Karlsruhe Institute of Technology

Advanced Substitution Model Selection Methods in RAxML-NG

Master's Thesis of

Christoph Stelz

At the KIT Department of Informatics

Institute of Theoretical Informatics

First examiner:	Prof. Dr. Alexandros Stamatakis
Second examiner:	Prof. Dr. Michael Beigl
First advisor:	M.Sc. Anastasis Togkousidis
Second advisor:	Dr. Oleksiy Kozlov

09. May 2025 – 10. November 2025

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

Advanced Substitution Model Selection Methods in RAxML-NG (Master's Thesis)

I declare that I have developed and written the enclosed thesis completely by myself. I have not used any other than the aids that I have mentioned. I have marked all parts of the thesis that I have included from referenced literature, either in their original wording or paraphrasing their contents. I have followed the by-laws to implement scientific integrity at KIT.

Karlsruhe, 10. November 2025

.....

(Christoph Stelz)

Abstract

With next generation sequencing, the cost to sequence a genome has decreased exponentially, even outpacing Moore's law. We therefore expect the use of phylogenetic inference tools to rise even further, with the tools facing a perpetual scalability challenge. The largest extent of studies (as well as users) now commends for increasingly automated methods as we observe a shift from manual workflows to reusable, highly automated pipelines. An important step in such phylogenetic analysis pipelines is model selection, which ranks the plethora of available DNA and protein substitution models according to their relative fit to the data. In this thesis, we present a novel implementation of a model selection procedure, integrated into RAxML-NG, a widely used tool for phylogenetic inference via Maximum Likelihood. It implements a number of heuristics to speed up execution and employs a robust parallelization scheme that also supports partitioned datasets and distributed memory systems. For 86.7% of amino acid, and 72.9% of single-gene DNA datasets, our implementation selects models with a BIC score difference of less than 10 compared to IQTree's ModelFinder and ModelTest-NG. With 48 cores, we reach a mean speedup per dataset of 3.54 (AA) and 5.82 (DNA) over IQTree. Over all datasets, the accumulated speedup with 48 cores over IQTree is 1.72 (AA) and 4.05 (DNA). By providing a fast, easily-accessible implementation without external dependencies, we hope to lower the barrier to prepend a model selection to phylogenetic analyses. This ensures a good model-data fit and potentially saves runtime during the tree search by limiting the use of computationally expensive mixture models. Furthermore, its integration into the inference tool allows future work to incorporate model selection results during the tree search, paving the way for more adaptive approaches and runtime optimizations.

Zusammenfassung

Durch neuartige Sequenzierungsmethoden sind die Kosten zur Sequenzierung eines Genoms exponentiell gesunken. Deshalb ist zu erwarten, dass die Verwendung von phylogenetischer Inferenzsoftware weiter ansteigen wird und sich die Software an stetig vergrößerende Datenmengen anpassen muss. Ein Großteil der Studien und Nutzer macht zunehmend automatisierte Methoden erforderlich. Es erfolgt ein Wechsel von manuellem Aufruf der Tools hin zu automatisierten und wiederverwendbaren Daten-Pipelines. Ein wichtiger Schritt in einer solchen phylogenetischen Pipeline ist die Modellauswahl, bei der die zahlreichen vorhandenen molekularen DNA- und Protein-Modelle anhand ihrer Fähigkeit, die vorliegenden Daten zu beschreiben, bewertet werden. In dieser Masterarbeit stellen wir eine neuartige Implementierung einer solchen Modellauswahl vor. Die Implementierung ist in das weit-verbreitete Inferenz-Tool RAxML-NG integriert. Sie verwendet mehrere Heuristiken, um die Ausführung zu beschleunigen und verwendet ein robustes Parallelisierungsschema, das auch partitionierte Datensätze und Cluster-Rechnersysteme gut unterstützt. Für 86,7% der Protein- und 72,9% der DNS-Datensätze beträgt die BIC-Differenz der ausgewählten Modelle weniger als 10 im Vergleich zu den Modellen von IQTree's ModelFinder und ModelTest-NG, Mit 48 CPU-Kernen erreicht unser Tool pro Datensatz einen durchschnittlichen Speedup von 3,54 (Protein) und 5,82 (DNS) im Vergleich zu IQTree. Über alle Datensätze gemessen beträgt der akkumulierte Speedup mit 48 CPU-Kernen 1,72 (Protein) und 4,05 (DNS). Mit diesem schnellen, leicht zugänglichen - da integriertem - Tool hoffen wir die Barriere für Modellauswahl zu senken und die Anwendung zu verbreiten. Dadurch ist gewährleistet, dass das ausgewählte Modell auch gut zum Datensatz passt. Ferner kann Laufzeit eingespart werden, da nicht komplexere Modelle als nötig verwendet werden. Außerdem erlaubt die Integration in das Inferenztool, in der Zukunft die Modellsuche in die Baumsuche miteinzubeziehen, um diese adaptiver und schneller zu gestalten.

Contents

1. Introduction

Phylogenetics studies the relatedness of species in the light of evolution. Before the discovery of DNA and the availability of sequence data, phylogenetic reconstruction relied on close observation of specimen and the detection of so-called homologies: similarities between different species that have developed through a shared evolutionary history. With the advent of DNA sequencing, phylogenetic methods are no longer constrained to the phenotype (observable traits), but primarily evaluate information from the genotype (the DNA sequence). This sudden breadth of information poses challenges best conquered with statistics. Computational phylogenetics therefore uses statistical models of DNA substitutions to explore evolutionary processes.

In certain cases, researchers have a good intuition on the type of model required for the analysis at hand and do not need to investigate further. Nevertheless, in the majority of cases, the plethora of available models and lack of a priori knowledge can render model selection a challenging and time-consuming task. Different methods exist to automatically select from a set of candidate models. Two popular tools with this functionality are ModelTest-NG [1] and IQTree's ModelFinder [2], [3], which use information criteria (such as the Bayesian Information Criterion, abbreviated BIC) to determine the "goodness of fit" of models.

In this thesis, we present a novel implementation of a model selection procedure, integrated into RAxML-NG, a widely used phylogenetic inference tool. Our implementation outperforms both IQTree and ModelTest-NG in multi-threaded shared-memory benchmarks on single-gene datasets. Its availability inside RAxML-NG lowers the barrier to perform a model selection, and may even be transparent to the user. Through its adaptive parallelization approach, it is well-suited to utilize all available computing resources, in cases where the user allocates a large amount of processing power in anticipation of the tree search following the model selection.

With the integration of model selection into RAxML-NG we were able to achieve:

1. a competitor for state of the art model selection tools that combines a large model candidate set (including freerate models) with runtime-saving heuristics and a versatile parallelization scheme
2. a simplification of the model selection process for users and in the context of pipelines
3. increased code maintainability, as there exists a large overlap of functionality between the tree inference tool RAxML-NG and the model selection procedure
4. a foundation for future work that can employ model testing during the tree search phase (i.e., by switching between hard-to-compute but accurate, and cheap but approximative models)

5. the option for future research to explore the link between phylogenetic models and their influence and bias on tree search results.

In the following sections, we will introduce phylogenetic inference, model selection, and available tools in detail (Section 2). In Section 3 we will describe our implementation, the optimizations and heuristics we employed, as well as the methods used to evaluate and benchmark our tool. In Section 4 and Section 5, we will present and discuss our findings, before concluding this thesis in Section 6.

2. Background

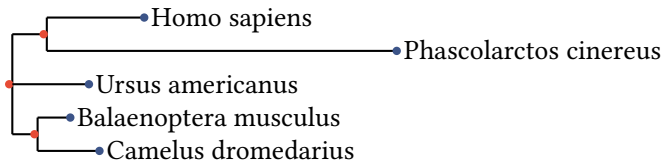
2.1. Phylogenetic Inference

2.1.1. Multiple Sequence Alignment and Phylogenetic Trees

The prerequirement for many phylogenetic methods is a Multiple Sequence Alignment (MSA): A matrix $S^{t \times n}$ represents the character sequence drawn from the state space S with length n of t taxa, with gap characters inserted such that "related" characters end up in the same column.

```
🐨 Phascolarctos cinereus CGCACAGATGCCTTGGAGCAAGGAGGATT
🐻 Ursus americanus       AGAACACAGGCCTTGGAACAAGGT-----
🐪 Camelus dromedarius    AGAACAGAGGCTTTGGAACAAGGAGGATT
🐋 Balaenoptera musculus  AGAACAGAGGCCTTGGAACAAGGAGGATT
👤 Homo sapiens           AGAACAGAGGCCTTAGAACAAGGAGGATT
```

(a) MSA excerpt



(b) A possible maximum-likelihood tree

Figure 1: MSA and a phylogenetic tree of five mammals based on the protein *Cyclin-dependent kinase 7* (CDK7), extracted from OrthoMAM [4].

For example, the MSA in Figure 1a is a matrix $S^{5 \times 30}$, and the state space comprises the four DNA nucleotides Adenine, Guanine, Cytosine, and Thymine alongside the gap character: $S = \{A, G, C, T, -\}$. The gap character represents deletions or insertions. For instance, the last nucleotides in the sequence of the American black bear (*Ursus americanus*) seem to be deleted. Another form of genetic mutation is the substitution: the Koala (*Phascolarctos cinereus*) has a Cytosine (C) in the first column, whereas all other taxa under consideration have an Adenine (A).

Given an MSA, the declared goal is to construct a phylogenetic tree, such as the one in Figure 1b . The leaf nodes (●) represent our taxa, whereas the inner nodes (●) can be thought of as extinct common ancestors. In an unrooted bifurcating tree as we have here, each inner node has exactly three neighbors. The number of trees grows super-exponentially with respect to the number of taxa: For t taxa, there are $\prod_{i=4}^{t}(2i-5)$ possible trees [5, p. 76]. This means for our small example from Figure 1 with 5 taxa there are 15 possible trees, for 10 taxa there are already 2 million possibilities, and for 50 taxa the unfathomable amount of $\sim 2.84 \cdot 10^{74}$ possible trees. From these considerations it is clear that evaluating all possible trees is a hopeless endeavour and that we must consider heuristics, statistical criteria, and approximations.

2.1.2. Tree Evaluation Criteria

An early and still relevant attempt at phylogenetic tree reconstruction is **Maximum Parsimony**. It embodies Occam's Razor: The explanation with the least amount of assumptions is to be preferred [6]. Typically, we do not know the states of the inner nodes, and we are therefore free to speculate. Maximum Parsimony minimizes the number of substitutions across the edges of a given tree and prefers trees that require less mutation to explain the observed sequence state at the tip nodes [5, p. 94]. Compared to other methods, it is cheap to evaluate and yields reasonable tree topologies.

A particular shortcoming of the Maximum Parsimony method is the fact that its criterion does not incorporate branch lengths. If we expect branches to be long, it seems likely that multiple mutations might occur along the way, possibly even ending up in the original state.

To capture such hidden mutations (hidden in the sense that they are transparent to us from the observed character states), we employ a **Markov model**. In particular, we model the substitution process of nucleotides or AAs using a Markov chain. The state space comprises the different bases (for DNA data) or amino acids (for AA data) and the transition between states represents a point mutation. The defining property of a Markov chain is the absence of memory: Let X_n be a random variable that denotes the state of the Markov chain at time point n, and let s_n be the observed state at time point n, then the value of X_{n+1} depends only on X_n:

$$P(X_{n+1} = s_{n+1} \mid X_n = s_n, X_{n-1} = s_{n-1}, ...) = P(X_{n+1} = s_{n+1} \mid X_n = s_n) \quad (1)$$

Equation 1 is called the Markov property [7, p. 1].

When we use Markov chains to model the substitution process, we assume that it has reached an equilibrium distribution [5, p. 8]. This means that as time progresses, the distribution $\pi = \left\{\pi_1, ..., \pi_{|S|}\right\}$, with π_i denoting the proportion of time spent in state i, is constant. The vector π is also called the base frequency vector.

A substitution rate matrix ("Q matrix") characterizes a continuous time Markov chain and denotes the transition probability between states during an infinitesimal time duration. Choosing "good" parameters for the Q matrix forms an optimization problem. We discuss the structure and parameter optimization of the Q matrix in Section 2.1.3. Since the Q matrix gives transition probabilities for infinitesimal durations, we can not use it to determine the transition probabilities for longer time periods, since multiple transitions with intermediate

states are possible. For this, we have to use the transition probability matrix ("P matrix"), which gives the transition probabilities for arbitrary non-infinitesimal continuous time t:

$$P(t) = e^{Qt} \tag{2}$$

The Chapman-Kolmogorov theorem [5, p. 8] guarantees that the transition probabilities in the P matrix account for all hidden intermediate mutations:

$$p_{s_1 \to s_2}(t_1 + t_2) = \sum_{k \in S} p_{s_1 k}(t_1) \cdot p_{k s_2}(t_2) \tag{3}$$

That is, the probability of a transition from state s_1 to state s_2 in time $t_1 + t_2$ is the same as the sum of all transition probabilities via an arbitrary intermediate state k. With the formulation of transition probabilities, we can now apply the principle of maximum likelihood estimation to our problem [8].

Likelihood is a function of the observation (our MSA $A \in S^{t \times n}$), all model parameters θ and a candidate tree T. Its value is the probability to observe the given data under those parameters: $L(A|\theta, T)$. Under the assumption that all sites of a sequence develop independently, the likelihood of a tree given an MSA is the product of all site-likelihoods, which in turn is the product of all transition probabilities across all tree branches, summed over all possible intermediate states [8].

Because we consider unrooted trees, we place a virtual root on an arbitrary branch in our tree to simplify the computation process. The likelihood evaluation algorithm performs a post-order traversal starting from the virtual root, and computes partial likelihoods. Given a parent node n_p with two child nodes n_i and n_j, it computes the partial per-site likelihood (stored in a so called conditional likelihood vector, CLV) of the parent node being in state s as follows:

$$L_p^{(s)} = \left(\sum_{s_2 \in S} p_{s \to s_2}(b_i) \cdot L_i^{(s_2)} \right) \cdot \left(\sum_{s_2 \in S} p_{s \to s_2}(b_j) \cdot L_j^{(s_2)} \right) \tag{4}$$

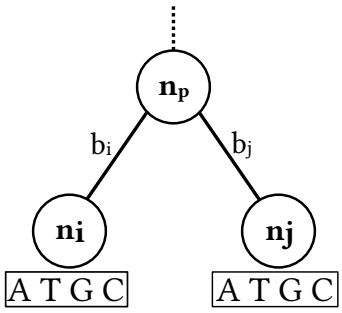


Figure 2: Triplet

At the tip nodes, the MSA defines the tip states:

$$L_i^{(s)} = \begin{cases} 1 \text{ if MSA has character } s \text{ at site } i \\ 0 \text{ otherwise} \end{cases} \tag{5}$$

Once the evaluation algorithm has computed the CLVs of all inner nodes, it obtains the final likelihood by considering the two nodes adjacent to the virtual root, which we will call n_l and

n_r. Let b denote the length of the branch that separates them, and π_s the base frequency of state s. Then the final likelihood value for the given site is

$$\mathrm{L} = \sum_{s_1 \in S} \pi_{s_1} L_l^{(s_1)} \left(\sum_{s_2 \in S} p_{s_1 \to s_2}(b) L_r^{(s_2)} \right) \tag{6}$$

To obtain the likelihood across all sites, we need to multiply the per-site likelihoods. Since these are numerically small, we instead take the sum of the logarithm of the site likelihoods, which is equivalent. This log-likelihood (lnL) is *the* guiding criterion in ML methods. In the following sections, we will discuss the details of the model.

