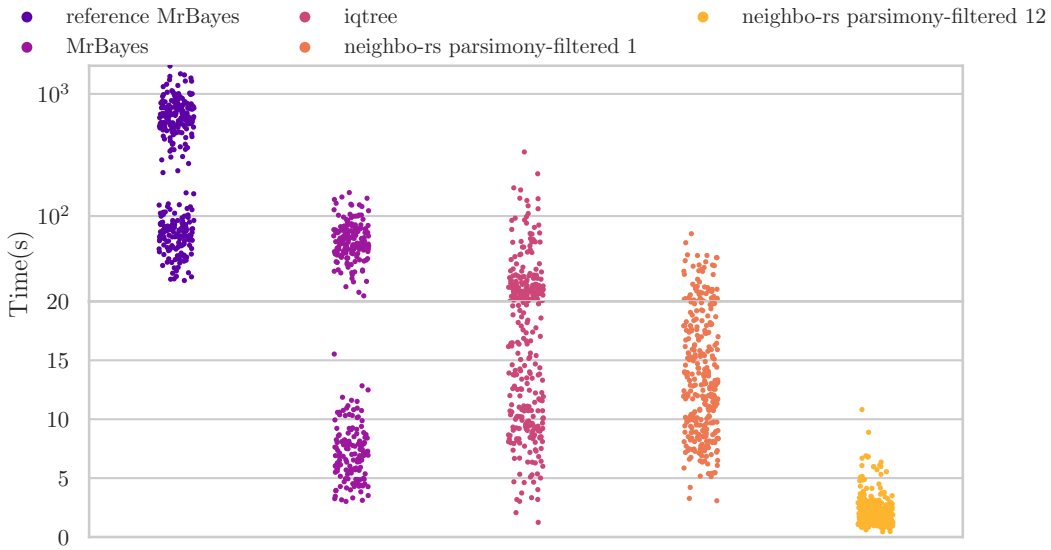
Figure 4.4.: Comparing the metrics for the parameter tuning sets with those of the full datasets when using the optimal parameters for the parameter tuning sets shows that the chosen parameters work well for most MSAs.

Difficulty	Tool	PCC				ASDSF				USR				USR$_R$				USR$_T$			
		Mean	Std	Min	Max	Mean	Std	Min	Max	Mean	Std	Min	Max	Mean	Std	Min	Max	Mean	Std	Min	Max
Easy (0.0-0.3)	MrBayes	0.993	0.021	0.828	0.999	0.047	0.056	0.005	0.390	0.205	0.121	0.020	0.553	0.149	0.103	0.006	0.471	0.084	0.065	0.006	0.330
	IQ-TREE	0.962	0.026	0.822	0.992	0.165	0.055	0.073	0.399	0.466	0.066	0.281	0.664	0.181	0.129	0.0	0.574	0.382	0.072	0.236	0.574
	neighbo-rs	0.900	0.058	0.765	0.981	0.275	0.101	0.113	0.514	0.524	0.206	0.094	0.958	0.148	0.112	0.0	0.785	0.464	0.244	0.0	0.958
Medium (0.3-0.7)	MrBayes	0.989	0.026	0.785	0.999	0.069	0.066	0.014	0.432	0.300	0.139	0.046	0.738	0.226	0.124	0.020	0.667	0.135	0.087	0.0	0.514
	IQ-TREE	0.934	0.044	0.723	0.982	0.231	0.078	0.117	0.555	0.565	0.111	0.355	0.889	0.423	0.184	0.029	0.886	0.329	0.088	0.150	0.677
	neighbo-rs	0.890	0.064	0.657	0.979	0.305	0.102	0.121	0.685	0.551	0.186	0.101	0.983	0.269	0.147	0.029	0.836	0.438	0.235	0.0	0.982
Hard (0.7-1.0)	MrBayes	0.951	0.069	0.620	0.996	0.186	0.111	0.057	0.608	0.765	0.140	0.467	0.963	0.684	0.162	0.391	0.934	0.578	0.224	0.162	0.923
	IQ-TREE	0.720	0.155	0.307	0.954	0.523	0.152	0.202	0.863	0.905	0.101	0.629	0.997	0.886	0.130	0.478	0.994	0.598	0.284	0.171	0.995
	neighbo-rs	0.844	0.105	0.656	0.978	0.380	0.151	0.148	0.670	0.850	0.083	0.649	0.970	0.692	0.170	0.341	0.920	0.755	0.134	0.396	0.967

Table 4.1.: Key metric values for all tools on combined data grouped by difficulty. *neighbo-rs* with parsimony filtering is abbreviated as *neighbo-rs* to preserve space.

