

Empirical Numerical Properties of Maximum Likelihood Phylogenetic Inference

Master's Thesis of

Julia Haag

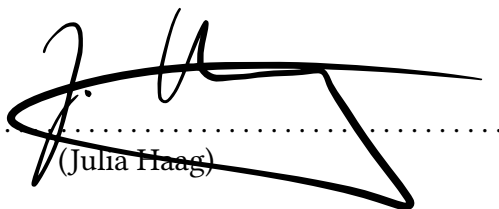
at the Department of Informatics
Institute of Theoretical Informatics

Reviewer: Prof. Dr. Alexandros Stamatakis
Second reviewer: Prof. Dr. Peter Sanders
Advisor: M.Sc. Lukas Hübner

01. May 2021 – 31. October 2021

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, 27.10.2021


.....
(Julia Haag)

Abstract

Phylogenetic trees represent hypothetical evolutionary relationships between organisms. Approaches for inferring phylogenetic trees include the Maximum Likelihood (ML) method. This method relies on numerical optimization routines that use internal numerical thresholds. We analyze the influence of these thresholds on the likelihood scores and runtimes of tree inferences for the ML inference tools RAxML-NG, IQ-Tree, and FastTree. We analyze 22 empirical datasets and show that we can speed up the tree inference in RAxML-NG and IQ-Tree by changing the default values of two such numerical thresholds. Using 15 additional simulated datasets, we show that these changes do not affect the accuracy of the inferred phylogenetic trees. For RAxML-NG, increasing the likelihood thresholds $lh_epsilon$ and $spr_lh_epsilon$ to 10 and 10^3 respectively results in an average speedup of 1.9 ± 0.6 . Increasing the likelihood threshold $lh_epsilon$ in IQ-Tree results in an average speedup of 1.3 ± 0.4 . In addition to the numerical analysis, we attempt to predict the difficulty of datasets, with the aim of preventing an unnecessarily large number of tree inferences for datasets that are easy to analyze. We present our prediction experiments and discuss why this task proved to be more challenging than anticipated.

Zusammenfassung

Phylogenetische Bäume repräsentieren hypothetische evolutionäre Beziehungen zwischen Organismen. Ein Ansatz zur Berechnung phylogenetischer Bäume ist die Maximum Likelihood (ML) Methode. Implementierungen zur Baumsuche anhand der ML Methode nutzen numerische Optimierungsverfahren, die interne numerische Schwellenwerte benutzen. In dieser Arbeit untersuchen wir den Einfluss dieser numerischen Schwellenwerte auf die Güte und Laufzeit von Baumsuchen dreier ML Programme RAxML-NG, IQ-Tree und FastTree. Wir analysieren 22 empirische Datensätze und zeigen, dass wir die Baumsuche von RAxML-NG und IQ-Tree durch Verändern zweier numerischer Schwellenwerte beschleunigen können. Unter Verwendung von 15 zusätzlichen, simulierten Datensätzen zeigen wir, dass diese Änderungen keinen Einfluss auf die Genauigkeit der berechneten phylogenetischen Bäume haben. Für RAxML-NG empfehlen wir, den Schwellenwert *lh_epsilon* auf 10 zu erhöhen und den Schwellenwert *spr_lh_epsilon* auf 10^3 . Unter diesen Änderungen beobachten wir eine durchschnittliche Laufzeitbeschleunigung von 1.9 ± 0.6 auf den empirischen Datensätzen. Für IQ-Tree empfehlen wir, den Schwellenwert *lh_epsilon* ebenfalls auf 10 zu erhöhen. Auf den empirischen Datensätzen beobachten wir durch diese Änderung eine durchschnittliche Beschleunigung von 1.3 ± 0.4 . Zusätzlich zu dieser numerischen Analyse, versuchen wir vorherzusagen, ob ein Datensatz schwer oder einfach zu analysieren ist. Das Ziel dabei ist es, für einfache Datensätze eine unnötig hohe Anzahl von Baumsuchen zu vermeiden. In der vorliegenden Arbeit präsentieren wir unsere Experimente zur Vorhersage der Schwierigkeit und diskutieren, warum sich diese Aufgabe als herausfordernder erwies als erwartet.

Contents

Abstract	i
Zusammenfassung	ii
I. Introduction	1
1. Introduction	2
2. Phylogenetic Inference	4
3. Maximum Likelihood Tree Inference	5
3.1. Multiple Sequence Alignment	5
3.2. Substitution Models	6
3.3. Likelihood Computation	7
3.4. Likelihood Optimization	8
3.4.1. Topology Optimization	8
3.4.2. Branch Length Optimization	9
3.4.3. Model Parameter Optimization	9
3.5. Significance Tests	10
3.5.1. Kishino-Hasagawa Test	10
3.5.2. Shimodaira-Hasagawa Test	11
3.5.3. Approximately Unbiased Test	11
3.5.4. Expected Likelihood Weight Test	11
3.6. Maximum Likelihood Tree Inference Tools	11
3.6.1. RAxML-NG	12
3.6.2. IQ-Tree	12
3.6.3. FastTree	13
3.7. Numerical Epsilon Thresholds	14
II. Numerical Properties of Maximum Likelihood Inference	15
4. Motivation	16
5. Experimental Setup	17
5.1. Data Generation Pipeline	17
5.2. Numerical Thresholds	19

5.3. Datasets	20
6. Influence of Numerical Thresholds on Likelihoods and Runtimes	21
6.1. Evaluation Phase	21
6.1.1. Minimum Branch Length	22
6.1.2. Likelihood Epsilon	22
6.1.3. Remaining Thresholds	24
6.2. Tree Search Phase	25
6.2.1. Likelihood Epsilons	25
6.2.2. Minimum Branch Length	34
6.2.3. Remaining Thresholds	39
7. Comparison of Inference Tools	40
7.1. Likelihood and Runtime	40
7.2. Influence of Evaluation Functionality	42
 III. Predicting Dataset Difficulty	 44
8. Problem Description	45
9. Training Data	46
9.1. Definition of Difficulty	46
9.2. Difficult Difficulty Labels	46
10. Difficulty Prediction	49
10.1. Feature Generation	49
10.1.1. MSA Features	49
10.1.2. Tree Features	50
10.2. Difficulty Prediction	51
10.2.1. Machine Learning Algorithms	51
10.2.2. Performance Metrics	53
10.2.3. Prediction Results	54
11. Further Experiments	56
 IV. Conclusion and Future Work	 57
12. Discussion	58
13. Outlook	60
Bibliography	62

A. Appendix	68
A.1. Numerical Properties of Maximum Likelihood Inference	68
A.1.1. Data Generation Pipeline	68
A.1.2. Influence of Numerical Thresholds	69

List of Figures

1.1.	Costs to sequence a human genome and costs for processing power according to Moore's Law (log scale). Figure obtained from <i>The Cost of Sequencing a Human Genome</i> [34].	2
2.1.	Phylogenetic tree of animals. Based on Fig. 9 in <i>Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics</i> [55].	4
3.1.	Common tree topology optimization strategies. a) Subtree Pruning and Re-grafting (SPR). b) Nearest Neighbor Interchange (NNI). c) Tree Bisection and Reconnection (TBR).	9
5.1.	The workflow of our data generation pipeline for one dataset and one tree inference tool. For each value combination of Thresh1 and Thresh2, we run 5 tree searches with the respective values. The tree with the highest ML score after the tree search phase (the highlighted tree) is then re-evaluated. After the evaluation phase, this tree is included in a set of <i>candidate trees</i> . We compare all trees in this candidate tree set using the IQ-Tree significance tests. Trees passing <i>all</i> tests form part of the set of <i>plausible trees</i>	18
6.1.	Morel <i>et al.</i> [49] observe worse ML scores after re-evaluating a tree with RAxML-NG and IQ-Tree under higher <i>minBranchLen</i> settings.	22
6.2.	Influence of the <i>minBranchLen</i> threshold on the ML scores and runtimes of the RAxML-NG evaluation phase. The runtimes refer to the runtime of the evaluation phase. The plots summarize the data over all datasets. The dashed vertical line indicates the mean, and the solid vertical line the median value. The highlighted box indicates the default <i>minBranchLen</i> value for RAxML-NG.	23
6.3.	Influence of the <i>minBranchLen</i> threshold on the ML scores and runtimes of the IQ-Tree evaluation phase. The runtimes refer to the runtime of the evaluation phase. The plots summarize the data over all datasets. The dashed vertical line indicates the mean, and the solid vertical line the median value. The highlighted box indicates the default <i>minBranchLen</i> value for IQ-Tree.	23
6.4.	Influence of the <i>lh_epsilon</i> threshold on the ML scores and runtimes of the RAxML-NG evaluation phase. The runtimes refer to the runtime of the evaluation phase. The plots summarize the data over all datasets. The dashed vertical line indicates the mean, and the solid vertical line the median value. The highlighted box indicates the default <i>lh_epsilon</i> value for RAxML-NG.	24

6.5.	Influence of the <i>lh_epsilon</i> threshold on the ML scores and runtimes of the RAxML-NG tree search phase. The ML scores refer to the ML scores <i>after</i> the evaluation phase. The runtimes refer to the runtime of the tree search phase. The plots summarize the data over all datasets. The dashed vertical line indicates the mean, and the solid vertical line the median value. The highlighted box indicates the default <i>lh_epsilon</i> value for RAxML-NG.	26
6.6.	Average proportions of <i>lh_epsilon</i> values among trees in the plausible tree set for RAxML-NG over all datasets. The darkest bar indicates the current default value (10^{-1}) and the brightest bar our suggested new default value (10). The error bar depicts the standard deviation.	27
6.7.	Variances of topological distances to the true tree per <i>lh_epsilon</i> setting on 15 simulated datasets. The y-axis shows the variations in percentage points relative to the average topological distance per dataset.	28
6.8.	Influence of the <i>spr_lh_epsilon</i> threshold on the ML scores and runtimes of the RAxML-NG tree search phase. The ML scores refer to the ML scores <i>after</i> the evaluation phase. The runtimes refer to the runtime of the tree search phase. The plots summarize the data over all datasets. The dashed vertical line indicates the mean, and the solid vertical line the median value. The highlighted box indicates the default <i>spr_lh_epsilon</i> value for RAxML-NG.	28
6.9.	Average proportions of <i>spr_lh_epsilon</i> values among trees in the plausible tree set for RAxML-NG over all datasets. The darkest bar indicates the current default value (10^{-1}) and the brightest bar our suggested new default value (10^3). The error bar depicts the standard deviation.	29
6.10.	Variances of topological distances to the true tree per <i>spr_lh_epsilon</i> setting on 15 simulated datasets. The y-axis shows the variations in percentage points relative to the average topological distance per dataset.	29
6.11.	Distribution of speedups across all datasets for RAxML-NG. We compare runtimes for tree searches under the default setting <i>lh_epsilon</i> = 10^{-1} and <i>spr_lh_epsilon</i> = 10^{-1} , with tree searches under our new suggested settings <i>lh_epsilon</i> = 10 and <i>spr_lh_epsilon</i> = 10^3	30
6.12.	Influence of the <i>lh_epsilon</i> threshold on the ML scores and runtimes of the IQ-Tree tree search phase. The ML scores refer to the ML scores <i>after</i> the evaluation phase. The runtimes refer to the runtime of the tree search phase. The plots summarize the data over all datasets. The dashed vertical line indicates the mean, and the solid vertical line the median value. The highlighted box indicates the default <i>lh_epsilon</i> value for IQ-Tree.	31
6.13.	Average proportions of <i>lh_epsilon</i> values among trees in the plausible tree set for IQ-Tree over all datasets. The darkest bar indicates the current default value (10^{-3}) and the brightest bar our suggested new default value (10). The error bar depicts the standard deviation.	32
6.14.	Variances of topological distances to the true tree per <i>lh_epsilon</i> setting on 15 simulated datasets. The y-axis shows the variations in percentage points relative to the average topological distance per dataset.	33

6.15. Distribution of speedups across all datasets for IQ-Tree. We compare runtimes for tree searches under the default setting $lh_epsilon = 10^{-3}$, with tree searches under our new suggested setting $lh_epsilon = 10$	33
6.16. Average proportions of $lh_epsilon$ values among trees in the plausible tree set for datasets D1481, D1604, D1718, and D2445. The left figure shows the proportions for random IQ-Tree. The right figure shows the proportions for de-randomized IQ-Tree. The error bar depicts the standard deviation.	34
6.17. Earth mover’s distances of the proportions of $lh_epsilon$ values in the set of plausible trees to a flat histogram for RAxML-NG, the random IQ-Tree variant and the de-randomized IQ-Tree variant. The plot shows the distances for the four datasets D1481, D1604, D1718, and D2445.	35
6.18. Influence of the <i>minBranchLen</i> threshold on the ML scores and runtimes of the RAxML-NG tree search phase. The ML scores refer to the ML scores <i>after</i> the evaluation phase. The runtimes refer to the runtime of the tree search phase. The plots summarize the data over all datasets. The dashed vertical line indicates the mean, and the solid vertical line the median value. The highlighted box indicates the default <i>minBranchLen</i> value for RAxML-NG.	36
6.19. Influence of the <i>minBranchLen</i> threshold on the ML scores and runtimes of the IQ-Tree tree search phase. The ML scores refer to the ML scores <i>after</i> the evaluation phase. The runtimes refer to the runtime of the tree search phase. The plots summarize the data over all datasets. The dashed vertical line indicates the mean, and the solid vertical line the median value. The highlighted box indicates the default <i>minBranchLen</i> value for IQ-Tree.	37
6.20. Influence of the <i>minBranchLen</i> threshold on the ML scores and runtimes of the FastTree tree search phase. The ML scores refer to the ML scores <i>after</i> the tree search phase. The runtimes refer to the runtime of the tree search phase. The plots summarize the data over all datasets. The dashed vertical line indicates the mean, and the solid vertical line the median value. The highlighted box indicates the default <i>minBranchLen</i> value for FastTree.	38
7.1. Comparison of ML scores and runtimes of RAxML-NG, IQ-Tree, and FastTree for all datasets. All values are relative to the RAxML-NG values, therefore all RAxML-NG values collapse into a single point.	41
7.2. ML score improvement for different <i>minBranchLen</i> values after the evaluation phase relative to the ML score after the tree search phase. The plot shows the improvements over all datasets for RAxML-NG.	43
7.3. Average ML scores after the tree search phase and after the evaluation phase. This plot shows the data for one exemplary datasets (D1481) for RAxML-NG.	43
9.1. Inferred ground-truth labels for the 222 training datasets. The annotations highlight the empirical datasets. Dots indicate easy datasets and stars indicate difficult datasets. The dashed lines depict the decision threshold for each feature. Only datasets above both thresholds are labeled as being difficult (highlighted area).	48

A.1.	Influence of the numerical thresholds on the ML scores and runtimes of the RAxML-NG evaluation phase. Each plot summarizes the data over all datasets. The dashed vertical line indicates the mean, and the solid vertical line the median value. The highlighted box indicates the default value of the respective numerical threshold in RAxML-NG.	71
A.2.	Influence of the numerical thresholds on the ML scores and runtimes of the IQ-Tree evaluation phase. Each plot summarizes the data over all datasets. The dashed vertical line indicates the mean, and the solid vertical line the median value. The highlighted box indicates the default value for the respective numerical threshold in IQ-Tree.	73
A.3.	Influence of the numerical thresholds on the ML scores and runtimes of the RAxML-NG tree search phase. Each plots summarizes the data over all datasets. The dashed vertical line indicates the mean, and the solid vertical line the median value. The highlighted box indicates the default value for the respective numerical threshold in RAxML-NG.	77
A.4.	Influence of the numerical thresholds on the ML scores and runtimes of the IQ-Tree tree search phase. Each plot summarizes the data over all datasets. The dashed vertical line indicates the mean, and the solid vertical line the median value. The highlighted box indicates the default value for the respective numerical threshold in IQ-Tree.	79
A.5.	Influence of the numerical thresholds on the ML scores and runtimes of FastTree. Each plot summarizes the data over all datasets. The dashed vertical line indicates the mean, and the solid vertical line the median value. The highlighted box indicates the default value for the respective numerical threshold in FastTree.	80

List of Tables

5.1.	Varied numerical thresholds with value ranges and applicable inference tools. The values in braces state the default value for the respective inference tool.	19
5.2.	Overview of the datasets we use for our analyses. All datasets are empirical datasets.	20
10.1.	The set of MSA features we suggest for difficulty prediction. An asterisk indicates that we implement this feature in our experiments.	50
10.2.	The set of tree features we suggest for difficulty prediction. An asterisk indicates that we implement this feature in our experiments.	51
10.3.	Comparison of different classification algorithms for difficulty prediction. Since we focus on the number of correct predictions of “difficult”, the TPR column is highlighted.	55
A.1.	The command lines we use to conduct tree inferences, re-evaluations and significance tests with RAxML-NG, IQ-Tree, and FastTree.	68

Part I.
Introduction

1. Introduction

Over the last two decades, the amount of available biological sequence data was increasing exponentially [23]. Novel sequencing techniques allow for faster and cheaper sequencing of long DNA strands and entire genomes [46]. The cost for sequencing a human genome has decreased at a higher pace, than the cost of compute power according to Moore’s Law [7] (Figure 1.1).

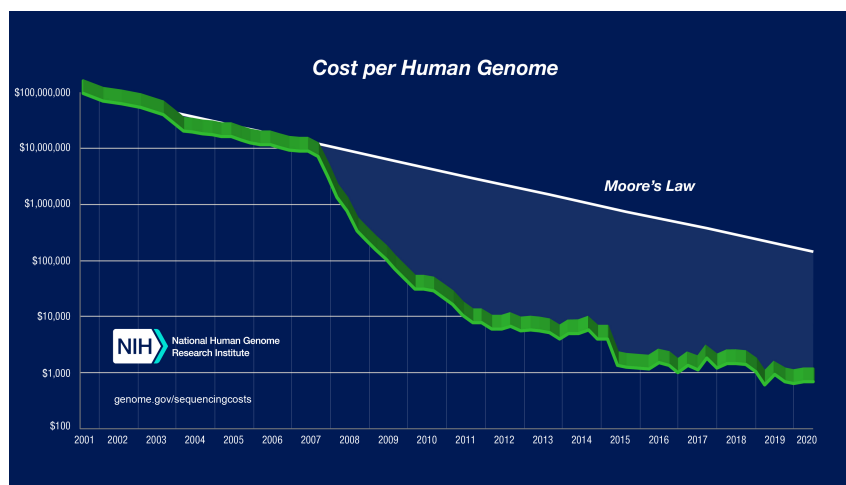


Figure 1.1.: Costs to sequence a human genome and costs for processing power according to Moore’s Law (log scale). Figure obtained from *The Cost of Sequencing a Human Genome* [34].

This growing gap between compute power and available biological sequence data calls for fast and systematic analyses of these data. One use case of biological sequence data is the inference of phylogenetic trees. These trees represent hypothetical evolutionary relationships between a group of organisms or species. The number of possible phylogenetic trees grows exponentially with the number of organisms. In September 2021, the catalog of life, a list of currently known species, contained approximately 2.3 million living species [2]. Even for 51 organisms, the number of possible phylogenetic trees already exceeds the estimated number of atoms in the universe [43]. In this thesis, we focus on one particular phylogenetic inference method, the Maximum Likelihood (ML) model. As finding the best ML tree is \mathcal{NP} -hard [10], we have to rely on search heuristics when inferring phylogenetic trees. Inferring trees under the ML model is time-consuming, with large datasets requiring several days of CPU time [39]. This is problematic, not only due to the limits of available compute power. With climate change being one of the biggest challenge of our time [73], we, as computer scientists and researchers, have the responsibility to limit the amount of computational resources required to solve a problem to a reasonable amount. In phylogenetics, this means that we should attempt

to decrease the amount of computing time necessary to infer equally likely trees, and also not run unnecessary tree inferences. In this thesis, we address both problems. In Part II we investigate whether the internal numerical thresholds of the tree search heuristics influence the runtime and ML scores. We show that by changing the default settings for two numerical thresholds, we can improve the runtimes, while yielding equally likely trees for two widely used ML inference tools. Additionally, in Part III we attempt to predict how difficult a dataset is to analyze. For easy-to-analyze datasets, we need to conduct fewer tree searches than for datasets that are difficult to analyze. Predicting this correctly could prevent the execution of unnecessary tree inferences. By the time of finishing this thesis, this task remains unsolved, as it proved to be more challenging than anticipated.

We start this thesis by explaining the fundamentals of phylogenetic trees (Chapter 2) and tree inference under the ML model (Chapter 3). In Chapter 4, we motivate the analysis of the numerical thresholds, and proceed to explain our analysis workflow in Chapter 5. In Chapter 6, we present the result of our numerical analysis, and show that we can speed up the tree inferences for two ML inference tools by changing two default numerical threshold settings. We further compare three ML inference tools according to ML scores and runtimes (Chapter 7). In Chapters 8 to 11, we demonstrate our efforts to predict how difficult a dataset is to analyze, provide an explanation on why this task is challenging, and suggest further experiments. Finally, in Chapter 12 we summarize the findings of this thesis, and in Chapter 13 we discuss possible future work.

2. Phylogenetic Inference

A phylogenetic tree represents hypothetical evolutionary relationships for a group of organisms [13]. The leaves of a phylogenetic tree are extant organisms called *taxa*. The inner nodes represent extinct *hypothetical* common ancestors. Phylogenetic trees have many important applications in biology and medicine, for example in drug development research [27], forensics [47], or the analysis of SARS-CoV-2 genomes [49]. Figure 2.1 shows an exemplary phylogenetic tree of animals.

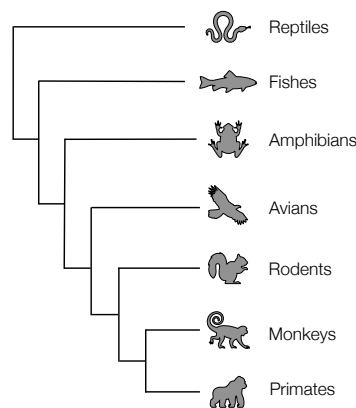


Figure 2.1.: Phylogenetic tree of animals. Based on Fig. 9 in *Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics* [55].

When inferring phylogenies, we reconstruct unrooted, strictly binary trees [63]. Rooted phylogenetic trees contain a common ancestor for all taxa. In contrast, unrooted trees merely show the relationships among the taxa without identifying the common ancestor [13]. A binary tree represents an evolutionary branching process, where a lineage always splits into two sublineages. We can represent multifurcations (splits into more than two sublineages) by connecting multiple bifurcating branches via a branch of length zero or close-to-zero [12]. The branch lengths of the tree typically indicate a relative evolutionary distance between two nodes.

In the past, researchers used to build phylogenetic trees based on morphological traits [42]. During the last decades, researchers developed a variety of computer-based phylogenetic inference methods [64]. These methods rely on biological sequence data of the organisms, usually Deoxyribonucleic Acid (DNA) or Amino Acid (AA) data. Using sequence data instead of morphological data, we can infer phylogenies with higher statistical confidence [42]. In this thesis, we focus on aspects of phylogenetic tree inference under the maximum likelihood (ML) model. We explain the ML model in the following chapter.

3. Maximum Likelihood Tree Inference

The starting point for inferring a phylogenetic tree is a set of homologous sequences from the organisms under study. Sequences are homologous if they share a common ancestor. Researchers have developed different methods to reconstruct trees given homologous sequences [76]. Many of these methods operate on a data structure called Multiple Sequence Alignment (MSA). In an MSA, regions with a hypothetical common evolutionary history are aligned to each other (more on MSA in the next section) [43]. Methods using the MSA as input can be categorized into two classes: distance-based methods, such as the neighbor-joining algorithm [57], and character-based methods, such as maximum parsimony [15, 18] and maximum likelihood (ML) [17]. Distance-based approaches build trees based on the assumption that the fewer changes there are between two sequences, the closer related the respective organisms are. This, however, does not account for multiple substitutions along the same branch ($G \rightarrow A \rightarrow T$)[76].

Instead of trying to minimize the number of mutations, the ML method tries to find the most likely tree among all possible trees given an explicit statistical model of sequence evolution. This is the tree that best explains the given data. Note that the likelihood of a tree is not the probability of this tree being the correct one. The likelihood is the probability of observing the data D given the parameter vector θ :

$$L(\theta|D) = P(D|\theta) \quad (3.1)$$

In case of phylogenetic tree inference, the parameter vector $\theta = (T, b, M, \phi)$ comprises the tree topology T , the set of branch lengths b and the substitution model M with internal parameters ϕ .

3.1. Multiple Sequence Alignment

To obtain the Multiple Sequence Alignment (MSA) from a set of homologous sequences, the sequences are filled with gaps, such that the regions that likely share a common evolutionary history are aligned to each other [43]. A simple heuristic for computing the MSA is to minimize the distance between aligned sequence sites. The resulting MSA data D can be represented as a matrix

$$D = \begin{bmatrix} s_1^1 & s_2^1 & \dots & s_M^1 \\ s_1^2 & s_2^2 & \dots & s_M^2 \\ \dots & \dots & \dots & \dots \\ s_1^N & s_2^N & \dots & s_M^N \end{bmatrix} \quad (3.2)$$

where s_i^k denotes the i -th character of the k -th aligned sequence. In the following, we will refer to the i -th column as i -th site s_i .

The properties of the MSA impact the dataset analysis [66]. The higher the proportion of gaps in the alignment, the more difficult it is to analyze. Despite their higher proportion of

gaps, multi-gene MSAs with few taxa are, easier to analyze than MSAs with many taxa and few genes. Alignments with a larger number of sites are in general easier to analyze. The higher the so-called *phylogenetic signal* of the data, the more accurate the inferred phylogenetic tree. This phylogenetic signal is a measurement of how informative the data is about the underlying evolutionary process [43].

Based on an MSA, we can attempt to infer a phylogenetic tree, using, for example, the maximum parsimony approach [15, 18]. But, to infer a ML tree, we need to model sequence evolution. This is done via a so-called *substitution model*.

3.2. Substitution Models

The substitution model describes the change of sequences over evolutionary time. It consists of the equilibrium frequencies $\vec{\Pi}$ and the substitution rates R [76].

The equilibrium frequencies describe the prior probability of occurrence for each character in the sequence data. For example, for DNA data, this vector contains four frequencies: $\vec{\Pi} = (\pi_A, \pi_C, \pi_G, \pi_T)$ where π_A denotes the probability of a nucleotide being the character A .

Molecular sequence evolution is modeled as a continuous time Markov Chain, where state transitions represent character substitutions and where the current state depends only on the previous state. Substitution rates are represented as a matrix R .

In phylogenetics, we assume time reversibility. This means that the probability of starting in state i and mutating into state j over time t is the same as starting in state j and mutating into state i over time t :

$$\forall i, j \in \{A, C, G, T\} : \pi_i P_{i \rightarrow j}(t) = \pi_j P_{j \rightarrow i}(t) \quad (3.3)$$

Various models exist for describing the equilibrium frequencies $\vec{\Pi}$ and the substitution rates R . These models differ in the number of free parameters. The most flexible and general model for DNA data is the *General Time Reversible (GTR)* model [72]:

$$\forall i, j \in \{A, C, T, G\} : \pi_i \neq \pi_j \quad R = \begin{pmatrix} \alpha & & & & & \\ \beta & \gamma & & & & \\ \delta & \epsilon & \zeta & & & \end{pmatrix} \quad (3.4)$$

The GTR model has 10 free parameters. If we set the equilibrium frequencies in $\vec{\Pi}$ and the rates in R relative to each other, we can reduce the degrees of freedom to 8. All other models, for example the Jukes-Cantor (JC) model [35] with only a single free parameter, are a subset of the GTR model. In our analyses, we therefore only use the GTR model.

