CherryML 2.0: Almost Linear Time Estimation of Phylogenetic Models

Wilson Wu

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2024-107
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2024/EECS-2024-107.html

May 15, 2024

Acknowledgement

I would like to thank Yun S. Song and Sebastian Prillo for their help and guidance throughout the entire project. Thank you for mentoring me throughout the research process with our insightful discussions on theory and computational experiments. It was a great pleasure working with you two throughout the year.

CherryML 2.0: Almost Linear Time Estimation of Phylogenetic Models

by

Wilson Wu

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Yun S. Song, Co-chair
Professor Satish Rao, Co-chair

Spring 2024

CherryML 2.0: Almost Linear Time Estimation of Phylogenetic Models

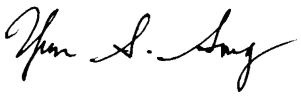
by Wilson Wu

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:



Professor Yun S. Song
Research Advisor

May 8, 2024

(Date)

* * * * * * *

Professor Satish Rao
Second Reader

May 15, 2024

(Date)

CherryML 2.0: Almost Linear Time Estimation of Phylogenetic Models

Abstract

CherryML 2.0: Almost Linear Time Estimation of Phylogenetic Models

by

Wilson Wu

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Yun S. Song, Co-chair

Professor Satish Rao, Co-chair

Phylogenetic models of protein evolution are parameterized by a rate matrix Q describing the rate at which amino acids mutate. Maximum likelihood based methods typically alternately optimize phylogenetic trees over the multiple sequence alignments given a rate matrix and the rate matrix given the trees. Classically, this process was bottlenecked by the matrix estimation step. Recently, CherryML showed that performing matrix estimation using composite likelihood estimate yielded accurate rate matrices whilst being orders of magnitudes faster. In this paper, we present CherryML 2.0 which applies composite likelihood to the tree estimation step, speeding up the tree estimation step by orders of magnitude with minimal impact in accuracy. Furthermore, CherryML 2.0's time complexity is almost linear in the length of sequences and number of sequences per multiple sequence alignments. We show that CherryML 2.0 achieves similar accuracy to CherryML 1.0 on real datasets, and it's able to reconstruct a rate matrix with less than 2% median error on simulated datasets.

Contents

List of Figures

Acknowledgments

I would like to thank Yun S. Song and Sebastian Prillo for their help and guidance throughout the entire project. Thank you for mentoring me throughout the research process with our insightful discussions on theory and computation experiments.

Chapter 1

Introduction

The evolution of amino acids is classically modelled with phylogenetic models parameterized by a rate matrix Q. This rate matrix describes the rate at which each amino acid mutates into one another. Q is usually inferred empirically from datasets – a process requiring trade offs between accuracy of the estimated matrix and computational efficiency.

Early works used counting based approaches in order to estimate this rate matrix. The Dayhoff model [2] first estimated phylogenetic trees over multiple MSA's. Then using maximum parsimony, it inferred ancestral sequences. The resulting matrix can then be estimated by simply counting the number of transitions between each pair of amino acids that occur across each branch in the trees. The model by Jones, Taylor, and Thornton, often referred to as the JTT model [5], similarly counted the number of transitions between each pair of amino acids in an estimated tree. However instead of using maximum parsimony to infer ancestral states, it instead iteratively counts the mutations across, then prunes pairs of nearest neighbors from the estimated tree. These methods are inherently biased as they assume only one mutation happens at a given site for each given branch, and the estimation methods do not take into account evolutionary time.

Maximum likelihood estimate (MLE) based approaches were later created to estimate more accurate rate matrices. The model by Whelan and Goldman [12], which we refer to as the WAG model, was among the first such models which used MLE to estimate rate matrices. WAG alternately estimates the MLE phylogenetic tree from the rate matrix and MLE rate matrix from the tree in a coordinate ascent like algorithm. This procedure is used to estimate Q from

$$\arg\max_{Q,T} P(D|T,Q) = \arg\max_{Q,T} \prod_i P(D_i|T_i,Q)$$

Here, $D = (D_1, D_2, ..., D_m)$ are m multiple sequence alignments (MSA) and $T = (T_1, T_2, ..., T_m)$ are estimated phylogenetic trees corresponding to the MSA's. Computing an MLE phylogenetic tree is computationally expensive. However, WAG made the crucial insight that sufficiently accurate phylogenetic trees can be used to accurately estimate the MLE rate matrix. Thus, maximizing the likelihood function can be approximated by estimating phylo-

genetic trees with neighbor-joining. Then, MLE branch lengths can be efficiently estimated for these topologies resulting in the trees in T. Given these "good enough" phylogenetic trees, numerical optimization algorithms can then be used to estimate.

$$\hat{Q} = \arg\max_{Q} P(D|T, Q)$$

This insight allowed WAG to estimate the WAG matrix using 3,905 sequences across 182 MSA's whereas previous MLE-based methods could only analyze no more than 30 sequences. Then, the seminal model by Le and Gascuel [7], which we refer to as the LG model, efficiently incorporated site rate variation into the WAG model, allowing the LG model to efficiently learn rate matrices more accurate than WAG.