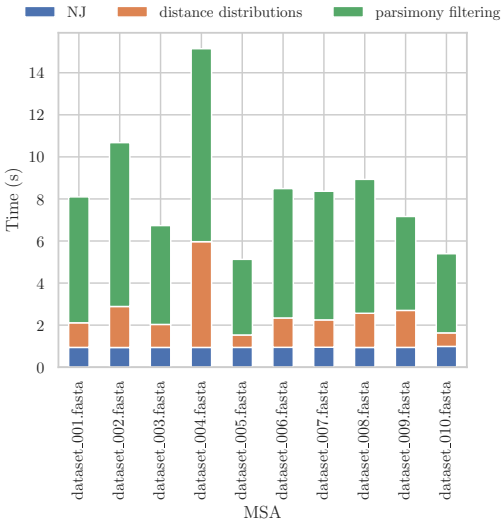
4.4. Runtime Analysis

Runtime is also a very important criterion in phylogenetic analysis. In Figure 4.5 we compare the runtime of all evaluated tools against each other. Starting at 20 seconds, the y-axis is scaled logarithmically to more compactly represent long runtimes, but also preserve details about shorter ones. The gap between the clusters (most notably for *MrBayes*) comes from AA MSAs taking substantially longer than their DNA counterparts. We observed no significant difference between empirical and simulated data that cannot be explained by the different sizes of the MSAs. The runtime of *neighbo-rs* is shown for 1 and 12 threads (hyper-threading) and includes the pre- and post-processing time for the distance distributions as well as parsimony filtering, which make up a large fraction of the runtime, especially when run sequentially. As a result, the runtime with 1 thread is very similar to the other tools. Their contribution can be seen in the detailed runtime evaluation for the simulated parameter tuning set in Figures 4.6a, 4.6b, 4.6c and 4.6d. These figures highlight the importance of utilizing the straightforward parallelization options for the computation of the distance distributions (parallelized over the individual distribution), the parsimony scores (here parallelized over the trees instead of the sites to have a more fair comparison between sequential and parallel execution) and generating the trees with NJ (again parallelized over the trees). The mean relative speedup from running *neighbo-rs* in parallel with 12 threads is 7.413 (Efficiency: 0.617). Because we mainly focused on quality and correctness in our implementation, the runtime still has much room for improvement. We discuss several options in Section 5.1.

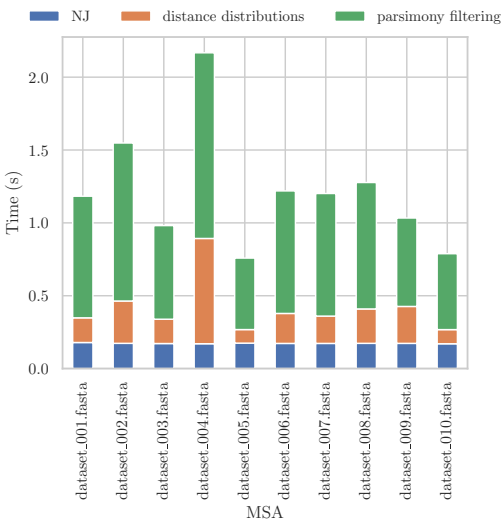


	reference MrBayes	MrBayes	IQ-TREE	neighbo-rs parsimony-filtered 1	neighbo-rs parsimony-filtered 12
mean (s)	414.33	39.67	26.11	15.14	2.19
std (s)	358.76	34.34	31.97	9.0	1.32

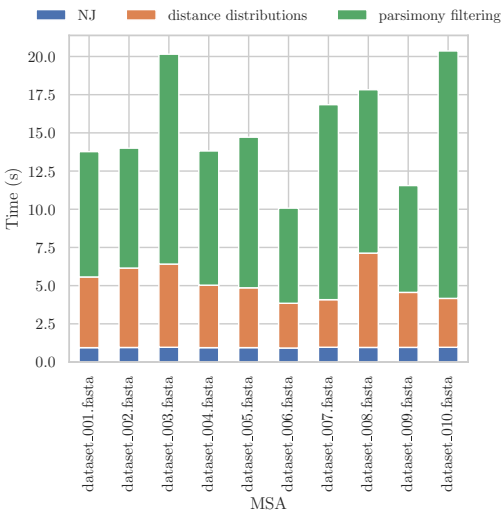
Figure 4.5.: 324 MSAs (full dataset), difficulty: 0.0–0.87.
In terms of absolute runtimes for the full dataset *neighbo-rs* is the fastest on average, even if the compute-intensive pre- and post-processing steps are done sequentially.



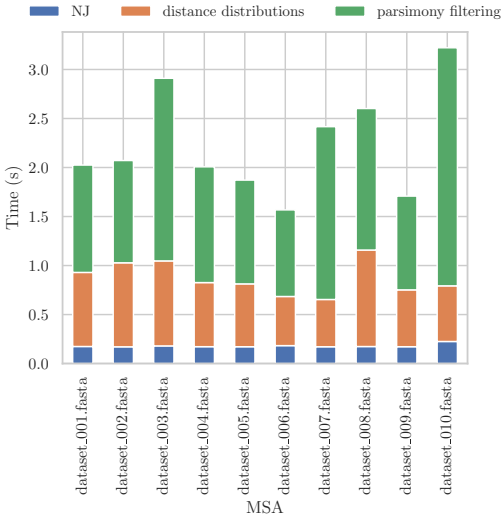
(a) 10 MSAs (simulated DNA parameter tuning set), difficulty: 0.39–0.54. 1 Thread



(b) 10 MSAs (simulated DNA parameter tuning set), difficulty: 0.39–0.54. 12 Threads



(c) 10 MSAs (simulated AA parameter tuning set), difficulty: 0.16–0.78. 1 Thread



(d) 10 MSAs (simulated AA parameter tuning set), difficulty: 0.16–0.78. 12 Threads

Figure 4.6.: The sequential and parallel runtimes on simulated DNA and AA data are split into the time required for pre-computing the distance distribution, generate the trees with NJ and parsimony filtering. The pre- and post-processing steps add significant runtime to *neighbo-rs*, but are required to get good results.