As different regions of the DNA encode different structures or functions, they underlie different evolutionary pressures [75]. This leads to differences in the mutation rates among sites. This phenomenon is called *rate heterogeneity*. To accommodate this, we multiply the rate matrix R for site s_i with a factor r_i . We model the factors for the different sites, by assuming that they are Γ distributed. The shape of the Γ distribution is determined by a parameter α . The smaller the α value, the higher the rate heterogeneity [75].

Given the MSA and the substitution model, we can evaluate the likelihood of a given tree topology. We explain how we can compute this likelihood in the following section.

3.3. Likelihood Computation

With the ML method, we only evaluate the likelihood of a given topology rather than suggesting a topology [76]. Therefore, in addition to the MSA and the substitution model, we need a candidate tree topology. During the ML search, we iteratively optimize this topology to obtain a tree with a good ML score. We can obtain such a candidate tree topology using, for example, the randomized stepwise addition order algorithm [8] or maximum parsimony [15, 18]. The explanation of the likelihood computation in this chapter is based on the explanation in *Computational Molecular Evolution* [76].

We compute the likelihood of the topology T with branch lengths b , and model parameters ϕ (summarized as parameter θ) given the MSA data D as:

$$L(\theta|D) = P(D|\theta) \quad (3.5)$$

To simplify the computation, we assume that sites evolve independently. That is:

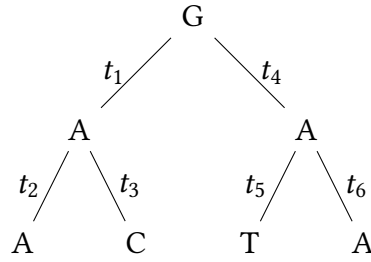
$$L(\theta|D) = P(D|\theta) = \prod_{i=1}^n P(s_i|\theta) \quad (3.6)$$

where s_i is the i -th column in the MSA.

The probabilities per site are usually very small. To prevent numerical underflow during multiplication, we compute the logarithm of the likelihood instead:

$$\log(L(\theta|D)) = \sum_{i=1}^n \log(P(s_i|\theta)) \quad (3.7)$$

In order to explain the computation of the per site probabilities $P(s_i|\theta)$, consider the following exemplary tree topology with a single site:

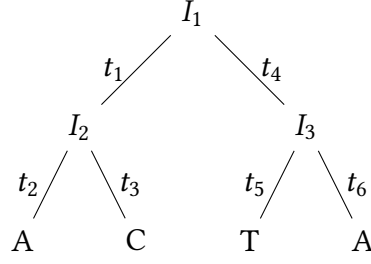


We evaluate the likelihood of this site by multiplying the transition probabilities given in the model M :

$$L(\theta|s_i) = P(s_i|\theta) = \pi_G \cdot P_{G \rightarrow A}(t_1) \cdot P_{A \rightarrow A}(t_2) \cdot P_{A \rightarrow C}(t_3) \\ \cdot P_{G \rightarrow A}(t_4) \cdot P_{A \rightarrow T}(t_5) \cdot P_{A \rightarrow A}(t_6)$$

The first factor π_G denotes the probability of observing nucleotide G at the root of the tree. The following factor $P_{G \rightarrow A}(t_1)$ reads as the probability of mutating from nucleotide G to nucleotide A in time t_1 .

We infer phylogenetics trees based on the observed sequences at the tree's tips. Therefore the inner states are unknown:



To compute the likelihood with unknown inner states, we have to consider all possible nucleotide states at the inner nodes. We achieve this by summing over probabilities for each possible combination:

$$L(\theta|s_i) = P(s_i|\theta) = \sum_{I_1=A}^T \sum_{I_2=A}^T \sum_{I_3=A}^T \pi_{I_1} \cdot P_{I_1 \rightarrow I_2}(t_1) \cdot P_{I_2 \rightarrow A}(t_2) \cdot P_{I_2 \rightarrow C}(t_3) \\ \cdot P_{I_1 \rightarrow I_3}(t_4) \cdot P_{I_3 \rightarrow T}(t_5) \cdot P_{I_3 \rightarrow A}(t_6)$$

This combinatorial problem can be efficiently computed using dynamic programming via the Felsenstein Pruning algorithm [16].

Finding the most likely tree is \mathcal{NP} -hard [10]. To find the most likely tree, we would have to evaluate all possible trees. The number of possible tree topologies, however, grows exponentially with the number of taxa. The number of possible unrooted trees for n taxa is $\prod_{i=3}^n (2i - 5)$. While an exhaustive search is guaranteed to find the globally optimal tree, this approach is computationally not feasible. In practice, we use heuristic search methods that iteratively improve a given topology [76]. In the following section, we present a selection of such optimization techniques.

3.4. Likelihood Optimization

In order to optimize the likelihood, we can improve the tree topology, the parameters of the substitution model, and the branch lengths with respect to the likelihood.

3.4.1. Topology Optimization

Commonly used methods to improve the tree topology are Subtree Pruning and Regrafting (SPR), Nearest Neighbor Interchange (NNI), and Tree Bisection and Reconnection (TBR) (Figure 3.1) [76].

With the SPR method, a subtree is selected and detached from the main tree (pruning) and reattached (regrafted) onto another branch. The NNI method detaches four subtrees by removing the internal branches and then rearranges these subtrees. With the TBR method, a branch is removed from the tree, yielding two subtrees. These subtrees are then reconnected by inserting a branch, connecting two branches of the subtrees.

The number of distinct trees generated for a tree with n taxa are in $O(n)$, $O(n^2)$ and $O(n^3)$ for NNI, SPR, and TBR, respectively [30].

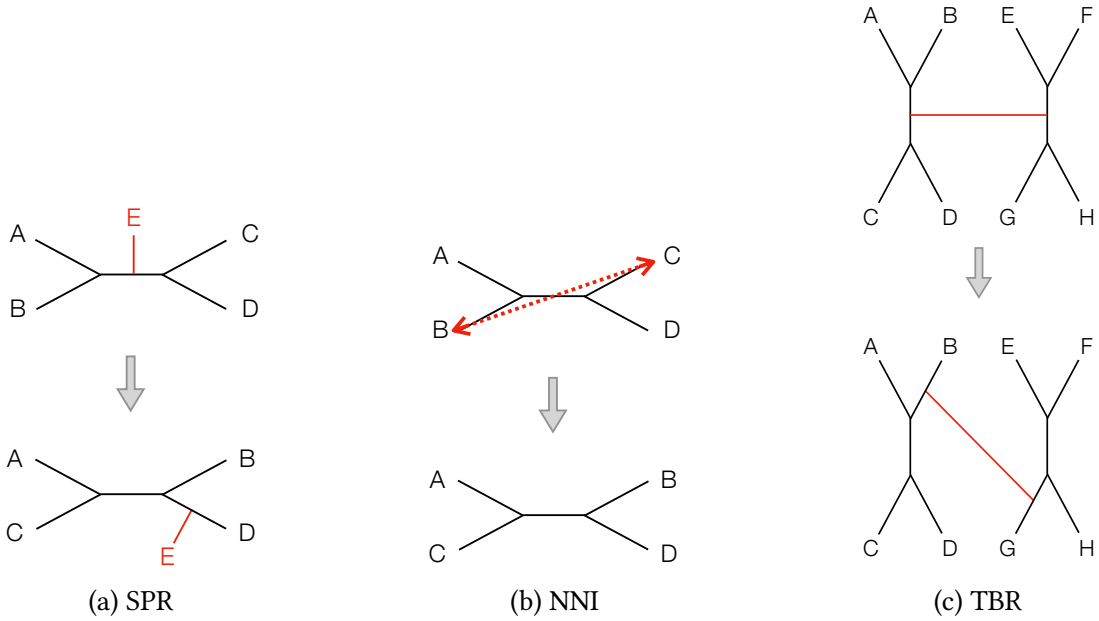


Figure 3.1.: Common tree topology optimization strategies. a) Subtree Pruning and Regrafting (SPR). b) Nearest Neighbor Interchange (NNI). c) Tree Bisection and Reconnection (TBR).

3.4.2. Branch Length Optimization

Given a fixed tree topology, we can improve the likelihood by adapting the branch lengths. The Newton-Raphson method is commonly used to solve this non-linear numerical optimization problem [40].

We evaluate each branch in the tree separately, that is, we only optimize a single branch length while the others remain fixed. The goal is to find the branch length b that maximizes the likelihood $L(b)$ of the tree. To find this branch length, we solve $L'(b) = 0$ using the following iterative approach:

$$b_{i+1} = b_i - \frac{L'(b_i)}{L''(b_i)} \quad (3.8)$$

where $L'(b)$ denotes the first derivative and $L''(b)$ the second derivative of $L(b)$. This process is repeatedly applied to all branches in the tree until the branch lengths eventually converge [63].

To ensure non-negative branch lengths, we constrain the search using a minimum branch length threshold $minBranchLen$. For numerical reasons, the $minBranchLen$ should be greater than 0 [76]. Furthermore, constraint optimization problems are easier to solve when simple upper and lower bounds are given [76]. Therefore, in addition to the $minBranchLen$ threshold, we also set a reasonable maximum branch length using the threshold $maxBranchLen$.

3.4.3. Model Parameter Optimization

Commonly used methods for optimizing model parameters other than the branch lengths are Brent's method [6] and the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method [20]. The

objective is to find the model parameters ϕ that maximize the likelihood of the tree, meaning we try to find the values for the vector ϕ where $L'(\phi) = 0$. Using Brent's method, we can for example optimize the α value of the Γ distribution of rate heterogeneity [65]. The BFGS method can be used to optimize the equilibrium frequencies and substitution rates [40].

Brent method Brent's algorithm is based on the Dekker method [11]. Dekker combines the secant and the bisection method to find the root. Brent modified Dekker's approach to converge faster and with fewer iterations [6]. The Brent method optimizes one-dimensional variables. This means that we can only optimize a single parameter at a time [52].

BFGS method The BFGS method is a Quasi-Newton approach. That means that instead of computing the second derivative, it approximates the inverse Hessian matrix. This inverse Hessian matrix is used to determine the move direction during an iterative search for the minimum. Using the BFGS method we can optimize multiple parameters at once as this method can find the optimum of a multidimensional variable. In phylogenetics we typically use the L-BFGS-B variant. This variant is optimized for limited memory and is extended to incorporate bound constraints in variables [79]. If we use the L-BFGS-B method for model parameter optimization, one constraint is, for example, that the equilibrium frequencies must sum to 1: $\pi_A + \pi_T + \pi_C + \pi_G = 1$. With the threshold *bfgs_factor* we can control the convergence tolerance of the L-BFGS-B optimization.

3.5. Significance Tests

To compare the tree topologies and likelihood scores in our analyses, we conduct significance tests among the set of inferred trees. We further call this set of inferred trees the *candidate set*. The goal of these tests is to detect whether two likelihoods for two trees under the same model of evolution are significantly different, or different by random chance. The tests we use are therefore designed to compare likelihood values rather than the topologies themselves. This allows us to compare contradicting tree topologies [43].

The following section provides a short summary of selected topology significance tests. For further details, we refer the interested reader to Lemey *et al.* [43] and Yang [76].

All tests we describe in the following use the RELL method to generate bootstrap samples. The *resampling estimated log likelihoods (RELL)* method approximates the non-parametric bootstrap. The classical non-parametric bootstrap re-samples alignment columns and infers a tree from each sample, including all likelihood optimizations. This is however computationally too intensive [43]. The RELL method instead re-samples site-log-likelihoods that are stored during the likelihood optimization procedure. The likelihood of this sampled tree is then approximated by multiplying the sampled site-log-likelihoods according to Equation 3.6.

3.5.1. Kishino-Hasagawa Test

The Kishino-Hasagawa test (KH) [37] compares tree likelihood values based on the distribution of likelihood differences. To obtain a test distribution, the KH test generates at least 1000 bootstrap samples using the RELL method and centers their likelihoods using the mean likelihood value. The test compares the pairwise likelihood differences for all possible pairs in the

candidate set against the distribution of likelihood differences. In our analyses, we compare all candidate trees against the tree with the best log likelihood. Therefore, we use the one-sided KH test. Furthermore, we also use the weighted KH test. The weighted test scales the likelihood differences using the likelihood variance of the candidate set.

3.5.2. Shimodaira-Hasagawa Test

The Shimodaira-Hasegawa test (SH) [60] compares the candidate likelihoods with the best likelihood in the candidate set. Analogous to the KH test, the SH test generates a likelihood distribution based on at least 1000 bootstrap samples and centers the likelihoods using the mean likelihood value. The SH test compares the differences between the best tree and each candidate tree against the likelihood difference distribution. In our analyses, we use the standard SH test and also the weighted variant.

3.5.3. Approximately Unbiased Test

Strimmer and Rambaut [71] show that the SH test is biased by the number of trees in the candidate set. To overcome this bias, Shimodaira [61] suggested the Approximately Unbiased test (AU). This test is based on a *multiscale bootstrap*. Instead of sampling replicates from the original input length n , the *multiscale bootstrap* samples replicates of varying lengths. For each length n_r , we draw at least 10 000 bootstrap samples. Using the RELL method, we compute the likelihood approximation l_r and scale the likelihood according to the sampling length: $l = \frac{n}{n_r} l_r$. Then, we apply the same test statistic as with the SH test.

3.5.4. Expected Likelihood Weight Test

Analogous to the other tests we present, the Expected Likelihood Weight test (ELW) [71] first generates a number of bootstrap samples B and approximates the likelihoods using the RELL method. For each tree T_i in the candidate set and each bootstrap sample b , the likelihood of the tree is weighted according to the sum of the likelihoods of all trees: $w_b^i = \frac{l_b^i}{\sum_x l_b^x}$. Next, we average the weights of each tree over all bootstrap samples $\bar{w}^i = \frac{1}{B} \sum_{b=1}^B w_b^i$ and sort these weights in descending order. We compute the cumulative sum of the sorted weights, and accept trees until this sum exceeds a predefined confidence threshold.

3.6. Maximum Likelihood Tree Inference Tools

As explained in Section 3.3, an exhaustive search for the most likely tree is not feasible due to the large tree space. Tree inference tools using the ML method therefore typically implement iterative improvement techniques, which they apply to an initial starting tree. This initial topology is obtained by using other heuristic tree inference methods, such as neighbor joining [57], randomized stepwise addition order [8], or maximum parsimony [15, 18]. In the following, we present three widely used ML inference tools.

3.6.1. RAxML-NG

RAxML-NG [39] was presented in 2019 as a successor of the widely used phylogenetic inference tool RAxML [67]. The starting tree for RAxML-NG can be either a random or a parsimony tree. This initial tree is then optimized using a greedy hill-climbing algorithm. Here, greedy means that only steps improving the likelihood of the tree are accepted. This hill-climbing consists of multiple rounds of optimizing the model parameters, the branch lengths and the tree topology. RAxML-NG optimizes the branch lengths and model parameters using the Newton-Raphson, L-BFGS-B, and Brent methods (Sections 3.4.2 and 3.4.3). With the parameter *num_iters* we can limit the number of branch length optimization iterations. RAxML-NG iterates a maximum of *num_iters* times over all branch lengths. RAxML-NG implements SPR moves as topology optimization strategy (Section 3.4.1). To save computational time, the algorithm only tests re-attachments up to a maximum distance around the pruning point for likelihood improvements. RAxML-NG sets this distance depending on the likelihood improvements during the first round of SPR moves [40, 63].

Algorithm 1 summarizes the key steps during tree search to obtain a single tree.

Algorithm 1 RAxML-NG tree search procedure

- 1: Generate a starting topology ▷ random or parsimony
 - 2: Branch length and model parameter optimization
 - 3: Repeated *fast* SPR rounds ▷ fast = no branch length optimization
 - 4: Model parameter optimization
 - 5: Repeated *fast* SPR rounds
 - 6: Model parameter optimization
 - 7: Repeated *slow* SPR rounds ▷ slow = with branch length optimization
 - 8: Model parameter optimization
-

3.6.2. IQ-Tree

IQ-Tree [48] is a widely used software package for phylogenetics that was first released in 2014. Similar to RAxML-NG, IQ-Tree also implements a greedy hill-climbing ML tree search. To obtain an initial tree topology, the user can choose between random tree generation, a parsimony tree, or an improved version of neighbor joining called *BIONJ* [22]. IQ-Tree's topology optimization strategy consists in repeated NNI moves (Section 3.4.1). Since NNI moves explore the tree space less than SPR moves [43], IQ-Tree additionally implements random moves. Using these random moves, IQ-Tree explores the tree space beyond the NNI neighborhood. Given the number of taxa N , the *stochastic* NNI step applies $\frac{1}{2}(N - 3)$ random NNI moves on the candidate tree. If the likelihood increases, the new tree is accepted and further optimized through *hill-climbing* NNI steps. If the likelihood decreases, all applied NNI moves are discarded. If the tree did not improve during 100 *stochastic* NNI rounds, the tree search terminates.

IQ-Tree optimizes branch lengths and model parameters using the Newton-Raphson, L-BFGS-B and Brent methods (Sections 3.4.2 and 3.4.3).

The IQ-Tree tree search algorithm to obtain a single tree is summarized in Algorithm 2.

Algorithm 2 IQ-Tree tree search procedure

- 1: Generate a starting topology ▷ random, parsimony or BIONJ
 - 2: Branch length and model parameter optimization
 - 3: Repeated *stochastic* NNI steps
 - 4: Repeated *hill-climbing* NNI steps
 - 5: Model parameter optimization
-

3.6.3. FastTree

FastTree [53] was first presented in 2009, followed a few months later in 2010 by the improved version FastTree 2 [54]. FastTree aims to reduce the time and space complexity of ML phylogenetic inference. The authors achieve this through various heuristics and shortcuts compared to other inference tools.

Algorithm 3 summarizes the FastTree tree search procedure. FastTree builds an initial tree topology using a modified neighbor-joining algorithm. Instead of explicitly computing the pairwise distance matrix of the input MSA, FastTree implements a set of heuristics. This leads to a lower memory consumption, and is in practice faster than computing the distance matrix [53]. In contrast to RAxML-NG and IQ-Tree, FastTree implements a minimum evolution step before maximizing the tree’s likelihood. The minimum evolution approach tries to obtain a tree that explains the data with as few mutations as possible, therefore minimizing the branch lengths. The minimum evolution steps include NNI and SPR moves. Instead of iterating until convergence, FastTree runs a predefined number of rounds. During the maximum likelihood phase, FastTree uses NNI steps to improve the tree topology. FastTree stops the NNI rounds based on two heuristics¹ and executes at most $2 \cdot \log(N)$ rounds, where N is the number of taxa in the MSA.

Algorithm 3 FastTree tree search procedure

- 1: Generate a starting topology ▷ heuristic neighbor joining

Minimum Evolution

- 2: Perform $4 \cdot \log_2(N)$ NNI rounds ▷ N = number of taxa
- 3: Perform 2 SPR rounds

Maximum Likelihood

- 4: Branch length optimization
 - 5: Single NNI round
 - 6: Model parameter optimization
 - 7: Repeated NNI rounds ▷ using stopping heuristics
 - 8: Branch length optimization ▷ only once for each branch
-

¹The authors refer to these heuristics as *Subtree skipping* and *Star topology test*. For details, we refer the interested reader to the original publication [54].

3.7. Numerical Epsilon Thresholds

In order to limit the number of iterations during the optimization procedures, each tool defines one or more log likelihood epsilon threshold values as early stopping criteria. If the log likelihood increases by less than the threshold, the corresponding subprocedure terminates. In our analyses, we focus on three different epsilon thresholds.

Iteration Epsilon (*lh_epsilon*) If the log likelihood after one iteration of optimizing all parameters (tree topology, branch lengths and model parameters) increases by less than this value, the iteration stops. We can set this epsilon value in each of the three ML inference tools.

Model Epsilon (*model_epsilon*) This epsilon value is used during the optimization of the parameters of the substitution model for a fixed tree topology. If the log likelihood improves by less than this threshold, the model parameter optimization terminates. We can set this value in RAxML-NG and IQ-Tree.

SPR Epsilon (*spr_lh_epsilon*) In RAxML-NG after each SPR round the log likelihood improvement is compared to the SPR epsilon. If it did not improve by more than this threshold, the SPR moves terminate. This parameter is a RAxML-NG specific epsilon parameter.

Part II.

Numerical Properties of Maximum Likelihood Inference

4. Motivation

To infer trees under the ML method, developers implement different heuristics and numerical optimization methods. In the previous chapter, we have seen that these methods rely on internal numerical thresholds, for example the minimum branch length (*minBranchLen*) threshold. In their paper on the phylogenetic analysis of SARS-CoV-2, Morel *et al.* [49] notice that this *minBranchLen* threshold has an impact on the resulting ML scores. The authors observe this phenomenon for both RAxML-NG and IQ-Tree. In this part of the thesis, we investigate whether we can reproduce this effect on other datasets and for other numerical thresholds as well. In addition to the phylogenetic inference tools RAxML-NG and IQ-Tree, we also test for this effect in FastTree. Furthermore, we investigate the influence of the numerical thresholds on the runtime of phylogenetic inferences.

RAxML-NG and IQ-Tree both implement a tree evaluation option. During the standard tree search, the programs optimize the tree topology, the branch lengths and the substitution model parameters. During the tree evaluation, only the branch lengths and the substitution model parameters are optimized, while the tree topology remains fixed. In our analyses, we test the influence of the numerical thresholds on both: tree search and evaluation.

We show that by increasing the default values for RAxML-NG's and IQ-Tree's likelihood epsilon values during the tree search, we can speed up the tree inference, while achieving equally good results. For RAxML-NG we measure an average speedup of 1.9 ± 0.6 and for IQ-Tree 1.3 ± 0.4 .

By comparing the ML scores and runtimes of RAxML-NG, IQ-Tree, and FastTree, we show that while FastTree is by far the fastest tool, RAxML-NG in general outperforms IQ-Tree and FastTree in terms of the absolute best ML score (Section 7).

In Section 5, we explain the workflow of our experiments, the datasets we analyze, and the numerical thresholds we investigate. In Section 6, we present our analysis on the influence of the numerical thresholds, with a focus on the likelihood epsilon values and the *minBranchLen* setting. Finally, in Section 7, we compare the ML scores and runtimes for the three inference tools RAxML-NG, IQ-Tree, and FastTree on different datasets and discuss the benefit of the evaluation functionality in RAxML-NG and IQ-Tree (Section 7.2).

5. Experimental Setup

We analyze the influence of distinct numerical thresholds on multiple datasets for RAxML-NG, IQ-Tree, and FastTree. In order to run the tree inferences in an automated manner, we implement a data generation pipeline. This pipeline takes an MSA and numerical threshold values as input, and outputs the ML scores and runtimes for all three inference tools under the different numerical threshold settings. We explain our pipeline in greater detail in Section 5.1. In Sections 5.2 and 5.3, we present the numerical thresholds and datasets we analyze.

5.1. Data Generation Pipeline

Given an MSA and numerical threshold values, our data generation pipeline runs phylogenetic tree inferences under all possible combinations of these values. Figure 5.1 summarizes the workflow of the pipeline with two toy numerical thresholds *Thresh1* and *Thresh2* on one dataset and for one ML inference tool. For each threshold value combination, we conduct five tree searches, each with a single starting tree, resulting in five final ML trees. We call this part of our pipeline the *tree search phase*. We select the best among these trees based on their ML scores. For RAxML-NG and IQ-Tree, we also re-evaluate this best tree. This means that the optimization algorithm does not alter the tree topology, but only optimizes the substitution model parameters and branch lengths. We call this phase the *evaluation phase*. For FastTree, we omit this evaluation phase, since FastTree does not provide an evaluation functionality analogous to RAxML-NG and IQ-Tree. Instead, we execute all following steps on the best tree after the tree search phase. We collect all re-evaluated trees in a *candidate tree set*. This candidate tree set has as many trees as there are threshold combinations. We run all significance tests as implemented in IQ-Tree on this candidate tree set. Due to the continuing debate about the most appropriate significance test for tree comparison (see Section 3.5), we use the same approach as Morel *et al.* [49] and include only those trees passing *all* significance tests into a so-called *plausible tree set*. Our pipeline stores the results in a database containing all ML scores and runtimes for the candidate tree set and the plausible tree set.

During the tree search phase and the evaluation phase, we use the GTR model with four discrete Γ rate categories as substitution model (Section 3.1). To ensure reproducibility, we set the seeds for the 5 tree searches to 0–4 and the seed for the evaluation and IQ-Tree significance tests to 0. FastTree, in its original implementation, does not allow for manual threshold setting. We therefore add command line switches for the thresholds we test. The adapted code is available at <https://www.github.com/tschuelia/param-fasttree>. We also use a modified RAxML-NG version, in which the thresholds *spr_lh_epsilon*, *num_iters*, and *bfgs_factor* can be set via the command line. This modification was not part of this thesis and is available at <https://www.github.com/lukashuebner/param-raxml-ng>. For the statistical tests, we use the default IQ-Tree settings for the number of bootstrap RELL replicates (10 000) and the

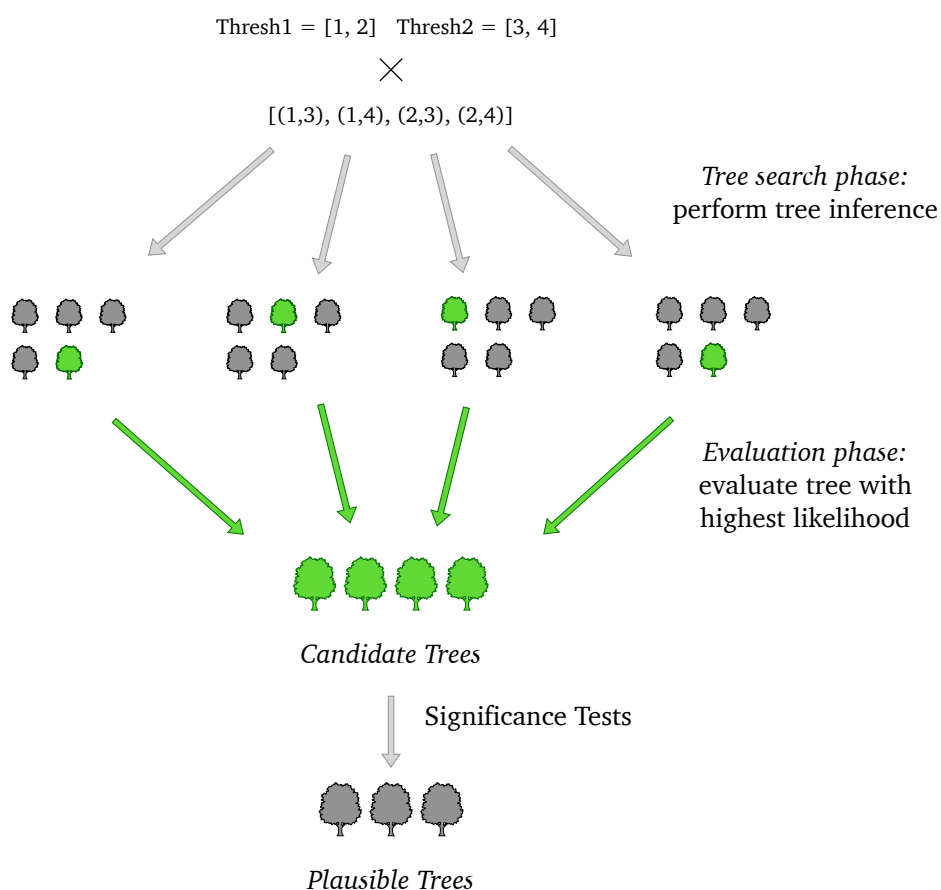


Figure 5.1.: The workflow of our data generation pipeline for one dataset and one tree inference tool. For each value combination of Thresh1 and Thresh2, we run 5 tree searches with the respective values. The tree with the highest ML score after the tree search phase (the highlighted tree) is then re-evaluated. After the evaluation phase, this tree is included in a set of *candidate trees*. We compare all trees in this candidate tree set using the IQ-Tree significance tests. Trees passing *all* tests form part of the set of *plausible trees*.

significance level ($\alpha = 0.05$). As stated in Section 3.5, some of the tests are biased by the number of trees in the candidate set. Therefore, before running the statistical tests on the candidate tree set, we filter duplicate tree topologies. Since we conduct the five tree searches independently, we use a single worker with two threads for each search as parallelization approach. The same applies for the tree evaluation phase.