Although LG made it possible to estimate MLE rate matrices, it is still computationally bottle necked by the numerical optimization step which estimates the rate matrices from the given phylogeny. This is because evaluating $P(D|T, Q) = \prod_i P(D_i|T_i, Q)$ requires the marginalization of ancestral states of each tree, which is typically done using Felsenstein's pruning algorithm [3]. Thus, any optimization algorithm will be bottle necked by this step. To speed this up, CherryML [11] showed that a composite likelihood can be used in lieu of the full joint likelihood and that optimizing for the maximum composite likelihood estimate (MCLE) rate matrix produces a rate matrix that's just as good as the MLE rate matrix. To do this, rather than evaluating $P(D|T, Q)$, CherryML evaluates

$$P(D|T, Q) \approx P(D|C, Q) \tag{1.1}$$

$$\approx \prod_i \prod_{(x,y)\in C_i} P(x|y, t_{x,y}, r, Q) \tag{1.2}$$

Here, the $C = (C_1, C_2, ..., C_{\frac{n}{2}})$'s are the set of cherries in their respective trees T_i. A cherry is two leaves which share the same parent, $t_{x,y}$ is the distance between the leaves x and y in the tree, and r is the site rates. Cherries are computed by recursively finding then pruning cherries from the subtrees of T until there's at most one leaf not included in a cherry. By decomposing the full joint likelihood of each MSA into the composite likelihoods of individual cherries, CherryML eliminates the need to estimate ancestral sequences when estimating Q. This massively speeds up the rate matrix estimation step.

Thus, the current most efficient method of estimating the MLE rate matrix from data comes from first approximating MLE phylogenetic trees, then using the cherries of those trees to approximate the joint likelihood of the data when numerically optimizing Q. The original CherryML paper uses FastTree 2.1 to estimate trees and composite likelihood to estimate the rate matrix. We refer to this method as CherryML 1.0. Since CherryML 1.0 significantly sped up the rate matrix estimation step, the bottleneck of it is now estimating the MLE phylogenetic trees. Even when using FastTree [10], CherryML reported that the tree estimation step accounts for over 90% of the end-to-end runtime.

Thus in this paper, we leverage the composite likelihoods used by CherryML to completely forego the tree estimation step. We instead estimate the MCLE set of cherries, their associated divergence times and site rates. These can then be directly used by CherryML to estimate Q. Further, we show that parsimony can be used to accurately estimate the set of cherries in lieu of composite likelihood. The resulting method is linear or almost linear in the number of sequences, length of sequences, and number of rates used. We title this new method CherryML 2.0 and demonstrate that it estimates rate matrices of comparable accuracy to CherryML 1.0 whilst being multiple orders of magnitudes faster.

Chapter 2

Results

CherryML 2.0 uses a divide and conquer algorithm to estimate the MCLE cherries based on maximum parsimony. Maximum parsimony attempts to pair cherries in a way to minimize the number of mutations across sites across all cherries. By directly estimating a set of cherries and divergence times, our method forgoes the expensive ancestral state reconstruction that is typically used during phylogeny estimation. We reproduce the figures from CherryML [11] to demonstrate the accuracy and speedups of MCLE cherries versus estimating phylogenetic trees. For each figure, we run 4 iterations of the end-to-end estimation to ensure convergence; although, we note that 2 or 3 iterations is typically enough in practice. We observe that on all datasets, our methods achieve comparable accuracy to CherryML 1.0 whilst being hundreds of times faster. Additionally, we include randomly selected nodes on all plots as a baseline for performance. Further, we ablate our method by injecting ground truth site rates and divergence times. We observe that choice of cherries does not significantly impact the quality of the estimated rate matrix; however, it does affect the quality of the estimated divergence times and site rates.

For each figure, we additionally show results for performing CherryML 2.0 whilst optimizing for likely cherries, rather than parsimonious cherries. We note that this doesn't make a significant different on any figure aside from Figure 2.5 in which ground truth site rates and branch lengths are injected. Thus in general, CherryML 2.0 using parsimony is preferable to using likelihood due to the time complexity of pairing with likelihood. However, this caveat does not apply in the absence of site rate variation. See Appendix 5.6 for applying CherryML 2.0 to the WAG model.

2.1 Simulated Data

We compare the median relative error of the estimated matrices across the estimation strategies on simulated data. Data is simulated the same way as the data for Figure 1b,c for CherryML, except we simulated the data with 4 rate categories. The Ground Truth and FastTree methods are taken directly from CherryML. The "Ground Truth" method directly