5. Conclusion

In this thesis we explored and evaluated multiple ideas on how to approximate phylogenetic tree distributions with a distanced-based method, Neighbor Joining. We put a special focus on simple strategies that perturb the input distances or add randomness to the NJ algorithm itself. It quickly became apparent that choosing the amount of noise to put on the pair-wise distances from a random normal-distributed scaling factor (Section 3.1.1) was not a good enough heuristic, so we switched to pair-wise distributions of distances supported by likelihood (Section 3.1.2). However, this only improved the results slightly and still left us with many unlikely trees, as well as split frequencies not matching those of the reference tree distribution very well. We mostly attribute this to dependencies in the pair-wise distances that share paths in the constructed tree, which we cannot account for and fix currently. Our ideas for adding randomness to the NJ algorithm itself (Section 3.2) did not hold up to the input perturbation strategies. Messing with the minimum evolution criterion of NJ had a more detrimental downstream effect than we expected, and if such strategies are to be explored in the future, there will be the need for mechanism to keep the chaos at bay. Overall, the results of this thesis show that more information is needed to approximate distributions inferred under MCMC methods, but because NJ is so fast to run, especially for small MSAs, there is a lot of breathing room to add pre- and post-processing steps. We see post-processing with parsimony filtering (Section 3.3) as the most promising result of this thesis. This method allows for a much broader exploration of the tree space – that can then be cleaned up afterward by discarding non-parsimonious trees – which is much faster than computing likelihoods and also serves as a great proxy for it. We assume that the drop in USR_T we saw as a result of parsimony filtering (see Figure 3.9) comes from eliminating unlikely trees at the fringes of the pair-wise distance distributions. Another positive result of this thesis was the increased effectiveness of our best strategy on difficult MSAs compared to the other tools (Section 4.1). In conclusion, approximating phylogenetic tree distributions with distance-based methods has by no means be solved in this thesis. However, we provide clear avenues of further approaches to explore and did our best to address the weaknesses of the presented strategies, as well as filter out methods that should not be given further thought.

5.1. Future Work

There are many things we could not address in this thesis due to its goal of keeping the methods simple and fast. In this section we list ideas we feel should be considered when doing further research on this topic.

5.1.1. Distance Distribution

First and foremost, the current approach to estimating the pair-wise distance distributions has some shortcomings that need to be addressed. Even though our preliminary evaluation on the parameter tuning datasets led us to believe that rejection sampling under likelihood scores works well, evaluating the full dataset presented some issues. Taking the priors from a Gamma distribution works well in most cases, but sometimes datasets are very spread out with distances clustered at either end, leading to very few accepted samples. One possible solution is to roughly estimate the distance of a pair of sequences based on their Hamming distance and then do a more restricted search around this suggested value. Here it might be beneficial to adjust the proposition mechanism to not draw from a prior distribution, but be more of a guided walk in the proximity of that value. This alleviates potential assumptions about the distribution of distances in the tree as a whole, but also between a single pair of sequences, which might not hold for especially empirical MSAs in practice. Another consideration is the interdependency of pair-wise distances. Ideally one would want to use the covariance between pairs of pair-wise distances such that the joined likelihood of distances sampled from both distribution remains high. One possible option would be to estimate these covariances from the (or many in the case of difficult MSAs) tree constructed from the ML distances and then adjust the distributions accordingly for the other trees. Another way would build upon our distance re-sampling strategy, but instead of the clunky mixing of distributions we use currently, have *profile-alignments* of the inner nodes from which new distance distributions are estimated. All these proposals would most likely be very time-consuming, but because the quality of NJ is so very dependent on accurate distances, it might be worth the effort.

5.1.2. Metrics

While the *ASDSF* is a commonly used metric in phylogenetic settings, the other frequency-based metrics are not. Therefore, we do not know how well the *Simple distance Hellinger distance* and *PCC* are suited to make comparisons in quality between split frequency distributions. Especially for the *PCC* it would be interesting to look into defining quality thresholds, because the definition of a "good" *PCC* value varies significantly for different applications. Additionally, further metrics for evaluating the topological similarity between trees of the distributions would be of great value. The split-based and frequency-based metrics are great for getting a rough idea of how the trees match overall, but there is a lack of information about the exact trees of the distributions. However, it is unclear to us how such a metric would look like and whether it could be computed efficiently.

5.1.3. Parsimony Filtering

Parsimony filtering is great at filtering unlikely trees and has a very positive impact on our results. However, in its current implementation it contributes a substantial amount of time to *neighbo-rs*, even with parallelization. While we haven't fully optimized this step in our implementation, simple parallelization is not enough to get reasonable runtimes. One potential solution could be to reuse results for common subtrees by

hashing them and storing their parsimony scores under this hash. Because we can access our internal tree representation while computing the parsimony scores, these hashes and scores can already be computed during NJ.

A future application of parsimony filtering could be the replacement of likelihood scores in other applications, like MCMC and bootstrap methods. Especially if exact likelihoods are not required, this could present a great opportunity to cut runtime. However, it is unclear whether the trade-off between quality and speed is worth it for these applications, and so we would advise careful experimental evaluation to judge the impact.

5.1.4. Difficult Data

A big problem with our current approach is deciding on the ideal noise parameters for a specific MSA (or even a set of MSAs). It would be of great value to have a (possibly machine learning based) predictor that can suggest a good parameter configuration based on some features of the alignment, like difficulty and type of data. We actively refrain from making statements about an "ideal" noise amount, because our limited amount of data is not suited to this. All we can do is optimizing the parameters for specific metrics on our parameter tuning set and judge the impact on the full data set. While this worked well enough for the data we used, it may not uphold in the general case.

On the note of difficulty, we would also like to explore options to use the methods implemented in *neighbo-rs* to generate starting trees for more sophisticated phylogenetic inference tools. For example, these trees could be used as starting trees for an ML tree search or be refined with local search based on NNI or SPR moves.

Bibliography

[BS20] Bastien Boussau and Celine Scornavacca. “Reconciling gene trees with species trees”. In: *Phylogenetics in the genomic era* (2020), pp. 3–2.

[CT05] Benny Chor and Tamir Tuller. “Maximum likelihood of evolutionary trees is hard”. In: *Annual International Conference on Research in Computational Molecular Biology*. Springer. 2005, pp. 296–310.

[DJS86] William HE Day, David S Johnson, and David Sankoff. “The computational complexity of inferring rooted phylogenies by parsimony”. In: *Mathematical biosciences* Volume 81 (1986), pp. 33–42.

[Fel73] Joseph Felsenstein. “Maximum likelihood and minimum-steps methods for estimating evolutionary trees from data on discrete characters”. In: *Systematic Biology* Volume 22 (1973), pp. 240–249.