We implement our data generation pipeline using the Snakemake workflow management system [38] and Python 3. We execute our data generation pipeline on two institutional clusters (Cascade and Haswell) at the Heidelberg Institute for Theoretical Studies (HITS) and one server of our research group. The Cascade cluster consists of 150 compute nodes with Intel Cascade Lake CPUs (Intel Xeon Gold 6230). Each CPU has 20 cores running at 2.1 GHz and 96GB RAM. The Haswell cluster has 224 nodes with Intel Haswell CPUs (E5-2630v3), each with 16 cores running at 2.4 GHz and 64 GB RAM. The server of our research group is a Cascade Lake CPU (Xeon Platinum 8260) with 48 cores running at 2.4 GHz and 754 GB RAM.

5.2. Numerical Thresholds

Depending on the tree inference tool, we examine different numerical thresholds (Chapter 3). In Table 5.1 we present the value ranges we test for each threshold and to what ML inference tool they are applicable, alongside the respective default value.

Threshold (Abbreviation)	Tested Values	Inference Tools (resp. default value)
Minimum Branch Length (<i>minBranchLen</i>)	$\{10^{-10}, 10^{-9}, \dots, 10^{-2}\}^*$	RAxML-NG (10^{-6}) IQ-Tree (10^{-6}) FastTree (5^{-9})
Maximum Branch Length (<i>maxBranchLen</i>)	$\{10, 10^2\}$	RAxML-NG (10^2) IQ-Tree (10)
Iteration Epsilon (<i>lh_epsilon</i>)	$\{10^{-3}, 10^{-2}, \dots, 10^3\}$	RAxML-NG (10^{-1}) IQ-Tree (10^{-3}) FastTree (10^{-1})
Model Epsilon (<i>model_epsilon</i>)	$\{10^{-3}, 10^{-2}, 10^{-1}\}$	RAxML-NG (10^{-3}) IQ-Tree (10^{-2})
SPR Epsilon (<i>spr_lh_epsilon</i>)	$\{10^{-3}, 10^{-2}, \dots, 10^3\}$	RAxML-NG (10^{-1})
Maximum number of branch length optimization iterations (<i>num_iters</i>)	$\{16, 32, 64\}$	RAxML-NG (32)
Convergence threshold for the BFGS optimization (<i>bfgs_factor</i>)	$\{10^5, 10^7, 10^9\}$	RAxML-NG (10^7)

*For FastTree we additionally run its default value 5^{-9} .

Table 5.1.: Varied numerical thresholds with value ranges and applicable inference tools. The values in braces state the default value for the respective inference tool.

5.3. Datasets

We investigate the numerical behavior of the thresholds on 22 datasets with 27 to 4869 taxa. In the following, we refer to the datasets using the number of taxa preceded by a “D”. Table 5.2 provides an overview of the datasets we use.

Name	# Taxa / # Sites	Notes / Reference
D27	27 / 1940	Data of the 28s rRNA gene of a broad taxonomic diversity [31].
D37	37 / 1 338 678	Data of 447 nuclear genes of <i>Eutheria</i> (often referred to as <i>SongD1</i>) [62].
D46	46 / 239 763	Data of 310 nuclear genes of seed plants (often referred to as <i>XiD4</i>) [74].
D101	101 / 1858	rRNA gene data of microsporidia [70].
D125	125 / 29 149	Mammalian DNA sequences [69].
D150	150 / 1269	rRNA gene data of microsporidia [70].
D218	218 / 2294	Prokaryotic sequences from the small ribosomal unit [29].
D354	354 / 460	Internal transcribed spacer (ITS) region of nuclear ribosomal DNA of <i>Acer</i> [28].
D500	500 / 1398	rbcL gene data (often referred to as <i>zilla</i> dataset) [9].
D714	714 / 1241	Dataset used for benchmarking ML inference tools [69].
D1288	1288 / 1200	Mammalian sequence data [69].
D1481	1481 / 1241	Dataset used for benchmarking ML inference tools [69].
D1512	1512 / 1577	Dataset used for benchmarking ML inference tools [69].
D1604	1604 / 1276	Dataset used for benchmarking ML inference tools [69].
D1718	1718 / 1371	rbcL gene data [68].
D1908	1908 / 1424	Fungal sequence data [69].
D2000	2000 / 1251	Dataset used for benchmarking ML inference tools [69].
D2308	2308 / 1224	Mammalian sequence data [69].
D2445	2445 / 1371	rbcL gene data [68].
D2554	2554 / 1232	rbcL gene data [69].
D3782	3782 / 1371	rbcL gene data [68].
D4869	4869 / 28 361	SARS-CoV2 data (snapshot 05/05/2020 obtained from gisaid.org)

Table 5.2.: Overview of the datasets we use for our analyses. All datasets are empirical datasets.

6. Influence of Numerical Thresholds on Likelihoods and Runtimes

In this chapter, we analyze the influence of the numerical thresholds on the ML scores and runtimes of RAxML-NG, IQ-Tree, and FastTree. This chapter has two parts:

1. Evaluation phase: In the first part, we focus on the influence of the numerical threshold variation during the evaluation phase. We show that the ML score and runtime are largely unaffected by the numerical threshold settings, as long as they are within a reasonable value range. Both RAxML-NG's and IQ-Tree's default values are in this range.

2. Tree search phase: In the second part, we focus on the influence of the numerical threshold variation during the tree search phase. Based on our findings, we suggest an increase of the default value for *lh_epsilon* in RAxML-NG and IQ-Tree, as well as an increase of the default value for RAxML-NG's *spr_lh_epsilon*. We demonstrate that these changes substantially improve the tree inference time, while yielding equally good results. We further show that the default values for the remaining numerical thresholds are suitable for all three ML inference tools.

It is important to note that we compare ML scores within a broad range of absolute likelihood values. The ML scores for the 22 empirical datasets we test range between approximately -6400 (D354) and $-12\,300\,000$ (D4869). Therefore, we compare likelihood improvement and degradation in percent rather than absolute log likelihood units. Since the ML scores are a log scale, however, the observed effects are greater than the percentages suggest.

For brevity, we only depict the influences of selected numerical thresholds. For the sake of completeness, we present the results of all thresholds in Appendix Section A.1.2. Furthermore, all figures, as well as more detailed figures for each dataset, are available as interactive figures at <https://www.thesis.juliahaag.de>.

6.1. Evaluation Phase

Morel *et al.* [49] notice a correlation of ML scores with the *minBranchLen* threshold setting when re-evaluating trees with RAxML-NG and IQ-Tree (Figure 6.1). We test for this phenomenon on different datasets and test for all other thresholds we present in Section 5.2. One exception is the *spr_lh_epsilon* threshold. RAxML-NG uses this specific threshold during the SPR rounds (for details, see Section 3.6). Note that this threshold is not used during the evaluation phase, when the topology is not optimized. During the tree search phase, we set the numerical thresholds to their respective default value, as presented in Table 5.1.

For the thresholds *minBranchLen* and *lh_epsilon* we observe an effect on ML scores and runtimes of RAxML-NG and IQ-Tree. For the remaining thresholds, we either notice no effect, or only a minor impact on runtimes. Based on our findings, we suggest a range of reasonable

values for each numerical threshold. For both tools, the respective default values are in this range.

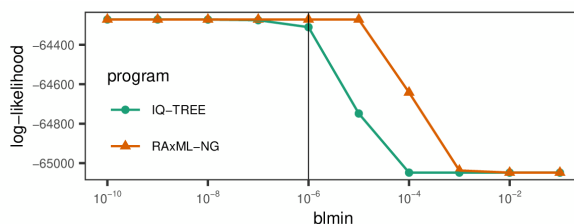


Figure 6.1.: Morel *et al.* [49] observe worse ML scores after re-evaluating a tree with RAxML-NG and IQ-Tree under higher *minBranchLen* settings.

6.1.1. Minimum Branch Length

RAxML-NG

Analogous to Morel *et al.* [49], we observe a correlation between the ML scores and the *minBranchLen* setting. However, as depicted in Figure 6.2a, we observe continuously worse ML scores with larger *minBranchLen* thresholds rather than distinct likelihood plateaus. We observe worse ML scores in the same order of magnitude as the authors. Morel *et al.* [49] report ML scores of up to 1.54 % worse than the best score, in our analysis we observe degradations between 0.1 % and 2.0 % with a mean of 1.55 % and two outliers (5.1 % worse for D354 and 12.5 % worse for D4869). For values $\leq 10^{-5}$ we observe, except for D4869, equally good ML scores. The *minBranchLen* threshold during the evaluation phase should therefore be set to $\leq 10^{-5}$. As Figure 6.2b shows, the runtimes for *minBranchLen* values $\leq 10^{-5}$ are approximately identical.

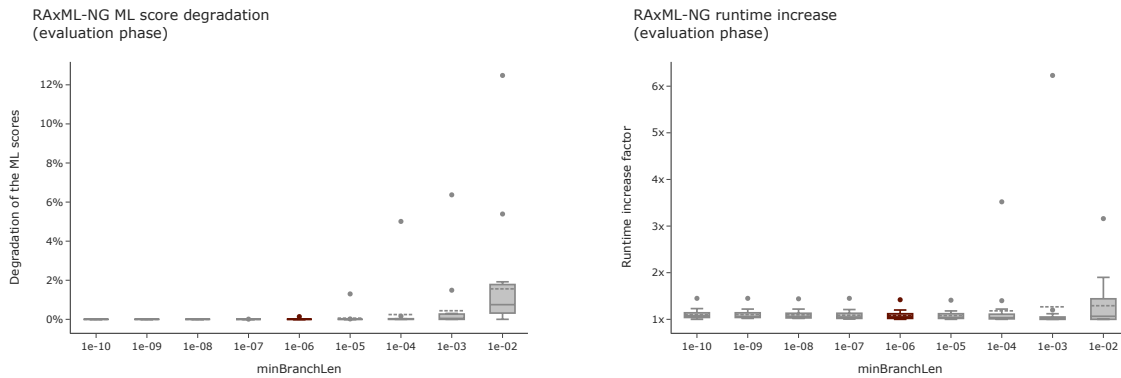
IQ-Tree

The runtimes for the IQ-Tree evaluation phase improve for *minBranchLen* settings $\geq 10^{-4}$. Averaged over all datasets, the evaluation phase with IQ-Tree’s default *minBranchLen* value 10^{-6} runs 1.4 ± 0.3 times longer than for 10^{-2} . For settings $\leq 10^{-4}$ we observe no clear trend in runtimes (Figure 6.3b). On 4 datasets, we observe worse ML scores for *minBranchLen* $\geq 10^{-3}$ (≤ 0.3 %; Figure 6.3a). We conclude that *minBranchLen* should be set to $\leq 10^{-4}$ during the IQ-Tree evaluation phase.

6.1.2. Likelihood Epsilon

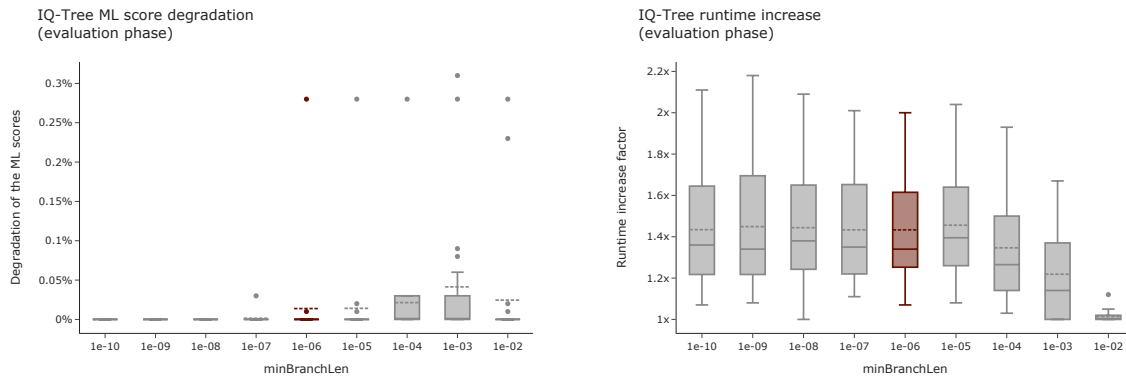
For IQ-Tree, the likelihood threshold *lh_epsilon* has no influence on either runtime or ML scores (log-likelihood variances are 0.0 and runtime variances ≤ 1 %). For RAxML-NG, we observe up to 0.2 % worse ML scores for values ≥ 10 and one extreme case of 1.5 % for *lh_epsilon* = 10^3 (D1288; Figure 6.4a). With smaller *lh_epsilon* threshold values, we observe increased runtimes (Figure 6.4b). Tree re-evaluation under RAxML-NG’s default *lh_epsilon* = 10^{-1} are on average 4 times slower than under *lh_epsilon* = 10^3 .

6. Influence of Numerical Thresholds on Likelihoods and Runtimes



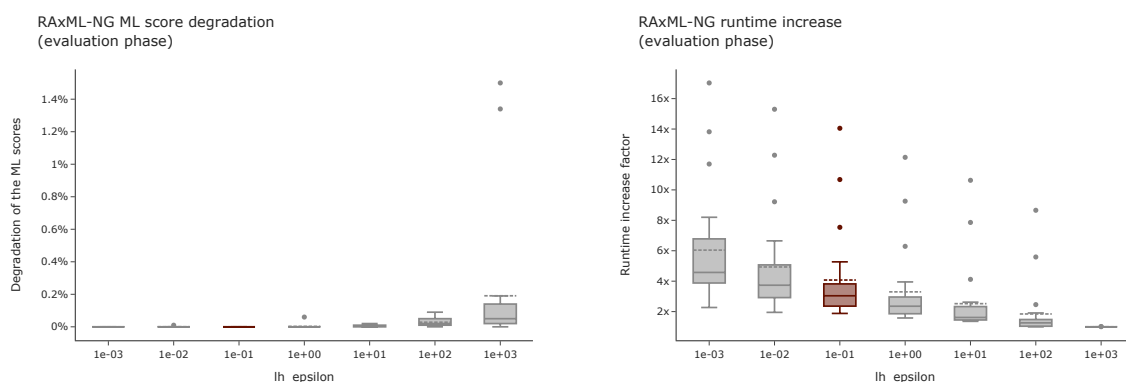
- (a) Degradation of ML scores across all datasets as a function of \minBranchLen values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores.
- (b) Increase of the evaluation time across all datasets as a function of \minBranchLen values. The increase is measured relative to the minimum runtime per dataset.

Figure 6.2.: Influence of the \minBranchLen threshold on the ML scores and runtimes of the RAxML-NG evaluation phase. The runtimes refer to the runtime of the evaluation phase. The plots summarize the data over all datasets. The dashed vertical line indicates the mean, and the solid vertical line the median value. The highlighted box indicates the default \minBranchLen value for RAxML-NG.



- (a) Degradation of ML scores across all datasets as a function of \minBranchLen values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores.
- (b) Increase of the evaluation time across all datasets as a function of \minBranchLen values. The increase is measured relative to the minimum runtime per dataset.

Figure 6.3.: Influence of the \minBranchLen threshold on the ML scores and runtimes of the IQ-Tree evaluation phase. The runtimes refer to the runtime of the evaluation phase. The plots summarize the data over all datasets. The dashed vertical line indicates the mean, and the solid vertical line the median value. The highlighted box indicates the default \minBranchLen value for IQ-Tree.



(a) Degradation of ML scores across all datasets as a function of $lh_epsilon$ values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores.

(b) Increase of the evaluation time across all datasets as a function of $lh_epsilon$ values. The increase is measured relative to the minimum runtime per dataset.

Figure 6.4.: Influence of the $lh_epsilon$ threshold on the ML scores and runtimes of the RAxML-NG evaluation phase. The runtimes refer to the runtime of the evaluation phase. The plots summarize the data over all datasets. The dashed vertical line indicates the mean, and the solid vertical line the median value. The highlighted box indicates the default $lh_epsilon$ value for RAxML-NG.

6.1.3. Remaining Thresholds

The thresholds $model_epsilon$, num_iters , and $bfgs_factor$ have no impact on the ML score. However, the runtimes for $model_epsilon$ increase with smaller $model_epsilon$ settings (on average 10.5 % increase for RAxML-NG, and 20 % for IQ-Tree). The $bfgs_factor$ threshold shows a similar effect: RAxML-NG runtimes increase on average 49 % with lower values. The runtimes for the num_iters threshold increase for more iterations (on average 3.7 %). We observe no impact on neither ML scores nor runtime for the $maxBranchLen$ threshold in RAxML-NG. For IQ-Tree we notice runtime variations ≤ 20 %. However, depending on the dataset, a different $maxBranchLen$ value yields faster execution times. The ML score is unaffected. The data for these thresholds is not shown in this chapter, we refer the interested reader to Appendix A.1.2 or the interactive plots on our website at <https://www.thesis.juliahaag.de/numericalProperties/influenceEval>.

Based on our findings, we suggest the following threshold settings during the evaluation phase:

Threshold	RAxML-NG	IQ-Tree
<i>minBranchLen</i>	$\leq 10^{-5}$	$\leq 10^{-4}$
<i>maxBranchLen</i>	$\in \{10, 10^2\}$	$\in \{10, 100\}$
<i>lh_epsilon</i>	≤ 10	≤ 1000
<i>model_epsilon</i>	≤ 0.1	≤ 0.1
<i>num_iters</i>	$\in \{16, 32, 64\}$	–
<i>bfgs_factor</i>	$\in \{10^5, 10^7, 10^9\}$	–

For both tools, the respective default values fulfill these criteria. Since the runtime of the evaluation phase is negligible compared to the tree search phase (Section 7.2), we set all numerical thresholds to their most conservative setting according to Table 5.1 during the evaluation phase, despite their longer runtimes. In the following, we investigate the influence of the numerical thresholds during the tree search phase more thoroughly.

6.2. Tree Search Phase

In this section, we focus on the influence of the numerical thresholds on the ML scores and runtimes, when varied during tree search phase. We specifically focus on the likelihood thresholds *lh_epsilon* and *spr_lh_epsilon*, and demonstrate a potential speedup of the tree search for RAxML-NG and IQ-Tree by changing these default values. We further discuss ML score and runtime variations caused by the *minBranchLen* threshold and show that the default values for all three inference tools are appropriate. For the remaining numerical thresholds, we only observe minor influences on ML scores and runtimes.

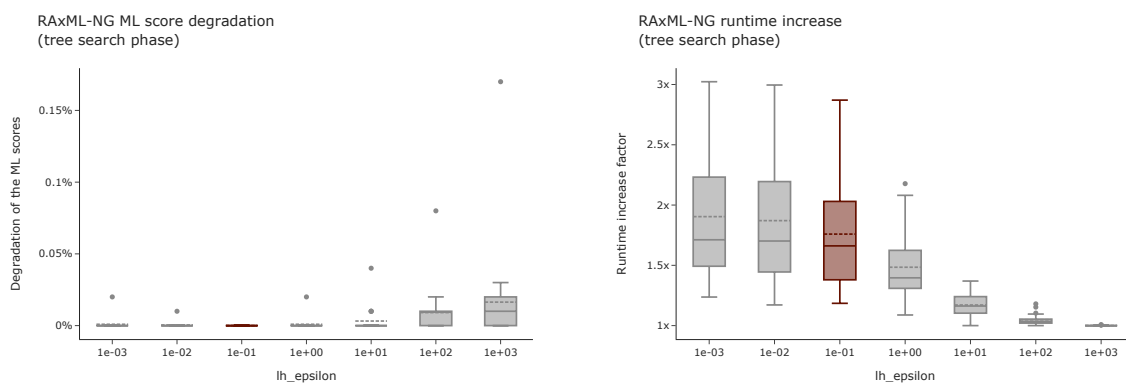
6.2.1. Likelihood Epsilons

The threshold with the highest runtime impact is the likelihood epsilon value *lh_epsilon*. We observe an impact for all three inference tools. The threshold *spr_lh_epsilon* also influences the runtime of RAxML-NG. For both thresholds, higher values improve the runtime. We demonstrate that increasing these likelihood epsilon values for RAxML-NG and IQ-Tree leads to equally good results with lower runtime, and therefore suggest the use of new default values. For FastTree, we observe that the default *lh_epsilon* value is appropriate.

RAxML-NG

The runtime of tree inferences using RAxML-NG improves for higher values of *lh_epsilon* (Figure 6.5b). Tree searches with the default setting 10^{-1} run on average 1.8 ± 0.5 times longer than with *lh_epsilon* = 10^3 . The ML scores for most datasets slightly worsens for settings ≥ 10 . Except for D354, the variances between different *lh_epsilon* settings are $\leq 0.03\%$ (Figure 6.5a). Given these observations, we form the hypothesis that the default *lh_epsilon* value for RAxML-NG should be increased from 10^{-1} to 10.

To support this hypothesis, we compute the proportions of *lh_epsilon* values among trees in the plausible tree set. Equal proportions for all *lh_epsilon* values indicate that this threshold does not influence whether a tree search returns a plausible tree. Figure 6.6 shows the proportions of



- (a) Degradation of ML scores across all datasets as a function of $lh_epsilon$ values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores.
- (b) Increase of the tree search time across all datasets as a function of $lh_epsilon$ values. The increase is measured relative to the minimum runtime per dataset.

Figure 6.5.: Influence of the $lh_epsilon$ threshold on the ML scores and runtimes of the RAxML-NG tree search phase. The ML scores refer to the ML scores *after* the evaluation phase. The runtimes refer to the runtime of the tree search phase. The plots summarize the data over all datasets. The dashed vertical line indicates the mean, and the solid vertical line the median value. The highlighted box indicates the default $lh_epsilon$ value for RAxML-NG.

$lh_epsilon$ values for different datasets. Analogous to the ML scores, we observe that $lh_epsilon$ values ≤ 10 return approximately equally frequently a plausible tree (with differences of less than 5 percentage points). To further test our hypothesis, we also run our data generation pipeline with 15 simulated datasets with 22–2288 taxa. We compare the topological distances of the resulting trees for different $lh_epsilon$ values to the true tree. Figure 6.7 shows these topological distances across all 15 simulated datasets. The y-axis depicts the variations of topological distances relative to the average distance across all $lh_epsilon$ settings per dataset. The variances for all $lh_epsilon$ settings are less than 1 percentage point. The distances of RAxML-NG’s default value $lh_epsilon = 10^{-1}$ vary more than the distances for our new suggested value $lh_epsilon = 10$. We conclude that both $lh_epsilon$ settings lead to equally good trees.

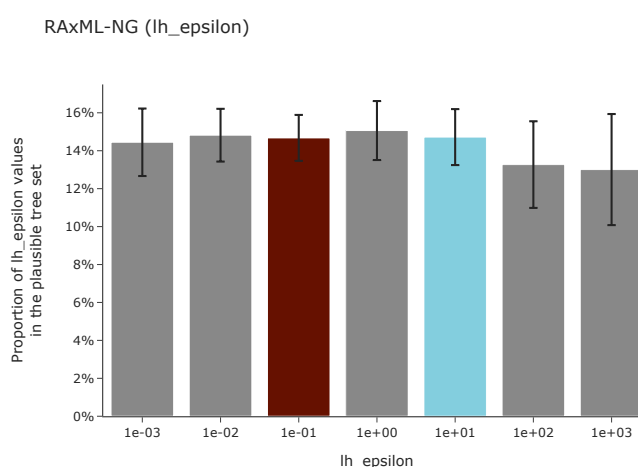


Figure 6.6.: Average proportions of $lh_epsilon$ values among trees in the plausible tree set for RAxML-NG over all datasets. The darkest bar indicates the current default value (10^{-1}) and the brightest bar our suggested new default value (10). The error bar depicts the standard deviation.

With RAxML-NG, we also investigate the influence of the $spr_lh_epsilon$ threshold. Similar to the $lh_epsilon$ threshold, the runtimes for $spr_lh_epsilon$ improve with higher values (Figure 6.8b). This threshold has no influence on the ML scores after the evaluation phase (Figure 6.8a). Analogous to our further analysis of the $lh_epsilon$ threshold, we compute the proportions of $spr_lh_epsilon$ values among the plausible trees. With a variance of less than 5 percentage points for different $spr_lh_epsilon$ values (Figure 6.9), we conclude that the $spr_lh_epsilon$ threshold does not influence whether a tree search returns a plausible tree. The topological distances are unaffected by the $spr_lh_epsilon$ threshold, with variances less than 1 percentage point. As Figure 6.10 shows, the variances for $spr_lh_epsilon = 10^3$ are higher than for $spr_lh_epsilon = 10^{-1}$, but the RF-distance is better on average. We therefore conclude that the default value for the RAxML-NG $spr_lh_epsilon$ threshold could be increased from 10^{-1} to 10^3 .