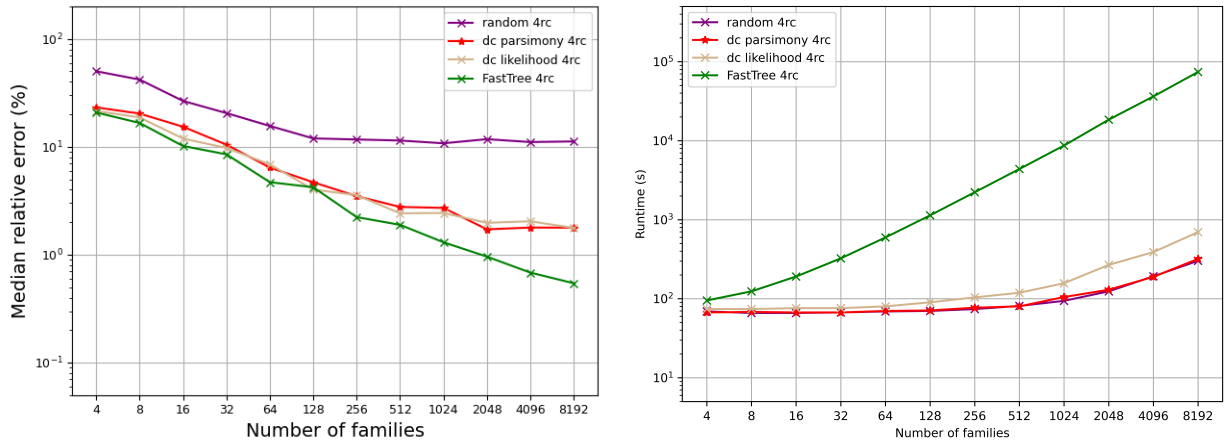


Figure 2.1: CherryML 2.0 has roughly half the statistical efficiency as CherryML 1.0 for small datasets and a bias of around 1 to 2 percent.

injects the the ground truth trees into the CherryML rate matrix estimator and is the theoretically optimal tree estimator. The FastTree method uses FastTree 2.1 to estimate trees and CherryML to estimate rate matrices, and this is what we refer to as CherryML 1.0.

CherryML 2.0 is labelled labelled "dc parsimony" which uses the divide and conquer algorithm for optimizing for parsimonious cherries. Additionally, we also include randomly selected cherries as a baseline for how well any cherry estimator should perform. We observe that selecting parsimonious cherries leads to an estimated rate matrix which has a 1 to 2 percent relative error compared to the ground truth matrix. For smaller datasets, it has roughly half the statistical efficiency as CherryML 1.0. However, CherryML 2.0 is hundreds of times faster end-to-end. Additionally, CherryML 2.0 performs significantly better than our random baseline, indicating that parsimonious cherries is a reasonable method for estimating the MCLE cherries, divergence times, and site rates.

2.2 Real Data

Next, we reproduced Figure 1e from CherryML [11] comparing our methods with CherryML 1.0 in our Figure 2.2. We train and test these methods on the Pfam dataset used by Le and Gascuel [7] and CherryML with a 3912/500 split of training data and test data. We perform 4 iterations of the cherry and rate matrix optimization initialized with the uniform rate matrix. Consistent with the simulated data, our methods perform comparably to CherryML 1.0 whilst being multiple orders of magnitude faster.

Next, we applied these same methods to the QMaker dataset [8] in Figure 2.3 using the same train/test split as CherryML 1.0. Here, we use 4 iterations for each method and observe

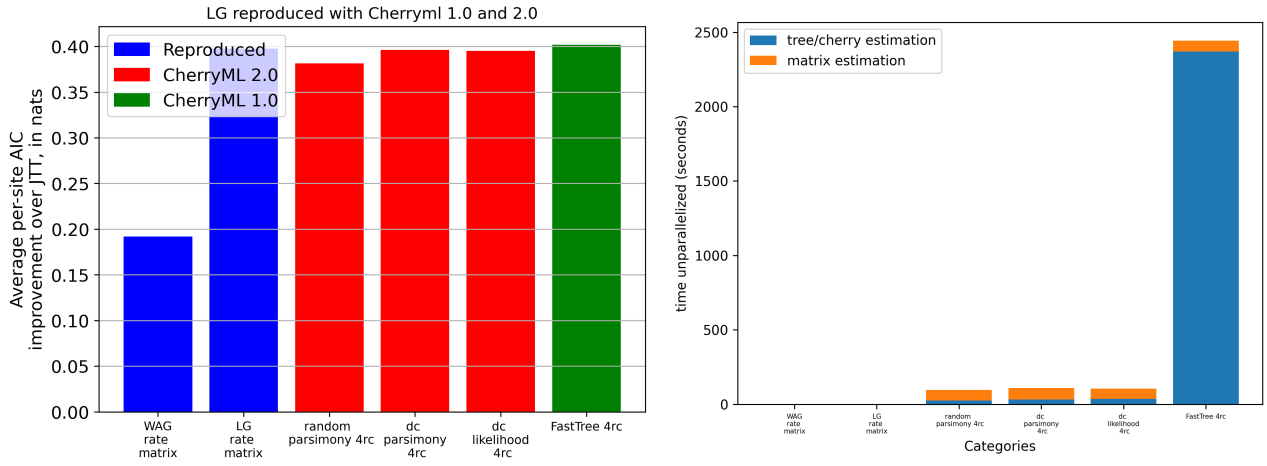


Figure 2.2: Compares the quality of the rate matrix estimated under the LG model between CherryML 1.0 and CherryML 2.0. This uses the same evaluation protocol of Figure 1e from CherryML 1.0 [11].

similar results to the above PFam datset. We observe that maximum parsimony estimates sufficiently accurate cherries for each dataset, and our method is not adversely affected by the presence of missing data.