[Fel81] Joseph Felsenstein. “Evolutionary trees from DNA sequences: a maximum likelihood approach”. In: *Journal of molecular evolution* Volume 17 (1981), pp. 368–376.

[Fit77] Walter M Fitch. “On the problem of discovering the most parsimonious tree”. In: *The American Naturalist* Volume 111 (1977), pp. 223–257.

[FSS23] Alberto Fernández, Natàlia Segura-Alabart, and Francesc Serratosa. “The MultiFurcating Neighbor-Joining Algorithm for Reconstructing Polytomic Phylogenetic Trees”. In: *Journal of Molecular Evolution* (2023), pp. 1–7.

[FY09] William Fletcher and Ziheng Yang. “INDELible: a flexible simulator of biological sequence evolution”. In: *Molecular biology and evolution* Volume 26 (2009), pp. 1879–1888.

[Gas97] Olivier Gascuel. “BIONJ: an improved version of the NJ algorithm based on a simple model of sequence data.” In: *Molecular biology and evolution* Volume 14 (1997), pp. 685–695.

[Hel09] Ernst Hellinger. “Neue begründung der theorie quadratischer formen von unendlichvielen veränderlichen.” In: *Journal für die reine und angewandte Mathematik* Volume 1909 (1909), pp. 210–271.

[HHBS22] Julia Haag, Dimitri Höhler, Ben Bettisworth, and Alexandros Stamatakis. “From Easy to Hopeless—Predicting the Difficulty of Phylogenetic Analyses”. In: *Molecular Biology and Evolution* Volume 39 (2022), msac254.

[HKW11] Heike Hofmann, Karen Kafadar, and Hadley Wickham. *Letter-value plots: Boxplots for large data*. had.co.nz, 2011.

[Koz+19] Alexey M Kozlov, Diego Darriba, Tomáš Flouri, Benoit Morel, and Alexandros Stamatakis. “RAxML-NG: a fast, scalable and user-friendly tool for maximum likelihood phylogenetic inference”. In: *Bioinformatics* Volume 35 (2019), pp. 4453–4455.

[Lak+08] Clemens Lakner, Paul Van Der Mark, John P Huelsenbeck, Bret Larget, and Fredrik Ronquist. "Efficiency of Markov chain Monte Carlo tree proposals in Bayesian phylogenetics". In: *Systematic biology* Volume 57 (2008), pp. 86–103.

[LDG15] Vincent Lefort, Richard Desper, and Olivier Gascuel. "FastME 2.0: a comprehensive, accurate, and fast distance-based phylogeny inference program". In: *Molecular biology and evolution* Volume 32 (2015), pp. 2798–2800.

[LLG08] Si Quang Le, Nicolas Lartillot, and Olivier Gascuel. "Phylogenetic mixture models for proteins". In: *Philosophical Transactions of the Royal Society B: Biological Sciences* Volume 363 (2008), pp. 3965–3976.

[Min+20] Bui Quang Minh, Heiko A Schmidt, Olga Chernomor, Dominik Schrempf, Michael D Woodhams, Arndt Von Haeseler, and Robert Lanfear. "IQ-TREE 2: new models and efficient methods for phylogenetic inference in the genomic era". In: *Molecular biology and evolution* Volume 37 (2020), pp. 1530–1534.

[MOP16] Diego Mallo, Leonardo de Oliveira Martins, and David Posada. "SimPhy: phylogenomic simulation of gene, locus, and species trees". In: *Systematic biology* Volume 65 (2016), pp. 334–344.

[Mor+21] Benoit Morel, Pierre Barbera, Lucas Czech, Ben Bettisworth, Lukas Hübner, Sarah Lutteropp, Dora Serdari, Evangelia-Georgia Kostaki, Ioannis Mamais, Alexey M Kozlov, et al. "Phylogenetic analysis of SARS-CoV-2 data is difficult". In: *Molecular biology and evolution* Volume 38 (2021), pp. 1777–1791.

[MWSS23] Benoit Morel, Tom A Williams, Alexandros Stamatakis, and Gergely J Szöllősi. "AleRax: A tool for gene and species tree co-estimation and reconciliation under a probabilistic model of gene duplication, transfer, and loss". In: *bioRxiv* (2023), pp. 2023–10.

[Par16] Emmanuel Paradis. "The distribution of branch lengths in phylogenetic trees". In: *Molecular Phylogenetics and Evolution* Volume 94 (2016), pp. 136–145.

[PDA10] Morgan N Price, Paramvir S Dehal, and Adam P Arkin. "FastTree 2–approximately maximum-likelihood trees for large alignments". In: *PloS one* Volume 5 (2010), e9490.

[PDS00] WH Piel, MJ Donoghue, and MJ Sanderson. "TreeBASE: a database of phylogenetic knowledge". In: *To the interoperable "Catalog of Life" with partners Species* (2000), pp. 41–47.

[Pea94] Karl Pearson. "Contributions to the mathematical theory of evolution". In: *Philosophical Transactions of the Royal Society of London. A* Volume 185 (1894), pp. 71–110.

[RF81] David F Robinson and Leslie R Foulds. "Comparison of phylogenetic trees". In: *Mathematical biosciences* Volume 53 (1981), pp. 131–147.

[Ron+12] Fredrik Ronquist, Maxim Teslenko, Paul Van Der Mark, Daniel L Ayres, Aaron Darling, Sebastian Höhna, Bret Larget, Liang Liu, Marc A Suchard, and John P Huelsenbeck. “MrBayes 3.2: efficient Bayesian phylogenetic inference and model choice across a large model space”. In: *Systematic biology* Volume 61 (2012), pp. 539–542.