Considering the suggested changes of both likelihood epsilon thresholds, we compute the speedup of all tree searches with the suggested new setting relative to the tree searches with the current default settings. To ensure comparability, we only compare tree searches with the exact same threshold settings for the remaining varied thresholds. Across all empirical datasets,

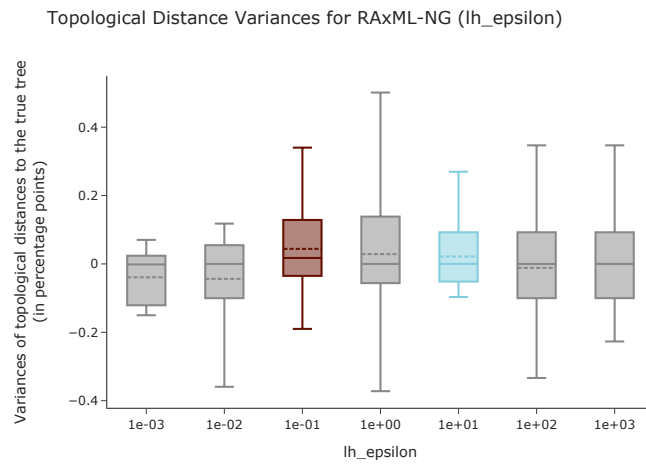
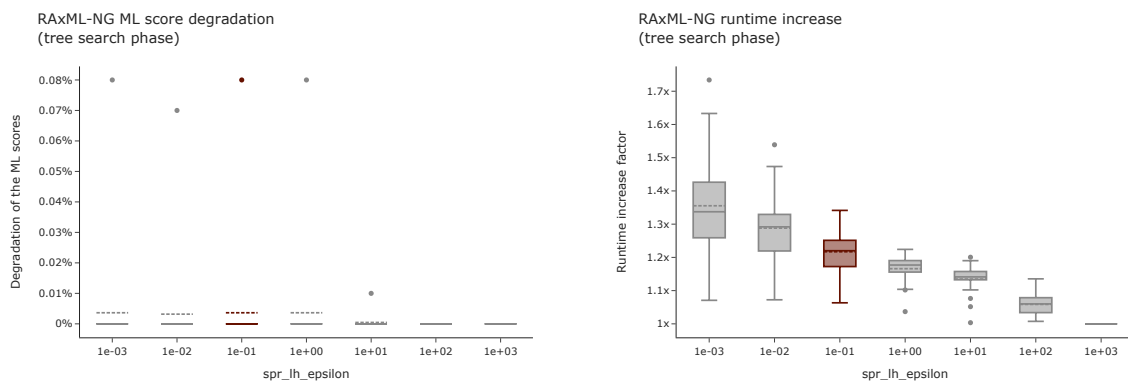


Figure 6.7.: Variances of topological distances to the true tree per $lh_epsilon$ setting on 15 simulated datasets. The y-axis shows the variations in percentage points relative to the average topological distance per dataset.



(a) Degradation of ML scores across all datasets as a function of $spr_lh_epsilon$ values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores. (b) Increase of the tree search time across all datasets as a function of $spr_lh_epsilon$ values. The increase is measured relative to the minimum runtime per dataset.

Figure 6.8.: Influence of the $spr_lh_epsilon$ threshold on the ML scores and runtimes of the RAxML-NG tree search phase. The ML scores refer to the ML scores *after* the evaluation phase. The runtimes refer to the runtime of the tree search phase. The plots summarize the data over all datasets. The dashed vertical line indicates the mean, and the solid vertical line the median value. The highlighted box indicates the default $spr_lh_epsilon$ value for RAxML-NG.

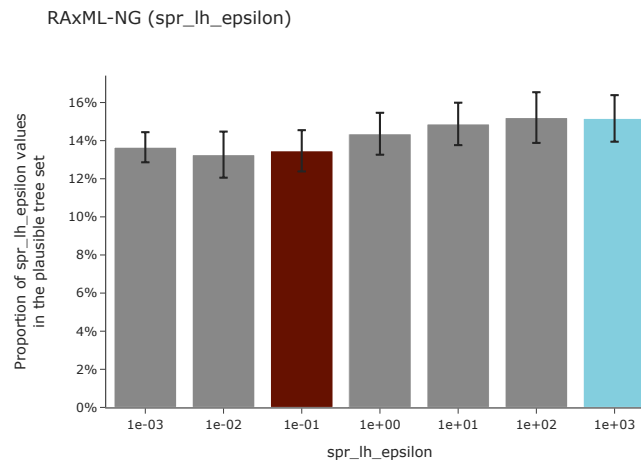


Figure 6.9.: Average proportions of $spr_lh_epsilon$ values among trees in the plausible tree set for RAxML-NG over all datasets. The darkest bar indicates the current default value (10^{-1}) and the brightest bar our suggested new default value (10^3). The error bar depicts the standard deviation.

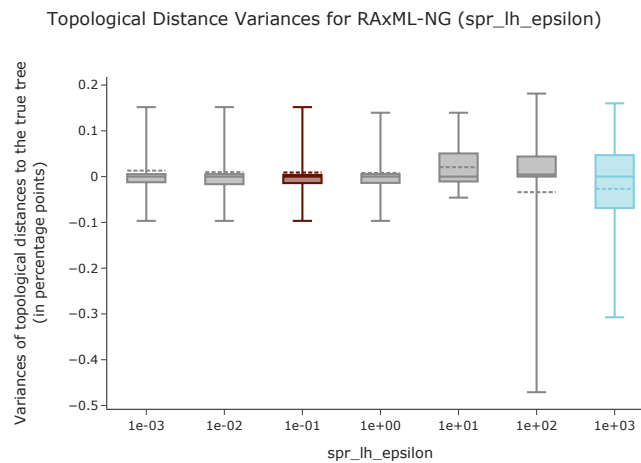


Figure 6.10.: Variances of topological distances to the true tree per $spr_lh_epsilon$ setting on 15 simulated datasets. The y-axis shows the variations in percentage points relative to the average topological distance per dataset.

we observe an average speedup of 1.9 ± 0.6 , with 97.5 % of all runs showing a speedup ≥ 1 . In 34.5 % of all tree searches, the runtime with the new settings was twice as fast as with the current default setting. In our analysis, more tree searches result in a speedup ≥ 3 (3.5 %) than in a speedup < 0.9 (1.2 %) (Figure 6.11).

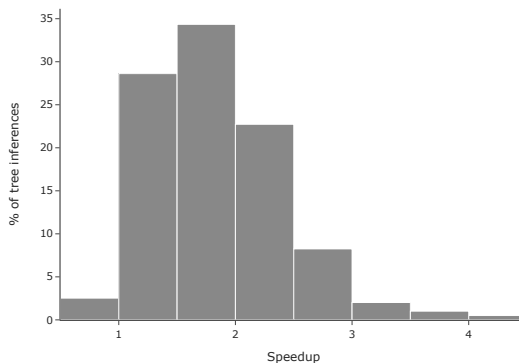


Figure 6.11.: Distribution of speedups across all datasets for RAxML-NG. We compare runtimes for tree searches under the default setting $lh_epsilon = 10^{-1}$ and $spr_lh_epsilon = 10^{-1}$, with tree searches under our new suggested settings $lh_epsilon = 10$ and $spr_lh_epsilon = 10^3$.

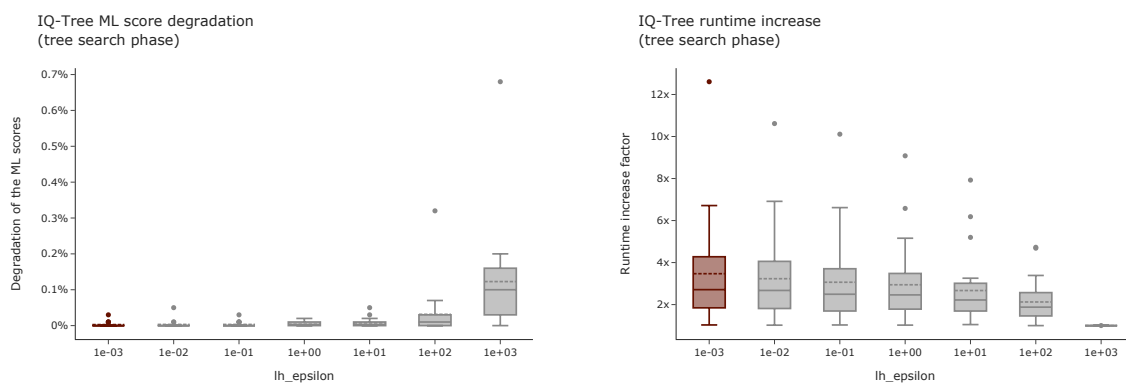
It is worth mentioning that these observations hold true independent of the absolute likelihood value. The ML scores for the datasets we test, range between approximately -6400 (D354) and $-12\,300\,000$ (D4869). It is, however, important to note that the evaluation phase after the tree search phase should not be omitted and executed with a small $lh_epsilon$ value (for example 10^{-3}).

Given these observations, we conclude that the default value for the $lh_epsilon$ threshold in RAxML-NG should be raised to 10 and the default value for the $spr_lh_epsilon$ threshold should be raised to 10^3 .

IQ-Tree

Analogous to RAxML-NG, the runtimes improve with higher $lh_epsilon$ thresholds. Tree searches with IQ-Tree's default threshold $lh_epsilon = 10^{-3}$ run on average thrice as long as tree searches with $lh_epsilon = 10^3$ (Figure 6.12b). However, IQ-Tree appears to be more sensitive to the $lh_epsilon$ threshold than RAxML-NG in terms of ML scores. Under higher $lh_epsilon$ settings, the ML score degradations are an order of magnitude worse than for RAxML-NG ($\leq 0.2\%$ for IQ-Tree vs. $\leq 0.03\%$ for RAxML-NG). For $lh_epsilon$ values ≤ 10 the ML scores are on average equally good. Hence, we suggest that the default $lh_epsilon$ threshold for IQ-Tree should be raised from 10^{-3} to 10.

To test this hypothesis, we use the same metrics as for RAxML-NG. First, we compute the proportions of $lh_epsilon$ values among trees in the plausible tree set. On average, the proportions for $lh_epsilon = 10^{-3}$ and $lh_epsilon = 10$ are approximately equal among each other (Figure 6.13). We further compute the topological distances to the true tree for the 15



- (a) Degradation of ML scores across all datasets as a function of $lh_epsilon$ values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores.
- (b) Increase of the tree search time across all datasets as a function of $lh_epsilon$ values. The increase is measured relative to the minimum runtime per dataset.

Figure 6.12.: Influence of the $lh_epsilon$ threshold on the ML scores and runtimes of the IQ-Tree tree search phase. The ML scores refer to the ML scores *after* the evaluation phase. The runtimes refer to the runtime of the tree search phase. The plots summarize the data over all datasets. The dashed vertical line indicates the mean, and the solid vertical line the median value. The highlighted box indicates the default $lh_epsilon$ value for IQ-Tree.

simulated datasets. For $lh_epsilon$ values ≤ 10 the distances are approximately equal with variances less than 1.5 percentage points (Figure 6.14).

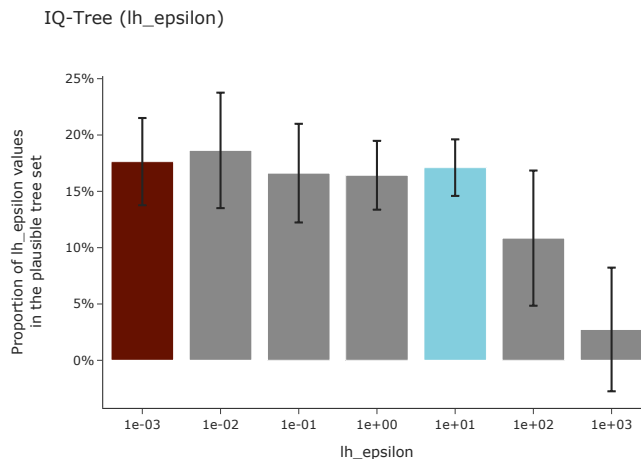


Figure 6.13.: Average proportions of $lh_epsilon$ values among trees in the plausible tree set for IQ-Tree over all datasets. The darkest bar indicates the current default value (10^{-3}) and the brightest bar our suggested new default value (10). The error bar depicts the standard deviation.

Implementing the suggested changes in IQ-Tree’s default $lh_epsilon$ threshold leads to an average speedup of 1.3 ± 0.4 with 74 % of all tree searches showing a speedup ≥ 1 . For 19 % of all tree searches, the speedup is ≥ 1.5 , but for 11 % we observe an increase in runtime with a speedup < 0.9 (Figure 6.15). We conclude that increasing the default value for $lh_epsilon$ in IQ-Tree results in equally likely trees with on average lower runtimes.

As described above, we observe noticeably worse ML scores for $lh_epsilon \geq 10^2$. This is reflected by the proportions of these $lh_epsilon$ values in the set of plausible trees (Figure 6.13). We suspect that this is caused by the random NNI moves in IQ-Tree’s search algorithm (Section 3.6.2). To verify this assumption, we modify IQ-Tree and disable the randomness in the search algorithm. As a consequence, IQ-Tree only optimizes the tree topology using standard NNI moves. We refer to the standard IQ-Tree as *random IQ-Tree* and to the IQ-Tree algorithm without random NNI moves as *de-randomized IQ-Tree*. We run our data generation pipeline on four datasets (D1418, D1604, D1718, and D2445) using the de-randomized IQ-Tree. Since standard NNI moves are prone to be stuck in a local optimum (Section 3.6.2), we expect the ML scores for de-randomized IQ-Tree to be worse than for random IQ-Tree, which we indeed observe in our analyses. To compare the influence of the $lh_epsilon$ threshold, we again compute the proportion of $lh_epsilon$ values among trees in the plausible tree set. Figure 6.16 compares the proportions for the four datasets using the random IQ-Tree (left figure) and using the de-randomized IQ-Tree (right figure). We see that, in contrast to random IQ-Tree, de-randomized IQ-Tree returns plausible trees under all $lh_epsilon$ thresholds for all four datasets.

In order to quantify these differences between the two IQ-Tree versions, we compute the earth mover’s distance between the histograms of $lh_epsilon$ values in the plausible tree set per dataset, and further compare them to a flat histogram. A flat histogram indicates that the $lh_epsilon$

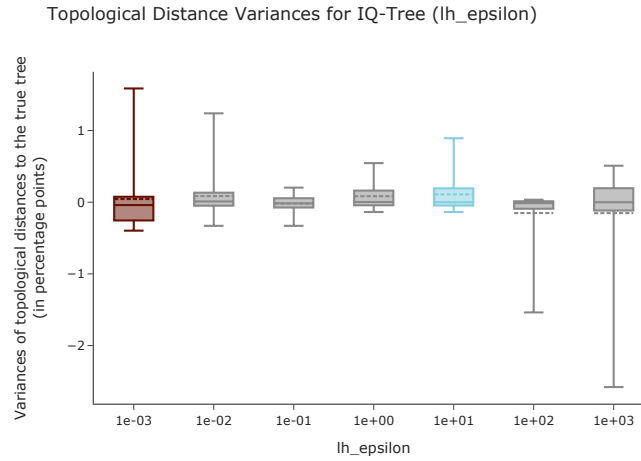


Figure 6.14.: Variances of topological distances to the true tree per $lh_epsilon$ setting on 15 simulated datasets. The y-axis shows the variations in percentage points relative to the average topological distance per dataset.

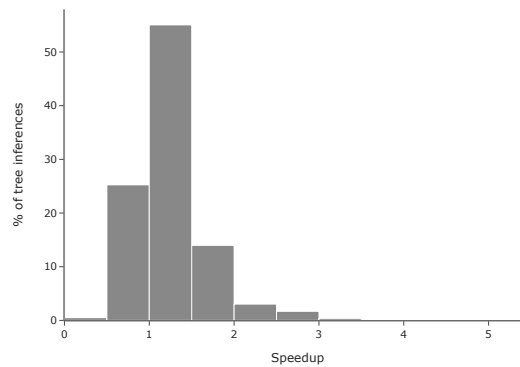


Figure 6.15.: Distribution of speedups across all datasets for IQ-Tree. We compare runtimes for tree searches under the default setting $lh_epsilon = 10^{-3}$, with tree searches under our new suggested setting $lh_epsilon = 10$.

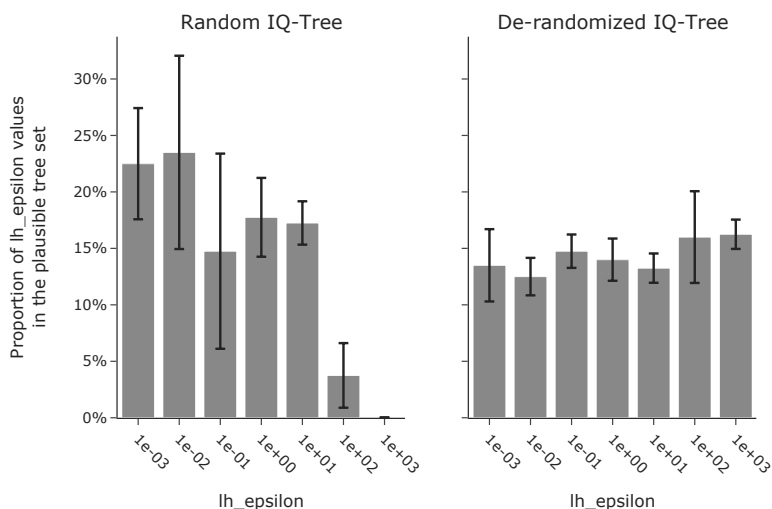


Figure 6.16.: Average proportions of $lh_epsilon$ values among trees in the plausible tree set for datasets D1481, D1604, D1718, and D2445. The left figure shows the proportions for random IQ-Tree. The right figure shows the proportions for de-randomized IQ-Tree. The error bar depicts the standard deviation.

threshold does not influence the tree inference. We observe that the distances between the de-randomized IQ-Tree histograms and the flat histogram are considerably smaller than the distances between the random IQ-Tree and the flat histogram (Figure 6.17). We conclude that large $lh_epsilon$ settings ($\geq 10^2$) distort the random NNI moves in IQ-Tree, causing a premature termination of the tree search. We additionally compare the RAxML-NG histograms of the respective datasets to both IQ-Tree histograms, and the flat histogram. We observe that the proportions of $lh_epsilon$ values with RAxML-NG is closest to the flat histogram for all four datasets. This indicates that RAxML-NG is less sensitive to the $lh_epsilon$ setting than IQ-Tree.

FastTree

With FastTree, we observe worse ML scores for $lh_epsilon$ thresholds > 1 . Considering the proportion of $lh_epsilon$ values, we notice that the proportion of a tree search returning a plausible tree decreases with higher $lh_epsilon$ thresholds. For most datasets, the thresholds 10^2 and 10^3 do not return plausible trees. This could be due to the fast heuristics in FastTree’s implementation and the lack of an evaluation functionality (Section 3.6.3). The runtimes for $lh_epsilon$ values ≤ 1 show no substantial speedup. FastTree’s default $lh_epsilon$ value 10^{-1} is therefore appropriate.

6.2.2. Minimum Branch Length

$minBranchLen$ settings $\geq 10^{-3}$ yield worse ML scores for all three ML inference tools. Due to the lack of an evaluation phase in FastTree, the degradations of ML scores are an order of

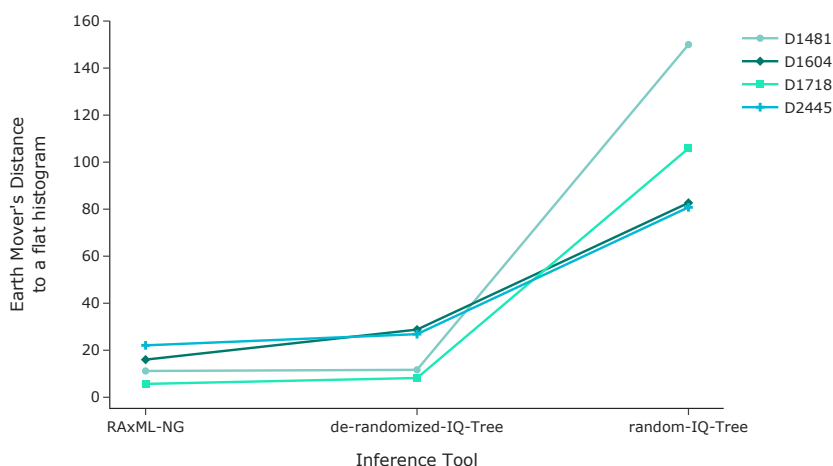


Figure 6.17.: Earth mover’s distances of the proportions of $lh_epsilon$ values in the set of plausible trees to a flat histogram for RAxML-NG, the random IQ-Tree variant and the de-randomized IQ-Tree variant. The plot shows the distances for the four datasets D1481, D1604, D1718, and D2445.

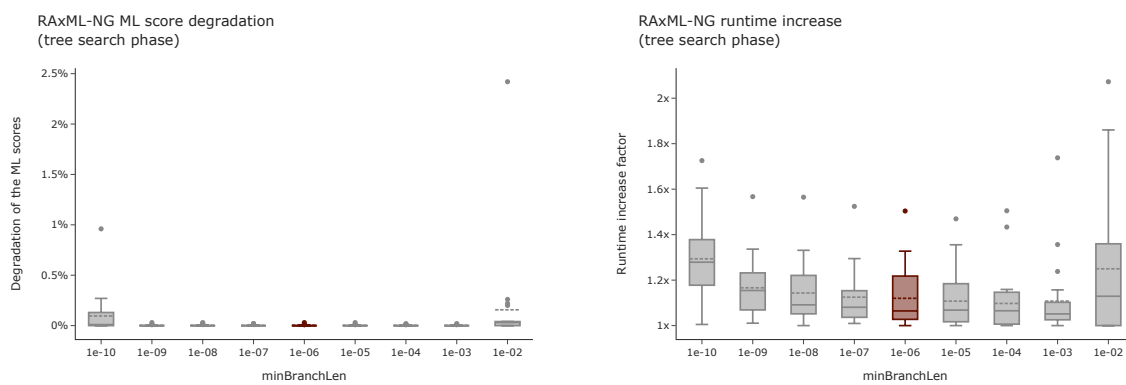
magnitude worse than for RAxML-NG and IQ-Tree. Depending on the tool, the runtimes follow different trends. We observe that the default values for all three inference tools are well-chosen.

RAxML-NG

The ML scores for trees inferred with RAxML-NG worsen on average by 0.16 % for *minBranchLen* settings $\geq 10^{-3}$. All *minBranchLen* values in the range 10^{-9} – 10^{-3} achieve equally good ML scores (variances ≤ 0.03 %). For RAxML-NG, the ML scores for 12 out of 22 datasets again worsens noticeably with a *minBranchLen* setting of 10^{-10} (Figure 6.18a). The ML scores for this setting are on average 0.1 % worse compared to *minBranchLen* = 10^{-9} . Similar to the ML scores, the runtimes for *minBranchLen* values in the range 10^{-9} – 10^{-3} are approximately equal, with a slight trend towards faster tree searches with higher *minBranchLen* settings, while the runs with *minBranchLen* 10^{-10} and 10^{-2} are slower. Given these observations, we conclude that a *minBranchLen* setting of 10^{-9} – 10^{-3} during the tree search is reasonable. RAxML-NG’s default value 10^{-6} falls in this range and is therefore appropriate.

IQ-Tree

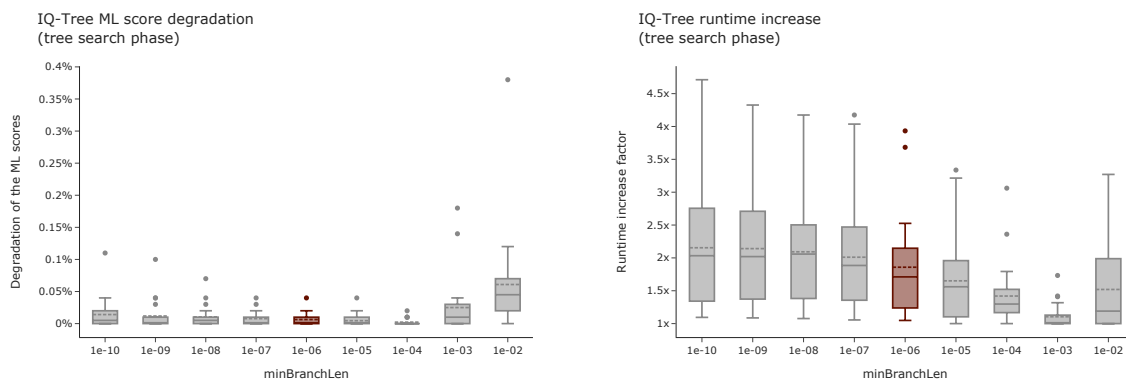
In general, *minBranchLen* settings $\geq 10^{-3}$ result in worse ML scores (Figure 6.19a). The ML scores under the highest setting *minBranchLen* = 10^{-2} are on average 0.1 % worse. Similar to RAxML-NG, for some datasets, we observe slightly worse ML scores for the *minBranchLen* setting 10^{-10} (≤ 0.11 %). We observe a high impact of the *minBranchLen* setting on the runtime of the IQ-Tree tree search phase (Figure 6.19b). The runtime of IQ-Tree increases with smaller



- (a) Degradation of ML scores across all datasets as a function of *minBranchLen* values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores.
- (b) Increase of the tree search time across all datasets as a function of *minBranchLen* values. The increase is measured relative to the minimum runtime per dataset.

Figure 6.18.: Influence of the *minBranchLen* threshold on the ML scores and runtimes of the RAxML-NG tree search phase. The ML scores refer to the ML scores *after* the evaluation phase. The runtimes refer to the runtime of the tree search phase. The plots summarize the data over all datasets. The dashed vertical line indicates the mean, and the solid vertical line the median value. The highlighted box indicates the default *minBranchLen* value for RAxML-NG.

$minBranchLen$ values. For $minBranchLen = 10^{-10}$ the tree search phase runs on average twice as long as for $minBranchLen = 10^{-3}$. Interestingly, the runtime also increases if $minBranchLen$ is set to 10^{-2} . These tree searches run on average $52 \pm 6\%$ longer than tree searches with $minBranchLen = 10^{-3}$. Considering these observations, the IQ-Tree default setting 10^{-6} for the $minBranchLen$ threshold appears to be a good trade-off between runtime and ML scores.

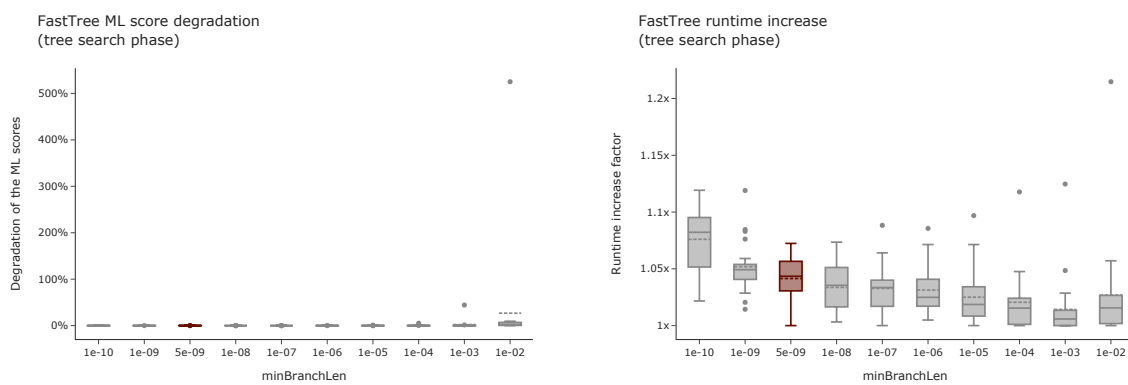


- (a) Degradation of ML scores across all datasets as a function of $minBranchLen$ values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores.
- (b) Increase of the tree search time across all datasets as a function of $minBranchLen$ values. The increase is measured relative to the minimum runtime per dataset.

Figure 6.19.: Influence of the $minBranchLen$ threshold on the ML scores and runtimes of the IQ-Tree tree search phase. The ML scores refer to the ML scores *after* the evaluation phase. The runtimes refer to the runtime of the tree search phase. The plots summarize the data over all datasets. The dashed vertical line indicates the mean, and the solid vertical line the median value. The highlighted box indicates the default $minBranchLen$ value for IQ-Tree.