2.1.3. DNA Substitution Models

Substitution models for DNA sequences differ in their substitution rate symmetries and the equilibrium distribution. A decisive assumption for efficient likelihood computation is time reversibility, as this allows arbitrary virtual root placement in order to traverse the unrooted tree. Time reversibility requires that for all $i, j \in \{1, ..., |S|\}, i \neq j$ the equality $\pi_i q_{ij} = \pi_j q_{ji}$ must hold.

Jukes and Cantor [9] proposed a simple model (commonly abbreviated **JC69**), which assumes an equal substitution rate λ between any two distinct characters. Because the rows of a substitution rate matrix must sum to zero, this leaves us with the following definition:

$$Q_{\mathrm{JC69}} = \begin{pmatrix} -3\lambda & \lambda & \lambda & \lambda \\ \lambda & -3\lambda & \lambda & \lambda \\ \lambda & \lambda & -3\lambda & \lambda \\ \lambda & \lambda & \lambda & -3\lambda \end{pmatrix}$$

Increasingly complex substitution models relax these constraints by introducing additional free parameters. Under this assumption, the most general substitution matrix named the General Time Reversible (**GTR**) model [10] is represented as follows:

$$Q_{\mathrm{GTR}} = \begin{pmatrix} \cdot & \alpha & \beta & \gamma \\ \alpha & \cdot & \delta & \varepsilon \\ \beta & \delta & \cdot & \zeta \\ \gamma & \varepsilon & \zeta & \cdot \end{pmatrix} \begin{pmatrix} \pi_1 & & & \\ & \pi_2 & & \\ & & \pi_3 & \\ & & & \pi_4 \end{pmatrix}$$

The requirement that rows must sum to zero allows the determination of the omitted values ("$\cdot$") on the main diagonal.

A concise way of expressing model matrices is to specify them in terms of symmetries between GTR parameters $\alpha, \beta, \gamma, \delta, \varepsilon,$ and ζ. Equal numbers mean that the parameter is repeated. For example, "000 00 0" designates JC69, as there is just a single parameter, and "012 34 5" denotes the GTR model. In total, there are 203 possible combinations [11]. However, there is a subset of 11 named and commonly used matrices, which we list in the appendix (Table A.4).

2.1.4. Amino Acid Substitution Models

For amino acid (AA) substitution models, the substitution rate matrices have dimension 20×20, a substantial increase compared to DNA data. After applying the above restrictions that

each row must sum to 0, symmetry must hold, and fixing one arbitrary scaling parameter, there are still 189 free parameters for the substitution matrix alone. This so called protein GTR model poses a challenge for regular optimization algorithms, as it is particularly slow to compute. To circumvent this problem, most analyses of AA datasets use precomputed rate matrices. Empirical studies have optimized these rate matrices on large datasets, sometimes with a focus on certain fields or applications such as mammalian or plant genomes. Table A.5 in the appendix lists the AA matrices supported by RAxML-NG and links them to their original publication.

2.1.5. Base Frequencies

There are multiple ways to estimate the value of the stationary distribution π:

1. In the simplest case, we assume *equal* frequencies for all states $i \in S$: $\pi_i = \frac{1}{|S|}$.
2. We can also count the number of times each state occurs in our observed data, that is the MSA, and divide by the total number of characters in the MSA. These are called *empirical* frequencies.
3. In the case of AA substitution models, the precomputed rate matrices are accompanied by a frequency vector estimated from the same dataset. We term these *model* frequencies.
4. It is also possible to *optimize* frequencies numerically with respect to the likelihood, using e.g. L-BFGS-B (Section 2.1.7).

2.1.6. Rate Heterogeneity Models

When we apply a given substitution model to our dataset, we assume that the substitution rate is uniform across all sites in the MSA. Since each site of a given sequence might face different evolutionary pressures, this assumption does not hold on empirical datasets. For instance, in protein-coding sequences the last nucleotide of a codon is often redundant, and therefore may undergo more frequent mutations, since this mutation does not change the encoded protein. The phenomenon that the substitution rate differs among sites is called rate heterogeneity among sites (RHAS).

A common approach is to employ a finite mixture model: we consider c substitution rate categories $r_1, ..., r_c$ with weights $w_1, ..., w_c$ under the constraints that the weights sum to 1, and that the average substitution rate is equal to 1:

$$\begin{aligned} w_1 + ... + w_c &= 1 \\ \sum_{i=1}^{c} w_i r_i &= 1 \end{aligned} \tag{7}$$

Since we do not know a priori which site fits best into which category, we compute the likelihood over the whole sequence for all categories separately, and later apply the weighted sum to obtain the final likelihood:

$$L = \sum_k w_k \cdot \left(\sum_i L_{k,i} \right) \tag{8}$$

In the following, we use the notation $L_{k,i}$ to refer to the likelihood value of the site with index i under rate category k. Multiple strategies to configure the substitution rate and weights exist and thus model the RHAS. We will briefly describe them in the following sections.

Invariant (+I)

One observation from real-world data is that some sites are highly conserved and virtually never mutate. It is possible to model this effect using a two-category mixture model, where one of the substitution rates is equal to zero: $r_1 = 0$. Let p denote the proportion of invariant sites, then the weights of the mixture model are $w_1 = p, w_2 = 1 - p$. Because of the restriction that the average substitution rate must equal one, we can deduce the value of the remaining rate [5, p. 111]:

$$\begin{aligned} 1 &= \sum_i w_i \cdot r_i \\ &= w_1 r_1 + w_2 r_2 = 0 \cdot p + w_2 r_2 \\ &= (1-p) r_2 \Leftrightarrow r_2 = \frac{1}{1-p} \end{aligned} \tag{9}$$

An inference tool can then optimize this single free parameter p with respect to the likelihood. For details on the optimization method, see Section 2.1.7.

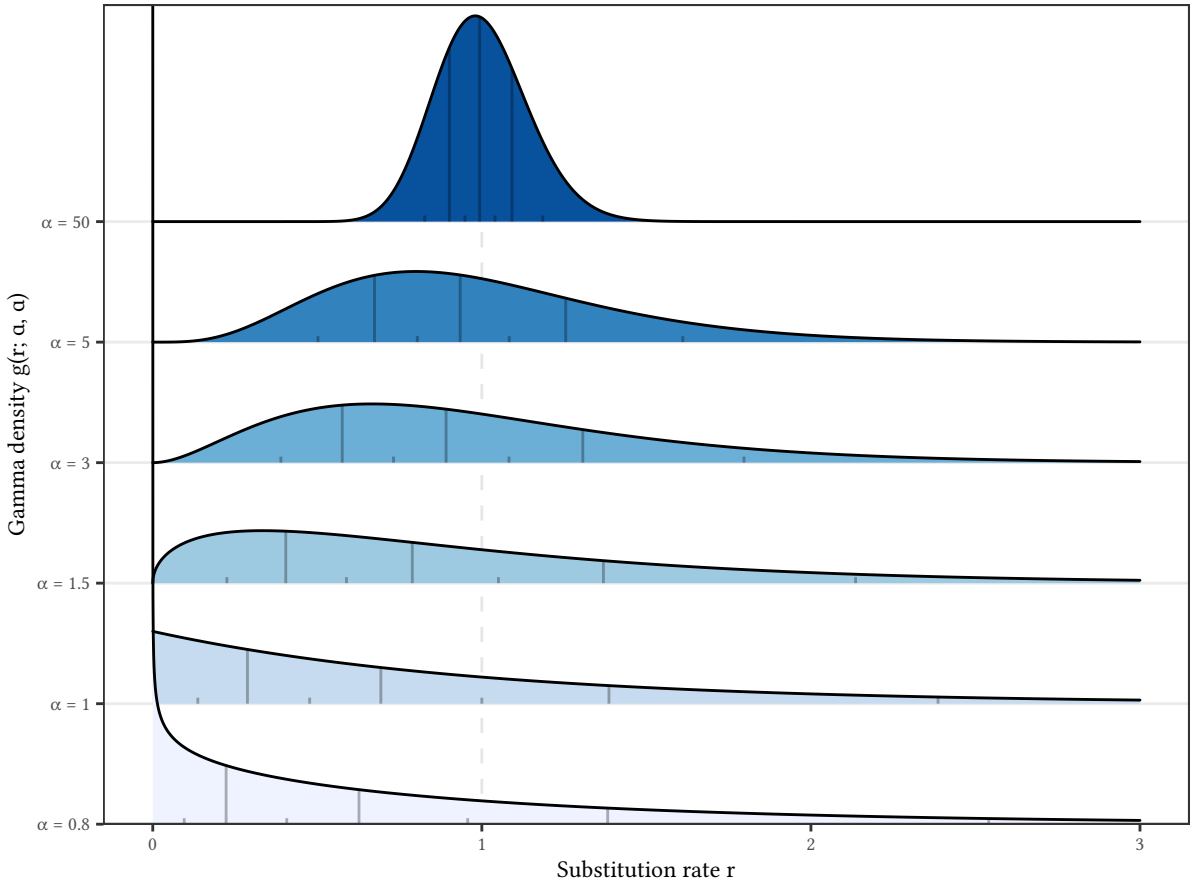


Figure 3: Probability density function of the gamma distribution for six distinct values of α. The gray vertical lines correspond to the quantiles 0.25, 0.5, and 0.75, respectively, and separate the distribution into four categories with equal probability mass. The small ticks at the bottom denote the mean substitution rate value for each category determined by Equation 11.

Gamma (+G)

A more flexible finite mixture model is the gamma model. Its underlying assumption is that the substitution rate of all sites follows a gamma distribution. The density function $g(r; \alpha, \beta)$ of the gamma distribution [5, 4.11], [12] is

$$g(r; \alpha, \beta) = \frac{\beta^\alpha r^{\alpha-1} e^{-\beta r}}{\Gamma(\alpha)}$$
$$\Gamma(\alpha) = \int_0^\infty t^{\alpha-1} e^{-t} dt \tag{10}$$

The mean value of the gamma distribution is $\frac{\alpha}{\beta}$ [13], therefore in order to satisfy the constraint that the mean substitution rate must be equal to 1, we set $\beta = \alpha$. Since the likelihood computation under the assumption of a continuous gamma distribution is prohibitively compute-intensive, Z. Yang [14] proposed a discretization into a fixed number of categories c. The discretization divides the gamma distribution into c intervals of equal probability, with the weights being $w_1, ..., w_c = \frac{1}{c}$. Let a and b be the quantiles as given by the cumulative gamma density function $F(x, \alpha)$ that restrict a particular substitution rate category i. Then, the mean substitution rate [14] of that category is

$$r_i = \frac{\int_a^b r \cdot g(r;\alpha,\alpha)dr}{\int_a^b g(r;\alpha,\alpha)dr} = c \cdot \int_a^b r \cdot g(r;\alpha,\alpha)dr \tag{11}$$

Figure 3 shows the gamma density function for six distinct values of α. For small values of α, the substitution rate values are spread out. For large values of α, the probability mass is much more centered around the mean value of 1. The same holds for the mean substitution rate of categories: When $\alpha = 50$, the means are approximately 0.83, 0.95, 1.04, and 1.19, thus covering only a small range of substitution rate values, whereas for $\alpha = 0.8$, the means range from 0.1 up to 2.54. We expect large values of α to fit datasets with a highly uniform substitution rate of sites well, whereas for highly rate-heterogenous datasets the optimal value of α w.r.t. likelihood may be small.

In some cases, the median substitution rate may provide the category rate instead. RAxML-NG denotes this by `+GA`, whereas a mixture model with `N` categories and mean category rates is denoted by `+GNm`

Freerate (+R)

If computational resources permit it, we can also forgo any further restrictions on the category weights and rates of the finite mixture model, and optimize the values according to likelihood with the data at hand. Given a fixed number of c categories, this approach introduces $2c - 2$ free parameters. For a discussion of optimization algorithms, refer to Section 2.1.7.

2.1.7. Parameter Optimization Methods

In order to optimize the continuous parameters during the ML tree search, RAxML-NG employs numerical optimization methods.

When optimizing the branch lengths, we can compute the likelihood's first and second derivatives, allowing us to apply the Newton-Raphson method. By solving the first derivative for zero, we can find a local minimum. For the optimization of scalar values where the derivative is unknown, e.g. the parameter α of the Γ-distribution (Section 2.1.6.2), or the proportion of invariant sites p in the invariant mixture model (Section 2.1.6.1), RAxML-NG uses Brent's method [15, Chapter 4] . Finally, to minimize multi-dimensional functions without explicit computation of the derivatives, e.g. the substitution rates of the Q matrix, the base frequencies of DNA substitution models, or the weights and category substitution rates of freerate mixture models (Section 2.1.6.3), it applies the quasi-Newton optimization algorithm L-BFGS-B [16], [17]. L stands for low-memory, B for bounded, and BFGS are the initials of the authors.

2.2. Information Criteria (IC)

In an effort to prevent over- and underfitting, information criteria seek to measure the "goodness of fit" by relating the optimized likelihood achieved under a given model to the number of free parameters.

2.2.1. AIC

H. Akaike [18] applied the information-theoretic Kullback-Leibler divergence to quantify the information loss of distinct candidate models when trying to approximate an unknown function. For ML estimation, Akaike provides the following asymptotic estimate:

$$\mathrm{AIC} = -2 \log L + 2k \tag{12}$$

L is the model's maximum likelihood and k is the number of free parameters. A lower AIC score indicates a better fitting model.

2.2.2. AICc

C. M. Hurvich and C.-L. Tsai [19] further provide a correction of the AIC for smaller sample sizes n:

$$\mathrm{AICc} = \mathrm{AIC} + \frac{2k(k+1)}{n-k-1} \tag{13}$$

2.2.3. BIC

G. Schwarz [20] proposed a different information criterion that uses a Bayesian argument. Based on the likelihood L, number of free parameter k and number of samples n, it is calculated as follows:

$$\mathrm{BIC} = -2 \log L + k \log n \tag{14}$$

Especially for large sample sizes, it penalizes higher parameter counts much more severely than the AIC.

2.3. Related Work

Note that in the remainder of this thesis, a candidate model refers to the choice of a substitution matrix, base frequency type, and RHAS model for a particular partition of the dataset.

2.3.1. ModelFinder

ModelFinder [3] is a model selection tool integrated into the phylogenetic toolbox IQTree [2]. It supports all RHAS models described in Section 2.1.6 (invariant, gamma, freerate, as well as the combination of invariant with either gamma or freerate). ModelFinder can parallelize using multiple threads by distributing the alignment sites across threads. To optimize freerate models, ModelFinder employs the Expectation-Maximization algorithm, which they claim is more accurate than BFGS [3, Online Methods]. To speed up the model testing, ModelFinder also employs heuristics, which we refer to as the *freerate heuristic* and the *RHAS heuristic*.