[SMP08] Martin Simonsen, Thomas Mailund, and Christian NS Pedersen. “Rapid neighbour-joining”. In: *Algorithms in Bioinformatics: 8th International Workshop, WABI 2008, Karlsruhe, Germany, September 15-19, 2008. Proceedings 8*. Springer. 2008, pp. 113–122.

[SN87] Naruya Saitou and Masatoshi Nei. “The neighbor-joining method: a new method for reconstructing phylogenetic trees.” In: *Molecular biology and evolution* Volume 4 (1987), pp. 406–425.

[TM86] S Tavaré and Robert M Miura. “Lectures on mathematics in the life sciences”. In: *Am. Math. Soc.* Vol. 17. 1986, pp. 57–86.

[Wey+14] Grady Weyenberg, Peter M Huggins, Christopher L Schardl, Daniel K Howe, and Ruriko Yoshida. “KDETREES: non-parametric estimation of phylogenetic tree distributions”. In: *Bioinformatics* Volume 30 (2014), pp. 2280–2287.

[Yan14] Ziheng Yang. *Molecular evolution: a statistical approach*. Oxford University Press, 2014.

[Yan96] Ziheng Yang. “Among-site rate variation and its impact on phylogenetic analyses”. In: *Trends in ecology & evolution* Volume 11 (1996), pp. 367–372.

A. Appendix

A.1. SimPhy Configuration

```
// SPECIES TREE
-RS 1 // number of replicates
-sb f:5e-09
-sd f:0.0
-sl f:50
-st ln:21.25,0.2
-su ln:-21.9,0.1
-gd f:0.0
-gb f:0.0
-gt f:0.0
-gg f:0.0
-ld sl:0.0,1.0,gd
-lb f:ld
-lt sl:0.0,1.0,gt
-lg f:gg
-lk 0
// POPULATION
-SP f:10
// LOCUS
-rl f:100 // locus (gene family) per replicate
// Subsitution rates heterogeneity parameters
-hs ln:1.5,1
-hl ln:1.551533,0.6931472
-hg ln:1.5,1
// GENERAL
-cs 42
-O AA_simulated // output directory
-OM 1 // output the mappings
-OC 1 // log the configuration file
-OD 1 // log the configuration file
-OP 1 // log the configuration file
```

Listing A.1: Amino Acid Configuration File

```
// SPECIES TREE
-RS 1 // number of replicates
-sb f:5e-09
-sd f:0.0
-sl f:50
-st ln:21.25,0.2
-su ln:-21.9,0.1
-gd f:0.0
-gb f:0.0
-gt f:0.0
-gg f:0.0
-ld sl:0.0,1.0,gd
-lb f:ld
-lt sl:0.0,1.0,gt
```

```
-lg f:gg
-lk 0
// POPULATION
-SP f:10
// LOCUS
-rl f:100 // locus (gene family) per replicate
// Subsitution rates heterogeneity parameters
-hs ln:1.5,1
-hl ln:1.551533,0.6931472
-hg ln:1.5,1
// GENERAL
-cs 42
-O DNA_simulated // output directory
-OM 1 // output the mappings
-OC 1 // log the configuration file
-OD 1 // log the configuration file
-OP 1 // log the configuration file
```

Listing A.2: DNA Configuration File

A.2. INDELible Configuration

```
[TYPE] AMINOACID 1
[SETTINGS] [fastaextension] fasta
[SIMPHY-UNLINKED-MODEL] modelA
  [submodel] LG // LG model
[rates] 0 $(e:2) 0 // Site-specific rate heterogeneities: 0 p-inv, alpha from an E(2) and using a
    ↪ continuous gamma distribution.
[SIMPHY-PARTITIONS] simple [1.0 modelA $(sl:0,0.25,193.8466468952688)]
[SIMPHY-EVOLVE] 1 dataset
```

Listing A.3: Amino Acid Configuration File

```
[TYPE] NUCLEOTIDE 1
[SETTINGS] [fastaextension] fasta
[SIMPHY-UNLINKED-MODEL] modelA
  [submodel] GTR $(rd:16,3,5,5,6,15) // GTR with rates from a Dirichlet
  [statefreq] $(d:36,26,28,32)  // frequencies for T C A G sampled from a Dirichlet
[rates] 0 $(e:2) 0 // Site-specific rate heterogeneities: 0 p-inv, alpha from an E(2) and using a
    ↪ continuous gamma distribution.
[SIMPHY-PARTITIONS] simple [1.0 modelA $(sl:0,0.25,193.8466468952688)]
[SIMPHY-EVOLVE] 1 dataset
```

Listing A.4: DNA Configuration File

A.3. MrBayes Configuration

```
set autoclose=yes nowarn=yes;
    set seed=42;
    execute alignment.nex;
    prset aamodelpr=fixed(lg);
    lset rates=equal;
    mcmc nruns=1 nchains=2 ngen=100000 samplefreq=100 file=alignment_mr_bayes_out
```

Listing A.5: Amino Acid Configuration File Short Run

```
set autoclose=yes nowarn=yes;
    set seed=42;
```

```
    execute alignment.nex;
    prset aamodelpr=fixed(lg);
    lset rates=equal;
    mcmc nruns=1 nchains=4 ngen=550000 samplefreq=500 file=alignment_mr_bayes_out_ref
```

Listing A.6: Amino Acid Configuration File Long Reference Run

```
set autoclose=yes nowarn=yes;
    set seed=42;
    execute alignment.nex;
    lset nst=6 rates=equal;
    mcmc nruns=1 nchains=2 ngen=110000 samplefreq=100 file=alignment_mr_bayes_out
```

Listing A.7: DNA Configuration File Short Run

```
set autoclose=yes nowarn=yes;
    set seed=42;
    execute alignment.nex;
    lset nst=6 rates=equal;
    mcmc nruns=1 nchains=4 ngen=550000 samplefreq=500 file=alignment_mr_bayes_out_ref
```