FastTree

Similar to the other ML inference tools, the ML scores for FastTree worsen with higher $minBranchLen$ settings. With FastTree, the degradations are an order of magnitude worse than for IQ-Tree and RAxML-NG. The trees for $minBranchLen = 10^{-2}$ are on average 1.7% less likely than the respective maximum likelihood tree (Figure 6.20a). This is most likely due to the lack of an evaluation mode that improves the ML scores under smaller $minBranchLen$ settings. We observe the highest decline of ML scores in our analysis for FastTree with $minBranchLen$ set to 10^{-2} on the D4869 dataset. The ML score with this $minBranchLen$ setting decreases by 525%. For $minBranchLen$ values $\leq 10^{-5}$ the ML scores are approximately equal (variances $\ll 0.1\%$). The runtime fluctuates depending on the dataset and the $minBranchLen$ setting. In general, there is a trend for smaller settings to run longer (Figure 6.20b). Given these observations, the default $minBranchLen$ value 5^{-9} appears to be a reasonable choice.



- (a) Degradation of ML scores across all datasets as a function of *minBranchLen* values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores.
- (b) Increase of the tree search time across all datasets as a function of *minBranchLen* values. The increase is measured relative to the minimum runtime per dataset.

Figure 6.20.: Influence of the *minBranchLen* threshold on the ML scores and runtimes of the FastTree tree search phase. The ML scores refer to the ML scores *after* the tree search phase. The runtimes refer to the runtime of the tree search phase. The plots summarize the data over all datasets. The dashed vertical line indicates the mean, and the solid vertical line the median value. The highlighted box indicates the default *minBranchLen* value for FastTree.

6.2.3. Remaining Thresholds

With the *maxBranchLen* threshold, we observe no influence on the ML scores of RAxML-NG or IQ-Tree, but we do observe an influence on the runtimes. With both, RAxML-NG and IQ-Tree, depending on the dataset, a different *maxBranchLen* setting runs faster on average ($\leq 16\%$ differences).

For the threshold *model_epsilon*, we observe minor variances in ML scores for IQ-Tree and RAxML-NG ($\leq 0.1\%$). For RAxML-NG, depending on the dataset, a different *model_epsilon* value appears to be the fastest setting with no clear trend. For IQ-Tree, we observe faster tree inferences with higher *model_epsilon* thresholds. We make a similar observation for the runtime of RAxML-NG's threshold *num_iters*. The ML score is not affected by the setting of *num_iters*. With the threshold *bfgs_factor* of RAxML-NG, we notice no influence on the ML scores, but a trend towards faster runs for higher values.

For all these thresholds, despite their minor variations in runtimes and ML scores, we conclude that the respective default settings in both RAxML-NG and IQ-Tree are appropriate.

The data for these thresholds is not shown in this chapter; we refer the interested reader to Appendix A.1.2 or the interactive plots on our website at <https://www.thesis.juliahaag.de/numericalProperties/influenceSearch>.

7. Comparison of Inference Tools

Over time, numerous authors compared different ML phylogenetic inference tools according to various metrics [39, 48, 78]. During our research, we find that RAxML-NG, IQ-Tree, and FastTree have been compared based on simulated datasets [36, 50, 77], but, to the best of our knowledge, no author has yet compared RAxML-NG, IQ-Tree, and FastTree on multiple *empirical* datasets in the same study. Given the data we have, it is evident to compare those three tools. In our analysis, we focus on the comparison of ML scores and runtimes. We compare RAxML-NG, IQ-Tree, and FastTree under the same conditions, according to our data generation pipeline (Section 5.1). As stated in the previous Section 5.1, we run our data generation on multiple different systems. To ensure comparability, we run all tools for one dataset on the same system. Since in contrast to RAxML-NG and IQ-Tree, FastTree does not provide an evaluation mode, we discuss the value of such a functionality in the second section of this chapter.

7.1. Likelihood and Runtime

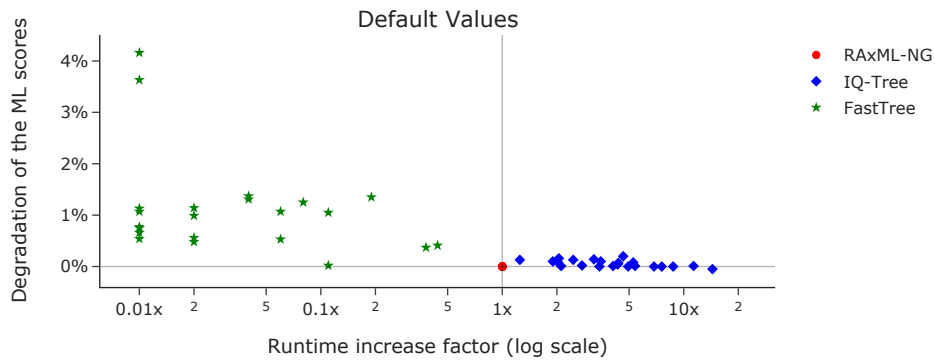
In Figures 7.1a and 7.1b we show the ML scores and runtimes for the three inference tools RAxML-NG, IQ-Tree, and FastTree. For RAxML-NG and IQ-Tree, the ML score refers to the ML score after the evaluation phase, and for FastTree after the tree search phase. The runtime value states the corresponding runtime for the tree search phase. All ML scores and runtimes are scaled relative to the RAxML-NG ML score and runtime, therefore all RAxML-NG values collapse into a single point.

The values in Figure 7.1a show the ML score of the best-found tree under the default numerical threshold settings. This means that we set the numerical threshold to their default values according to Table 5.1 during the tree search phase. Figure 7.1b shows the best found ML score and corresponding runtime under any numerical threshold setting.

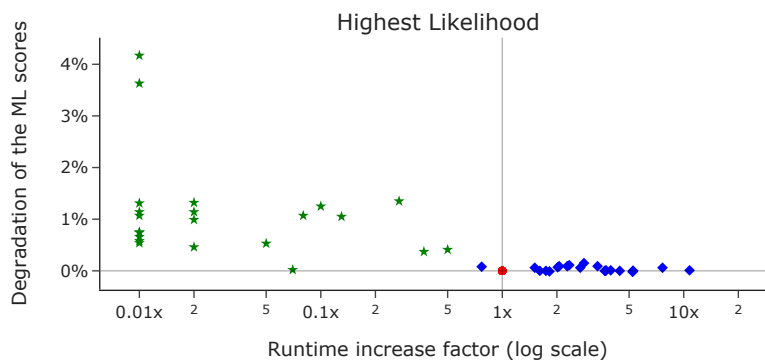
In terms of ML scores, RAxML-NG and IQ-Tree, as expected by the design of the search heuristics (Section 3.6), outperform FastTree on all datasets in our analysis. The ML scores of trees inferred with FastTree are up to 4 % worse than trees inferred with RAxML-NG. For 20 out of 22 datasets, RAxML-NG achieves equally good or higher ML scores than IQ-Tree (16/22 higher, 5/22 equal). Only for 1 out of the 22 datasets, IQ-Tree outperforms RAxML-NG. The differences in ML scores between RAxML-NG and IQ-Tree are $\leq 0.05\%$. For 21/22 datasets, IQ-Tree has the longest runtime, and for all datasets FastTree has the shortest runtime. FastTree runs on average 55 ± 44 times faster than RAxML-NG. RAxML-NG runs on average 5 ± 3 times faster than IQ-Tree. The high variance in runtimes is due to the broad range of datasets we use in our analysis. For smaller datasets, the runtime differences are less prominent than for larger datasets.

Figure 7.1c compares the results if we apply the suggested changes in default values of the likelihood epsilon values for RAxML-NG and IQ-Tree, as presented in the previous sections.

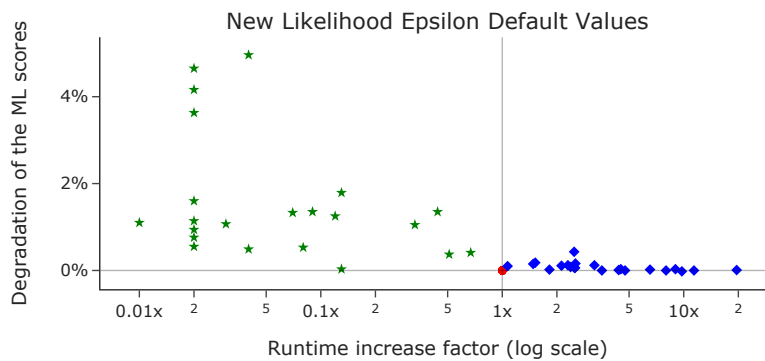
7. Comparison of Inference Tools



- (a) The highest achieved ML scores per dataset and the corresponding runtimes for the default settings of each tool.



- (b) The highest achieved ML scores per dataset and the corresponding runtimes if we allow any numerical threshold setting.



- (c) The highest achieved ML scores per dataset and the corresponding runtimes if we increase the likelihood epsilon default values for IQ-Tree and RAxML-NG.

Figure 7.1.: Comparison of ML scores and runtimes of RAxML-NG, IQ-Tree, and FastTree for all datasets. All values are relative to the RAxML-NG values, therefore all RAxML-NG values collapse into a single point.

Due to the changes, the tree inferences for RAxML-NG and IQ-Tree run faster, and the runtime gap to FastTree shrinks. Since the changes of likelihood epsilon values results in a higher speedup for RAxML-NG than for IQ-Tree, the runtime gap between RAxML-NG and IQ-Tree increases.

On our website at <https://www.thesis.juliah Haag.de/numericalProperties/comparison> the data is available as interactive plots and tables.

7.2. Influence of Evaluation Functionality

In addition to the standard phylogenetic tree inference, RAxML-NG and IQ-Tree both implement an evaluation mode. During this re-evaluation, the algorithm optimizes the branch lengths and substitution model parameters on a fixed tree topology. In our analysis, we re-evaluate the trees after the tree search phase under conservative numerical threshold settings (Section 6.1). Since we vary the numerical thresholds during the tree search phase, the threshold settings can be very restricting (for example $\text{minBranchLen} = 10^{-2}$). Using this re-evaluation, we allow the inference tool to improve the ML scores under more conservative threshold settings. FastTree does not provide a similar evaluation functionality. We therefore examine the improvements of the ML scores and runtime costs resulting from this re-evaluation for RAxML-NG and IQ-Tree. In our data, we observe an average runtime for the evaluation phase across all datasets and numerical thresholds for RAxML-NG of $5 \pm 5\%$ of the respective tree search time. For IQ-Tree, we observe an average runtime of $4 \pm 3\%$. The improvement of ML scores after the evaluation phase for RAxML-NG range between 0.0% (no improvement) and 2.6%, with an average of $0.2 \pm 0.5\%$. For IQ-Tree, we observe ML score improvements up to 3.6%, with an average of $0.7 \pm 1.1\%$. Averaged over all tree inferences and datasets, the average improvements appear to be minor. We observe, however, that the improvements of ML scores vary, depending on the value of the numerical thresholds during the tree search phase. For example, with the minBranchLen threshold for RAxML-NG, we see that the ML score for runs with minBranchLen set to 10^{-2} during the tree search phase, improves on average 1.4%. If the minBranchLen threshold is set to 10^{-6} , the improvement is only 0.02% (Figure 7.2).

We observe that when using this evaluation functionality, we can recover from “bad” threshold settings during the tree search phase. Figure 7.3 demonstrates this for the minBranchLen threshold on one exemplary dataset (D1481). The ML score after the evaluation phase is approximately equal, whereas after the tree search phase, we observe a decrease for higher minBranchLen values.

In contrast to the ML scores, the runtime of the evaluation phase is largely unaffected by the numerical threshold settings during the runtime. For the above example, the runtimes for the evaluation phase were on average 4.0% and 3.6% of the runtime of the corresponding tree search for $\text{minBranchLen} = 10^{-2}$ and 10^{-6} respectively. In Section 6.2.1 we have seen that we can exploit the evaluation mode to speed up the tree search using higher likelihood epsilon values, while obtaining equally good ML scores after the evaluation phase.

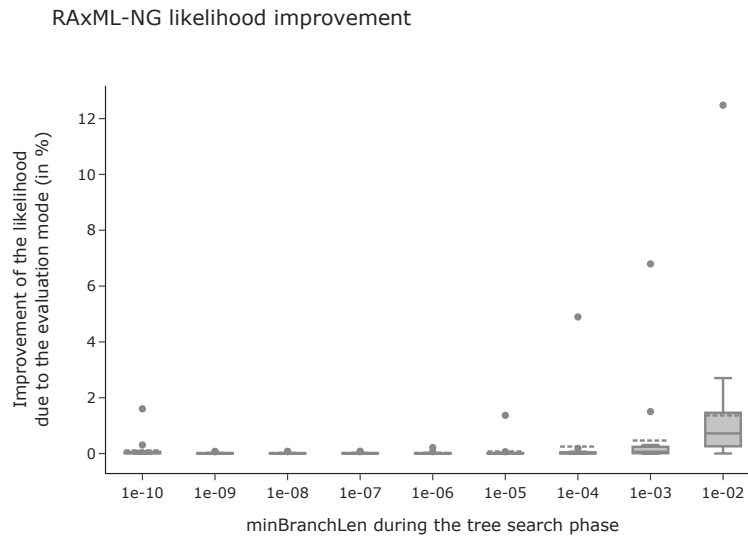


Figure 7.2.: ML score improvement for different *minBranchLen* values after the evaluation phase relative to the ML score after the tree search phase. The plot shows the improvements over all datasets for RAxML-NG.

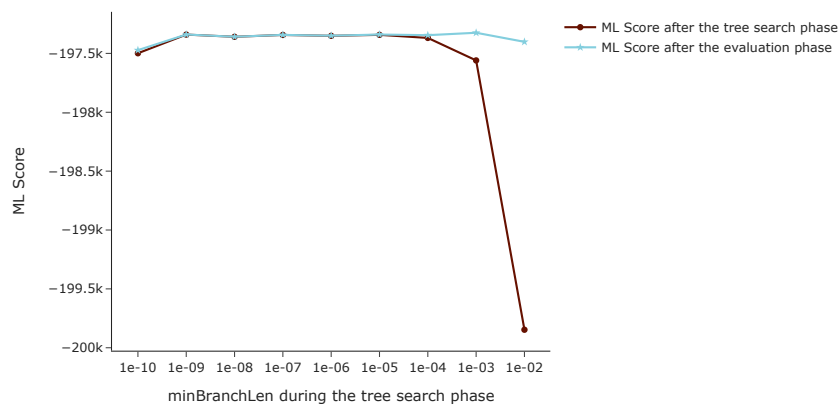


Figure 7.3.: Average ML scores after the tree search phase and after the evaluation phase. This plot shows the data for one exemplary datasets (D1481) for RAxML-NG.

Part III.

Predicting Dataset Difficulty

8. Problem Description

Our main goal when inferring phylogenetic trees under the ML model is to find the most likely tree. That is the tree that best explains the given data. Since the ML method is \mathcal{NP} -hard [10], we cannot ensure that an inferred tree is the *global* optimum, in most cases it is a *local* optimum. For this reason, we typically infer multiple ML trees from different starting trees to escape local optima. We can then summarize the obtained trees in a so-called consensus tree [76]. For example, RAxML-NG and IQ-Tree by default conduct 20 distinct tree searches. For some datasets, all tree searches lead to the same tree. This indicates an easy to analyze likelihood surface with a single optimum. On other datasets, each tree inference yields a different tree. For such datasets we need to conduct more tree searches to build a convincing consensus-tree, than we need for datasets that are easy to analyze. In order to save time and resources, it would be valuable to predict whether a dataset is easy or difficult to analyze *before* running multiple tree searches. The terms “easy” and “difficult”, when referring to the analysis of datasets, are not clearly defined. We address this issue in Section 9.1. In this part of the thesis, we describe our experiments to predict the difficulty of datasets using different machine learning approaches. We divide this part into three sections: In Section 9 we focus on the training data and the definition of ground-truth labels with respect to dataset difficulty. In Section 10 we suggest a set of features that can be useful for predicting the difficulty of datasets, briefly introduce the classification algorithms we use, and analyze our prediction results. Finally, in Section 11 we provide an overview of possible further experiments.

9. Training Data

In order to train a difficulty prediction algorithm, we require training data. For our prediction experiments, we use the same 22 empirical datasets as for our numerical analysis in Part II. To increase the amount of training data, we additionally use 200 simulated datasets. For 100 of these simulated datasets, we reduce the phylogenetic signal of the data (Section 3.1) by removing the second half of each sequence in each MSA file. With most prediction algorithms we train, as well as for performance evaluation, we rely on ground-truth difficulty labels for each dataset. To obtain such labels, we need a quantifiable definition of *difficulty*. We address this in Section 9.1. In Section 9.2 we explain the quantification and label generation for our training data.

9.1. Definition of Difficulty

In order to train a reliable difficulty prediction algorithm, we need to infer reliable ground-truth labels for our training datasets. To obtain such labels, we need a concrete definition of difficulty. Consulting the literature, we find multiple definitions. Morel *et al.* [49] and Stamatakis [66] describe a dataset as being difficult, if multiple tree searches lead to statistically indistinguishable, locally optimal tree topologies with “high” topological distances. Furthermore, Stamatakis [66] describes the likelihood surface of difficult datasets as *rough* and *rugged*, meaning the data exhibits multiple likelihood peaks. Lakner *et al.* [41] make a similar observation and report “topologies [...] separated [...] by a broad valley”. All of these statements describe a similar phenomenon: the tree space exhibits multiple, statistically indistinguishable local optima. The datasets D27 and D4869 are known to be difficult [31, 49]. Since we only know for these two datasets that they are difficult, we need to infer difficulty labels for the remaining 220 datasets. In order to do this, we use the above definitions to quantify difficulty. We explain this in the following section.

9.2. Difficult Difficulty Labels

To quantify the difficulty, we conduct 20 tree searches for each dataset using RAxML-NG, resulting in a set of 20 *search trees*. We then re-evaluate each of these trees, resulting in 20 *evaluation trees*. On these evaluation trees we perform all significance tests as implemented in IQ-Tree, and include trees passing all tests in the set of *plausible trees*. Given the tree search trees, evaluation trees, and plausible trees, we compute a number of different features in order to assign a difficulty label to each dataset. For each tree set, we compute the average topological distance between all trees, and the number of unique topologies. We use the Robinson-Foulds Distance (RF-distance) [56] as topological distance metric. According to the above definitions,

for an easy dataset we expect only few unique topologies with a low pair-wise RF-distance. Another indicator for difficulty might be the number of tree searches resulting in a plausible tree. If the number of trees in the plausible tree set is substantially smaller than the number of trees inferred, the dataset will presumably be difficult to analyze. During our analyses, we experiment with different combinations of these features to define ground-truth labels. Based on our knowledge that D27 and D4869 are difficult datasets, the RF-distance and the number of unique topologies in the set of plausible trees appear to be good candidates for difficulty label assignment. Rather than using the plain number of unique topologies, we scale the number of unique topologies with the number of trees in the plausible tree set. Given these features, we estimate the decision boundaries as:

1. The number of unique tree topologies in the plausible tree set is at least half the number of trees in the plausible tree set.
2. The RF-distance in the plausible tree set is at least 10 %.

In our further difficulty prediction experiments, we assign the label “difficult” to a dataset if it fulfills both of the above criteria. Figure 9.1 plots the features, decision boundaries, and the resulting labels for the 222 training datasets, with the 22 empirical datasets being highlighted. In our datasets, the labels are imbalanced. There are three times more easy datasets than there are difficult ones. This is important for choosing appropriate performance metrics. A performance metric that only accounts for correct predictions will yield good scores for predictors that predict “easy” for any dataset. We address this issue in Section 10.2.2.

For the 22 empirical datasets, we manually review the assigned labels. We notice that for some datasets, we do not concur with the inferred labels. For example, dataset D46 is labelled “easy”. In the plausible tree set, all trees have the exact same tree topology (RF-distance = 0). However, the trees group into two distinct ML scores. This indicates that while the tree topologies are equal, the branch lengths or substitution parameters differ. Yet, using significance tests, we cannot differentiate among these plausible trees. We observe a similar effect for D37 and D101. Given the limited amount of training data and the lack of definitive ground-truth labels, we cannot determine whether these effects are outliers, or if our label inference is not appropriate. Consulting Figure 9.1, we observe that most datasets have as many plausible trees as there are unique topologies in the plausible tree set (the x-value is 1). This indicates that this feature might not be particularly well-suited to separate the datasets into easy and difficult ones. In future experiments, we should address these issues, and infer labels based on other, or additional difficulty indicators (Section 11).

Despite the observed issues with the inferred labels, we train and explore different prediction algorithms using these labels. We explain these experiments in the following chapter.

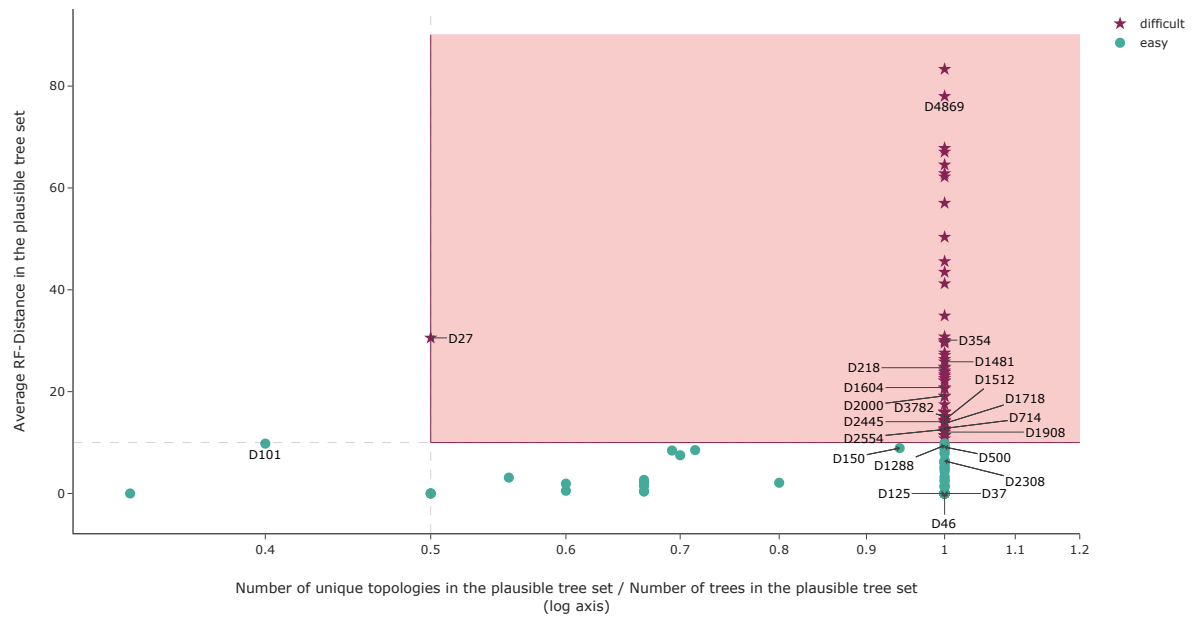


Figure 9.1.: Inferred ground-truth labels for the 222 training datasets. The annotations highlight the empirical datasets. Dots indicate easy datasets and stars indicate difficult datasets. The dashed lines depict the decision threshold for each feature. Only datasets above both thresholds are labeled as being difficult (highlighted area).

10. Difficulty Prediction

Given our training data, alongside the difficulty labels, we apply machine learning algorithms to predict whether a dataset is difficult or easy. Our goal is to predict the difficulty prior to conducting multiple tree searches. In Section 10.1 we present a set of features that is based on the MSA and a single tree inference. In Section 10.2 we briefly explain the machine learning algorithms we train for difficulty prediction, introduce performance metrics, and compare the different classification algorithms.

10.1. Feature Generation

Given the MSA and a single RAxML-NG tree search, we are able to compute a variety of different features. In the following, we present a set of such features that could be useful for difficulty prediction. A similar set of features has proven useful in predicting the best substitution model for phylogenetic analysis [1]. Due to the limited amount of time, we only implement a subset of these features for our experiments.

10.1.1. MSA Features

Table 10.1 summarizes the MSA features. As discussed in Section 3.1, the properties and the phylogenetic signal of the MSA impact the analysis of the dataset and are therefore good candidates for difficulty features. The features we implement for our experiments are indicated by an asterisk. Researchers developed different statistics to measure the phylogenetic signal in the MSA [25, 33, 44]. In our experiments, we use the Treelikeness Score [33]. For further experiments, we suggest implementing additional phylogenetic signal quantification measures.

MSA Feature	Description
# Taxa*	Number of taxa in the MSA.
# Sites*	Number of alignment columns in the MSA.
% Invariant sites*	Percentage of fully conserved sites in the MSA.
% Gaps*	Proportion of gaps in the MSA.
# Patterns*	Number of unique sites.
# Parallel patterns	Number of columns corresponding to the same template, for example, the columns ACCC, CAAA, AGGG are parallel and correspond to the template XYYY [25].
Char frequencies	Frequencies of each character in the MSA.
MSA Entropy*	Shannon Entropy [59] over all MSA sites. See below for a more detailed description.
Bollback Multinomial*	Multinomial test statistic [3]. See below for a more detailed description.

Treelikeness Score*	Phylogenetic signal quantification [33]. See below for a more detailed description.
Sum-of-pairs score	Sum over all per-site scores; the score for one site in the MSA is the sum over all pairwise scores between the characters of the site.

Table 10.1.: The set of MSA features we suggest for difficulty prediction. An asterisk indicates that we implement this feature in our experiments.

Further Explanation of MSA Features

MSA Entropy The Shannon Entropy [59] measures the information value of data. We compute the entropy for an MSA with N taxa as average Shannon Entropy over all M MSA sites:

$$H(\text{MSA}) = \frac{1}{M} \sum_{i=1}^M H(\text{site}_i) \quad (10.1)$$

$$\text{with } H(\text{site}_i) = - \sum_{j=1}^N P(\text{char}_{j,i}) \cdot \log(P(\text{char}_{j,i})) \quad (10.2)$$

Bollback Multinomial Bollback [3] designed this test statistic to quantify the frequency of site patterns. Let M be the length of the MSA, n the number of unique site patterns, $\xi(i)$ the i -th unique pattern, and $N_{\xi(i)}$ the number of times this pattern occurs. We compute the multinomial test statistic as:

$$T(\text{MSA}) = \left(\sum_{i=1}^n N_{\xi(i)} \cdot \ln(N_{\xi(i)}) \right) - M \cdot \ln(M) \quad (10.3)$$

Treelikeness Score The treelikeness score [33] is designed to quantify the phylogenetic signal of the data. The treelikeness score is based on the matrix of pairwise distances of all sequences in the MSA. Let N be the number of taxa, and D the pair-wise distance matrix with entries $d_{i,j} \forall i, j \in N$. For a set of four taxa (a *quartet*) $q = (x, y, u, v)$, we compute the treelikeness as:

$$\delta_q = \frac{d_{xv|yu} - d_{xu|yv}}{d_{xv|yu} - d_{xy|uv}} \quad (10.4)$$

$$\text{with } d_{xy|uv} = d_{xy} + d_{uv} \quad (10.5)$$

$$\text{and } d_{xy|uv} \leq d_{xu|yv} \leq d_{xv|yu} \quad (10.6)$$

The lower the score δ_q , the stronger the phylogenetic signal in the quartet q . For the set N of taxa, we compute this δ_q -score for every possible quartet q in N . The treelikeness of the entire data is then computed as the mean over all δ_q -scores. Due to the computational complexity, for datasets with many taxa (> 100), Holland *et al.* [33] suggest computing the δ_q -scores for a random sample of all possible quartets.