2.3 Importance of Selecting Cherries

Similar to section 2.1, we compare the median relative error of the estimated matrices across the estimation strategies on simulated data. For CherryML 2.0 and the random cherries, we inject the ground truth divergence and site rates in Figure 2.5. We observe that when these ground truth values are injected, the parsimony based pairer is biased and returns a matrix with an error of 2%. Meanwhile, picking random cherries is unbiased when given ground truth site rates and divergence times. Combined with section 2.1, we conclude that our parsimony based pairer is biased; however it returns better rate matrices in practice since those cherries can be used to accurately infer divergence times and site rates. Meanwhile, despite being unbiased, random cherries are insufficient for estimating accurate rates and times, leading inferior performance on all other metrics.

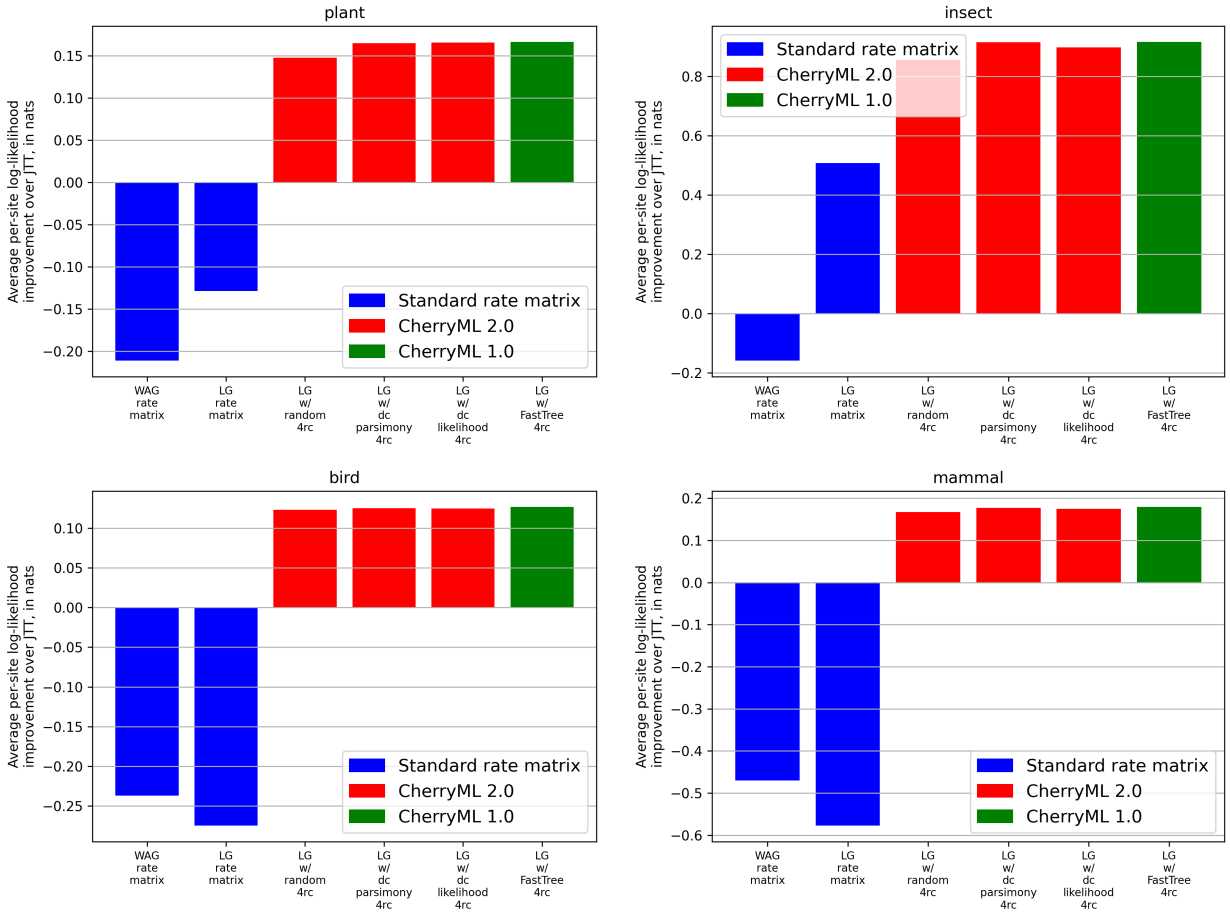


Figure 2.3: CherryML 2.0 reproduces rate matrices of similar quality to CherryML 1.0 across all datasets.