Listing A.8: DNA Configuration File Long Reference Run

A.4. Simulated MSA Metrics

Table A.1.: Simulated DNA MSA Difficulty

MSA	Difficulty	MSA	Difficulty
dataset_001.fasta	0.52	dataset_051.fasta	0.44
dataset_002.fasta	0.5	dataset_052.fasta	0.45
dataset_003.fasta	0.54	dataset_053.fasta	0.79
dataset_004.fasta	0.39	dataset_054.fasta	0.47
dataset_005.fasta	0.49	dataset_055.fasta	0.52
dataset_006.fasta	0.39	dataset_056.fasta	0.36
dataset_007.fasta	0.48	dataset_057.fasta	0.44
dataset_008.fasta	0.42	dataset_058.fasta	0.39
dataset_009.fasta	0.47	dataset_059.fasta	0.51
dataset_010.fasta	0.51	dataset_060.fasta	0.38
dataset_011.fasta	0.46	dataset_061.fasta	0.51
dataset_012.fasta	0.41	dataset_062.fasta	0.33
dataset_013.fasta	0.44	dataset_063.fasta	0.34
dataset_014.fasta	0.4	dataset_064.fasta	0.49
dataset_015.fasta	0.42	dataset_065.fasta	0.41
dataset_016.fasta	0.44	dataset_066.fasta	0.78
dataset_017.fasta	0.5	dataset_067.fasta	0.48
dataset_018.fasta	0.37	dataset_068.fasta	0.37
dataset_019.fasta	0.44	dataset_069.fasta	0.5
dataset_020.fasta	0.48	dataset_070.fasta	0.45
dataset_021.fasta	0.47	dataset_071.fasta	0.31
dataset_022.fasta	0.4	dataset_072.fasta	0.45
dataset_023.fasta	0.84	dataset_073.fasta	0.4
dataset_024.fasta	0.37	dataset_074.fasta	0.52
dataset_025.fasta	0.42	dataset_075.fasta	0.49
dataset_026.fasta	0.34	dataset_076.fasta	0.4
dataset_027.fasta	0.42	dataset_077.fasta	0.41
dataset_028.fasta	0.64	dataset_078.fasta	0.35
dataset_029.fasta	0.42	dataset_079.fasta	0.39
dataset_030.fasta	0.47	dataset_080.fasta	0.59
dataset_031.fasta	0.34	dataset_081.fasta	0.39
dataset_032.fasta	0.39	dataset_082.fasta	0.45
dataset_033.fasta	0.46	dataset_083.fasta	0.57
dataset_034.fasta	0.33	dataset_084.fasta	0.33
dataset_035.fasta	0.45	dataset_085.fasta	0.58
dataset_036.fasta	0.46	dataset_086.fasta	0.52
dataset_037.fasta	0.34	dataset_087.fasta	0.4
dataset_038.fasta	0.42	dataset_088.fasta	0.74
dataset_039.fasta	0.45	dataset_089.fasta	0.44
dataset_040.fasta	0.38	dataset_090.fasta	0.54
dataset_041.fasta	0.36	dataset_091.fasta	0.51
dataset_042.fasta	0.49	dataset_092.fasta	0.49
dataset_043.fasta	0.49	dataset_093.fasta	0.46
dataset_044.fasta	0.59	dataset_094.fasta	0.49
dataset_045.fasta	0.47	dataset_095.fasta	0.46
dataset_046.fasta	0.49	dataset_096.fasta	0.43
dataset_047.fasta	0.48	dataset_097.fasta	0.37
dataset_048.fasta	0.53	dataset_098.fasta	0.43
dataset_049.fasta	0.37	dataset_099.fasta	0.48
dataset_050.fasta	0.74	dataset_100.fasta	0.39

Table A.2.: Simulated AA MSA Difficulty

MSA	Difficulty	MSA	Difficulty
dataset_001.fasta	0.26	dataset_051.fasta	0.26
dataset_002.fasta	0.16	dataset_052.fasta	0.35
dataset_003.fasta	0.33	dataset_053.fasta	0.22
dataset_004.fasta	0.43	dataset_054.fasta	0.28
dataset_005.fasta	0.3	dataset_055.fasta	0.77
dataset_006.fasta	0.35	dataset_056.fasta	0.34
dataset_007.fasta	0.57	dataset_057.fasta	0.25
dataset_008.fasta	0.25	dataset_058.fasta	0.29
dataset_009.fasta	0.28	dataset_059.fasta	0.34
dataset_010.fasta	0.78	dataset_060.fasta	0.35
dataset_011.fasta	0.28	dataset_061.fasta	0.39
dataset_012.fasta	0.26	dataset_062.fasta	0.26
dataset_013.fasta	0.28	dataset_063.fasta	0.3
dataset_014.fasta	0.33	dataset_064.fasta	0.47
dataset_015.fasta	0.41	dataset_065.fasta	0.32
dataset_016.fasta	0.3	dataset_066.fasta	0.22
dataset_017.fasta	0.34	dataset_067.fasta	0.28
dataset_018.fasta	0.74	dataset_068.fasta	0.29
dataset_019.fasta	0.25	dataset_069.fasta	0.26
dataset_020.fasta	0.69	dataset_070.fasta	0.86
dataset_021.fasta	0.32	dataset_071.fasta	0.15
dataset_022.fasta	0.19	dataset_072.fasta	0.26
dataset_023.fasta	0.38	dataset_073.fasta	0.42
dataset_024.fasta	0.25	dataset_074.fasta	0.4
dataset_025.fasta	0.26	dataset_075.fasta	0.32
dataset_026.fasta	0.4	dataset_076.fasta	0.28
dataset_027.fasta	0.37	dataset_077.fasta	0.27
dataset_028.fasta	0.27	dataset_078.fasta	0.51
dataset_029.fasta	0.29	dataset_079.fasta	0.28
dataset_030.fasta	0.28	dataset_080.fasta	0.41
dataset_031.fasta	0.35	dataset_081.fasta	0.25
dataset_032.fasta	0.25	dataset_082.fasta	0.63
dataset_033.fasta	0.25	dataset_083.fasta	0.24
dataset_034.fasta	0.28	dataset_084.fasta	0.19
dataset_035.fasta	0.29	dataset_085.fasta	0.26
dataset_036.fasta	0.3	dataset_086.fasta	0.32
dataset_037.fasta	0.29	dataset_087.fasta	0.25
dataset_038.fasta	0.26	dataset_088.fasta	0.3
dataset_039.fasta	0.78	dataset_089.fasta	0.25
dataset_040.fasta	0.26	dataset_090.fasta	0.17
dataset_041.fasta	0.28	dataset_091.fasta	0.36
dataset_042.fasta	0.34	dataset_092.fasta	0.45
dataset_043.fasta	0.43	dataset_093.fasta	0.38
dataset_044.fasta	0.23	dataset_094.fasta	0.24
dataset_045.fasta	0.47	dataset_095.fasta	0.26
dataset_046.fasta	0.26	dataset_096.fasta	0.11
dataset_047.fasta	0.32	dataset_097.fasta	0.25
dataset_048.fasta	0.27	dataset_098.fasta	0.24
dataset_049.fasta	0.32	dataset_099.fasta	0.29
dataset_050.fasta	0.36	dataset_100.fasta	0.17