10.1.2. Tree Features

Our difficulty prediction aims to reduce the amount of resources used for analyzing easy datasets, and to increase user awareness about the degree of difficulty of the dataset. However,

for any dataset, we need to conduct at least one tree search to obtain one ML tree. We use this single tree search to enrich our set of features. The idea behind this is that we predict the difficulty *after* this single tree inference. Only if we identify the dataset as being difficult, we suggest conducting further tree searches. In Table 10.2 we present a set of potential features based on a single RAxML-NG ML tree inference. An asterisk after the feature indicates that we implement this feature in our experiments.

Tree Feature	Description
# SPR rounds*	Total number of RAxML-NG SPR rounds for a single tree inference (Section 3.6.1).
Parsimony score*	Parsimony score of the starting tree and the final tree.
Branch lengths*	Minimum, maximum, average, standard deviation, sum of all branch lengths.
RF-distance	RF-distance between the starting tree topology and the final tree topology
Likelihood difference	Likelihood difference between the RAxML-NG starting tree and the final tree, scaled relative to the likelihood of the final tree.
Substitution model parameters	Equilibrium frequencies, substitution rates, and α shape parameter of the Γ distribution (Section 3.1).
Patristic distances	Sum over the branch lengths between a pair of taxa, computed for all pairs. Minimum, maximum, average, standard deviation, sum of all patristic distances.

Table 10.2.: The set of tree features we suggest for difficulty prediction. An asterisk indicates that we implement this feature in our experiments.

10.2. Difficulty Prediction

In this section, we describe the difficulty prediction experiments we perform on the datasets using the set of computed features.

For our experiments, we use both supervised and unsupervised machine learning algorithms. Supervised algorithms rely on ground-truth labels for training. Unsupervised techniques learn to classify the data based on their features. We show that we cannot reliably predict the difficulty of datasets with either technique. In Section 10.2.1, we briefly explain the supervised and unsupervised machine learning approaches we use in our experiments.

In Section 10.2.2 we present a set of performance metrics according to which we compare all trained difficulty prediction algorithms. In Section 10.2.3, we show the results of applying the presented classifiers to our training data and discuss why the results are not satisfactory.

10.2.1. Machine Learning Algorithms

For our prediction experiments, we train a number of different machine learning algorithms. We use the implementations of the Python machine learning library scikit-learn [51]. We pose

the challenge of predicting the difficulty as a binary classification task. We refer to instances of the classification algorithms as *classifiers*. In the following, we briefly explain the algorithms we use. For a more detailed introduction we refer the interested reader to the respective publications cited below and to Goodfellow *et al.* [26].

Support Vector Machine (SVM) The SVM classifier [4] aims to separate the data in the feature space such that the gap between the two target classes is maximized. If the data is not linearly separable, the SVM makes use of the so-called kernel trick. The SVM maps the data into a higher dimensional feature space, where the data is linearly separable.

Decision Tree A decision tree classifier [5] is a tree-like structure, where each internal node represents a split into subgroups based on a specific feature. The leafs represent classification decisions. Predicting the label of a datum is done by traversing the decision tree starting at the root until a leaf is reached. In our analysis we limit the depth of the tree to 3, and require at least 3 samples per leaf.

Random Forest A random forest classifier [32] is an ensemble method that summarizes a number of individually trained decision trees. In our experiments, we use a random forest that consists of 10 decision trees, each with a maximum depth of 3.

Multilayer Perceptron (MLP) An MLP is a fully-connected neural network. To prevent overfitting, we only use a single hidden layer with 20 neurons. As activation function, we use the rectified linear unit function (ReLU). We set the learning rate to 10^{-3} and allow a maximum number of 200 training iterations.

Adaptive Boosting (AdaBoost) The AdaBoost classification algorithm is based on the concept of boosting [21]. A set of weak individual classifiers is combined into a single, stronger classification algorithm. Each weak classifier votes for a prediction. The votes are summed, weighted by the training error of the respective individual classifier.

Principal Components Analysis (PCA) The PCA [14] is an unsupervised dimensionality reduction algorithm. The idea behind PCA is to map the (high-dimensional) feature space into a lower dimensional space. This is achieved by rotating the feature space along the axes with the highest variance. In our experiments, we use the PCA to reduce the dimensionality of the set of MSA and tree features. We refer to the set of features before the PCA as *plain features*, and to the set of features after the PCA as *reduced features*.

K-Nearest Neighbor (K-NN) The K-Nearest Neighbor algorithm [19] predicts the label of a datum based on the labels of the k nearest neighbors in the feature space. During the learning phase, the algorithm only stores the feature vectors alongside the labels of the training data. For our experiments, we set the number of considered neighbors to $k = 3$. We apply the algorithm to the plain features, as well as the reduced features.

K-Means The K-Means algorithm [45] groups the input data into k clusters based on similarities in the feature space. Since difficulty prediction is a binary classification problem, we set the number of clusters to $k := 2$. We perform the clustering on the plain features, as well as the reduced features.

10.2.2. Performance Metrics

In order to evaluate the predictions, we split the datasets into a training set and a test set. The classifiers only train on the training set and do not see the test set during training. The training set comprises 70 % of the data, and the test set 30 %. To enhance the number of training samples, we use a technique called cross-validation. We train five instances of each classification algorithm, each instance on a different train–test split. To evaluate the performance of a prediction algorithm, we average over the performances of these five classifiers. We implement a set of performance metrics based on the so-called confusion matrix. This matrix shows the proportion of True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) predictions, with “Positive” corresponding to “difficult”:

		Ground-truth difficulty	
		Difficult	Easy
Predicted difficulty	Difficult	TP	FP
	Easy	FN	TN

Using this confusion matrix, we implement the following metrics:

Accuracy (ACC) Fraction of datasets the classifier predicts correctly

$$ACC = \frac{TP + TN}{TP + FP + FN + TN}$$

True positive rate (TPR) Fraction of difficult datasets the classifier predicts as being difficult; this is also called sensitivity or recall

$$TPR = \frac{TP}{TP + FN}$$

True negative rate (TNR) Fraction of easy datasets the classifier predicts as being easy; this is also called specificity

$$TNR = \frac{TN}{TN + FP}$$

Positive predictive value (PPV) Fraction of the predicted difficult datasets that is actually difficult; this is also called precision

$$PPV = \frac{TP}{TP + FP}$$

Negative predictive value (NPV) Fraction of the predicted easy datasets that is actually easy

$$NPV = \frac{TN}{TN + FN}$$

False positive rate (FPR) Fraction of easy datasets the classifier predicts as being difficult

$$FPR = \frac{FP}{FP + TN}$$

False Negative rate (FNR) Fraction of difficult datasets the classifier predicts as being easy

$$FNR = \frac{FN}{TP + FN}$$

It is important to note that using the accuracy alone as performance metric does not suffice for our task. As we have seen in Section 9.2, the labels in our training datasets are imbalanced: there are three times more easy datasets than there are difficult ones. As a consequence, a naïve classification algorithm that always predicts “easy” yields a high accuracy, but results in wrong predictions for difficult datasets. For our use case, our main goal is to obtain a high TPR. It is important, to correctly predict the difficult datasets, indicating that multiple tree searches should be conducted. In contrast, low FPRs and high PPVs are less important. While we try to reduce the necessary amount of resources for the analysis of easy datasets, it is less detrimental to waste resources than to obtain unreliable consensus trees.

As a baseline for the performances of the classifiers, we compare the results to two dummy classifiers. Such dummy classifiers predict labels based on certain rules. We use a dummy classifier that always predicts “easy”, and a dummy classifier that predicts labels based on stratified sampling from the labels in the training data.

10.2.3. Prediction Results

In this section, we evaluate the set of trained classifiers according to the presented performance metrics. We compute the performance for the prediction results on the unseen test data.

In Table 10.3 we list the performances of the classifiers we train, according to the presented metrics. Comparing the accuracies and the true positive rates, we see why the accuracy as single metric does not suffice to evaluate the performance in our case. While the accuracies imply that the classifiers learn to distinguish between “easy” and “difficult”, the low values of FPR and high values of FNR and TNRs indicate that the classifiers tend to predict “easy” rather than “difficult”. All classifiers perform well in predicting the easy datasets (high TNR), but are not reliable in predicting difficult data (low TPR). Given that the TPR for our use case is an important measure (Section 10.2.2), the results are not satisfactory. When predicting probabilities instead of labels, we notice that across all classifiers, many predictions are uncertain, especially negative predictions. This is reflected by the PPV and NPV values.

The classifier that overall performs worst is the K-Means algorithm. This classifier does not correctly predict a single difficult dataset. This can be due to the fact that with unsupervised learning methods, it is difficult to control the learned objective function [24]. The clustering may lead to two separated classes, but they do not necessarily represent dataset difficulty. Performing clustering on the reduced features does not improve the prediction results.

Comparing the classifiers regarding the features they rely on for data separation, we see that most classifiers decide on difficulty based on the number of taxa. This is most likely due to an imbalance in our training data. In our data, the average number of taxa for the difficult datasets is 581, whereas the average number of taxa for the easy datasets is 119. Besides the number of taxa, we observe that there is no consensus on feature importance. For example, the SVM classifier does not consider the Bollback Multinomial (feature importance 0 %) for its decision, while with the AdaBoost classifier, the respective feature importance is 16 %. However, both classifiers achieve the same TPR. We observe a similar effect when repeatedly training a single

Classifier	ACC	TPR	TNR	PPV	NPV	FPR	FNR
dummy (constant “easy” prediction)	0.7	0.0	1.0	—	0.7	0.0	1.0
dummy (stratified label sampling)	0.55	0.25	0.68	0.25	0.68	0.32	0.75
SVM	0.74	0.38	0.89	0.60	0.77	0.11	0.62
Decision Tree	0.68	0.19	0.89	0.43	0.72	0.11	0.81
Random Forest	0.74	0.38	0.89	0.60	0.77	0.11	0.62
MLP	0.36	0.25	0.41	0.15	0.56	0.59	0.75
AdaBoost	0.77	0.5	0.89	0.67	0.80	0.11	0.50
K-NN	0.68	0.25	0.86	0.44	0.73	0.14	0.75
K-Means (2 Clusters; raw features)	0.7	0.0	1.0	—	0.7	0.0	1.0
K-Means (2 Clusters; after PCA)	0.7	0.0	1.0	—	0.7	0.0	1.0

Table 10.3.: Comparison of different classification algorithms for difficulty prediction. Since we focus on the number of correct predictions of “difficult”, the TPR column is highlighted.

classification algorithm on the same data. During the different trainings, different features are considered as being most informative for data separation. We presume that these inconsistencies and the bad performance of the classifiers are caused by two effects. As discussed in Section 9.2, the ground truth labels we define may be incorrect or noisy. Also, the features we compute on the MSA and via a single RAxML-NG tree search might not sufficiently describe the difficulty of a dataset. In the next chapter, we suggest further measures to address these issues.

11. Further Experiments

By the time of completing this thesis, the task of predicting the difficulty of datasets remains unsolved. We presented a number of approaches that result in unsatisfactory predictions. These results can be caused by either unreliable ground truth labels, or insufficient features (Section 10.2). Future research in this area should therefore investigate the inference of correct labels for training datasets. This can be achieved by using additional or different indicators of difficulty, or a combination of multiple indicators. In Section 9.2 we presented different features based on the tree search trees, evaluation trees, and plausible trees. In further experiments, we could include additional features such as the standard deviation of likelihood values as an indicator of multiple likelihood peaks. More work should be put into generating easy and difficult simulated datasets, to enhance the amount of training data. In our experiments, we use unsupervised and supervised learning methods. Researchers developed further methods that rely on supervised learning on only a small set of labeled training data and a large set of unlabeled data [58]. Such techniques could limit the required amount of correct ground-truth difficulty labels while yielding satisfactory results.

To improve the classification, we suggest including the currently unimplemented features we presented in Section 10.1. We further suggest incorporating the content of the MSA: how many genes does the MSA comprise, and how closely related are the taxa presumably. This requires incorporating information from previous phylogenetic analyses and the tree of life. Further work can be done in improving the classifiers, for example by penalizing uncertain predictions or favoring predicting “easy”.

Part IV.

Conclusion and Future Work

12. Discussion

In the introduction of this thesis (Chapter 1), we formulated two goals: Decrease the inference time while yielding equally likely phylogenetic trees, and prevent unnecessary tree inferences. We addressed the first goal in Part II by investigating the influence of distinct numerical thresholds on the ML scores and runtimes of RAxML-NG, IQ-Tree, and FastTree. We showed that we can decrease the tree inference time for RAxML-NG and IQ-Tree by changing the default values for two thresholds. We further showed that the quality of the resulting trees is not affected by these changes. In order to prevent unnecessary tree inferences, we attempt to predict the difficulty of datasets in Part III. By the time of completing this thesis, this challenge remains unsolved. In the following, we discuss the key findings of both challenges in more detail.

Numerical Properties of Maximum Likelihood Inference Tools that infer a phylogenetic tree under the ML model rely on search heuristics. These heuristics use several internal numerical thresholds. We investigated the influence of selected numerical thresholds on the ML scores and runtimes of RAxML-NG, IQ-Tree, and FastTree. For all three tools, we investigated the influence of the threshold settings on tree inference. For RAxML-NG and IQ-Tree, we additionally investigated the resulting ML scores and runtimes when varying the numerical threshold during the evaluation of fixed trees. For the evaluation phase, we do not notice a substantial influence of any threshold, as long as the threshold values are in a reasonable range of values. We suggested such a range of values in Section 6.1. If we vary the thresholds during the tree search phase, we notice influences of the $lh_epsilon$ threshold on all three inference tools, and an influence of the RAxML-NG-specific threshold $spr_lh_epsilon$. We show that by changing the respective default values, we can decrease the runtime of tree inferences for RAxML-NG and IQ-Tree. For RAxML-NG, we suggest a change of the default $lh_epsilon$ threshold to 10 and the default $spr_lh_epsilon$ threshold to 10^3 . Under these new settings, we observe a speedup of 1.9 ± 0.6 for our test data. For IQ-Tree, we suggest a change of the default $lh_epsilon$ value to 10, resulting in an average speedup of 1.3 ± 0.4 . We showed that trees inferred under these new threshold settings are equally likely as trees inferred under the current default settings. For the remaining numerical thresholds, we confirmed that their current default settings are reasonable. By comparing the ML scores and runtimes of RAxML-NG, IQ-Tree, and FastTree over all datasets, we showed that RAxML-NG and IQ-Tree outperform FastTree in terms of ML scores, but FastTree is by far the fastest among the three ML inference tools.

Predicting Dataset Difficulty We presented our attempts to predict how difficult a dataset is to analyze. We reviewed and quantified definitions of difficulty from the literature, and assigned ground-truth difficulty labels based on this quantification of difficulty to the training datasets. We presume that the inferred labels appear to insufficiently represent the difficulty of data. Hence, further work in inferring reliable ground-truth labels is needed. Despite these challenges in inferring correct labels, we attempted to train classification machine learning

algorithms. We showed that the resulting predictions are not satisfactory and presented a set of possible further experiments.

13. Outlook

In this thesis, due to the limited amount of time, we only investigated the influence of seven distinct numerical thresholds. The tree inference heuristics implemented in RAxML-NG, IQ-Tree, and FastTree use additional thresholds. During the preparation phase, we hand-selected the thresholds we believed to influence the ML scores and the runtimes to the largest degree. In further experiments, additional thresholds could be included in the analyses.

In our analyses, we focused on assessing the influences of the thresholds separately. In future analyses, we could investigate correlated influences of multiple numerical thresholds.

Tree inferences with FastTree are considerably faster than with RAxML-NG or IQ-Tree. However, RAxML-NG and IQ-Tree yield more accurate trees. An interesting experiment would be to first infer trees using FastTree and then re-evaluate these trees with RAxML-NG or IQ-Tree. Since the runtimes of the evaluation modes in RAxML-NG and IQ-Tree are only a small fraction of the tree inference time ($5 \pm 5\%$ for RAxML-NG, $4 \pm 3\%$ for IQ-Tree; Section 6.1), this could combine the runtime advantages of FastTree with the accuracy advantages of RAxML-NG or IQ-Tree. Another experiment could be to use a FastTree tree as starting topologies in RAxML-NG and IQ-Tree.

We showed that IQ-Tree is sensitive to large *lh_epsilon* settings, and our analysis indicates that this is caused by the randomness in the IQ-Tree search heuristics. An interesting experiment would be to introduce a second likelihood epsilon value that is used during the hill-climbing optimization cycles of the *stochastic* NNI moves (Section 3.6.2). Setting this additional likelihood epsilon value to a small value and using a large *lh_epsilon* value for the outer optimization cycle could result in a substantial speedup.

In future research, we aim to solve the task of difficulty prediction. In Chapter 11 we presented a number of possible further measures to obtain more reliable ground-truth difficulty labels and additional training data. We further presented possible approaches to improve the results of the difficulty prediction algorithms.

Glossary

bfgs_factor Controls the convergence of the L-BFGS-B method used for optimization of substitution rates and stationary frequencies. 10, 17, 19, 24, 25, 39, 68, 71, 76

lh_epsilon Epsilon value for ML score improvement after one iteration of full optimization (tree topology, branch lengths and model parameters). i, ii, vi–viii, 14, 19, 21, 22, 24–28, 30–35, 58, 60, 68, 70, 73, 75, 79, 80

maxBranchLen Maximum branch length. Upper limit for the branch length values. 9, 19, 24, 25, 39, 68, 69, 72, 74, 78

minBranchLen Minimum branch length. Lower limit for the branch length values. vi, viii, 9, 16, 19, 21–23, 25, 34–38, 42, 43, 68, 69, 72, 74, 78, 80

model_epsilon Epsilon value for model parameter improvement. 14, 19, 24, 25, 39, 68, 70, 73, 75, 79

num_iters Parameter to control the maximum number of iterations during branch length optimization in RAxML-NG. 12, 17, 19, 24, 25, 39, 68, 71, 77

spr_lh_epsilon Epsilon value for ML score improvement after one SPR round in RAxML-NG. i, ii, vii, 14, 17, 19, 21, 25, 27–30, 58, 68, 76

RF-distance Robinson–Foulds (RF) distance. 27, 46, 47, 51

Bibliography

- [1] S. Abadi, O. Avram, S. Rosset, T. Pupko, and I. Mayrose, “ModelTeller: Model Selection for Optimal Phylogenetic Reconstruction Using Machine Learning”, *Molecular Biology and Evolution*, vol. 37, no. 11, pp. 3338–3352, 2020. DOI: 10.1093/molbev/msaa154.
- [2] O. Bánki, Y. Roskov, L. Vandepitte, R. E. DeWalt, D. Remsen, P. Schalk, T. Orrell, M. Keping, J. Miller, R. Aalbu, R. Adlard, E. Adriaenssens, C. Aedo, E. Aesch, N. Akkari, M. A. Alonso-Zarazaga, B. Alvarez, F. Alvarez, and G. e. a. Anderson, “Catalogue of Life Checklist”, 2021, Version 2021-09-21. DOI: 10.48580/d4sv.
- [3] J. P. Bollback, “Bayesian Model Adequacy and Choice in Phylogenetics”, *Molecular Biology and Evolution*, vol. 19, no. 7, pp. 1171–1180, 2002. DOI: 10.1093/oxfordjournals.molbev.a004175.
- [4] B. E. Boser, I. M. Guyon, and V. N. Vapnik, “A Training Algorithm for Optimal Margin Classifiers”, in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, Association for Computing Machinery, 1992, pp. 144–152. DOI: 10.1145/130385.130401.
- [5] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Wadsworth and Brooks, 1984.
- [6] R. P. Brent, “An algorithm with guaranteed convergence for finding a zero of a function”, *The Computer Journal*, vol. 14, no. 4, pp. 422–425, 1971. DOI: 10.1093/comjnl/14.4.422.
- [7] E. Britannica. “Moore’s law”. (2019), [Online]. Available: <https://www.britannica.com/technology/Moores-law> (visited on 10/13/2021).
- [8] L. L. Cavalli-Sforza and A. W. F. Edwards, “Phylogenetic analysis. Models and estimation procedures”, *Evolution*, vol. 21, no. 3, pp. 550–570, 1967. DOI: 10.1111/j.1558-5646.1967.tb03411.x.
- [9] M. W. Chase, D. E. Soltis, R. G. Olmstead, D. Morgan, D. H. Les, B. D. Mishler, M. R. Duvall, R. A. Price, H. G. Hills, Y.-L. Qiu, K. A. Kron, J. H. Rettig, E. Conti, J. D. Palmer, J. R. Manhart, K. J. Sytsma, H. J. Michaels, W. J. Kress, K. G. Karol, W. D. Clark, M. Hedren, B. S. Gaut, R. K. Jansen, K.-J. Kim, C. F. Wimpee, J. F. Smith, G. R. Furnier, S. H. Strauss, Q.-Y. Xiang, G. M. Plunkett, P. S. Soltis, S. M. Swensen, S. E. Williams, P. A. Gadek, C. J. Quinn, L. E. Eguiarte, E. Golenberg, G. H. Learn, S. W. Graham, S. C. H. Barrett, S. Dayanandan, and V. A. Albert, “Phylogenetics of Seed Plants: An Analysis of Nucleotide Sequences from the Plastid Gene *rbcl*”, *Annals of the Missouri Botanical Garden*, vol. 80, no. 3, pp. 528–580, 1993. DOI: 10.2307/2399846.
- [10] B. Chor and T. Tuller, “Maximum likelihood of evolutionary trees: hardness and approximation”, *Bioinformatics*, vol. 21, pp. i97–i106, 2005. DOI: 10.1093/bioinformatics/bti1027.

- [11] T. J. Dekker, “Finding a zero by means of successive linear interpolation”, in *Constructive Aspects of the Fundamental Theorem of Algebra*, B. Dejon and P. Henrici, Eds., New York: Interscience, 1969.
- [12] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison, *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998. DOI: 10.1017/CB09780511790492.
- [13] EMBL-EBI. “Phylogenetics – An introduction”. (2021), [Online]. Available: <https://www.ebi.ac.uk/training/online/courses/introduction-to-phylogenetics/> (visited on 08/23/2021).
- [14] K. F.R.S. Pearson, “LIII. On lines and planes of closest fit to systems of points in space”, *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901. DOI: 10.1080/14786440109462720.
- [15] J. S. Farris, “Methods for Computing Wagner Trees”, *Systematic Biology*, vol. 19, no. 1, pp. 83–92, 1970. DOI: 10.1093/sysbio/19.1.83.
- [16] J. Felsenstein, “Maximum Likelihood and Minimum-Steps Methods for Estimating Evolutionary Trees from Data on Discrete Characters”, *Systematic Zoology*, vol. 22, no. 3, pp. 240–249, 1973. DOI: 10.2307/2412304.
- [17] J. Felsenstein, “Evolutionary trees from DNA sequences: A maximum likelihood approach”, *Journal of Molecular Evolution*, vol. 17, pp. 368–376, 2005.
- [18] W. M. Fitch, “Toward Defining the Course of Evolution: Minimum Change for a Specific Tree Topology”, *Systematic Zoology*, vol. 20, no. 4, pp. 406–416, 1971. DOI: 10.2307/2412116.
- [19] E. Fix and J. L. Hodges, “Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties”, *International Statistical Review / Revue Internationale de Statistique*, vol. 57, no. 3, pp. 238–247, 1989. DOI: 10.2307/1403797.
- [20] R. Fletcher, *Practical Methods of Optimization*. John Wiley & Sons, Ltd, 2000. DOI: 10.1002/9781118723203.ch3.
- [21] Y. Freund and R. E. Schapire, “Experiments with a New Boosting Algorithm”, in *Proceedings of the Thirteenth International Conference on International Conference on Machine Learning*, ser. ICML’96, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996, pp. 148–156.
- [22] O. Gascuel, “BIONJ: An Improved Version of the NJ Algorithm Based on a Simple Model of Sequence Data”, *Molecular biology and evolution*, vol. 14, pp. 685–95, 1997. DOI: 10.1093/oxfordjournals.molbev.a025808.
- [23] N. GenBank. “GenBank and WGS Statistics”. (2021), [Online]. Available: <https://www.ncbi.nlm.nih.gov/genbank/statistics/> (visited on 10/13/2021).
- [24] Z. Ghahramani, “Unsupervised Learning”, in *Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2 - 14, 2003, Tübingen, Germany, August 4 - 16, 2003, Revised Lectures*, O. Bousquet, U. von Luxburg, and G. Rätsch, Eds. Springer Berlin Heidelberg, 2004, pp. 72–112. DOI: 10.1007/978-3-540-28650-9_5.

- [25] N. Goldman, “Simple diagnostic statistical tests of models for DNA substitution.”, *Journal of Molecular Evolution*, vol. 37, no. 6, pp. 650–661, 1993. DOI: 10.1007/BF00182751.
- [26] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, <http://www.deeplearningbook.org>.
- [27] I. Gregoret, Y.-M. Lee, and H. V. Goodson, “Molecular Evolution of the Histone Deacetylase Family: Functional Implications of Phylogenetic Analysis”, *Journal of Molecular Biology*, vol. 338, no. 1, pp. 17–31, 2004. DOI: 10.1016/j.jmb.2004.02.006.
- [28] G. W. Grimm, S. S. Renner, A. Stamatakis, and V. Hemleben, “A Nuclear Ribosomal DNA Phylogeny of *Acer* Inferred with Maximum Likelihood, Splits Graphs, and Motif Analysis of 606 Sequences”, *Evolutionary Bioinformatics*, vol. 2, 2006. DOI: 10.1177/117693430600200014.
- [29] S. Guindon and O. Gascuel, “A Simple, Fast, and Accurate Algorithm to Estimate Large Phylogenies by Maximum Likelihood”, *Systematic Biology*, vol. 52, no. 5, pp. 696–704, 2003. DOI: 10.1080/10635150390235520.
- [30] L. S. Heath and N. Ramakrishnan, *Problem Solving Handbook in Computational Biology and Bioinformatics*, 1st. Springer-Verlag, 2010. DOI: 10.1007/978-0-387-09760-2.
- [31] S. B. Hedges, K. D. Moberg, and L. R. Maxson, “Tetrapod phylogeny inferred from 18S and 28S ribosomal RNA sequences and a review of the evidence for amniote relationships.”, *Molecular Biology and Evolution*, vol. 7, no. 6, pp. 607–633, 1990. DOI: 10.1093/oxfordjournals.molbev.a040628.
- [32] T. K. Ho, “Random decision forests”, in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, vol. 1, 1995, pp. 278–282. DOI: 10.1109/ICDAR.1995.598994.
- [33] B. R. Holland, K. T. Huber, A. Dress, and V. Moulton, “ δ Plots: A Tool for Analyzing Phylogenetic Distance Data”, *Molecular Biology and Evolution*, vol. 19, no. 12, pp. 2051–2059, 2002. DOI: 10.1093/oxfordjournals.molbev.a004030.
- [34] N. H. G. R. Institute. “The Cost of Sequencing a Human Genome”. (2020), [Online]. Available: <http://genome.gov/sequencingcosts> (visited on 10/13/2021).
- [35] T. H. Jukes and C. R. Cantor, “CHAPTER 24 - Evolution of Protein Molecules”, in *Mammalian Protein Metabolism*, H. MUNRO, Ed., Academic Press, 1969, pp. 21–132. DOI: 10.1016/B978-1-4832-3211-9.50009-7.
- [36] P. Kalaghatgi, “Phylogeny inference under the general Markov model using MST-backbone”, *bioRxiv*, 2020. DOI: 10.1101/2020.06.30.180315.
- [37] H. Kishino and M. Hasegawa, “Evaluation of the maximum likelihood estimate of the evolutionary tree topologies from DNA sequence data, and the branching order in hominoidea”, *Journal of molecular evolution*, vol. 29, no. 2, pp. 170–179, 1989. DOI: 10.1007/bf02100115.
- [38] J. Köster and S. Rahmann, “Snakemake—a scalable bioinformatics workflow engine”, *Bioinformatics*, vol. 28, no. 19, pp. 2520–2522, 2012. DOI: 10.1093/bioinformatics/bts480.