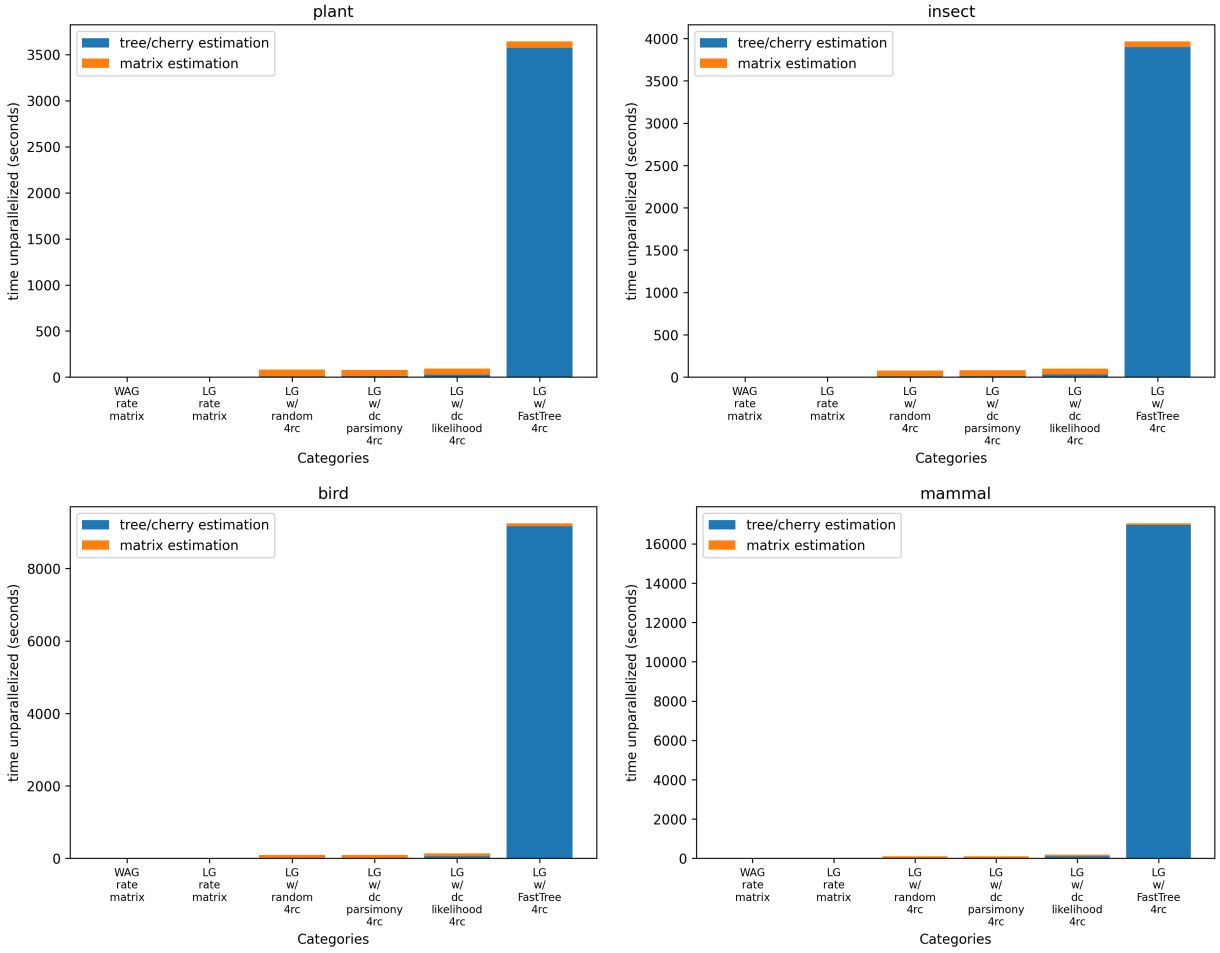


Figure 2.4: Runtimes for Figure 2.3

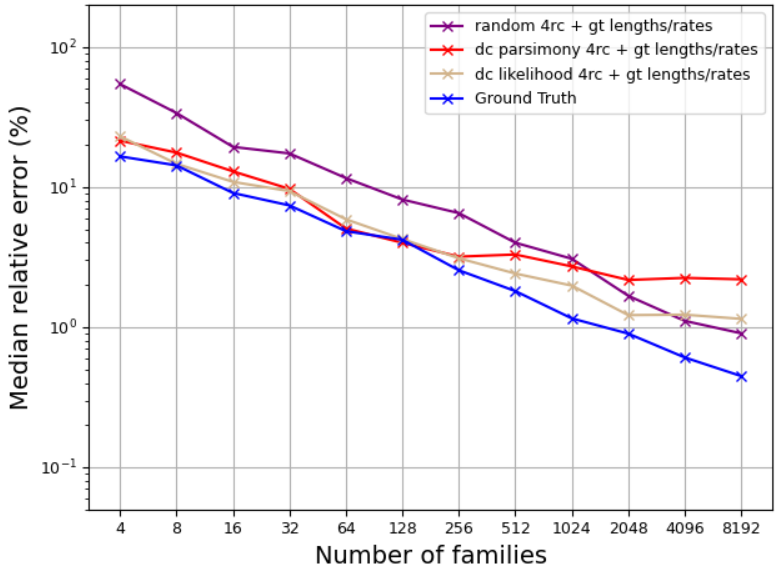


Figure 2.5: CherryML 2.0 is shown to be biased due to the use of parsimony.

Chapter 3

Methods

CherryML 1.0 uses the cherries of estimated MLE trees to estimate the MLE rate matrix. These cherries are essentially pairs of sequences with some distance defined based on the branch lengths of the estimated trees. Thus, our new method uses CherryML to optimize the rate matrix optimization; however replaces the tree estimation step with two new methods: 1) pairing 2) branch length and site rate estimation. In the following sections, we first discuss our method to estimate MCLE branch lengths and site rates. Then, we explain the pairing algorithms used to estimate the MCLE cherries.