The freerate heuristic aims to limit the additional computational load that comes with considering freerate models. If, as is the default setting, we consider freerate category counts from the interval $c \in [2, 10]$, this choice implies nine evaluations per substitution matrix. Given the fact that the BIC score for a fixed dataset rises linearly with the number of free parameters (Equation 14), and that each additional category adds two new free parameters, it seems likely

that the BIC score ceases to improve beyond a certain point. ModelFinder exploits this fact and evaluates freerate models in ascending order of category count, stopping evaluations once the BIC score no longer improves.

The RHAS heuristic assumes that the well-suitedness of RHAS models for a dataset is independent of the substitution matrix. This implies that we do not need to test all combinations of RHAS models and substitution matrices. Instead, ModelFinder evaluates all RHAS models (under the application of the previous freerate heuristic) for a single "reference" substitution matrix, such as the `JC` substitution matrix. For the remaining substitution matrices, it only considers RHAS models whose BIC score difference to the best observed RHAS model for the reference matrix is less than 10. According to guidelines, a BIC score difference exceeding 10 means that it is very likely that the model with the lower score fits the dataset significantly better [21].

There are references in the source code to a command-line argument `--thread-model`, which activates a parallelization mode where each thread evaluates a single model. This flag does not show up in the help message or documentation. There seem to be some incongruency surrounding this parallelization mode and the heuristics that ModelFinder applies, leading us to believe that the developers abandoned this mode.

2.3.2. ModelTest-NG

D. Darriba *et al.* [1] introduced ModelTest-NG in 2020 as the successor to the previously established model testing tools jModelTest 2 [22] and ProtTest 3 [23]. It builds on the Phylogenetic Likelihood Library (PLL) [24], which contains highly-optimized, vectorized procedures for likelihood computations. ModelTest-NG represents a significant step up in performance over its two predecessors, reaching average speedups of 510 on empirical DNA data, and 36.9 on empirical protein data. The authors further report a speedup over ModelFinder of 1.24 on empirical DNA data, and 1.19 on empirical protein data.

On a high-level, its model selection algorithm consists of the following steps:

1. It builds a set of candidate models, partly based on user input: It computes the cartesian product between a) substitution matrices under consideration b) frequency type (equal or ML-optimized frequencies for DNA data, and equal or empirically counted for AA datasets), and c) RHAS models (uniform, invariant, gamma, invariant and gamma, freerate)
2. Iterates over all candidate models
3. Assigns one thread to the given combination of partition and candidate model
4. Optimizes the model parameters and branch lengths according to the likelihood
5. Optionally, it performs a limited number of topological moves to improve the tree under the given model.
6. After obtaining an ML estimate, it computes the score under the three information criteria AIC, AICc, and BIC.
7. After the evaluation of all candidate models for a given partition has ended, it selects the model with the minimum IC score.
8. If the MSA is multi-partitioned, it returns to step 2. and repeats the process for the next partition, until it has processed all partitions.

Importantly, ModelTest-NG does not apply heuristics to reduce the set of candidate models. Furthermore, the number of categories for freerate models must be explicitly specified when invoking ModelTest-NG.

2.3.3. Machine Learning

There also have been multiple attempts to apply machine learning methods to directly predict the best-fit model based on features of the input dataset, thereby circumventing the time-consuming step of optimizing the likelihood of candidate models [25], [26]. Due to time constraints, we did not evaluate these methods in the context of this thesis.

2.3.4. RAxML-NG

RAxML-NG is a tree inference tool using the maximum-likelihood method, developed by our lab. Older versions linked the `libpll` [24] to carry out the core likelihood computations, whereas newer (development) versions of RAxML-NG use `coraxlib` [27], a `libpll` successor. RAxML-NG supports arbitrary DNA substitution matrices, all base frequency types and RHAS models mentioned in Section 2.1.5 and Section 2.1.6, as well as a superset of the constant rate matrices supported by ModelTest-NG and IQTree. Previously, RAxML-NG did not support model selection and required the user to explicitly specify a model at the program invocation.

3. Methods

We integrate model selection capabilities into RAxML-NG. Previously, RAxML-NG operated under the assumption that the model is a fixed part of the input data and specified on a per-partition basis. The new implementation triggers a model selection process in two cases:

1. The user specifically requests model selection with a command-line flag
2. The user passed `--model auto` while primarily conducting a separate analysis, e.g. ML tree search

The model selection process runs after the starting tree generation and uses a fixed tree topology to evaluate the likelihood of the input data under different models. It scores the model candidates based on information criteria which favor models with a higher optimized log-likelihood score, yet also penalize the number of model parameters in an effort to prevent overfitting. In case of a partitioned dataset, RAxML-NG treats the model selection process for each partition independently.

3.1. Model Optimization

Our model selection implementation utilizes a preexisting RAxML-NG method to optimize a given candidate model. It comprises the following steps:

1. Computation of the log-likelihood score of the current model parameters
2. Iterative optimization of the branch lengths
3. In the case of a rate-heterogeneity model that accounts for invariant sites, optimization of the proportion p_{inv} using Brent's method
4. In the case of a model that allows base frequencies as free parameters, optimization of the base frequencies via the L-BFGS-B method
5. In the case of the gamma rate-heterogeneity model, optimization of the parameter α of the Γ-distribution using Brent's method
6. Optimization of substitution rates with L-BFGS-B
7. In the case of a freerate rate-heterogeneity model, optimization of the category weights and rates, either with L-BFGS-B or Expectation-Maximization, depending on the mode of operation
8. Recomputation of the log-likelihood score of the optimized model parameters
9. As long as likelihood improvement is greater than ε_{lh}, repetition of the process starting from step 1

3.2. Applied Optimizations

Model optimization is a time-intensive process, which is aggravated by the large number of candidate models under consideration during model selection. Therefore, we need to apply optimizations and shortcuts to keep the runtime tolerable.

3.2.1. Heuristics

The fact that each candidate model comprises three independent parts – substitution rate matrix, base frequencies, and rate heterogeneity model with a varying number of categories – yields a large number of possible combinations. For example, for DNA data there are 203 possible substitution matrices [11], two base frequency types (equal and ML-optimized), five types of rate heterogeneity models (uniform, invariant, gamma, invariant and gamma, freerate with two to eight categories), leading to a total of 4 466 candidate models, Even if we restrict ourselves to the eleven most common substitution matrices, we still need to examine 242 candidates. In order to conserve computational resources, we attempt to reduce the set of candidate models with heuristics.

Freerate Heuristic

Similarly to ModelFinder [3], we implement a procedure to automatically discover the number of freerate categories. First, the user specifies an interval of category counts that ModelFinder should examine (by default [2, 10]). The model selection procedure then evaluates the freerate models in order of ascending category counts, keeping track of the score given by the information criterion. As soon as the IC score no longer improves by additional rate categories, we abort the evaluation of all models that comprise more categories. The reasoning behind this is that once the score starts to drop, adding further categories is unlikely to yield any further improvement, since the information criterion penalizes the number of free parameters[1].

Unlike IQTree, we parallelize over candidate models instead of alignment sites by default. This complicates the application of our heuristic, since the evaluation of models depends on the results of previous evaluations. A sufficiently large parallel processor could run the evaluation of all freerate models on a small dataset concurrently. In such a case, the heuristic would be unable to reduce the number of model evaluations.

To avoid having to revert to sequential model evaluation, we adopt a greedy scheduling scheme: If the heuristic has no information on whether the given freerate candidate model can be skipped, we speculatively initiate its evaluation. This means that some evaluations may turn out to be unnecessary in retrospect. However, speculative scheduling increases CPU utilization and prevents idling. We further benefit from the observation that for most datasets, four categories provide a good fit. To this end, we evaluate freerate models with up to four categories first, followed by the other RHAS models which do not suffer from this dependency problem, and only then move on to freerate models with higher category counts. Intuitively,

[1] Each additional freerate category adds two parameters (weight and rate).

this increases the probability that once the program is ready to evaluate the freerate models with higher category counts, the data needed to assess whether it should indeed evaluate them, or whether it can skip them, is already available. Crucially, we do this across all partitions, meaning that on datasets with a large number of partitions it is less likely that unresolved dependencies cause superfluous candidate evaluations.

RHAS Heuristic

One reason for the high number of candidate models is that any substitution model (Q matrix) can be combined with any of the rate-heterogeneity models. Similar to ModelFinder [3], we implement a heuristic to detect which RHAS models work well for the dataset at hand, and restricts further testing to those models. RAxML-NG evaluates all RHAS models on a reference matrix – `GTR` for DNA data, `LG` for AA data. For all remaining substitution models, it only considers RHAS models where the IC score difference to the best observed model is higher than a fixed threshold. Equivalent to IQTree, we use the BIC score by default and set the threshold to $\Delta := 10$.

3.2.2. Expectation-Maximization

Profiling the model optimization procedure of RAxML-NG revealed that the optimization of freerate models takes a considerable amount of time (data not shown). Especially with a high number of categories, the L-BFGS-B optimization procedure takes a long time to converge. An alternative approach to the category weight optimization is Expectation-Maximization, as described by S. Kalyaanamoorthy *et al.* [3]. Let $L_{k,s}$ denote the likelihood of alignment site s under category k which has weight $w_k^{(i)}$ in iteration i. In total there are c rate categories. Then the posterior probability of site s to belong to category k is

$$p_{k,s} = \frac{w_k^{(i)} L_{k,s}}{\sum_{j=1}^{c} w_j^{(i)} L_{j,s}} \tag{15}$$

The updated weight of a category is the mean of the posterior probabilities across all sites:

$$w_k^{(i+1)} = \sum_{s=1}^{n} p_{k,s} \tag{16}$$

We extended coraxlib to support the EM algorithm when optimizing freerate weights. A particular implementation challenge arises under the use of threading or MPI. The parallelization of coraxlib builds on the use of a single reduction operator, which reduces n numbers across the p processing elements (PE) participating in the computation:

$$\text{reduce} : \{+, \max, \min\} \times \mathbb{F}^{n \times p} \to \mathbb{F}^n \tag{17}$$

However, RAxML-NG's load balancer [28], [29] can split up a partition and distribute its sites across many threads. Since the reduction operation poses our only synchronization primitive, the threads communicate only globally and not on a per-partition basis. We work around this problem by introducing a method that optimizes the freerate weights of all given partitions at once. It stores the required parameters in a tightly packed array (see Figure 4), which it can

pass to reduce without further processing. If one partition converges sooner than the others, its threads must still participate in the operation to ensure global convergence.

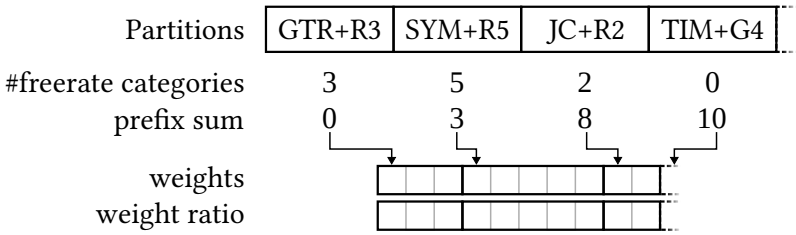


Figure 4: For multi-partitioned datasets, we store per-category data in a tightly-packed one-dimensional array to allow easy reduction of values via coraxlib primitives.

3.2.3. Parallelization Across Models and Opportunistic Multi-threading

The most fine-grained parallelization level available for ML computations is parallelization across sites: different CPU cores compute the per-site log-likelihood of a different subset of sites, and a sum over the intermediate results delivers the final log-likelihood. This sum-operation demands synchronization between threads. Since likelihood evaluation is *the* most frequent operation in RAxML-NG, with multiple thousands of calls per second, depending on the dataset, the overhead is substantial. For model selection in particular, we have an additional opportunity for parallelism: As long as the program has sufficient memory available, it can test multiple candidate models at the same time. This decreases the synchronization overhead, since fewer threads – and ideally only a single thread – participate in the likelihood computation.

The heuristics described above introduce soft data dependencies between the model evaluations, which complicate the parallelization across models. In general, there are two types of dependencies, corresponding to the two heuristics applied:

1. All candidate models depend on the evaluations of the reference matrix, since they decide which RHAS models are relevant to the given dataset (*RHAS heuristic*, Section 3.2.1.2).
2. Candidate models with a freerate or Invariant freerate RHAS model depend on the evaluation of all candidate models with the same substitution matrix and RHAS model, but lower category counts (*freerate heuristic*, Section 3.2.1.1).

For example, a freerate model with five or more categories does not need to be evaluated if the IC score increased after advancing from three to four categories. Similarly, if the Invariant RHAS model did not yield a good fit during evaluation on the reference matrix, we entirely omit it during the evaluation of other substitution matrices.

IQTree's ModelFinder solves the dependency problem by focusing on parallelization over alignment sites. The evaluation of models in sequence guarantees the availability of results from previous evaluations, ensuring optimal application of the heuristics. However, with the synchronization overhead discussed at the beginning of this section, the limited length of single-gene alignments, and the large number of cores available in modern machines, scaling across alignment sites has its limits. ModelTest-NG on the other hand utilizes exactly one thread per candidate model, but does not implement heuristics with dependent evaluations.

Our implementation provides fine-grained, dynamic parallelization: It evaluates multiple models concurrently, with multiple threads cooperating on a single model by distributing the alignment sites. By approaching the evaluation of candidate models in the order described below, we attempt to resolve the dependencies ahead of time. Should the results a heuristic depends on still be pending when an evaluation is about to start, we eagerly evaluate it regardless.

The model selection process of our tool starts with assembling a list of all possible candidate models of all partitions. If there are q partitions in the dataset and m different choices for a model, we expect to have $q \cdot m$ candidate models. In an effort to increase the distance between the evaluation dependency and dependent, we assign each candidate a priority based on the substitution matrix m and the category count c and sort the list of candidates with descending priority[2]:

$$\text{priority}(m, c) = \begin{cases} 0 \text{ if } m \text{ is reference and } c \leq 4 \\ 1 \text{ if } m \text{ is reference and } c > 4 \\ 2 \text{ if } c \leq 4 \\ 3 \text{ if } c > 4 \end{cases} \tag{18}$$

The initialization routine then estimates the thread count p based on the recommendations RAxML-NG uses for tree search, which takes the number of alignment patterns n, the number of states, and the number of RHAS categories c into account [30]:

$$\begin{aligned} |\text{CLV}| &= c \cdot n \cdot \begin{cases} 4 & \text{for DNA} \\ 20 & \text{for AA} \end{cases} \\ x &:= \frac{|\text{CLV}|}{\begin{cases} 4\,000 & \text{if priority} < 2 \\ 80\,000 & \text{otherwise} \end{cases}} \\ p &:= \frac{|\text{CLV}|}{\begin{cases} 4 - \log_2(x) & \text{if } x < 8 \\ \log_2(x) - 2 & \text{otherwise} \end{cases}} \end{aligned} \tag{19}$$

RAxML-NG spawns a fixed number of threads for the model selection phase, which typically coincides with the number of CPU cores. If possible, it pins threads to specific CPU cores to avoid issues with non-uniform memory layouts. Our dynamic parallelization scheme relies on the worker threads to spontaneously assemble a "team" of threads to work on a given model evaluation. Figure 5 visualizes this process: A worker thread with no assigned candidate models tries to enter a critical section to get a new work assignment. The critical section assigns candidate models with status WAITING in the order described above, i.e. under consideration of the priority. Once enough threads have joined the team of a candidate model, the evaluation can begin (RUNNING) and the critical section will assign the next model in the list. After the evaluation has concluded, the first thread to join the team writes back the result and the evaluation is now FINISHED. The written results can influence future evaluations: Should they allow the application of a shortcut, the respective candidate model will be SKIPPED and any assigned threads will acquire a new work item.