- [39] A. M. Kozlov, D. Darriba, T. Flouri, B. Morel, and A. Stamatakis, “RAxML-NG: a fast, scalable and user-friendly tool for maximum likelihood phylogenetic inference”, *Bioinformatics*, vol. 35, no. 21, pp. 4453–4455, 2019. DOI: 10.1093/bioinformatics/btz305.
- [40] O. Kozlov, “Models, Optimizations, and Tools for Large-Scale Phylogenetic Inference, Handling Sequence Uncertainty, and Taxonomic Validation”, Ph.D. dissertation, Karlsruher Institut für Technologie (KIT), 2018. DOI: 10.5445/IR/1000081661.
- [41] C. Lakner, P. van der Mark, J. P. Huelsenbeck, B. Larget, and F. Ronquist, “Efficiency of Markov Chain Monte Carlo Tree Proposals in Bayesian Phylogenetics”, *Systematic Biology*, vol. 57, no. 1, pp. 86–103, 2008. DOI: 10.1080/10635150801886156.
- [42] M. S. Lee and A. Palci, “Morphological Phylogenetics in the Genomic Age”, *Current Biology*, vol. 25, no. 19, R922–R929, 2015. DOI: 10.1016/j.cub.2015.07.009.
- [43] P. Lemey, M. Salemi, and A.-M. Vandamme, *The Phylogenetic Handbook: A Practical Approach to Phylogenetic Analysis and Hypothesis Testing*, 2nd ed. Cambridge University Press, 2009. DOI: 10.1017/CB09780511819049.
- [44] J. Lyons-Weiler, G. A. Hoelzer, and R. J. Tausch, “Relative apparent synapomorphy analysis (RASA). I: The statistical measurement of phylogenetic signal.”, *Molecular Biology and Evolution*, vol. 13, no. 6, pp. 749–757, 1996. DOI: 10.1093/oxfordjournals.molbev.a025635.
- [45] J. Macqueen, “Some methods for classification and analysis of multivariate observations”, in *In 5-th Berkeley Symposium on Mathematical Statistics and Probability*, 1967, pp. 281–297.
- [46] E. R. Mardis, “The impact of next-generation sequencing technology on genetics”, *Trends in Genetics*, vol. 24, no. 3, pp. 133–141, 2008. DOI: 10.1016/j.tig.2007.12.007.
- [47] M. L. Metzker, D. P. Mindell, X.-M. Liu, R. G. Ptak, R. A. Gibbs, and D. M. Hillis, “Molecular evidence of HIV-1 transmission in a criminal case”, *Proceedings of the National Academy of Sciences*, vol. 99, no. 22, pp. 14 292–14 297, 2002. DOI: 10.1073/pnas.222522599.
- [48] B. Q. Minh, H. A. Schmidt, O. Chernomor, D. Schrempf, M. D. Woodhams, A. von Haeseler, and R. Lanfear, “IQ-TREE 2: New Models and Efficient Methods for Phylogenetic Inference in the Genomic Era”, *Molecular Biology and Evolution*, vol. 37, no. 5, pp. 1530–1534, 2020. DOI: 10.1093/molbev/msaa015.
- [49] B. Morel, P. Barbera, L. Czech, B. Bettisworth, L. Hübner, S. Lutteropp, D. Serdari, E.-G. Kostaki, I. Mamais, A. M. Kozlov, P. Pavlidis, D. Paraskevis, and A. Stamatakis, “Phylogenetic Analysis of SARS-CoV-2 Data Is Difficult”, *Molecular Biology and Evolution*, vol. 38, no. 5, pp. 1777–1791, 2020. DOI: 10.1093/molbev/msaa314.
- [50] M. Park, P. Zaharias, and T. Warnow, “Disjoint Tree Mergers for Large-Scale Maximum Likelihood Tree Estimation”, *Algorithms*, vol. 14, no. 5, 2021. DOI: 10.3390/a14050148.
- [51] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine Learning in Python”, *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

- [52] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing*. USA: Cambridge University Press, 1988, ISBN: 052135465X.
- [53] M. N. Price, P. S. Dehal, and A. P. Arkin, “FastTree: Computing Large Minimum Evolution Trees with Profiles instead of a Distance Matrix”, *Molecular Biology and Evolution*, vol. 26, no. 7, pp. 1641–1650, 2009. DOI: 10.1093/molbev/msp077.
- [54] M. N. Price, P. S. Dehal, and A. P. Arkin, “FastTree 2 – Approximately Maximum-Likelihood Trees for Large Alignments”, *PLOS ONE*, vol. 5, no. 3, pp. 1–10, 2010. DOI: 10.1371/journal.pone.0009490.
- [55] S. Ranganathan, K. Nakai, and C. Schonbach, *Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics*. Elsevier Science, 2018, ISBN: 9780128114322.
- [56] D. Robinson and L. Foulds, “Comparison of phylogenetic trees”, *Mathematical Biosciences*, vol. 53, no. 1, pp. 131–147, 1981. DOI: 10.1016/0025-5564(81)90043-2.
- [57] N. Saitou and M. Nei, “The neighbor-joining method: a new method for reconstructing phylogenetic trees.”, *Molecular Biology and Evolution*, vol. 4, no. 4, pp. 406–425, 1987. DOI: 10.1093/oxfordjournals.molbev.a040454.
- [58] M. Seeger, “Learning With Labeled and Unlabeled Data”, 2001.
- [59] C. E. Shannon, “A mathematical theory of communication”, *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948. DOI: 10.1002/j.1538-7305.1948.tb01338.x.
- [60] H. Shimodaira and M. Hasegawa, “Multiple Comparisons of Log-Likelihoods with Applications to Phylogenetic Inference”, *Molecular Biology and Evolution*, vol. 16, no. 8, 1999. DOI: 10.1093/oxfordjournals.molbev.a026201.
- [61] H. Shimodaira, “An Approximately Unbiased Test of Phylogenetic Tree Selection”, *Systematic biology*, vol. 51, pp. 492–508, 2002. DOI: 10.1080/10635150290069913.
- [62] S. Song, L. Liu, S. V. Edwards, and S. Wu, “Resolving conflict in eutherian mammal phylogeny using phylogenomics and the multispecies coalescent model”, *Proceedings of the National Academy of Sciences*, vol. 109, no. 37, pp. 14 942–14 947, 2012. DOI: 10.1073/pnas.1211733109.
- [63] A. Stamatakis, “Distributed and Parallel Algorithms and Systems for Inference of Huge Phylogenetic Trees based on the Maximum Likelihood Method”, Dissertation, Technische Universität München, München, 2004.
- [64] A. Stamatakis, “Phylogenetics: Applications, Software and Challenges”, *Cancer Genomics & Proteomics*, vol. 2, no. 5, pp. 301–305, 2005.
- [65] A. Stamatakis, “Phylogenetic models of rate heterogeneity: a high performance computing perspective”, in *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*, 2006. DOI: 10.1109/IPDPS.2006.1639535.
- [66] A. Stamatakis, “Phylogenetic Search Algorithms for Maximum Likelihood”, in *Algorithms in Computational Molecular Biology*. John Wiley & Sons, Ltd, 2011, ch. 25, pp. 547–577. DOI: 10.1002/9780470892107.ch25.

- [67] A. Stamatakis, “RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies”, *Bioinformatics*, vol. 30, no. 9, pp. 1312–1313, 2014. DOI: 10.1093/bioinformatics/btu033.
- [68] A. Stamatakis, M. Göker, and G. W. Grimm, “Maximum Likelihood Analyses of 3,490 rbcL Sequences: Scalability of Comprehensive Inference versus Group-Specific Taxon Sampling”, *Evolutionary Bioinformatics*, vol. 6, EBO.S4528, 2010. DOI: 10.4137/EB0.S4528.
- [69] A. Stamatakis, P. Hoover, and J. Rougemont, “A Rapid Bootstrap Algorithm for the RAxML Web Servers”, *Systematic Biology*, vol. 57, no. 5, pp. 758–771, 2008. DOI: 10.1080/10635150802429642.
- [70] C. Stewart, D. Hart, D. Berry, G. Olsen, E. Wernert, and W. Fischer, “Parallel Implementation and Performance of FastDNAm1 - A Program for Maximum Likelihood Phylogenetic Inference”, in *SC '01: Proceedings of the 2001 ACM/IEEE Conference on Supercomputing*, 2001. DOI: 10.1145/582034.582054.
- [71] K. Strimmer and A. Rambaut, “Inferring confidence sets of possibly misspecified gene trees.”, in *Proceedings. Biological sciences*, 2002, pp. 137–142. DOI: 10.1098/rspb.2001.1862.
- [72] S. Tavaré, “Some probabilistic and statistical problems on the analysis of DNA sequences”, *Lectures on Mathematics in the Life Sciences*, vol. 17, pp. 57–86, 1986.
- [73] C. W. Team, R. Pachauri, and L. M. (eds.), “Climate Change 2014: Synthesis Report. Contribution of Working Groups I, II and III to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change”, 2014.
- [74] Z. Xi, L. Liu, J. S. Rest, and C. C. Davis, “Coalescent versus Concatenation Methods and the Placement of Amborella as Sister to Water Lilies”, *Systematic Biology*, vol. 63, no. 6, pp. 919–932, 2014. DOI: 10.1093/sysbio/syu055.
- [75] Z. Yang, “A space-time process model for the evolution of DNA sequences.”, *Genetics*, vol. 139, no. 2, pp. 993–1005, 1995. DOI: 10.1093/genetics/139.2.993.
- [76] Z. Yang, “Computational Molecular Evolution”, *Oxford University Press*, P. H. Harvey and R. M. May, Eds., 2006. DOI: 10.1093/acprof:oso/9780198567028.001.0001.
- [77] C. Young, S. Meng, and N. Moshiri, “An Evaluation of Phylogenetic Workflows in Viral Molecular Epidemiology”, *bioRxiv*, 2020. DOI: 10.1101/2020.11.24.396820.
- [78] X. Zhou, X.-X. Shen, C. T. Hittinger, and A. Rokas, “Evaluating Fast Maximum Likelihood-Based Phylogenetic Programs Using Empirical Phylogenomic Data Sets”, *Molecular Biology and Evolution*, vol. 35, no. 2, pp. 486–503, 2017. DOI: 10.1093/molbev/msx302.
- [79] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal, “Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-Scale Bound-Constrained Optimization”, *ACM Trans. Math. Softw.*, vol. 23, no. 4, pp. 550–560, 1997. DOI: 10.1145/279232.279236.

A. Appendix

A.1. Numerical Properties of Maximum Likelihood Inference

A.1.1. Data Generation Pipeline

In our analysis pipeline we use RAxML-NG, IQ-Tree, and FastTree to infer phylogenetic trees. We additionally re-evaluate trees with RAxML-NG and IQ-Tree. Furthermore, we use the significance tests implemented in IQ-Tree. Table A.1 states the command lines used for the respective task and tree inference tool.

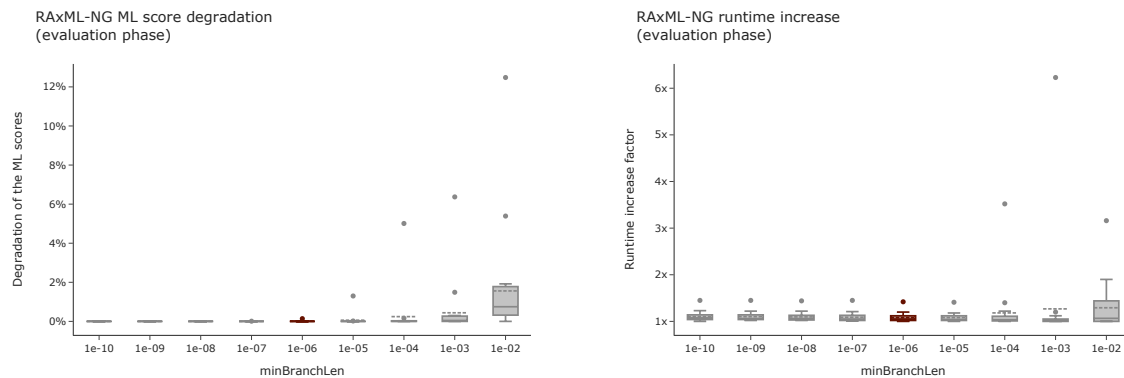
Mode	Tool	Command Line
Search	RAxML-NG	<code>raxml-ng --msa <MSA> --model GTR+G --tree pars{1} --blmin <minBranchLen> --blmax <maxBranchLen> --lh-epsilon <lh_epsilon> --param-eps <model_epsilon> --brlen-smoothings <num_iters> --spr-lheps <spr_lh_epsilon> --bfgs-factor <bfgs_factor> --seed <SEED> --prefix <OUTDIR> --threads 2</code>
Search	IQ-Tree	<code>iqtree -s <MSA> -m GTR+FO+G4 -ninit 1 -blmin <minBranchLen> -blmax <maxBranchLen> -me <model_epsilon> -eps <lh_epsilon> -seed <SEED> -pre <OUTDIR> -nt 2</code>
Search	FastTree	<code>fasttree -gtr -gamma -cat 4 -nt -blmin <minBranchLen> -lheps <lh_epsilon> -seed <SEED> <<MSA>> <OUTDIR></code>
Evaluation	RAxML-NG	<code>raxml-ng --eval --msa <MSA> --model GTR+G --tree <TREE> --blmin <minBranchLen> --blmax <maxBranchLen> --lh-epsilon <lh_epsilon> --param-eps <model_epsilon> --brlen-smoothings <num_iters> --bfgs-factor <bfgs_factor> --seed 0 --prefix <OUTDIR> --threads 2</code>
Evaluation	IQ-Tree	<code>iqtree -s <MSA> -m GTR+FO+G4 -te <TREE> -blmin <minBranchLen> -blmax <maxBranchLen> -me <model_epsilon> -eps <lh_epsilon> -seed 0 -pre <OUTDIR> -nt 2</code>
Significance Tests	IQ-Tree	<code>iqtree -s <MSA> -m GTR+FO+G4 -z <ALL_TREES> -te<BEST_TREE> -n 0 -zb 10000 -zw -au -pre <OUTDIR> -nt 2</code>

Table A.1.: The command lines we use to conduct tree inferences, re-evaluations and significance tests with RAxML-NG, IQ-Tree, and FastTree.

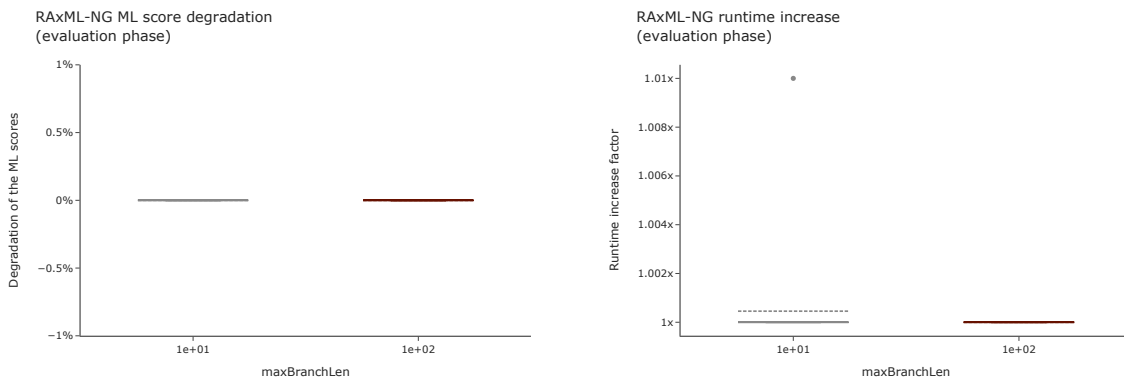
A.1.2. Influence of Numerical Thresholds

A.1.2.1. RAxML-NG Evaluation Phase

The following figures show the influence of all numerical thresholds on the ML scores and runtimes of RAxML-NG if varied during the evaluation phase. Due to the broad range of absolute likelihood values, we plot the degradation of ML scores in percent relative to the best ML score per dataset. The runtimes state how much longer the evaluation phase runs compared to the fastest evaluation phase per dataset.

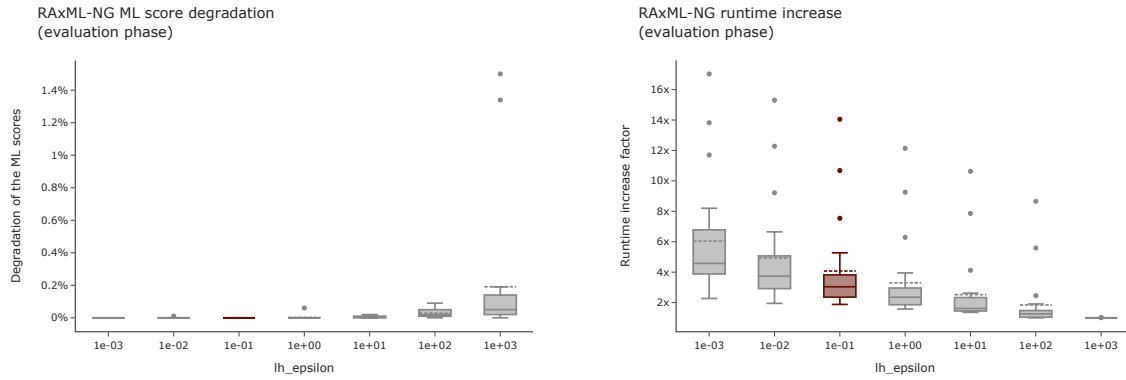


- (a) Degradation of ML scores across all datasets as a function of $minBranchLen$ values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores.
- (b) Increase of the tree search time across all datasets as a function of $minBranchLen$ values. The increase is measured relative to the minimum runtime per dataset.

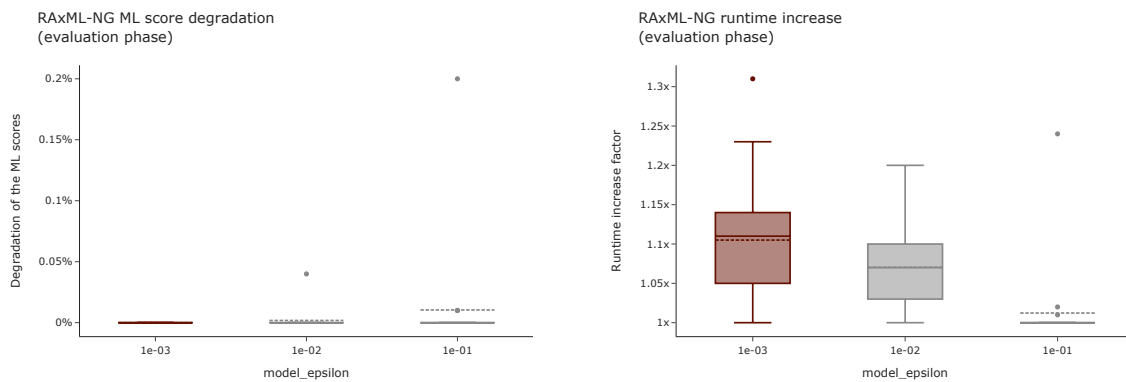


- (c) Degradation of ML scores across all datasets as a function of $maxBranchLen$ values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores.
- (d) Increase of the tree search time across all datasets as a function of $maxBranchLen$ values. The increase is measured relative to the minimum runtime per dataset.

A. Appendix

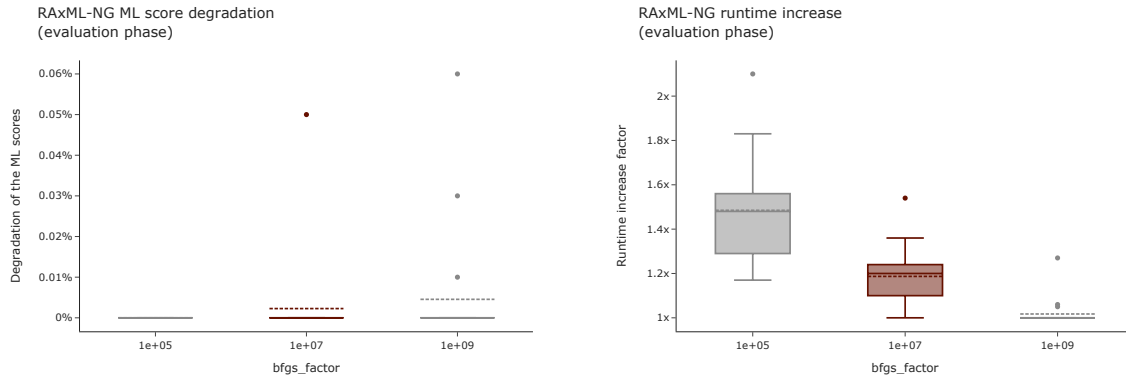


- (e) Degradation of ML scores across all datasets as a function of $lh_epsilon$ values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores.
- (f) Increase of the tree search time across all datasets as a function of $lh_epsilon$ values. The increase is measured relative to the minimum runtime per dataset.

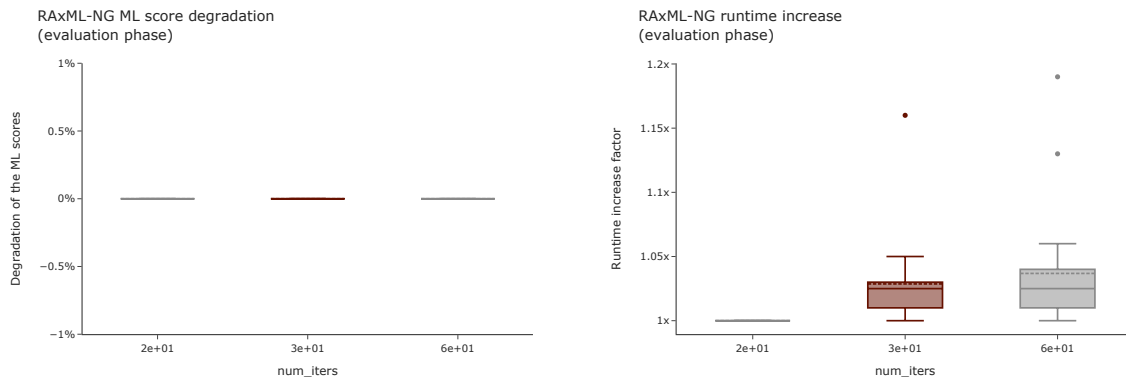


- (g) Degradation of ML scores across all datasets as a function of $model_epsilon$ values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores.
- (h) Increase of the tree search time across all datasets as a function of $model_epsilon$ values. The increase is measured relative to the minimum runtime per dataset.

A. Appendix



- (i) Degradation of ML scores across all datasets as a function of *bfgs_factor* values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores.
- (j) Increase of the tree search time across all datasets as a function of *bfgs_factor* values. The increase is measured relative to the minimum runtime per dataset.

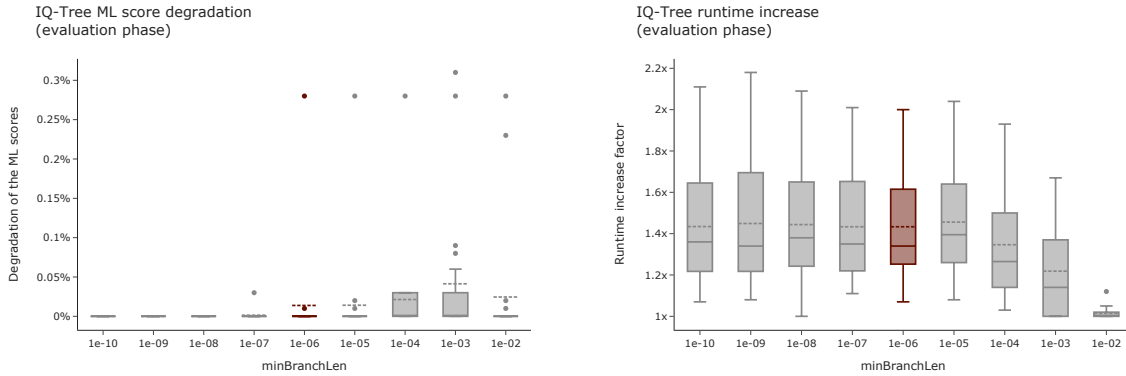


- (k) Degradation of ML scores across all datasets as a function of *num_iters* values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores.
- (l) Increase of the tree search time across all datasets as a function of *num_iters* values. The increase is measured relative to the minimum runtime per dataset.

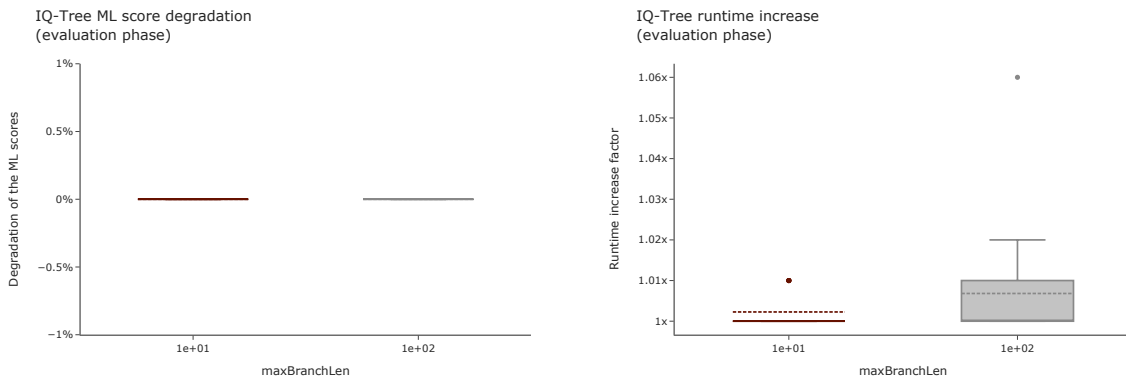
Figure A.1.: Influence of the numerical thresholds on the ML scores and runtimes of the RAxML-NG evaluation phase. Each plot summarizes the data over all datasets. The dashed vertical line indicates the mean, and the solid vertical line the median value. The highlighted box indicates the default value of the respective numerical threshold in RAxML-NG.