3.1 Estimating Branch Lengths and Site Rates

We estimate the divergence times of cherries and the site rates via coordinate ascent. We first use parsimony to initialize a set of site rates, then we alternately optimize the divergence times given the rates and optimize the rates given the times. For a given MSA, this process takes $O(nls + l \log l + r + k \cdot (nl \log b + rl))$ time where n is the number of sequences, l is the length of sequences, s is the number of characters, r is the number of rate categories, and k is the maximum number of iterations of coordinate ascent.

3.2 Branch Lengths

Similar to CherryML 1.0, we use zeroth order optimization to estimate the MLE divergence time for each cherry. We use a set of $b = 129$ geometrically spaced points centered at 0.03 with a spacing of 1.1. Then for a given cherry (x, y) find the the quantized time point satisfying

$$\hat{t}_{x,y} = \arg\max_t P(x|y,t,r,Q)P(y|x,t,r,Q) \tag{3.1}$$

$$= \arg\max_t \log\left[P(x|y,t,r,Q)\right] + \log\left[P(y|sx,t,r,Q)\right] \tag{3.2}$$

Since sites are treated as independent, these log likelihoods can be efficiently evaluated by precomputing the matrix exponentials across all time points and site rates, then reusing these matrices across all MSA's and iterations of the algorithm. Concretely, we compute the transition matrices Q where $Q[t, r, i, j]$ yields the probably of i mutating to j over time t and a site rate of r. We perform this using the `r8mat_expm1` function from [9].

This optimization problem can be solved in $O(b)$ iterations by simply evaluating the likelihood at each time point. However, we empirically observe that the likelihood function is quasiconvex. Thus, the global maximum of this discrete function can be found in $O(\log b)$ using a binary search like algorithm.

The runtime to pre-compute the matrix exponential is negligible, amortized across all MSA's, and estimating divergence times for a given cherry takes $O(l \log b)$ time where l is the length of the sequences, and $O(nl \log b)$ for an MSA.

3.3 Initializing Site Rates

We perform the following parsimony based procedure to estimate its site rate. First for all sites, we count the number of mutations across all pairs of sequences in the MSA. Next, we sort the sites in increasing order of mutations. Finally, we partition the sorted sites into rate categories where the bin sizes follow a discrete gamma distribution with shape = 3 and scale = 1/3. Figure 3.1 shows how to compute the size of each bin given the fixed rates of the CAT model. We take the geometric means of neighboring rates, and the bins size of each discrete rate is the area under the gamma PDF between the previous and next means. See Appendix section 5.3 for details.

3.4 Optimizing Site Rates

Like FastTree [10], we use the CAT model of site rate variation. Under the CAT model, we use r discrete rate categories which are geometrically spaced between $\frac{1}{r}$ and r. Each site is assigned to a rate category via Bayesian estimation, where the priors distributed as a gamma distribution with shape=3 and scale=1/3. Specifically, if r_i is a rate category and $f(x)$ is the pdf of the described gamma distribution, then $f(r_i)$ yields the prior for rate r_i. Like FastTree, we also rescale branch lengths such that the mean rate becomes 1.0.

We optimize each site rate individually by evaluating the composite likelihood of a site across all cherries. Specifically for the rate at site i, we optimize the following function.

$$\hat{r_i} = \arg\max_{r_i} \left[f(r_i) \cdot \prod_{(x,y)\in C} P(x_i|y_i, t_{x,y}, r_i, Q) P(y_i|x_i, t_{x,y}, r_i, Q) \right] \tag{3.3}$$

$$= \arg\max_{r_i} \left[\log\left[f(r_i)\right] + \sum_{(x,y)\in C} \log\left[P(x_i|y_i, t_{x,y}, r_i, Q)\right] + \log\left[P(y_i|x_i, t_{x,y}, r_i, Q)\right] \right] \tag{3.4}$$

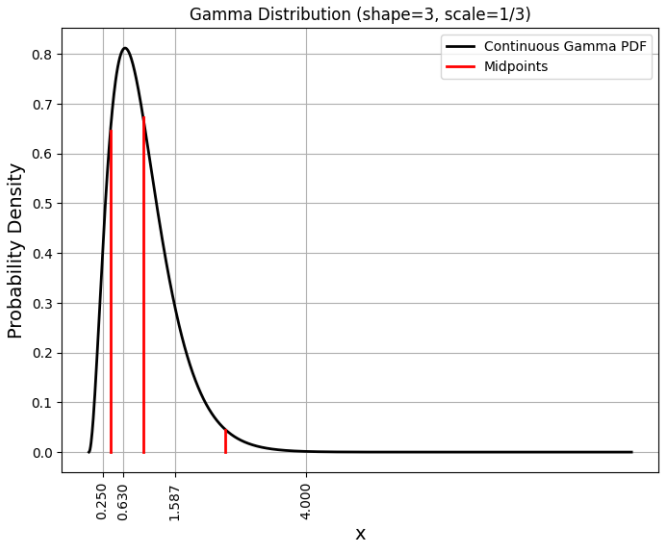


Figure 3.1: Assuming $r = 4$, we use a discrete gamma distribution with 4 rates. To compute the likelihood of each discrete rate, partition the gamma pdf into 4 bins defined by the geometric means of the rates. The likelihood of each rate is the area under the gamma's pdf for its bin.