[2] That is, increasing priority number.

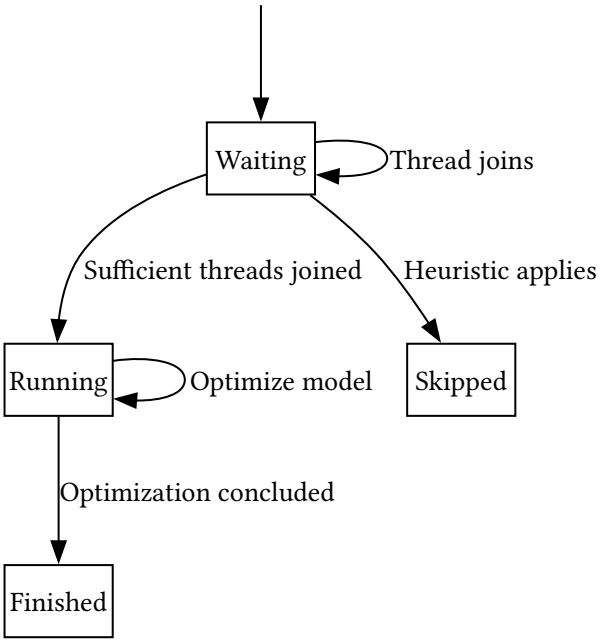


Figure 5: State diagram of a candidate model's evaluation status.

Two crucial methods underpin this implementation: a reduction callback for the max and $\sum$ operations, and the `barrier` function. While RAxML-NG has preexisting implementations of these procedures, they are not suitable for the dynamic parallelism of our model selection process, since in our case we require synchronization between a constantly changing set of threads.

The barrier procedure serves as a synchronization primitive of threads. A barrier call returns only once all other threads have also made the same call. Because threads share the same memory space, we use an atomic counter. Each thread increments the counter exactly once upon entering the barrier. A coordinating thread (by default the first thread to have joined the given evaluation) waits until the counter is equal to the expected number of threads, resets it to zero and releases all threads from the barrier by setting a `proceed` flag. While RAxML-NG already implements a thread barrier, both as a global operation and in the context of a thread group, the pre-existing implementations were not fit for the dynamic parallelism of our model selection routine. We therefore reimplemented the barrier method in a class called `ModelEvaluator`. This class has a one-to-one mapping to a given candidate model, and holds a private counter and flag value.

Building on the barrier method, the likelihood library coraxlib used by RAxML-NG requires only a single method to support arbitrary parallelization contexts: the reduction method as introduced in Section 3.2.2. This method takes n double-precision floating-point values as input, as well as a flag specifying the reduction operator ($\sum$, max, min). The reduction method as implemented in RAxML-NG works in two phases. In the first phase, the p calling threads all write their input values into a shared reduction buffer $R \in \mathbb{F}^{n \times p}$. After the call to the barrier method, the second phase begins. Each thread performs a column-wise reduction of the buffer into local memory. To adapt this reduction operator for our parallelization scheme, we allocate one reduction buffer per evaluation. Furthermore, we use the team-local barrier

method mentioned above, instead of the global barrier method. Because the coraxlib API simply expects a function pointer to the reduction method, we straightforwardly pass a pointer to the newly implemented reduction operation, without needing to modify any parts of the library itself. In this way we implement a highly tunable parallelization scheme in shared memory.

3.2.4. MPI Parallelization

For datasets beyond a certain size, a single compute node does not suffice. Moreover, the user may also perform a model selection as part of the tree search, with a large amount of compute resources already allocated to RAxML-NG. In these cases, it is important to support the computation on distributed memory machines via the Message Passing Interface (MPI). The ModelTest-NG MPI implementation expects a fine-grained allocation of MPI ranks: one MPI rank corresponds to one thread. Since RAxML-NG already supports a hybrid mode for tree search, we extend this framework to the model selection procedure. In a hybrid parallelization approach, each MPI rank has multiple running threads. A typical setup, which we optimize for in our implementation, would be a single MPI rank per node, with each running one thread per CPU core. We restrict the cooperation on a single candidate model to a single machine, meaning that each candidate model belongs to a single MPI rank at any given point, and alignment sites are not distributed across the process boundary. This restriction allows us to rely on shared memory for the barrier and reduction methods described in the previous section, such that the model optimization itself can happen without any reliance on the MPI interface.

The extension to support MPI parallelization focuses on two things: ensuring exclusivity of the candidate model scheduling, such that no two ranks evaluate the same candidate model twice, and communicating the evaluation results, such that the heuristics can still work even when they depend on an evaluation result which was not computed on the same rank.

For the former part, we adopt a strategy that is also implemented in ModelTest-NG. Typically, MPI requires sender and receiver to execute the same code. To avoid having a single rank preoccupied with the coordination of candidate model scheduling, and thus prevented from contributing to the model evaluation, we use MPI Remote Memory Access (RMA) [31, Chapter 12] . Thereby, MPI ranks can allocate a certain “window” into their memory, which is also available to other ranks without requiring explicit cooperation of the target rank. In our case, we allocate a global 64-bit index on rank 0 that points to the candidate model next in line. If an MPI rank then wants to evaluate and “claim” a certain candidate model in order to assign local threads to it, it calls the MPI RMA procedure `MPI_Fetch_and_op`, which atomically increments the model index on rank 0 and returns the value prior to the incrementation. The atomicity of that operation ensures that the scheduler assigns a candidate model only once. If the hardware supports it, an explicit participation of rank 0 in the operation is not required. Figure 6 illustrates this: all MPI ranks modify the index resident on rank 0 via RMA operations. The index points into the list of candidate models discussed in the previous section, which each MPI rank holds locally.

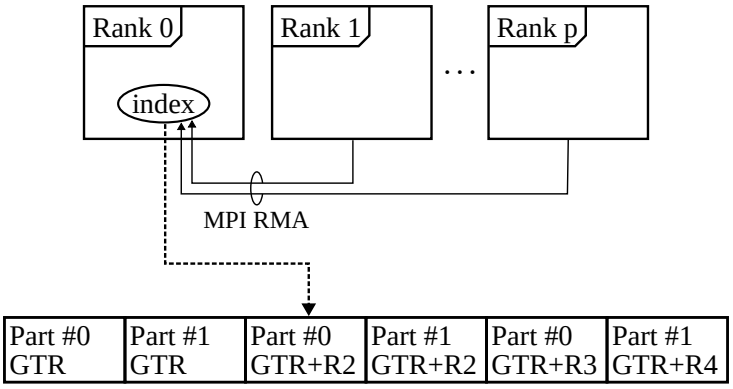


Figure 6: The scheduling of candidate models happens through one-sided MPI communication.

To share results, we allocate two MPI RMA windows on rank 0. The first is a byte array large enough to accommodate the results of all candidate models. Each evaluation result consists of the achieved log-likelihood (lnL), the optimized model parameters, the partition index and the index of the candidate models (to allow other ranks to correctly update the result in their memory). The second window is a displacement counter which indexes the results window. This solves the following two use cases: 1) An MPI rank seeks to announce the result of an evaluation, or 2) it wants to ingest the results of other MPI ranks in order to make informed decisions using heuristics. For the first use case (announce/write), we need a distinct memory range in the array to write to, otherwise the program could accidentally overwrite results.

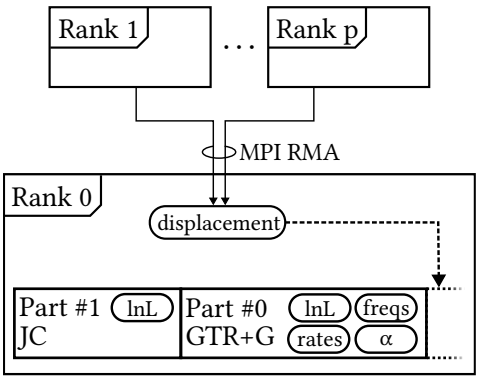


Figure 7: MPI ranks announce results by writing them consecutively into a window of the root rank.

For the second use case (ingest/read), an MPI rank ideally only reads a minimal subset of the array to incrementally update its results. Furthermore, the length of the serialized model parameters differs, depending on the model type: A candidate model with the Uniform RHAS model needs less space than a freerate model with eight categories. Especially if a dataset has numerous partitions, reading the entire array is costly, with the model parameters taking up approximately 256 bytes for DNA data. To solve this problem, we store the evaluation results consecutively into the results array, and keep track of the current writehead using the displacement counter. If an MPI rank announces a result, it enters a critical section which guarantees mutual exclusion. It reads the current writehead displacement and writes its result into the array starting at that address. Afterwards, it increments the displacement counter according to the number of bytes it has written and exits the critical section. When reading the

new results, an MPI rank can simply check whether the displacement counter has increased beyond the last seen value, and read that range specifically from the results array.

3.3. Checkpointing

With model selection taking up a considerable amount of runtime, it is important that RAxML-NG saves intermediate results. This prevents losing progress in case the program prematurely exits, for example if it encounters an error. Also, the user or a surrounding HPC scheduler (such as SLURM) could choose to terminate RAxML-NG with the model selection routine still running. Because of the large number of candidate models, the workload has high granularity by nature: we can save the model optimization results of each candidate model. For datasets with numerous small partitions, this could lead to an excessive amount of checkpoint writing, since model evaluations can finish in short succession. To avoid unnecessary pressure on the disk I/O, we have set a minimum time period of one second between checkpoint writes.

MPI parallelization poses a further hindrance to checkpointing, as only a single rank should write the checkpoint file. Since our implementation already communicates the evaluation results to keep the heuristics updated, we utilize this mechanism to also incorporate the results from foreign MPI ranks into the checkpoint.

3.4. Evaluation of Model Selection Accuracy

3.4.1. EvoNAPS

To evaluate our model selection algorithm, we draw a sample from EvoNAPS [32]. This is a well-structured relational database and contains MSAs from various sources, such as TreeBASE [33], OrthoMAM [4], and PANDIT [34], alongside the results of a model selection, model parameter optimization and maximum likelihood tree estimation. Since the pre-existing model selection is the result of an older IQTree version, we did not reuse it for our purposes. EvoNAPS stores partitions of datasets separately, though most entries are single-gene alignments.

In total, EvoNAPS contains 48 707 DNA alignments and 21 800 AA alignments. Because of insufficient time and resources for an exhaustive analysis of all alignments in the database, we took a random sample of 500 DNA and 500 AA alignments. Samples were not stratified, in an attempt to reflect the average use case.

We received EvoNAPS as a SQL dump from a MariaDB database. In order to reduce the dependency on a running database server, we converted it into a SQLite database [35] using an automated script [36]. In this way, the downstream analysis only requires access to a single database file. A small Python script exports the sequences contained in the database into a FASTA file.

3.4.2. Pipeline

We implemented the evaluation procedure as a reproducible and re-usable Snakemake pipeline [37]. Snakemake's file-based dependency graph is a good fit, since all tools under consideration (RAxML-NG, IQTree, ModelTest-NG) have a command-line interface and use compatible file-based input/output formats (FASTA for MSAs and Newick for trees).

To isolate the software environment, we use conda-forge [38] and Bioconda [39], with our implementation bundled in a custom package[3].

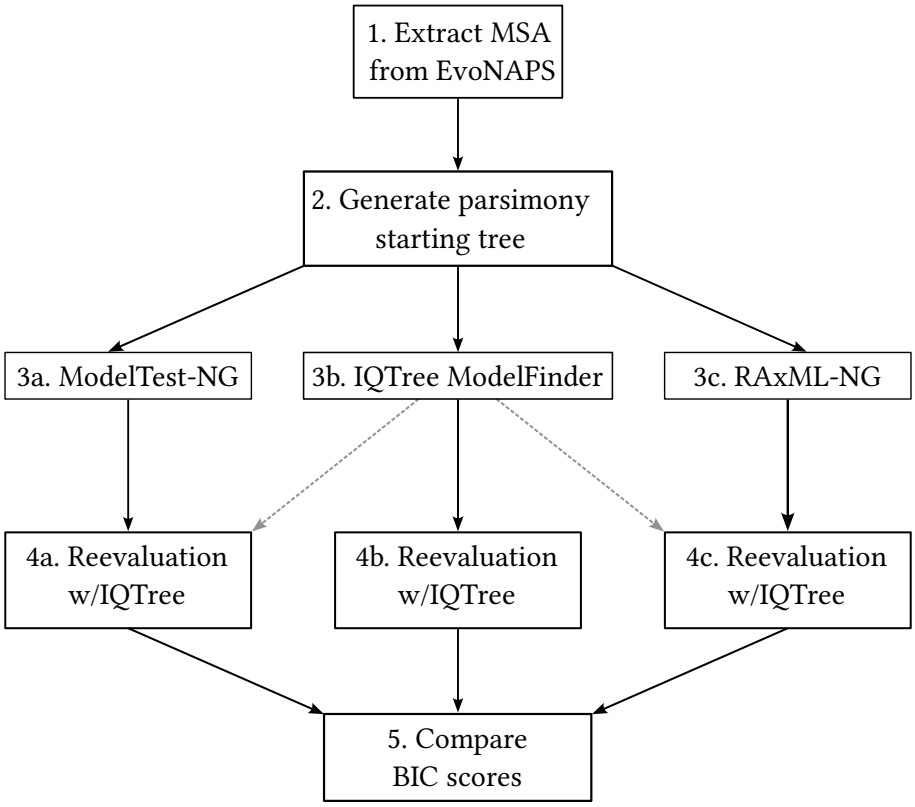


Figure 8: Overview of the evaluation pipeline steps

Figure 8 gives an overview over the pipeline steps. For any given EvoNAPS dataset, we extract the MSA and generate a parsimony tree with RAxML-NG. With this parsimony tree as starting tree, we infer the model with ModelTest-NG, IQTree's ModelFinder and our implementation in RAxML-NG. IQTree performs a small number of topological moves on the given starting tree to optimize its likelihood. In order to have comparable BIC scores, we reevaluate the inferred models of all tools under consideration with IQTree on its optimized topology. After extracting the BIC score from IQTree log files, we compute the BIC weights using R [40], [41].