A.5. Empirical MSA Metrics

Table A.3.: Empirical DNA MSA Metrics

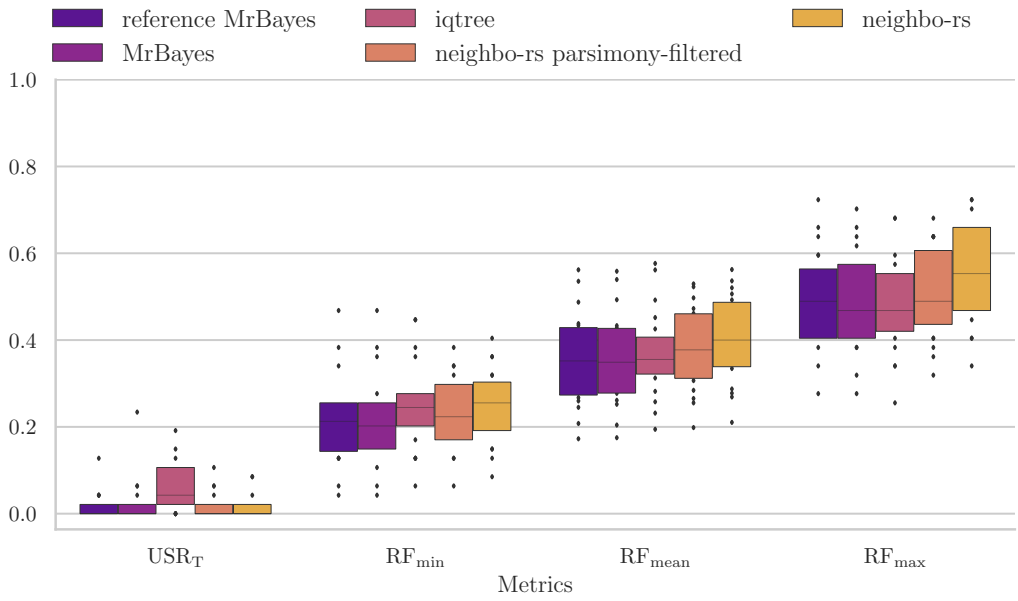
MSA	Difficulty	Taxa	Sites
alignment_19682_0	0.22	42	229
alignment_14876_1	0.23	36	272
alignment_10148_2	0.34	73	138
alignment_15827_1	0.36	30	117
alignment_21988_0	0.41	51	289
alignment_20312_4	0.43	33	133
alignment_21988_1	0.46	51	291
alignment_10264_0	0.49	34	103
alignment_16855_5	0.50	39	283
alignment_12630_5	0.54	44	290
alignment_15750_3	0.54	30	251
alignment_16855_3	0.54	51	277
alignment_17791_1	0.54	45	276
alignment_604_0	0.54	39	145
alignment_10102_2	0.56	55	282
alignment_15021_7	0.56	43	173
alignment_11178_23	0.59	43	135
alignment_22509_1	0.59	37	243
alignment_27596_28	0.60	30	295
alignment_10264_3	0.62	39	187
alignment_10102_1	0.63	46	271
alignment_10943_1	0.63	35	274
alignment_21888_4	0.66	41	288

MSA	Difficulty	Taxa	Sites
alignment_22503_1	0.66	70	288
alignment_15654_1	0.67	37	289
alignment_15750_2	0.68	31	279
alignment_11642_2	0.70	91	297
alignment_10464_0	0.71	81	157
alignment_12828_0	0.76	49	150
alignment_10644_4	0.78	42	265
alignment_27108_0	0.78	41	229
alignment_12145_0	0.79	50	149
alignment_20997_2	0.79	76	100
alignment_15442_4	0.81	42	240
alignment_21656_0	0.81	49	286
alignment_25465_1	0.81	33	293
alignment_26579_5	0.81	49	280
alignment_15070_4	0.82	80	171
alignment_15621_0	0.83	65	278
alignment_16634_1	0.83	52	197
alignment_20997_1	0.83	58	100
alignment_11949_2	0.85	65	225
alignment_27596_42	0.85	71	281
alignment_18144_1	0.86	74	277
alignment_27596_24	0.86	65	187
alignment_23738_1	0.87	87	216