Similar to optimizing the divergence times, optimal rate for a given site can be found in $O(\log r)$ time via binary search. However, since r is typically quite small, we implemented this by evaluating the composite likelihood for each rate and choosing the most likely rate. Thus, our implementation takes $O(rl)$ to optimize site rates; however, this can be improved to $O(l \log r)$ via binary search if desired.

3.5 Estimating Cherries

Now we propose an exact algorithm for estimating a set of likely cherries and propose optimizations in the following section.

Since each sequence can belong to at most one cherry, we can find an optimal set of cherries via a reduction to the minimum-weight perfect matching problem. The minimum-weight perfect matching problem, given a weighted graph, attempts to find a perfect matching whilst minimizing the weight of the matching. Our reduction works as follows: create a complete graph where nodes represent sequences and edge weights represent the negative log likelihood that those two nodes form a cherry. Then, find the minimum-weight perfect matching and return the edges as cherries. In the case of an odd number of sequences, add an additional node with distance negative infinity to all other nodes. This problem can be solved in $O(n^4)$ on a complete graph, where n is the number of nodes. Efficient implementations of this

algorithm can be found in Blossom V [6] and NetworkX [1].

However, this approach is slow for large MSA's and is asymptotically slower than FastTree with respect to the number of sequences. In fact, any matching algorithm, approximate or exact, will require require at least quadratic time in the number of sequences, as the pairwise distances between all sequences must be computed. Additionally, computing the negative log likelihood can take a significant amount of time, see section 3.6.

Thus, in order to create an almost linear time algorithm which achieves a substantial speedup over CherryML 1.0, we propose the following divide and conquer algorithm for estimating MCLE cherries. Additionally, we optimize our cherries for maximum parsimony rather than log likelihood for additional speedups.

3.6 Estimating MCLE Cherries

To estimate the MCLE set of cherries, we optimize the following composite likelihood function.

$$\arg\max_C P(D|C,Q) \approx \arg\max_C \prod_{(x,y)\in C} P((x,y) \text{ is a cherry}) \tag{3.5}$$

$$\approx \arg\max_C \prod_{(x,y)\in C} \max_{t,r}[P(x|y,t,r,Q)P(y|x,t,r,Q)] \tag{3.6}$$

$$= \arg\max_C \sum_{(x,y)\in C} \max_{t,r}[\log(P(x|y,t,r,Q)) + \log(P(y|x,t,r,Q))] \tag{3.7}$$

$$= \arg\min_C - \sum_{(x,y)\in C} \max_{t,r}[\log(P(x|y,t,r,Q)) + \log(P(y|x,t,r,Q))] \tag{3.8}$$

We approximate the probability that (x, y) is a cherry as $\max_t P(x|y,t,r,Q)P(y|x,t,r,Q)$ since it computes the probability that two sequences can mutate to each other over some fixed amount of time. This allows us to optimize over the same composite likelihood during the cherry estimation phase as the matrix estimation phase. Thus, we can use the procedure in section 3.1 to find the optimal divergence times and site rates, which can then be used to compute the log likelihood.

One obvious issue arises with this idea. Estimating the likelihoods requires optimizing for divergence times and site rates; however, our procedure for this requires us to first estimate the set of cherries. Thus to avoid this issue whilst still being able to efficiently estimate the likelihoods, we simply use the initial site rates as defined by section 3.3 to estimate the divergence times described in section 3.2 without performing coordinate ascent.

3.7 Using Parsimony over Likelihood

Due to the cost of estimating MCLE cherries, we instead estimate a set of cherries based on parsimony. Although optimizing composite likelihood (section 3.6) may be a more natural way to optimize cherries, we found that in practice this makes no difference whilst being significantly slower. To implement this process for cherries, we simply estimate cherries which minimize the hamming distance amongst all pairs of cherries. We exclude sites with missing data when computing the hamming distance, and we normalize the hamming distances. This prevents pairs with missing data from being always being significantly closer than pairs without missing data.

3.8 Divide and Conquer Pairing

The divide and conquer pairer is motivated by the following heuristic: sequences which greatly differ are highly unlikely to be a cherry in the MLE tree whilst similar sequences are. Thus, we can recursively partition the sequences into sets such that sequences in the same set are likely to be cherries, whilst pairs of sequences across two sets are not. The details of this pairer are as follows. The distance function used should be the normalized hamming distance between sequences. This allows the algorithm to optimize for parsimonious cherries.

The worst case runtime of this algorithm is $O(n^2 p)$ where n is the number of sequences and p is the complexity of computing the distance between two sequences. In practice, we observe that divide and conquer significantly outperforms the $\Omega(n^2)$ matching based algorithms described the appendix. Assuming that the sets are not consistently uneven, we do expect it to run in $O(n \log(n) p)$ time. Note that this algorithm can be modified to work in expected $O(n \log n)$ time. We briefly describe this method in the appendix 5.1; however, we don't use it for CherryML 2.0 as it offers very little speed boosts whilst significantly affecting the accuracy of the method.