Let $B \in \mathbb{R}^{n \times 1}$ be a vector of BIC scores with $m := \arg\min_i(B_i)$ defined as the index of the minimal entry. Then the BIC weight is defined as

$$w_i^{\mathrm{BIC}}(B) = \frac{e^{-\frac{1}{2}(B_i - B_m)}}{\sum_{k=1}^{n} e^{-\frac{1}{2}(B_k - B_m)}} \tag{20}$$

[3] https://anaconda.org/stelzch/raxml-ng-modeltest

One interpretation of the BIC weight is that it represents the probability that the given model is the best model according to the BIC [42].

For ModelTest-NG, we enable the evaluation of freerate models and use the default set of substitution models and base frequency modes (DNA: equal or ML-optimized, AA: model-defined or empirical counts). Because it does not support evaluation of multiple freerate category counts in a single run, we leave the category count at the default setting of 4.

For RAxML-NG, we enable the auto-discovery of freerate category counts in the range $c \in [2, 10]$. We manually enable the consideration of the following RHAS models: equal (`+E`), invariant (`+I`), gamma (`+G`), invariant and gamma (`+I+G`), freerate (`+R`), and invariant and freerate (`+I+R`). Our implementation considers candidate models with equal or ML-optimized base frequencies in the case of DNA data, and model-defined or counted (empirical) base frequencies in the case of AA data.

IQTree's ModelFinder considers the same set of RHAS models. We instruct it to keep identical sequences (`--keep-ident`) to avoid incompatibility with the starting tree. For DNA data, IQTree either uses equal or counted (empirical) base frequencies. While IQTree supports models with ML-optimized base frequencies, it does not consider them during model selection.

The interoperation of the three tools poses some challenges. Especially for DNA model matrices, the naming is not consistent between the tools. Table A.4 lists the different names and aliases known to IQTree and the library underlying both RAxML-NG and ModelTest-NG. We also need to address the differences in the base frequency type of a substitution model: As described in Section 2.1.5, the base frequencies can either be equal (denoted `+FE` in RAxML-NG and ModelTest-NG, and `+FQ` in IQTree), empirically counted from the MSA (`+FC` in RAxML-NG and ModelTest-NG, `+F` in IQTree), or optimized with respect to likelihood (`+FO` in all tools). ModelTest-NG uses ML-optimized base frequencies, but does not append `+FO` to the model name in its output, thus requiring a manual workaround. Additionally, IQTree has a software bug where the model names outputted by ModelFinder do not match the conventions used by IQTree when parsing the model specified on the command line. To our understanding, ModelFinder appends a `+F` to the model name whenever it uses empirically counted frequencies, with the absence of a `+F` denoting equal frequencies. However, it does not update the matrix name to the corresponding variant with equal frequencies. For example, it outputs `GTR` instead of `GTR+FQ`, or the even more canonical `SYM+FQ`. Our pipeline circumvents this problem by appending a `+FQ` whenever `+F` is missing from the model name, before passing it as argument to the IQTree reevaluation call.

3.5. Runtime Benchmarks

To benchmark the runtime of the tools, we uniformly sampled 100 datasets from the evaluation sample, excluding two datasets for their excessive runtime requirements. We then re-ran all three tools in sequence with 1, 2, 4, 8, 16, 32, and 48 threads assigned and measured the runtime through Snakemake's benchmarking functionality. The benchmarks ran on a shared-memory

machine with two Intel Xeon Platinum 8260 clocked at 2.4 GHz, with a total of 48 cores and 96 threads, paired with 754 GB of main memory, and running Alma Linux 8.10 with kernel version 4.18.0-553.62.1. Section A.1.1 lists the version of the tools under test.

4. Results

We evaluated a uniformly chosen sample of 500 DNA datasets and 500 AA datasets with IQTree, ModelTest-NG, and RAxML-NG on a shared-memory machine. Figure 9 shows the number of taxa and alignment sites of all datasets in the EvoNAPS database, and the datasets included in our evaluation sample. We ran the analyses with a single thread assigned to each job, with Snakemake utilizing all cores of the machine. For 2 DNA datasets, at least one of the tools did not successfully terminate. Additionally, for 11 AA datasets, either ModelTest-NG or RAxML-NG chose a substitution-matrix unsupported by IQTree (stmtREV or DEN, see Table A.5). In both cases, we discarded the datasets from our analysis, leaving us with 987 datasets in total. Table A.3 in the appendix shows further information about the sample.

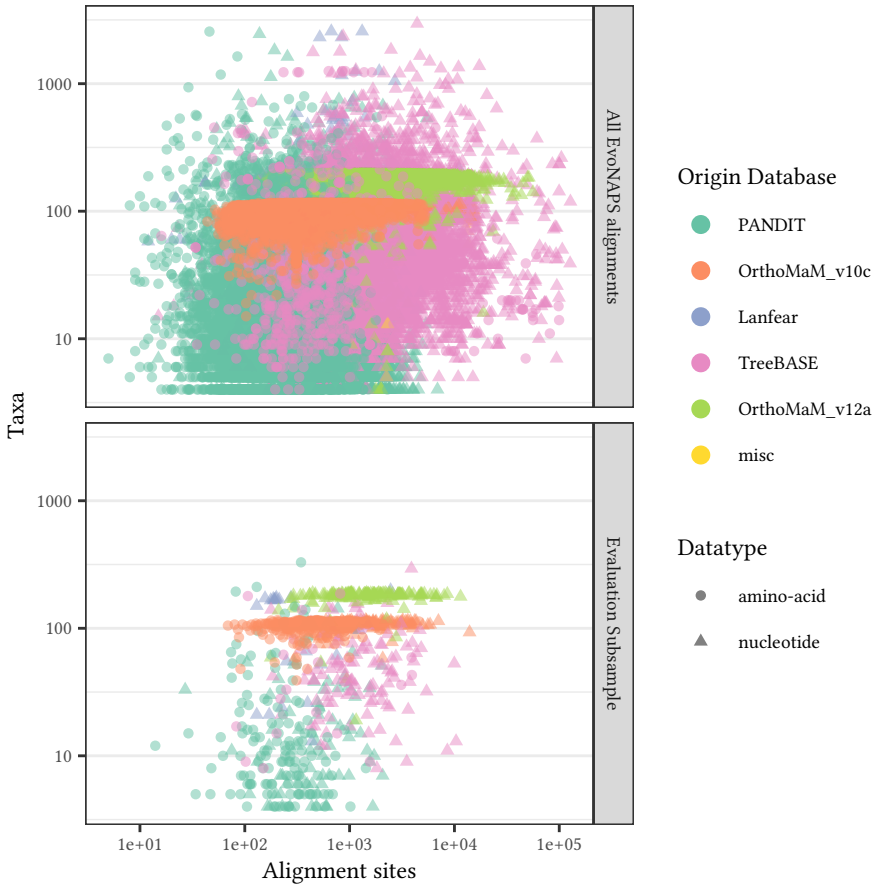


Figure 9: Scatter plot of taxa over sites for EvoNAPS alignments, colored by the origin database. Logarithmic x- and y-axis.

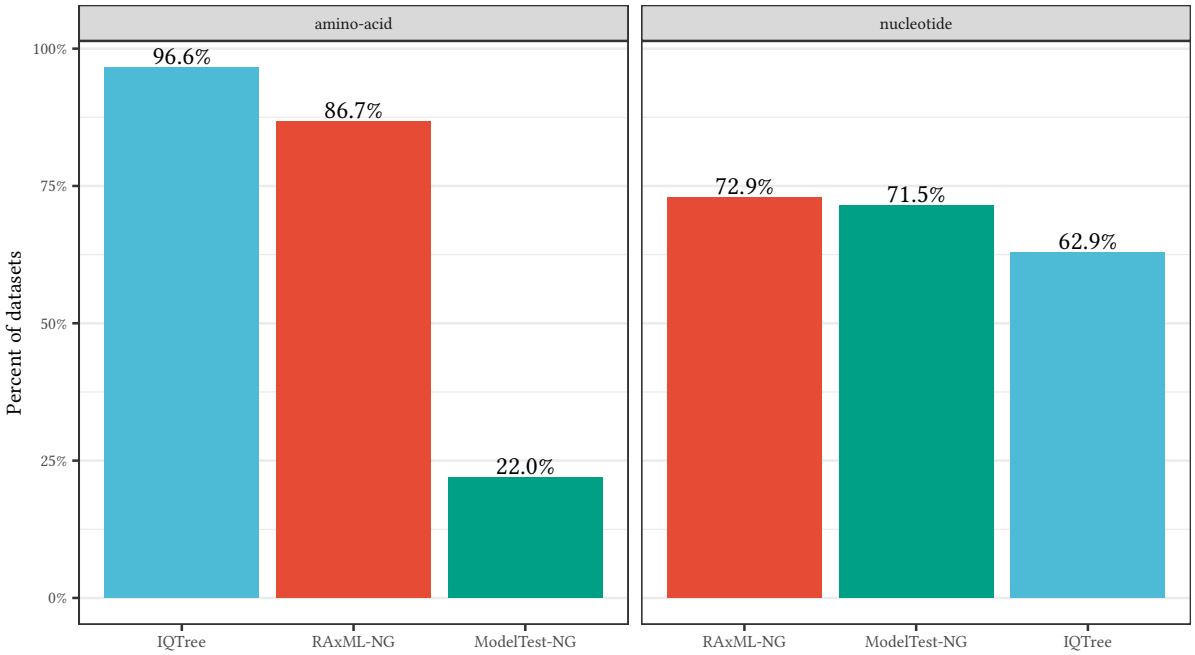


Figure 10: Proportion of datasets where each tool chose a candidate model that had a difference of less than ten in the BIC score to the best known model on the given dataset.

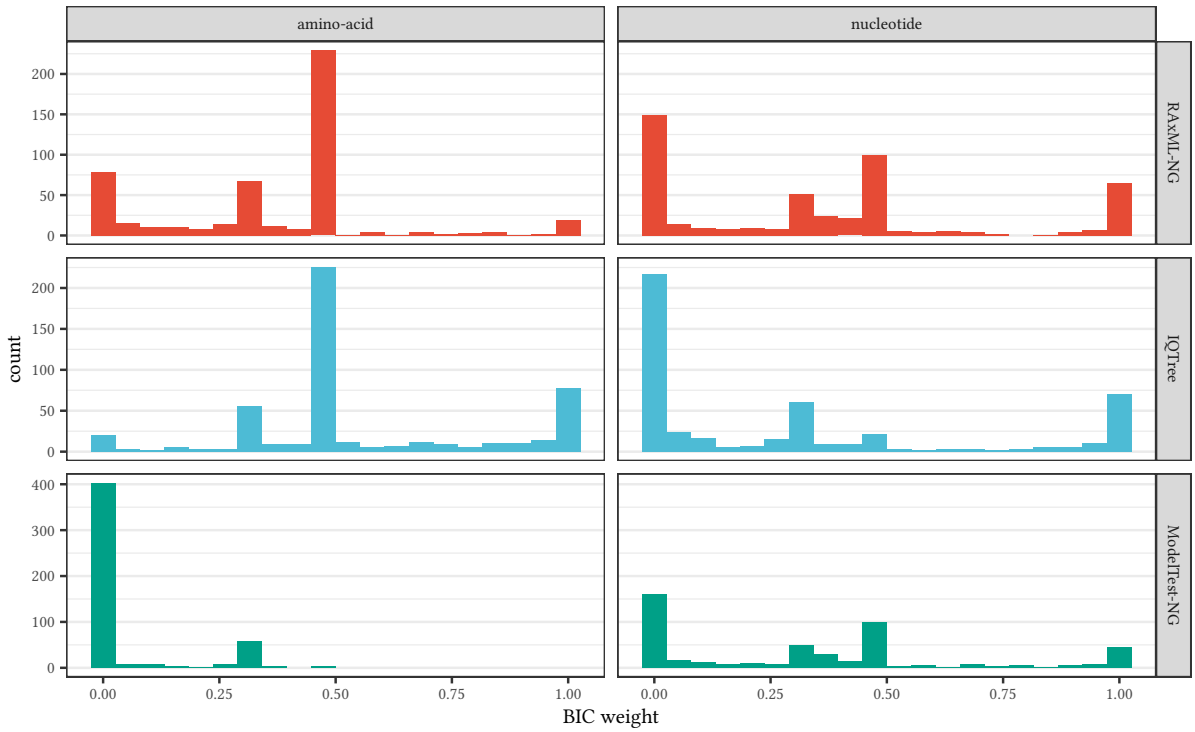


Figure 11: Histograms of the BIC weights for the chosen model of each tool, separated by datatype.

We consider a BIC score difference of greater than 10 as strong evidence that the model with the lower score fits best [21]. In Figure 10, we count on how often datasets each tool provided a model that had BIC score difference (ΔBIC) of less than 10 to the best observed BIC score among the three tools. For AA datasets, IQTree meets this condition on 96.6% of all datasets, followed by RAxML-NG on 86.7%, and ModelTest-NG on 22.0% of all datasets.

For nucleotide datasets, the ranks appear shifted: RAxML-NG finds a good model on 73% of datasets, ModelTest-NG on 71.5%, and IQTree on 62.9% of datasets.

By observing the distribution of the BIC weight for each tool (Figure 11), a similar image presents itself. For AA datasets, in a predominant amount of instances IQTree and RAxML-NG infer a comparably well-fitting model (BIC weight of 0.5), with IQTree sometimes finding a considerably better model than the others (BIC weight near 1), followed by datasets where all three tools perform equally well (BIC weight near 0.33). On DNA datasets, ModelTest-NG performs better, with IQTree delivering worse models (BIC weight near 0).

To further understand how the model choices of the tools differ, in Figure 12 we count how often a tool selects each RHAS model, freerate category count, and base frequency type, separated by datatype (AA or nucleotide datasets). All tools rarely choose the Uniform (`+E`) and Invariant (`+I`) RHAS models. RAxML-NG has a tendency to pick the invariant gamma (`+I+G`) RHAS model. This seems to yield better results for AA datasets than for nucleotide datasets, as indicated by the faint bar that depicts datasets where the model was chosen, but the best observed model discovered by another tool reached a better BIC score with a difference of larger than 10. On AA datasets, IQTree uniformly picks from Gamma and freerate, as well as their Invariant counterparts, with a slight perference towards models with fewer parameters (Gamma and Invariant Gamma). On nucleotide datasets, the opposite is true: IQTree chooses parameter-rich models like Invariant freerate more often. ModelTest-NG has a preference for freerate models on AA datasets, which is even more pronounced on nucleotide datasets.

The number of freerate categories is distributed around a mean of 3 for AA datasets and 4 for nucleotide datasets. On the latter, RAxML-NG more frequently picks higher category counts than IQTree. ModelTest-NG does not test multiple freerate categories.

For AAs, all three tools prefer constant base frequencies as specified by the substitution model. However, ModelTest-NG picks empirical frequencies twice as often as the other tools. On nucleotide datasets, it is apparant that IQTree uses empirical base frequencies, whereas ModelTest-NG and RAxML-NG use ML-optimized base frequencies. The latter also pick equal frequencies less often than IQTree.