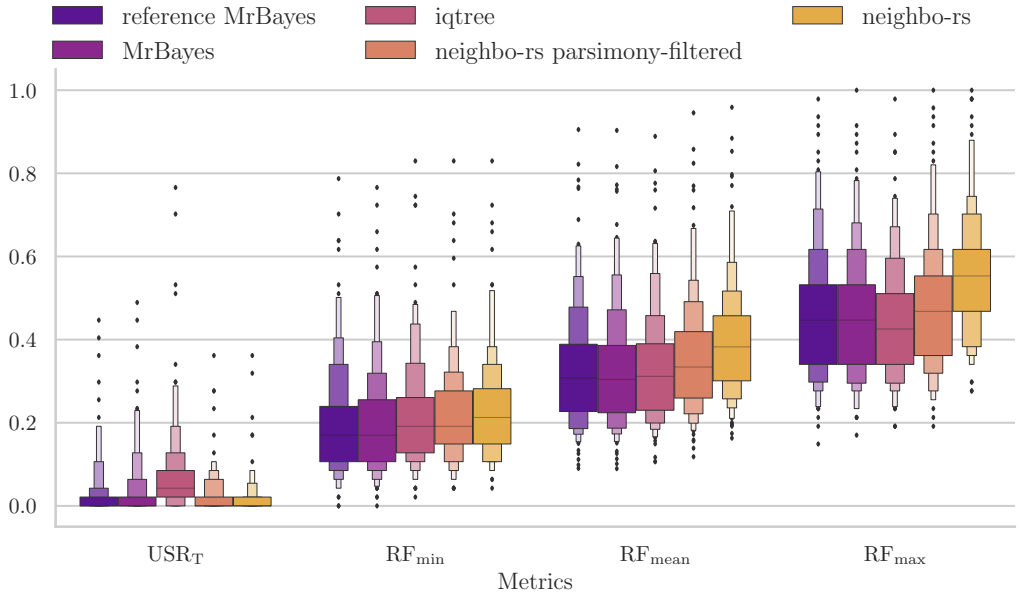
Table A.4.: Empirical AA MSA Metrics

MSA	Difficulty	Taxa	Sites
alignment_11802_1	0.00	35	290
alignment_14994_0	0.00	32	209
alignment_19998_1	0.00	32	191
alignment_20196_7	0.00	41	239
alignment_21817_2	0.01	51	276
alignment_20145_1	0.02	37	109
alignment_22408_2	0.02	31	229
alignment_17168_0	0.03	60	226
alignment_17171_0	0.03	60	226
alignment_21560_1	0.03	35	228
alignment_21188_0	0.04	39	122
alignment_22408_5	0.04	47	271
alignment_13985_0	0.05	46	149
alignment_13985_1	0.05	46	149
alignment_13985_9	0.05	46	149
alignment_20196_9	0.05	37	229
alignment_15931_4	0.06	52	136
alignment_15865_1	0.07	38	197
alignment_20196_20	0.08	42	212
alignment_17486_0	0.09	38	250
alignment_10068_0	0.11	32	291
alignment_17164_1	0.11	44	240
alignment_25084_2	0.12	55	219
alignment_15931_0	0.13	52	119
alignment_18551_1	0.13	78	104
alignment_10148_0	0.15	73	230
alignment_20196_10	0.15	62	234
alignment_14979_1	0.16	88	235
alignment_14979_0	0.18	76	283
alignment_20196_15	0.18	33	249
alignment_21362_0	0.18	68	123
alignment_14407_0	0.19	37	217
alignment_14408_0	0.19	37	217
alignment_14212_0	0.20	43	279
alignment_16190_4	0.20	80	201
alignment_15021_2	0.21	53	285
alignment_27836_0	0.21	37	189
alignment_20196_19	0.23	41	205
alignment_10791_7	0.24	58	126
alignment_28360_7	0.26	65	219
alignment_13444_2	0.27	81	284
alignment_13878_1	0.27	38	193
alignment_21817_1	0.27	81	247
alignment_28360_31	0.27	68	222
alignment_13882_0	0.28	46	133
alignment_13878_0	0.29	52	196
alignment_20145_0	0.29	39	107
alignment_10521_0	0.30	38	179
alignment_10791_0	0.31	33	144
alignment_28360_30	0.31	55	164
alignment_13985_11	0.32	96	114
alignment_20196_2	0.33	41	101
alignment_21817_9	0.33	85	240
alignment_13985_2	0.35	98	114
alignment_13985_3	0.35	98	114
alignment_12306_0	0.40	99	268
alignment_12306_10	0.40	84	127
alignment_16190_0	0.42	55	121
alignment_12306_3	0.43	100	199
alignment_11894_0	0.46	59	164
alignment_28360_18	0.47	67	222
alignment_12306_7	0.50	100	138
alignment_23036_0	0.50	82	162
alignment_16190_3	0.55	76	115
alignment_20196_16	0.56	77	206
alignment_23593_2	0.57	51	268
alignment_28360_16	0.57	68	125
alignment_28360_17	0.58	67	168
alignment_23745_0	0.59	42	250
alignment_20196_22	0.61	96	170
alignment_28360_8	0.62	68	123
alignment_15522_0	0.68	51	208
alignment_23745_2	0.69	39	250
alignment_23745_1	0.70	41	250
alignment_28360_27	0.71	67	182
alignment_23593_1	0.74	79	256
alignment_23593_0	0.76	87	268
alignment_19516_1	0.83	70	173

A.6. Reference Distribution



(a) Reference tree metrics for simulated parameter tuning data.



(b) Reference tree metrics for the full simulated dataset.

Figure A.1.: Comparing the fraction of missed splits, as well as minimum, mean and maximum RF-distance between the tools for the "true" tree generated by *SimPhy* shows that while the other metrics show large discrepancies between the distributions, all reliably hit the splits of the reference tree and the RF-distances are very similar overall.