A.1.2.2. IQ-Tree Evaluation Phase

The following figures show the influence of all numerical thresholds on the ML scores and runtimes of IQ-Tree if varied during the evaluation phase. Due to the broad range of absolute likelihood values, we plot the degradation of ML scores in percent relative to the best ML score per dataset. The runtimes state how much longer the evaluation phase runs compared to the fastest evaluation phase per dataset.

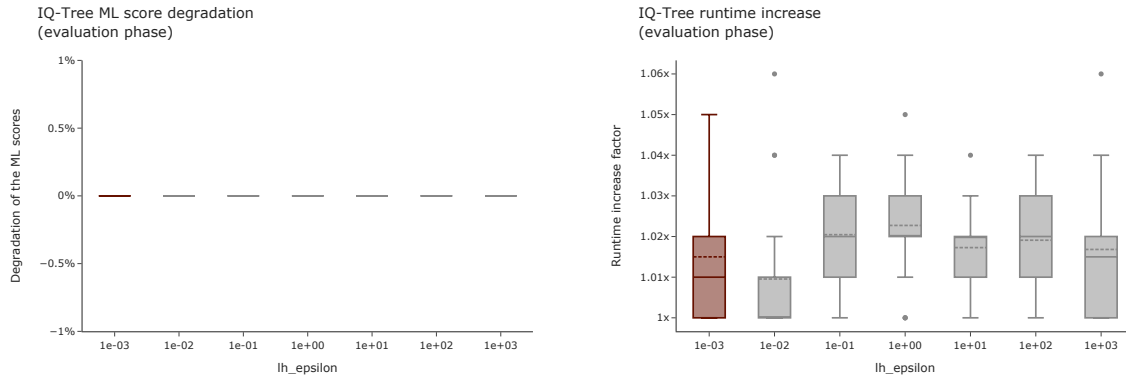


(a) Degradation of ML scores across all datasets as a function of $minBranchLen$ values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores. (b) Increase of the tree search time across all datasets as a function of $minBranchLen$ values. The increase is measured relative to the minimum runtime per dataset.

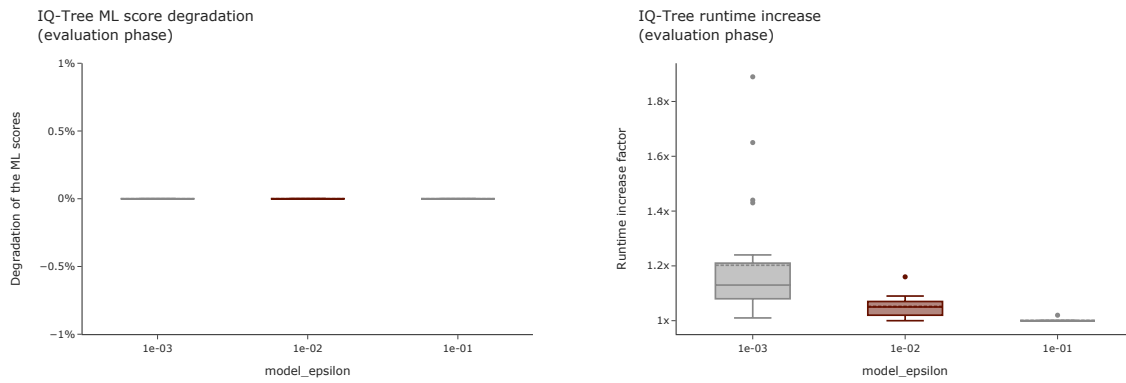


(c) Degradation of ML scores across all datasets as a function of $maxBranchLen$ values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores. (d) Increase of the tree search time across all datasets as a function of $maxBranchLen$ values. The increase is measured relative to the minimum runtime per dataset.

A. Appendix



- (e) Degradation of ML scores across all datasets as a function of $lh_epsilon$ values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores.
- (f) Increase of the tree search time across all datasets as a function of $lh_epsilon$ values. The increase is measured relative to the minimum runtime per dataset.

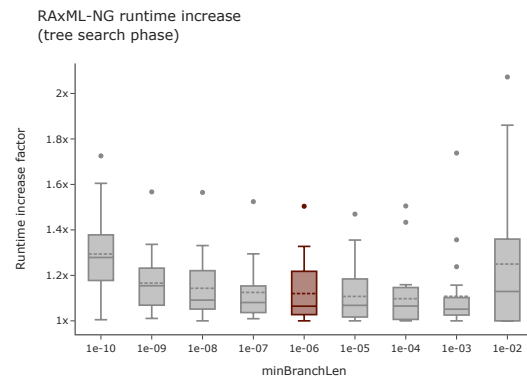
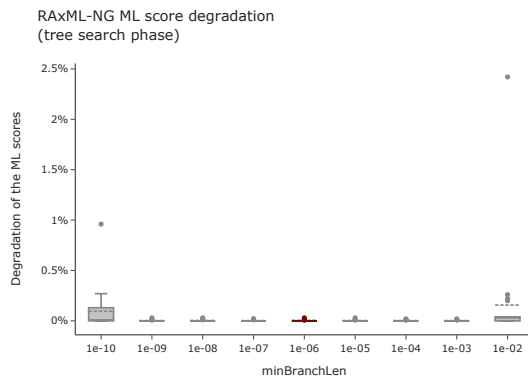


- (g) Degradation of ML scores across all datasets as a function of $model_epsilon$ values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores.
- (h) Increase of the tree search time across all datasets as a function of $model_epsilon$ values. The increase is measured relative to the minimum runtime per dataset.

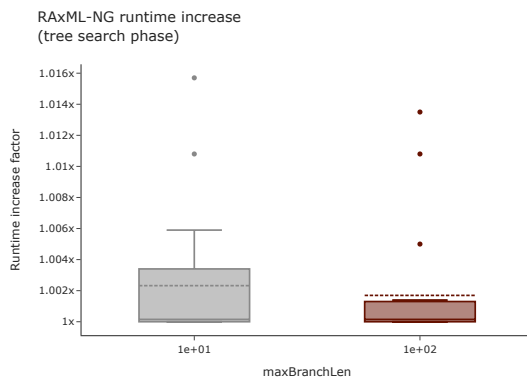
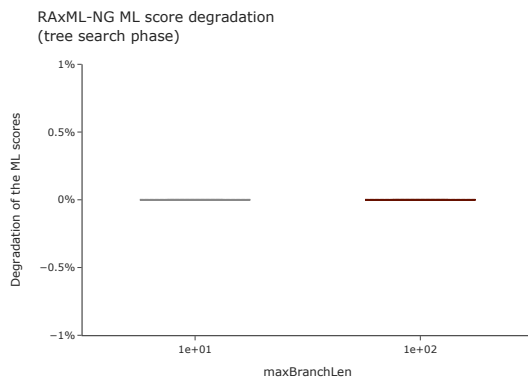
Figure A.2.: Influence of the numerical thresholds on the ML scores and runtimes of the IQ-Tree evaluation phase. Each plot summarizes the data over all datasets. The dashed vertical line indicates the mean, and the solid vertical line the median value. The highlighted box indicates the default value for the respective numerical threshold in IQ-Tree.

A.1.2.3. RAxML-NG Tree Search Phase

The following figures show the influence of all numerical thresholds on the ML scores and runtimes of RAxML-NG if varied during the tree search phase. The ML scores refer to the ML scores after the evaluation phase, and the runtimes refer to the runtimes of the tree search phase. Due to the broad range of absolute likelihood values, we plot the degradation of ML scores in percent relative to the best ML score per dataset. The runtimes state how much longer the tree inference runs compared to the fastest tree inference per dataset.

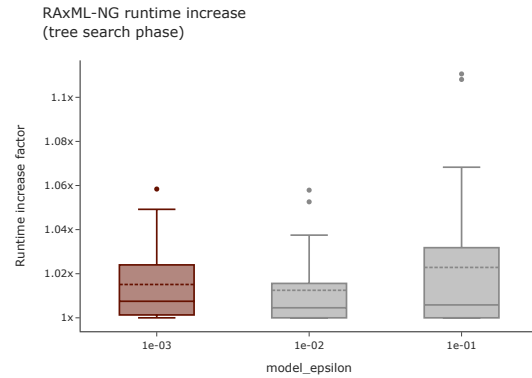
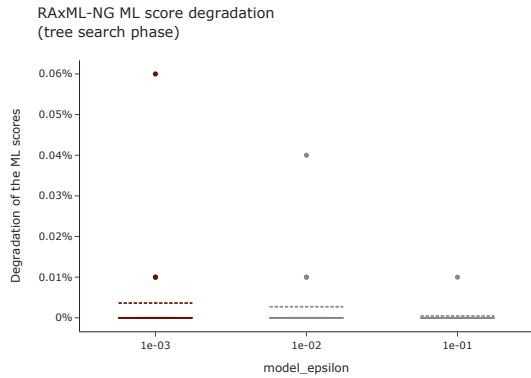


- (a) Degradation of ML scores across all datasets as a function of *minBranchLen* values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores.
- (b) Increase of the tree search time across all datasets as a function of *minBranchLen* values. The increase is measured relative to the minimum runtime per dataset.



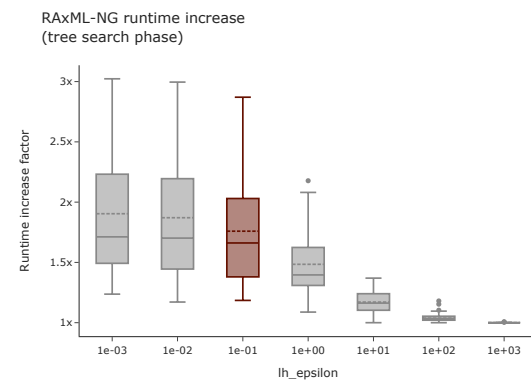
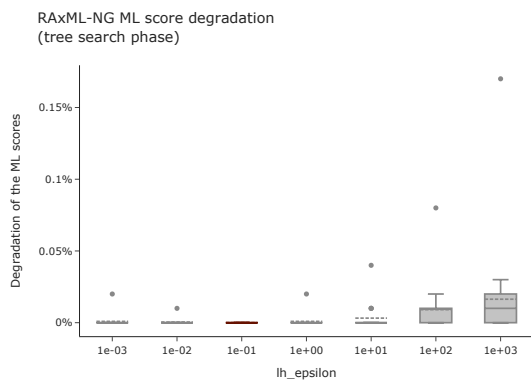
- (c) Degradation of ML scores across all datasets as a function of *maxBranchLen* values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores.
- (d) Increase of the tree search time across all datasets as a function of *maxBranchLen* values. The increase is measured relative to the minimum runtime per dataset.

A. Appendix



(e) Degradation of ML scores across all datasets as a function of $model_epsilon$ values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores.

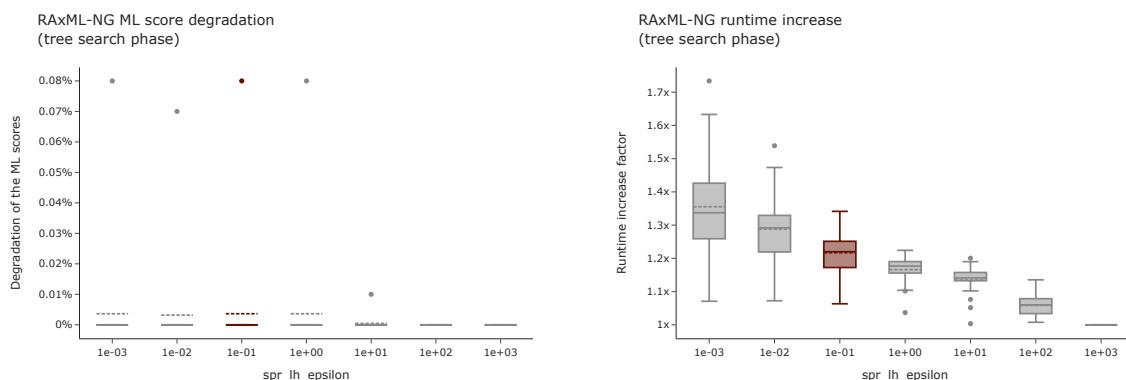
(f) Increase of the tree search time across all datasets as a function of $model_epsilon$ values. The increase is measured relative to the minimum runtime per dataset.



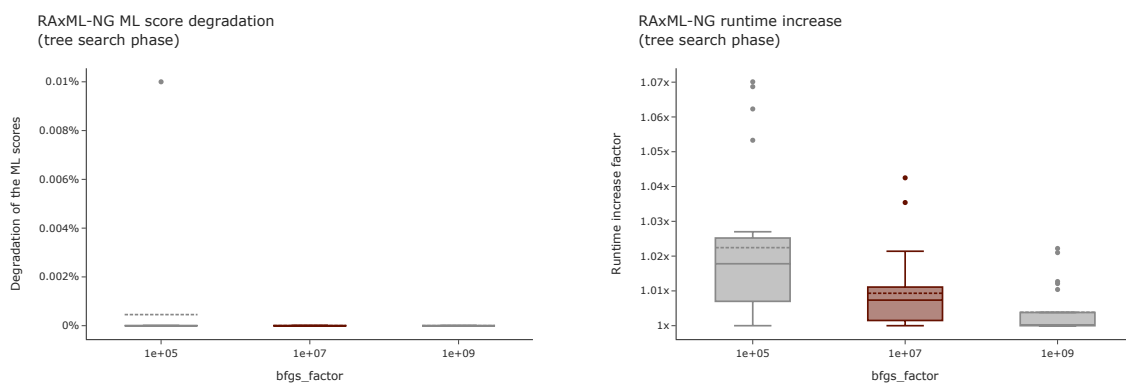
(g) Degradation of ML scores across all datasets as a function of $lh_epsilon$ values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores.

(h) Increase of the tree search time across all datasets as a function of $lh_epsilon$ values. The increase is measured relative to the minimum runtime per dataset.

A. Appendix

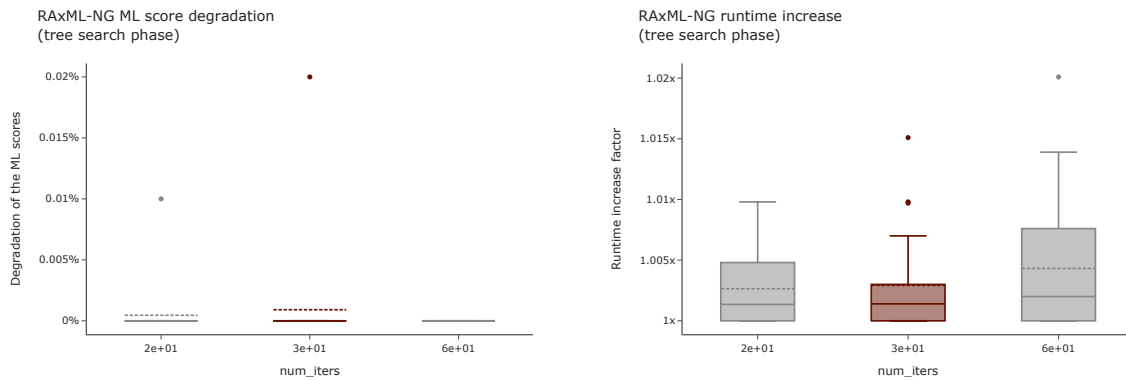


- (i) Degradation of ML scores across all datasets as a function of $spr_lh_epsilon$ values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores.
- (j) Increase of the tree search time across all datasets as a function of $spr_lh_epsilon$ values. The increase is measured relative to the minimum runtime per dataset.



- (k) Degradation of ML scores across all datasets as a function of $bfgs_factor$ values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores.
- (l) Increase of the tree search time across all datasets as a function of $bfgs_factor$ values. The increase is measured relative to the minimum runtime per dataset.

A. Appendix

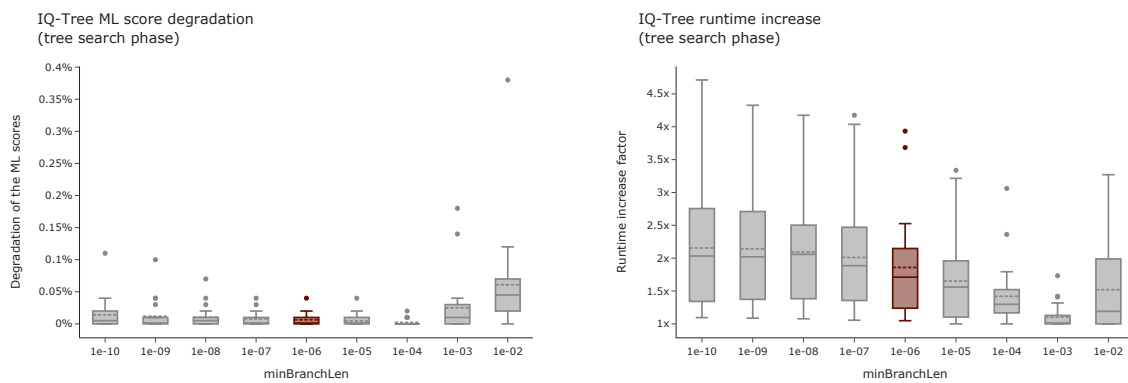


- (m) Degradation of ML scores across all datasets as a function of *num_iters* values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores.
- (n) Increase of the tree search time across all datasets as a function of *num_iters* values. The increase is measured relative to the minimum runtime per dataset.

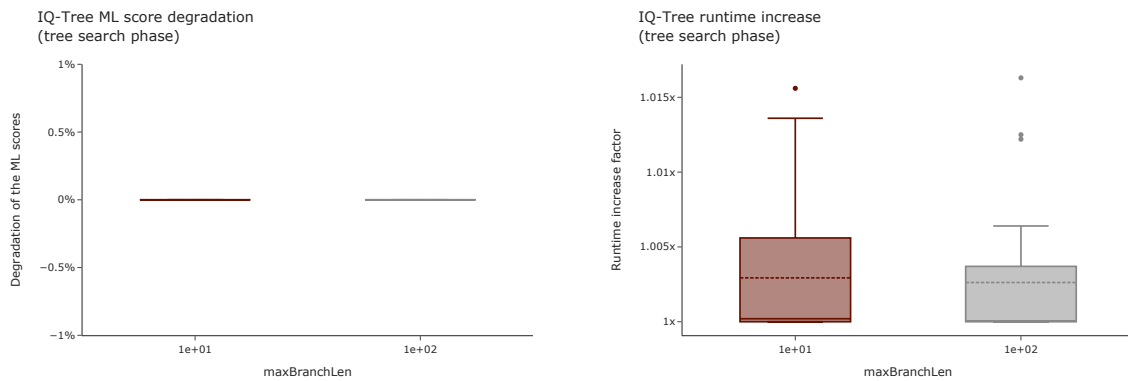
Figure A.3.: Influence of the numerical thresholds on the ML scores and runtimes of the RAxML-NG tree search phase. Each plots summarizes the data over all datasets. The dashed vertical line indicates the mean, and the solid vertical line the median value. The highlighted box indicates the default value for the respective numerical threshold in RAxML-NG.

A.1.2.4. IQ-Tree Tree Search Phase

The following figures show the influence of all numerical thresholds on the ML scores and runtimes of IQ-Tree if varied during the tree search phase. The ML scores refer to the ML scores after the evaluation phase, and the runtimes refer to the runtimes of the tree search phase. Due to the broad range of absolute likelihood values, we plot the degradation of ML scores in percent relative to the best ML score per dataset. The runtimes state how much longer the tree inference runs compared to the fastest tree inference per dataset.

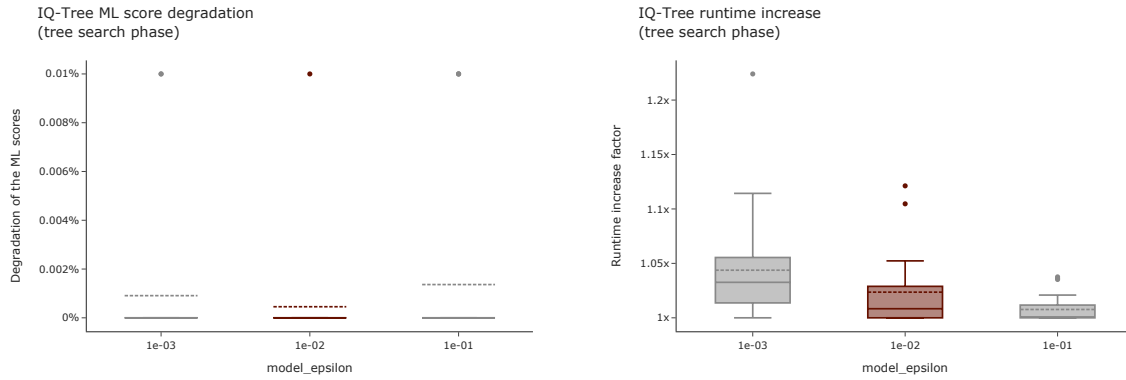


(a) Degradation of ML scores across all datasets as a function of *minBranchLen* values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores. (b) Increase of the tree search time across all datasets as a function of *minBranchLen* values. The increase is measured relative to the minimum runtime per dataset.

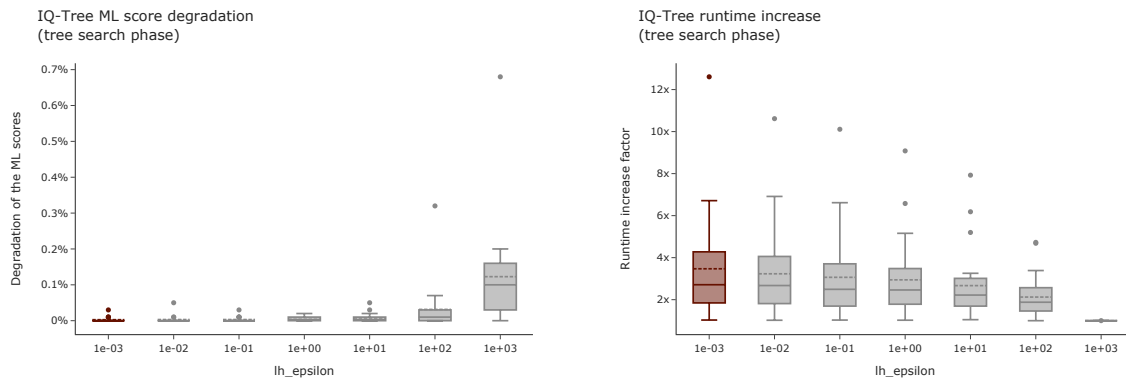


(c) Degradation of ML scores across all datasets as a function of *maxBranchLen* values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores. (d) Increase of the tree search time across all datasets as a function of *maxBranchLen* values. The increase is measured relative to the minimum runtime per dataset.

A. Appendix



- (e) Degradation of ML scores across all datasets as a function of $model_epsilon$ values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores.
- (f) Increase of the tree search time across all datasets as a function of $model_epsilon$ values. The increase is measured relative to the minimum runtime per dataset.

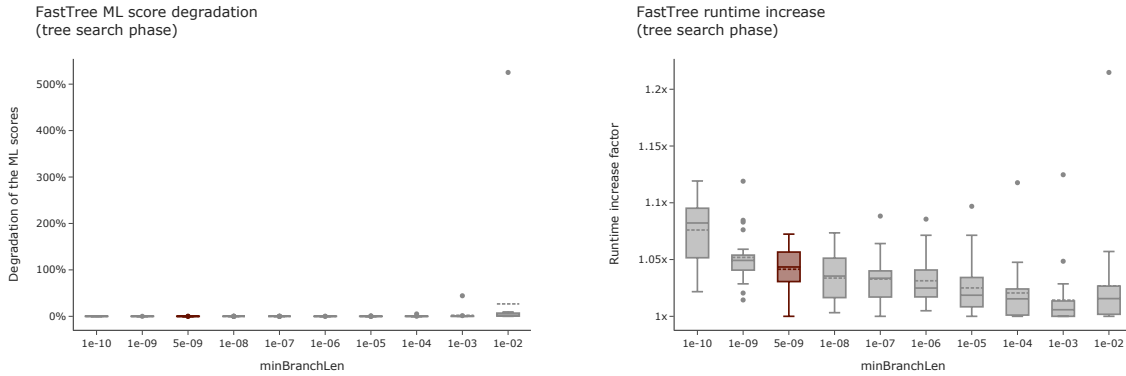


- (g) Degradation of ML scores across all datasets as a function of $lh_epsilon$ values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores.
- (h) Increase of the tree search time across all datasets as a function of $lh_epsilon$ values. The increase is measured relative to the minimum runtime per dataset.

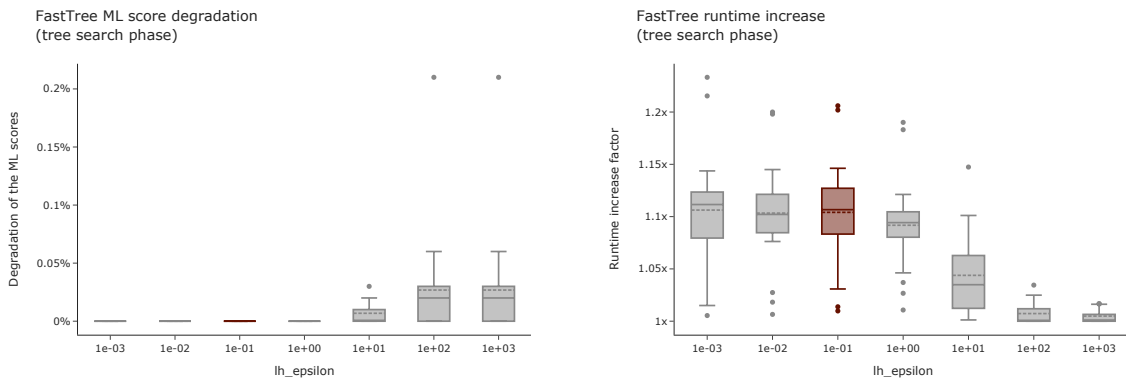
Figure A.4.: Influence of the numerical thresholds on the ML scores and runtimes of the IQ-Tree tree search phase. Each plot summarizes the data over all datasets. The dashed vertical line indicates the mean, and the solid vertical line the median value. The highlighted box indicates the default value for the respective numerical threshold in IQ-Tree.

A.1.2.5. FastTree Tree Search Phase

The following figures show the influence of all numerical thresholds on the ML scores and runtimes of FastTree if varied during the tree search phase. Due to the broad range of absolute likelihood values, we plot the degradation of ML scores in percent relative to the best ML score per dataset. The runtimes state how much longer the tree inference runs compared to the fastest tree inference per dataset.



(a) Degradation of ML scores across all datasets as a function of \minBranchLen values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores. (b) Increase of the tree search time across all datasets as a function of \minBranchLen values. The increase is measured relative to the minimum runtime per dataset.



(c) Degradation of ML scores across all datasets as a function of $lh_epsilon$ values. The degradation is measured relative to the highest ML score per dataset. Higher percentages indicate worse ML scores. (d) Increase of the tree search time across all datasets as a function of $lh_epsilon$ values. The increase is measured relative to the minimum runtime per dataset.

Figure A.5.: Influence of the numerical thresholds on the ML scores and runtimes of FastTree. Each plot summarizes the data over all datasets. The dashed vertical line indicates the mean, and the solid vertical line the median value. The highlighted box indicates the default value for the respective numerical threshold in FastTree.