Figure 13 shows which rate matrices the tools chose most often. The tools pick the substitution matrices by B. Q. Minh *et al.* [43] (which have the prefix "`Q.`") in 80.4% of all AA datasets. For nucleotide datasets, the distribution is more balanced.

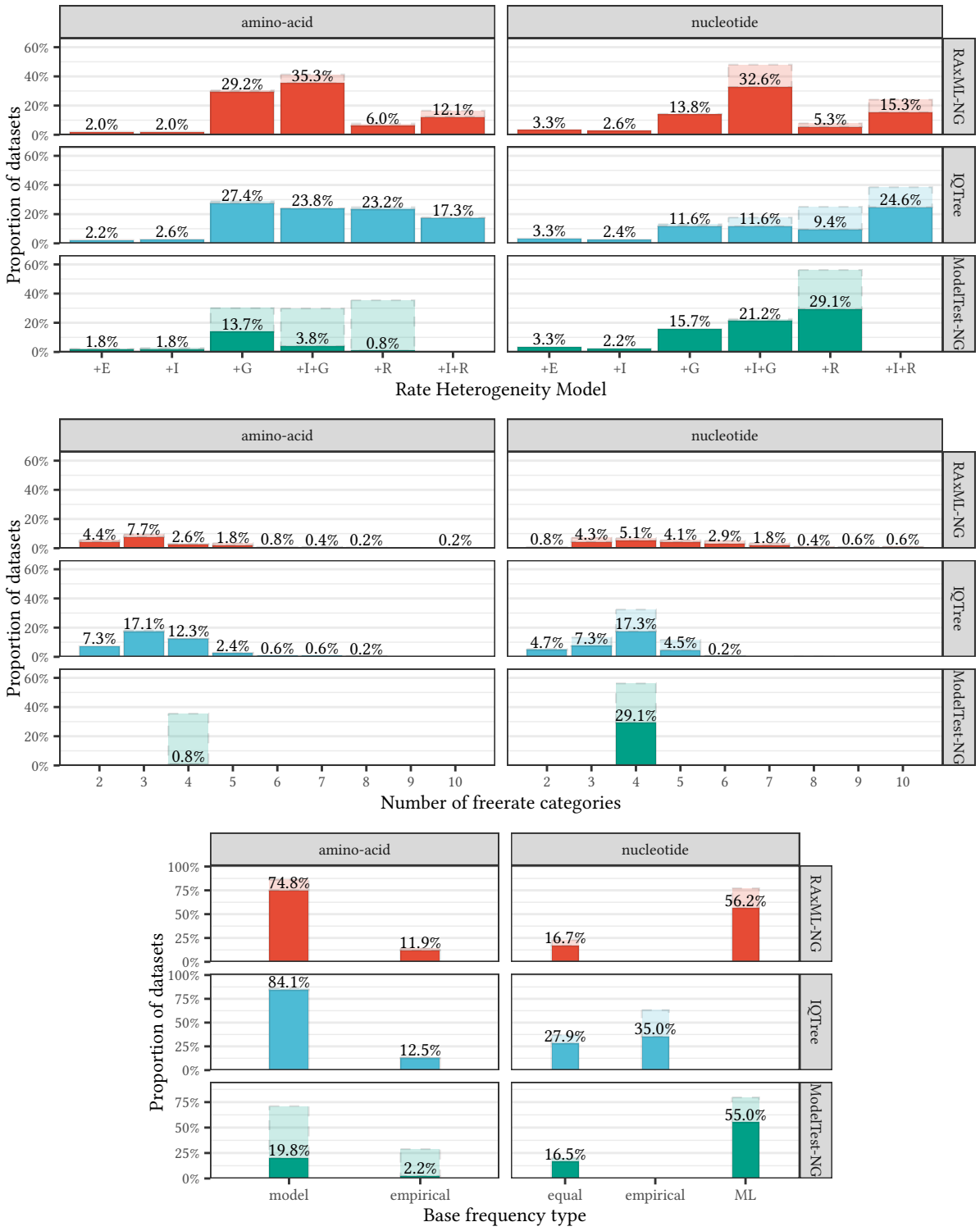


Figure 12: RHAS model type, number of categories and frequency type of the model selected by each tool. Percentage labels on the opaque bars refer to models with ΔBIC < 10 to the best known model among all tools for the given dataset, the faint bars extend to all reported models regardless of tool comparison.

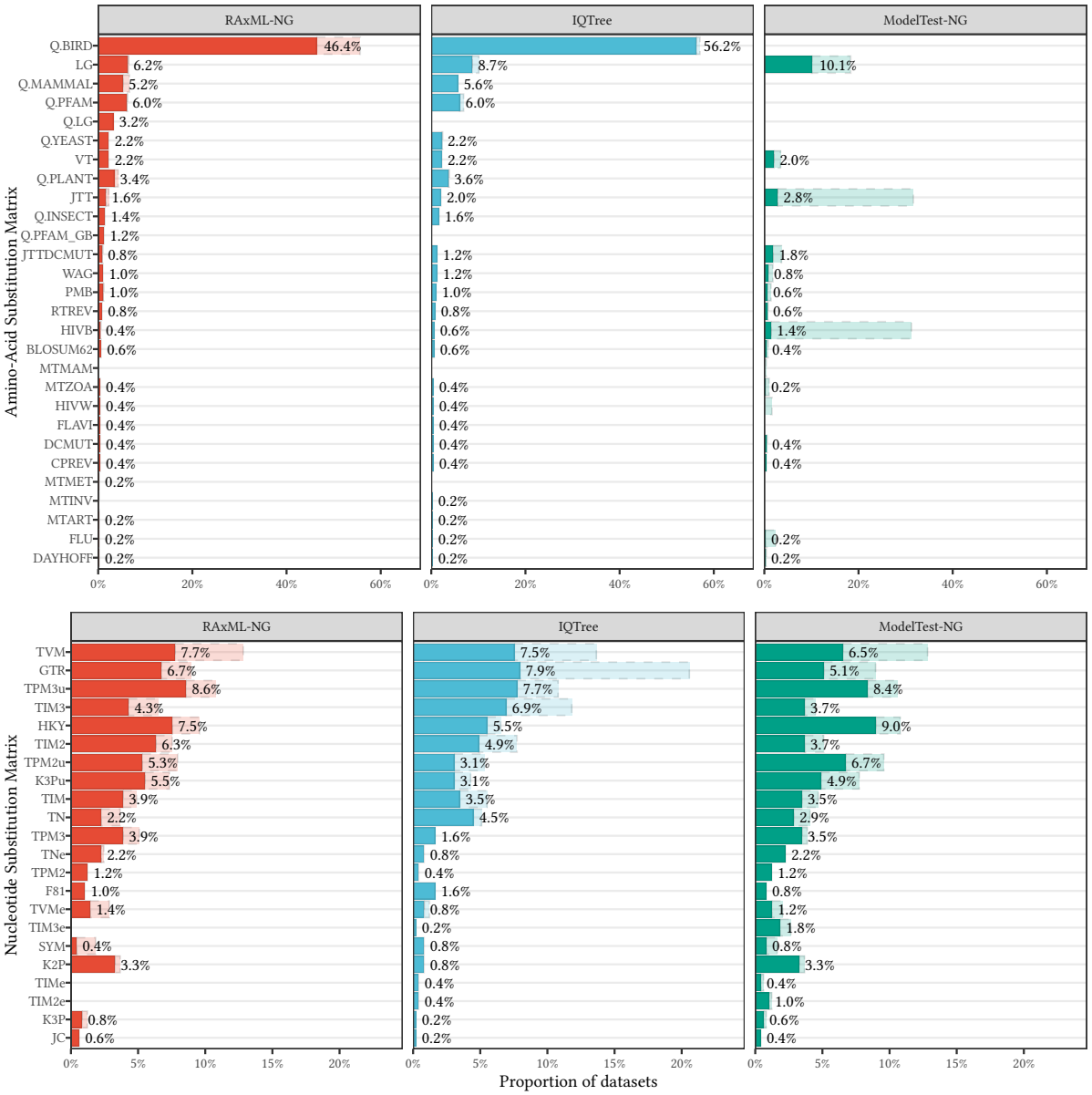


Figure 13: Frequency of rate matrices selected by each tool. Percentage labels on the opaque bars refer to models with $\Delta\text{BIC} < 10$ to the best known model among all tools for the given dataset, the faint bars extend to all reported models regardless of tool comparison.

To benchmark runtime, we ran 100 datasets on a shared-memory machine. Because we sample uniformly and not stratified from EvoNAPS, the datasets are limited in size. The median number of patterns is 456.5 (SD 924.3) and the median number of taxa is 106 (SD 57.9). Two datasets did not finish in time and were excluded for their excessive runtime requirements. Table 1 lists key data of ModelTest-NG's and RAxML-NG's speedup over IQTree, calculated on a per-dataset basis. With a single-thread assigned, ModelTest-NG achieves a mean speedup of 0.41 and RAxML-NG 1.1 on DNA data. We presume that in the single-threaded case the use of heuristics is especially important, as it allows the tool to optimize less candidates. Additionally, since IQTree does not evaluate ML-optimized base frequencies, it has an advantage over ModelTest-NG and RAxML-NG. While RAxML-NG implements the heuristics present in IQTree, it does not optimize the likelihood of the starting tree before beginning the evaluation of candidate models. Similarly, it is missing a thorough round of optimization of the chosen model at the very end.

For AA datasets, ModelTest-NG achieves a mean speedup of 0.61 and RAxML-NG 1.48. Note however, that while RAxML-NG in this instance seems faster than IQTree, it did not yield better results on AA datasets, as detailed in the previous section.

Datatype	Threads	Tool	Mean	Median	Min	Max	Q10	Q90
amino-acid	1	RAxML-NG	1.48	1.35	0.21	7.82	0.48	5.49
		ModelTest-NG	0.61	0.58	0.11	3.03	0.27	1.78
	2	RAxML-NG	1.39	1.27	0.21	6.57	0.62	4.43
		ModelTest-NG	0.73	0.64	0.12	3.24	0.33	1.93
	4	RAxML-NG	1.55	1.38	0.11	6.59	0.76	4.55
		ModelTest-NG	0.88	0.85	0.14	3.72	0.4	2.19
	8	RAxML-NG	1.94	1.88	0.11	8.1	1.02	4.36
		ModelTest-NG	1.09	1.03	0.21	4.46	0.45	2.37
	16	RAxML-NG	2.35	2.41	0.38	6.3	1.24	4.65
		ModelTest-NG	1.31	1.21	0.26	5.04	0.55	2.77
	32	RAxML-NG	3.14	3.41	0.14	8.48	1.69	5.98
		ModelTest-NG	1.67	1.8	0.45	4.9	0.74	3.52
	48	RAxML-NG	3.54	3.52	0.17	13	2.12	6.39
		ModelTest-NG	1.86	1.96	0.5	6.02	0.81	4.37
nucleotide	1	RAxML-NG	1.1	1.14	0.21	4.69	0.46	2.82
		ModelTest-NG	0.41	0.46	0.09	1.96	0.19	0.78
	2	RAxML-NG	1.46	1.64	0.32	4.15	0.63	3.15
		ModelTest-NG	0.64	0.69	0.15	2.34	0.31	1.22
	4	RAxML-NG	1.94	2.22	0.33	5.48	0.81	3.81
		ModelTest-NG	1.05	1.15	0.28	3.97	0.5	2.07
	8	RAxML-NG	2.59	3.07	0.32	9.78	0.8	5.08
		ModelTest-NG	1.89	1.92	0.43	7.28	0.94	4.22
	16	RAxML-NG	3.84	4.4	0.46	12.4	1.46	7.5
		ModelTest-NG	2.88	3.17	0.43	11.6	1.14	5.98
	32	RAxML-NG	5.09	5.93	0.61	21	1.97	11.4
		ModelTest-NG	4.11	4.43	0.49	14.4	1.53	9.09
	48	RAxML-NG	5.82	6.97	0.57	23.1	2.1	12.9
		ModelTest-NG	5.05	5.28	0.57	17.1	2.15	11.5

Table 1: Speedups over IQTree on a per-dataset basis, given for each datatype and thread count. Mean is geometric mean.

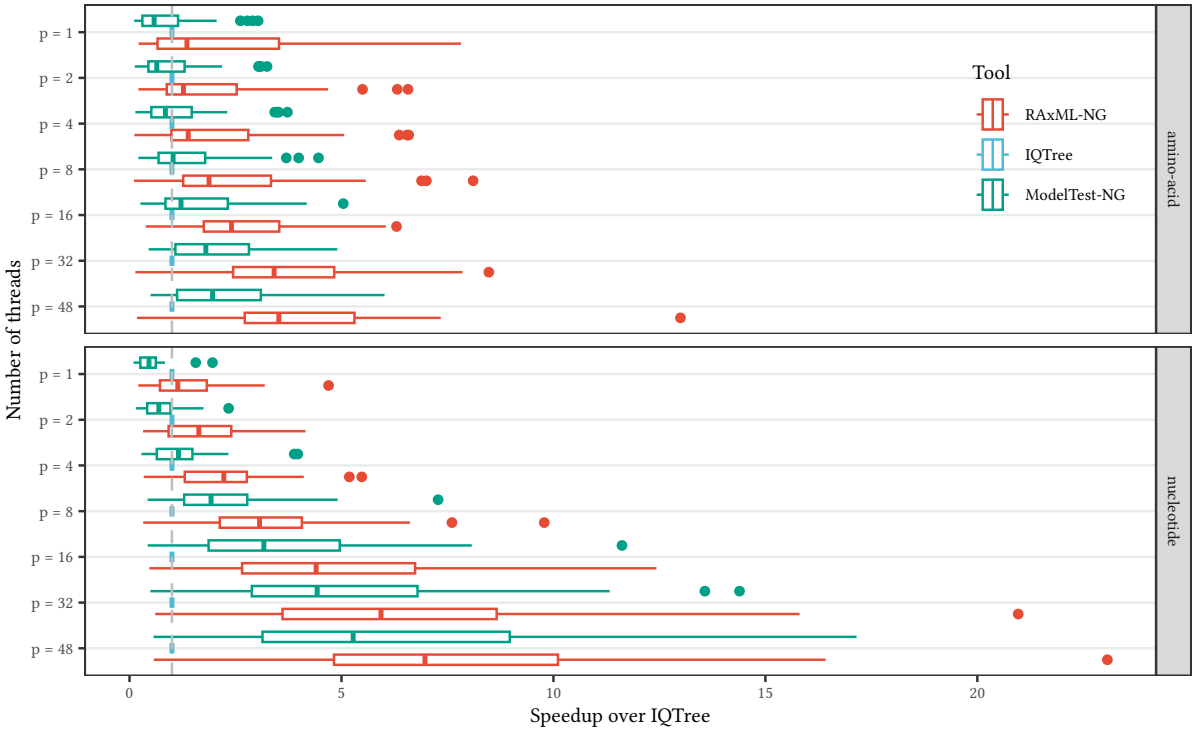


Figure 14: Speedup over IQTree on nucleotide and AA datasets with varying thread count

Figure 14 shows the speedups of ModelTest-NG and RAxML-NG over IQTree, calculated for each dataset and thread count. The speedups of both ModelTest-NG and RAxML-NG over IQTree show an increasing trend with higher thread counts, which we attribute to the different parallelization scheme and limited dataset size. For higher thread counts, IQTree prints a warning, as the datasets only have a small number of sites.

In addition to the speedups per dataset shown above, we also computed the accumulated speedups, that is, the ratio of sums over all runtimes for a given datatype, thread count, and tool. Datasets with longer runtimes have a larger influence on this metric than datasets that are fast to compute. For a single thread, ModelTest-NG reaches an accumulated speedup over IQTree of 1.33 on AA data, and 0.686 on DNA data. RAxML-NG in comparison reaches speedups over IQTree of 1.06 on AA and 1.69 on DNA. For the highest tested thread count of 48 threads, ModelTest-NG's speedups over IQTree are 2.12 on AA data and 3.97 on DNA data, and RAxML-NG's speedups over IQTree are 4.05 on DNA data and 1.72 on AA data. See Table 2 for the complete list of speedup values.

In terms of scaling behavior, we analysed the parallel efficiency, also known as the speedup per processor. If a sequential execution requires time t_1, and a parallel execution with p threads requires time t_p, then the parallel efficiency ε is defined as

$$\varepsilon = \frac{t_1}{t_p * p} \tag{21}$$

Figure 15 shows the parallel efficiency of our benchmarking runs for all three tools. For IQTree, the parallel efficiency diminishes logarithmically. ModelTest-NG exhibits a near-linear decrease in parallel efficiency for up to 32 cores. Additionally, for some datasets, it seems to exhibit a super-linear speedup ($\varepsilon > 1$) when going from a single thread to two threads,

possibly caused by caching effects. The parallel efficiency of RAxML-NG is, simply put, all over the place. Multiple factors could be at play here. Since we only run a single repetition per tool, dataset, and thread count, the effect could be a result of measurement noise. Additionally, since we implement a dynamic threading scheme with a best-effort scheme for load balancing and ad-hoc application of heuristics, both the idle time of threads could vary immensely, as could the true number of candidate model evaluations, that is, the primary workload.

Finally, Figure 16 shows which tool gave the fastest answer with $\Delta\mathrm{BIC} < 10$ to the best known model for the dataset when we assign up to p threads to the problem. IQTree is competitive in the single-threaded case, but if a large amount of threads can be assigned to the problem, RAxML-NG finishes faster due to better scaling behavior on small datasets.

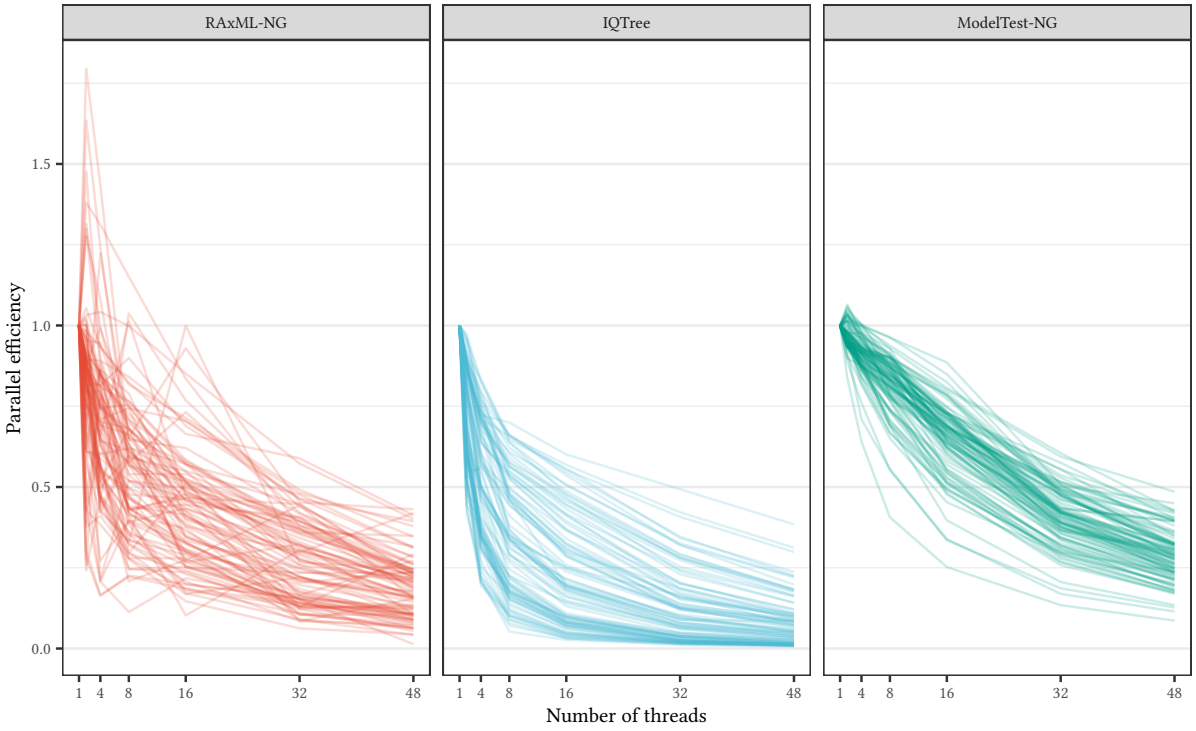


Figure 15: Parallel efficiency of all tools. Each line represents a dataset.

Figure 16: For each dataset and upper bound on the thread count, we consider which tool gave the fastest answer with $\Delta\text{BIC} < 10$.

Datatype	Threads	Tool	Accumulated Speedup	Accumulated Runtime (s)
amino-acid	1	RAxML-NG	1.69	24,028
		ModelTest-NG	1.33	30,532
	2	RAxML-NG	1.11	20,559
		ModelTest-NG	1.44	15,821
	4	RAxML-NG	1.02	13,582
		ModelTest-NG	1.65	8,432
	8	RAxML-NG	0.906	8,989
		ModelTest-NG	1.75	4,639
	16	RAxML-NG	1.62	3,123
		ModelTest-NG	1.81	2,792
	32	RAxML-NG	1.3	3,067
		ModelTest-NG	1.94	2,055
	48	RAxML-NG	1.72	2,291
		ModelTest-NG	2.12	1,857
nucleotide	1	RAxML-NG	1.06	4,860
		ModelTest-NG	0.686	7,536
	2	RAxML-NG	1.2	2,919
		ModelTest-NG	0.904	3,892
	4	RAxML-NG	1.53	1,768
		ModelTest-NG	1.32	2,048
	8	RAxML-NG	1.57	1,370
		ModelTest-NG	1.89	1,139
	16	RAxML-NG	2.39	768
		ModelTest-NG	2.56	717
	32	RAxML-NG	3.38	600
		ModelTest-NG	3.32	610
	48	RAxML-NG	4.05	587
		ModelTest-NG	3.97	598

Table 2: Accumulated speedups (sum over all runtimes) per thread count and datatype

5. Discussion

The extent of our benchmarks and evaluations covers only a narrow range of single-partitioned datasets with a low number of patterns (median 456.5 for the benchmark dataset). Furthermore, the reliance on the model reevaluation with IQTree on a different tree topology than the one used by RAxML-NG and ModelTest-NG for model optimization, the choice of using the BIC score as a sole comparison measure, as well as examining only a single "best model" per tool and dataset puts a limit on the explanatory power of our accuracy evaluation.

Nonetheless, we believe to have shown that, on nucleotide datasets with comparable size to our evaluation sample, RAxML-NG's model selection procedure can deliver results on-par with ModelTest-NG and IQTree's ModelFinder. We attribute the slight improvement over IQTree on nucleotide datasets to the fact that IQTree does not consider ML-optimized base frequencies and only uses empirically counted frequencies. Given that the former have the same number of free parameters, but yield a better likelihood, we believe optimized frequencies to be superior with respect to the information criterion BIC. For AA datasets, our tool performs slight sub-par to IQTree, but in the context of our evaluation setup, outperforms ModelTest-NG accuracy-wise. The poor performance on ModelTest-NG on AA datasets has two potential reasons. For one, it does not include the constant rate matrices derived by B. Q. Minh *et al.* [43], which RAxML-NG and IQTree select as best-fitting for 80.4% of all AA datasets. Secondly, given the fact that all AA rate matrices under consideration are constant and thus do not add to the number of free parameters, the choice of the RHAS models becomes evermore important. With the default arguments, ModelTest-NG only considers freerate models with four categories and does not support checking multiple category counts in a single run.

While we do not have enough data to make statements for larger datasets, we show that parallelizing over candidate models is faster for small datasets when enough CPU cores are available. Additionally, when considering freerate models with different category counts, heuristics are important to keep the workload below a tolerable level.

6. Conclusion

In this thesis, we present a novel implementation of a model selection algorithm that builds on the ideas of existing tools. Through a dynamic parallelization scheme and broad support for models, it has competitive accuracy and runtime. On single-gene datasets, it is able to outperform the existing tools. Its integration into the widely-used tree search tool RAxML-NG makes it instantly accessible to a large group of users. Further work is necessary to tune the performance of our tool and validate its accuracy on a larger sample of datasets.

6.1. Future Work

For a more complete comparison of the three tools investigated in this thesis, further work is necessary. We need to quantify the performance and accuracy on larger datasets, possibly with stratified sampling with respect to dataset sources, size, and difficulty [44]. In order to evaluate how well the tools recover the models of simulated MSAs, simulation studies are of further interest. Additionally, one could observe how the selected models perform during tree search, and if and how the plausible tree set changes under the model.

Empirical data may help refining our heuristics for load balancing (Equation 18 and Equation 19), adapting them to the characteristics of model optimization. Furthermore, the parallel efficiency exhibits chaotic behavior (see Figure 15) we need to be address. For instance, in order to avoid excessive wait times, our scheduler could permit the evaluation to start with less threads. Additionally, as the heuristics sometimes deliver their information “too late” for our eager evaluation scheduling, preemptive scheduling could help prevent wasting time on evaluating candidate models that have not proven worthwhile.

Bibliography

[1] D. Darriba *et al.*, 'ModelTest-NG: A New and Scalable Tool for the Selection of DNA and Protein Evolutionary Models', *Molecular Biology and Evolution*, vol. 37, no. 1, pp. 291–294, Jan. 2020, doi: 10.1093/molbev/msz189.

[2] T. Wong *et al.*, 'IQ-TREE 3: Phylogenomic Inference Software Using Complex Evolutionary Models'. Accessed: Apr. 14, 2025. [Online]. Available: https://ecoevorxiv.org/repository/view/8916/

[3] S. Kalyaanamoorthy *et al.*, 'ModelFinder: Fast Model Selection for Accurate Phylogenetic Estimates', *Nature Methods*, vol. 14, no. 6, pp. 587–589, June 2017, doi: 10.1038/nmeth.4285.

[4] V. Ranwez *et al.*, 'OrthoMaM: A Database of Orthologous Genomic Markers for Placental Mammal Phylogenetics', *BMC Evolutionary Biology*, vol. 7, no. 1, p. 241, 2007, doi: 10.1186/1471-2148-7-241.

[5] Z. Yang, *Computational Molecular Evolution*, 1st edn. Oxford University Press, 2006. doi: 10.1093/acprof:oso/9780198567028.001.0001.

[6] 'Occam's Razor'. Oxford University Press, 2005. doi: 10.1093/oi/authority.20110803100244343.

[7] R. Durrett, *Essentials of Stochastic Processes*, 2nd ed. 2012. in Springer Texts in Statistics. New York, NY: Springer New York, 2012. doi: 10.1007/978-1-4614-3615-7.

[8] J. Felsenstein, 'Evolutionary Trees from DNA Sequences: A Maximum Likelihood Approach', *Journal of Molecular Evolution*, vol. 17, no. 6, pp. 368–376, Nov. 1981, doi: 10.1007/BF01734359.

[9] T. H. Jukes and C. R. Cantor, 'Evolution of Protein Molecules', *Mammalian Protein Metabolism*. Academic Press, New York, pp. 21–123, 1969.

[10] S. Tavaré, 'Some Probabilistic and Statistical Problems in the Analysis of DNA Sequences', *Lectures on Mathematics in the Life Sciences*, vol. 17. pp. 57–86, 1986. Accessed: Dec. 02, 2024. [Online]. Available: https://archive.org/details/someprobabilisticandstatisticalproblemsintheanalysisofdnasequences

[11] M. Hoff *et al.*, 'Does the Choice of Nucleotide Substitution Models Matter Topologically?', *BMC Bioinformatics*, vol. 17, no. 1, p. 143, Mar. 2016, doi: 10.1186/s12859-016-0985-x.

[12] E. W. Weisstein, 'Gamma Function'. Wolfram Research, Inc. Accessed: Oct. 16, 2025. [Online]. Available: https://mathworld.wolfram.com/GammaFunction.html

[13] E. W. Weisstein, 'Gamma Distribution'. Wolfram Research, Inc. Accessed: Oct. 16, 2025. [Online]. Available: https://mathworld.wolfram.com/GammaDistribution.html

[14] Z. Yang, 'Maximum Likelihood Phylogenetic Estimation from DNA Sequences with Variable Rates over Sites: Approximate Methods', *Journal of Molecular Evolution*, vol. 39, no. 3, pp. 306–314, Sept. 1994, doi: 10.1007/BF00160154.

[15] R. P. Brent, *Algorithms for Minimization without Derivatives*. in Prentice-Hall Series in Automatic Computation. Englewood Cliffs, N.J: Prentice-Hall, 1973.

[16] C. Zhu *et al.*, 'Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-Scale Bound-Constrained Optimization', *ACM Transactions on Mathematical Software*, vol. 23, no. 4, pp. 550–560, Dec. 1997, doi: 10.1145/279232.279236.

[17] R. H. Byrd *et al.*, 'A Limited Memory Algorithm for Bound Constrained Optimization', *SIAM Journal on Scientific Computing*, vol. 16, no. 5, pp. 1190–1208, Sept. 1995, doi: 10.1137/0916069.

[18] H. Akaike, 'A New Look at the Statistical Model Identification', *IEEE Transactions on Automatic Control*, vol. 19, no. 6, pp. 716–723, Dec. 1974, doi: 10.1109/TAC.1974.1100705.

[19] C. M. Hurvich and C.-L. Tsai, 'Regression and Time Series Model Selection in Small Samples', *Biometrika*, vol. 76, no. 2, pp. 297–307, 1989, doi: 10.1093/biomet/76.2.297.

[20] G. Schwarz, 'Estimating the Dimension of a Model', *The Annals of Statistics*, vol. 6, no. 2, pp. 461–464, 1978, Accessed: Dec. 02, 2024. [Online]. Available: https://www.jstor.org/stable/2958889

[21] R. E. Kass and A. E. Raftery, 'Bayes Factors', *Journal of the American Statistical Association*, vol. 90, no. 430, pp. 773–795, June 1995, doi: 10.1080/01621459.1995.10476572.

[22] D. Darriba *et al.*, 'jModelTest 2: More Models, New Heuristics and High-Performance Computing', *Nature methods*, vol. 9, no. 8, p. 772, July 2012, doi: 10.1038/nmeth.2109.

[23] D. Darriba *et al.*, 'ProtTest 3: Fast Selection of Best-Fit Models of Protein Evolution', *Bioinformatics*, vol. 27, no. 8, pp. 1164–1165, Apr. 2011, doi: 10.1093/bioinformatics/btr088.

[24] T. Flouri *et al.*, 'The Phylogenetic Likelihood Library', *Systematic Biology*, vol. 64, no. 2, pp. 356–362, Mar. 2015, doi: 10.1093/sysbio/syu084.

[25] S. Abadi *et al.*, 'ModelTeller: Model Selection for Optimal Phylogenetic Reconstruction Using Machine Learning', *Molecular Biology and Evolution*, vol. 37, no. 11, pp. 3338–3352, Nov. 2020, doi: 10.1093/molbev/msaa154.

[26] S. Burgstaller-Muehlbacher *et al.*, 'ModelRevelator: Fast Phylogenetic Model Estimation via Deep Learning', *Molecular Phylogenetics and Evolution*, vol. 188, p. 107905, Nov. 2023, doi: 10.1016/j.ympev.2023.107905.

[27] 'Coraxlib'. [Online]. Available: https://codeberg.org/Exelixis-Lab/coraxlib/

[28] K. Kobert *et al.*, 'The Divisible Load Balance Problem and Its Application to Phylogenetic Inference', *Algorithms in Bioinformatics*, vol. 8701. in Lecture Notes in Computer Science, vol. 8701. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 204–216, 2014. doi: 10.1007/978-3-662-44753-6_16.

[29] B. Morel *et al.*, 'A Novel Heuristic for Data Distribution in Massively Parallel Phylogenetic Inference Using Site Repeats', in *2017 IEEE 19th International Conference on High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE 3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, Bangkok: IEEE, Dec. 2017, pp. 81–88. doi: 10.1109/HPCC-SmartCity-DSS.2017.11.

[30] A. M. Kozlov *et al.*, 'RAxML-NG: A Fast, Scalable and User-Friendly Tool for Maximum Likelihood Phylogenetic Inference', *Bioinformatics*, vol. 35, no. 21, pp. 4453–4455, Nov. 2019, doi: 10.1093/bioinformatics/btz305.

[31] 'MPI: A Message-Passing Interface Standard'. [Online]. Available: https://www.mpi-forum.org/docs/mpi-4.1/mpi41-report.pdf

[32] F. Reden, 'EvoNAPS: A Database for Natural Parameter Settings of Evolutionary Models', mathesis, Vienna, Austria, 2023. [Online]. Available: https://doi.org/10.25365/thesis.76008

[33] W. H. Piel *et al.*, 'TreeBASE: A Database of Phylogenetic Information.', in *Proceedings of the 2nd International Workshop of Species*, 2000.

[34] S. Whelan, 'PANDIT: An Evolution-Centric Database of Protein and Associated Nucleotide Domains with Inferred Trees', *Nucleic Acids Research*, vol. 34, no. 90001, pp. D327–D331, Jan. 2006, doi: 10.1093/nar/gkj087.

[35] D. R. Hipp, 'SQLite'. [Online]. Available: https://sqlite.org/

[36] J.-L. Lacroix *et al.*, 'Mysql2sqlite'. Accessed: Oct. 13, 2025. [Online]. Available: https://github.com/mysql2sqlite/mysql2sqlite

[37] F. Mölder *et al.*, 'Sustainable Data Analysis with Snakemake'. Accessed: Feb. 07, 2025. [Online]. Available: https://f1000research.com/articles/10-33

[38] Conda-Forge Community, 'The Conda-Forge Project: Community-based Software Distribution Built on the Conda Package Format and Ecosystem'. Accessed: Oct. 20, 2025. [Online]. Available: https://doi.org/10.5281/ZENODO.4774216

[39] B. Grüning *et al.*, 'Bioconda: Sustainable and Comprehensive Software Distribution for the Life Sciences', *Nature Methods*, vol. 15, no. 7, pp. 475–476, July 2018, doi: 10.1038/s41592-018-0046-7.

[40] R Core Team, 'R: A Language and Environment for Statistical Computing'. R Foundation for Statistical Computing, Vienna, Austria, 2025. [Online]. Available: https://www.r-project.org/

[41] H. Wickham *et al.*, 'Welcome to the tidyverse', *Journal of Open Source Software*, vol. 4, no. 43, p. 1686, 2019, doi: 10.21105/joss.01686.

[42] E.-J. Wagenmakers and S. Farrell, 'AIC Model Selection Using Akaike Weights', *Psychonomic Bulletin & Review*, vol. 11, no. 1, pp. 192–196, Feb. 2004, doi: 10.3758/BF03206482.

[43] B. Q. Minh *et al.*, 'QMaker: Fast and Accurate Method to Estimate Empirical Models of Protein Evolution', *Systematic Biology*, vol. 70, no. 5, pp. 1046–1060, Sept. 2021, doi: 10.1093/sysbio/syab010.

[44] J. Haag *et al.*, 'From Easy to Hopeless—Predicting the Difficulty of Phylogenetic Analyses', *Molecular Biology and Evolution*, vol. 39, no. 12, p. msac254, Dec. 2022, doi: 10.1093/molbev/msac254.

[45] M. Dayhoff *et al.*, 'A Model of Evolutionary Change in Proteins', *Atlas of protein sequence and structure*, vol. 5, pp. 345–352, 1978.

[46] C. Kosiol and N. Goldman, 'Different Versions of the Dayhoff Rate Matrix', *Molecular Biology and Evolution*, vol. 22, no. 2, pp. 193–199, Feb. 2005, doi: 10.1093/molbev/msi005.

[47] S. Henikoff and J. G. Henikoff, 'Amino Acid Substitution Matrices from Protein Blocks.', *Proceedings of the National Academy of Sciences*, vol. 89, no. 22, pp. 10915–10919, Nov. 1992, doi: 10.1073/pnas.89.22.10915.

[48] D. T. Jones *et al.*, 'The Rapid Generation of Mutation Data Matrices from Protein Sequences', *Bioinformatics*, vol. 8, no. 3, pp. 275–282, 1992, doi: 10.1093/bioinformatics/8.3.275.

[49] J. Adachi and M. Hasegawa, 'Model of Amino Acid Substitution in Proteins Encoded by Mitochondrial DNA', *Journal of Molecular Evolution*, vol. 42, no. 4, pp. 459–468, Apr. 1996, doi: 10.1007/BF02498640.

[50] Z. Yang *et al.*, 'Models of Amino Acid Substitution and Applications to Mitochondrial Protein Evolution', *Molecular Biology and Evolution*, vol. 15, no. 12, pp. 1600–1611, Dec. 1998, doi: 10.1093/oxfordjournals.molbev.a025888.

[51] J. Adachi *et al.*, 'Plastid Genome Phylogeny and a Model of Amino Acid Substitution for Proteins Encoded by Chloroplast DNA', *Journal of Molecular Evolution*, vol. 50, no. 4, pp. 348–358, Apr. 2000, doi: 10.1007/s002399910038.

[52] T. Müller and M. Vingron, 'Modeling Amino Acid Replacement', *Journal of Computational Biology*, vol. 7, no. 6, pp. 761–776, Dec. 2000, doi: 10.1089/10665270050514918.

[53] S. Whelan and N. Goldman, 'A General Empirical Model of Protein Evolution Derived from Multiple Protein Families Using a Maximum-Likelihood Approach', *Molecular Biology and Evolution*, vol. 18, no. 5, pp. 691–699, May 2001, doi: 10.1093/oxfordjournals.molbev.a003851.

[54] M. W. Dimmic *et al.*, 'rtREV: An Amino Acid Substitution Matrix for Inference of Retrovirus and Reverse Transcriptase Phylogeny', *Journal of molecular evolution*, vol. 55, no. 1, p. 65, 2002.

[55] S. Veerassamy *et al.*, 'A Transition Probability Model for Amino Acid Substitutions from Blocks', *Journal of Computational Biology*, vol. 10, no. 6, pp. 997–1010, Dec. 2003, doi: 10.1089/106652703322756195.

[56] F. Abascal *et al.*, 'MtArt: A New Model of Amino Acid Replacement for Arthropoda', *Molecular Biology and Evolution*, vol. 24, no. 1, pp. 1–5, Oct. 2006, doi: 10.1093/molbev/msl136.

[57] D. C. Nickle *et al.*, 'HIV-Specific Probabilistic Models of Protein Evolution', *PLoS ONE*, vol. 2, no. 6, p. e503, June 2007, doi: 10.1371/journal.pone.0000503.

[58] S. Q. Le and O. Gascuel, 'An Improved General Amino Acid Replacement Matrix', *Molecular Biology and Evolution*, vol. 25, no. 7, pp. 1307–1320, Apr. 2008, doi: 10.1093/molbev/msn067.

[59] O. Rota-Stabelli *et al.*, 'MtZoa: A General Mitochondrial Amino Acid Substitutions Model for Animal Evolutionary Studies', *Molecular Phylogenetics and Evolution*, vol. 52, no. 1, pp. 268–272, July 2009, doi: 10.1016/j.ympev.2009.01.011.

[60] C. C. Dang *et al.*, 'FLU, an Amino Acid Substitution Model for Influenza Proteins', *BMC Evolutionary Biology*, vol. 10, no. 1, p. 99, Dec. 2010, doi: 10.1186/1471-2148-10-99.

[61] Y. Liu *et al.*, 'Mitochondrial Phylogenomics of Early Land Plants: Mitigating the Effects of Saturation, Compositional Heterogeneity, and Codon-Usage Bias', *Systematic Biology*, vol. 63, no. 6, pp. 862–878, Nov. 2014, doi: 10.1093/sysbio/syu049.

[62] V. S. Le *et al.*, 'Improved Mitochondrial Amino Acid Substitution Models for Metazoan Evolutionary Studies', *BMC Evolutionary Biology*, vol. 17, no. 1, p. 136, Dec. 2017, doi: 10.1186/s12862-017-0987-y.

[63] T. Le Kim *et al.*, 'Building a Specific Amino Acid Substitution Model for Dengue Viruses', in *2018 10th International Conference on Knowledge and Systems Engineering (KSE)*, 2018, pp. 242–246. doi: 10.1109/KSE.2018.8573341.

[64] T. K. Le and L. S. Vinh, 'FLAVI: An Amino Acid Substitution Model for Flaviviruses', *Journal of Molecular Evolution*, vol. 88, no. 5, pp. 445–452, July 2020, doi: 10.1007/s00239-020-09943-3.

A Appendix

A.1 Data Availability

The implementation is available on GitHub[4] and has been merged into RAxML-NG v2.0-beta3.

A.1.1 Software Versions

Sourced from Bioconda [39]:

- `snakemake=9.11.8=hdfd78af_0`
- `raxml-ng=1.2.2=h6747034_2`
- `modeltest-ng=0.1.7=hf316886_3`
- `iqtree=3.0.1=h503566f_0`

Custom package[5], implementation presented in this thesis:

- `raxml-ng-modeltest=2.0.0.dev1=0_gbdffad2_2`

[4] https://github.com/stelzch/raxml-ng/tree/feature/moose
[5] https://anaconda.org/stelzch/raxml-ng-modeltest

A.2 Evaluation Subsample

Datatype	Database	N	Median Taxa	Median Sites	Median Patterns
aa	OrthoMaM_v10c	336	109	451.5	371.5
	PANDIT	138	10	233	209
	TreeBASE	22	40.5	462	382.5
dna	Lanfear	21	150	213	154
	OrthoMaM_v10c	133	107	1,173	835
	OrthoMaM_v12a	158	183	1,435.5	886.5
	PANDIT	71	12	483	338
	TreeBASE	108	45.5	1,150	381

Table A.3: Key data of the evaluation sample

A.3 List of DNA substitution models

Rate symmetry	Frequencies	coraxlib name(s)	IQTree name(s)
000 00 0	equal	JC	JC, JC69
000 00 0	free	F81	F81
010 01 0	equal	K80	K2P, K80
010 01 0	free	HKY	HKY, HKY85
010 02 0	equal	TN93ef, TrNef	TNe, TNef, TrNef, TNe, TrNe
010 02 0	free	TN93, TrN	TN, TrN, TN93
012 21 0	equal	K81, TPM1	K3P, K81, TPM1
012 21 0	free	K81uf, TPM1uf	K3Pu, K81uf, K81u, K3Puf, TPM1uf, TPM1u
010 21 2	equal	TPM2, TPM2ef	TPM2
010 21 2	free	TPM2uf	TPM2u, TPM2uf
012 01 2	equal	TPM3, TPM3ef	TPM3
012 01 2	free	TPM3uf	TPM3u, TPM3uf
012 23 0	equal	TIM1, TIM1ef	TIMe, TIMef, TIMe, TIM1ef, TIM1e
012 23 0	free	TIM1uf	TIM, TIM1
010 23 2	equal	TIM2, TIM2ef	TIM2e, TIM2ef
010 23 2	free	TIM2uf	TIM2
012 03 2	equal	TIM3, TIM3ef	TIM3e, TIM3ef
012 03 2	free	TIM3uf	TIM3
012 31 4	equal	TVMef	TVMe, TVMef
012 31 4	free	TVM	TVM
012 34 5	equal	SYM	SYM
012 34 5	free	GTR	GTR, REV

Table A.4: List of DNA model names for RAxML-NG (coraxlib) and IQTree. The first name is the canonical name, all following names are aliases. Note that for the most part, cross-compatible aliases exist, except for TN93ef and TIM1, TIM2, and TIM3. For the latter, the programs also differ on the default frequency setting.

A.4 List of Amino Acid Substitution Matrices

Matrix name	Citation
DAYHOFF	M. Dayhoff *et al.* [45], C. Kosiol and N. Goldman [46]
BLOSUM62	S. Henikoff and J. G. Henikoff [47]
JTT	D. T. Jones *et al.* [48]
mtREV	J. Adachi and M. Hasegawa [49]
mtMAM	Z. Yang *et al.* [50]
cpREV	J. Adachi *et al.* [51]
VT	T. Müller and M. Vingron [52]
WAG	S. Whelan and N. Goldman [53]
rtREV	M. W. Dimmic *et al.* [54]
PMB	S. Veerassamy *et al.* [55]
mtART	F. Abascal *et al.* [56]
HIVB, HIVW	D. C. Nickle *et al.* [57]
LG	S. Q. Le and O. Gascuel [58]
mtZOA	O. Rota-Stabelli *et al.* [59]
FLU	C. C. Dang *et al.* [60]
stmtREV	Y. Liu *et al.* [61]
mtMET, mtVER, mtINV	V. S. Le *et al.* [62]
DEN	T. Le Kim *et al.* [63]
FLAVI	T. K. Le and L. S. Vinh [64]
Q.BIRD, Q.INSECT, Q.MAMMAL, Q.PLANT, Q.YEAST, Q.LG, Q.PFAM, Q.PFAM_GB	B. Q. Minh *et al.* [43]

Table A.5: List of AA substitution matrices available in RAxML-NG.