Algorithm 1 Cherry Pair Finding Algorithm

```
1: function GET_CHERRIES(S)                          ▷ S is the set of sequences
2:     x, y ← FIND_DISTANT_PAIR(S)
3:     X, Y ← PARTITION(S, x, y)
4:     Pairs_x, Unpaired_x ← GET_CHERRIES(X)
5:     Pairs_y, Unpaired_y ← GET_CHERRIES(Y)
6:     Pairs ← Pairs_x ∪ Pairs_y
7:     if Unpaired_x ≠ {} and Unpaired_y ≠ {} then
8:         return Pairs, Unpaired_x ∪ Unpaired_y
9:     end if
10:    return Pairs, {}
11: end function
12: function FIND_DISTANT_PAIR(S)
13:    x ← random sequence from S
14:    y ← farthest sequence from x
15:    z ← farthest sequence from y
16:    return y, z
17: end function
18: function PARTITION(S, x, y)
19:    X ← ∅, Y ← ∅
20:    for s ∈ S do
21:        if s is closer to x than y then
22:            X ← X ∪ {s}
23:        else
24:            Y ← Y ∪ {s}
25:        end if
26:    end for
27:    return X, Y
28: end function
```

Chapter 4

Discussion

We have presented CherryML 2.0, which is able to accurately reconstruct rate matrices of phylogenetic model in almost linear time. The runtime of the cherry estimation step is $O(nl\log(n)\log(b))$. This, combined with the $O(nl\log(b) + gbs^3)$ complexity of the matrix estimation step yields an end-to-end complexity of $O(nl\log(n)\log(b)+gbs^3)$, which is almost linear in the length of sequences, number of sequences per MSA, and number of rate categories. We showed that this method is hundreds of times faster than CherryML 1.0 whilst still being able to accurately reconstruct rate matrices. Further, we showed that parsimony is a sufficiently accurate metric to estimate a good set of cherries.

4.1 Future Work

Improving Statistical Efficiency

One immediate improvement that can be performed to improve the performance of CherryML 2.0 is to include more pairs of sequences in the composite likelihood, even if they no longer form a set of cherries in any phylogenetic tree. Although composite likelihood is known to be unbiased, it is not guaranteed to be accurate for small MSA's. In the case of estimating divergence times and site rates, a larger number of pairs may be necessary to accurately estimate them. We conclude this as using random cherries with ground truth lengths and rates is unbiased, yet random cherries with estimated lengths and rates is biased. We suspect that using some almost linear number of pairs would be sufficient for this purpose. Additionally, we suspect that this idea can be used to increase the statistical efficiency of CherryML 1.0 without significantly impacting its runtime.

This process must involve weighting the resulting pairs, however, to prevent the double counting of transitions. For example, pairs can be weighted based on their topological distance in some phylogenetic tree. Hence, more distant pairs are weighted less than the true cherries. Alternatively, individual sites of each pair can also be weighted based on how common that amino acid is in that subtree. This prevents transitions between the left and

right subtrees at a node from being double counted.

Models of Site Rate Variation

Given our results with the CAT model, we suspect that composite likelihood may also be used to efficiently estimate site rates under other models. For example, the discrete gamma model used by [4] is typically considered more accurate than the CAT model; however, it is significantly slower to compute. We have some preliminary results along this line, which shows that using the CAT model with different priors can result in better estimated rate matrices. See appendix section 5.4 for details.

Improving Space Complexity

Additionally, we can reduce the space complexity of the program by modifying the way we precompute the transition matrices. Rather than computing $Q[t, r, i, j]$, we instead compute $Q[x, i, j]$ where x is the divergence time. In order to get the probability of a mutation time from i to j over time t and site rate r, simply binary search the nearest discrete time point for $t \cdot r$ and use the probability $Q[t \cdot r, i, j]$. This approach may also speed up the program when r is large as $Q[t, r, i, j]$ may be too big to fit in memory.

Chapter 5

Appendix

5.1 Guaranteed Almost Linear Time Divide and Conquer

We can make the divide and conquer algorithm guaranteed $O(nl \log n \log b)$ with the following modification to the divide and conquer algorithm. Rather than partitioning all sequences based on whether they are more likely to form a cherry with x or y, first compute the difference in likelihood for each sequence with x and for each sequence with y. Then use the quickselect algorithm to find the median of this list of differences. Partition the sequences based on if the difference is greater than or less than the median. This scheme guarantees 50/50 splits, thus guaranteeing an $O(nl \log n \log b)$ time complexity. This idea can also be modified to guarantee any arbitrary splits where the smaller split is some fraction of n. In practice, we observed that this algorithm degrades the accuracy of the resulting rate matrix without substantially improving the time complexity; thus, we did not include this method in the main paper.

5.2 Quadratic Time Matching Based Algorithm for Finding Cherries

A greedy algorithm can be used to approximately match the most likely cherries. The greedy pairing scheme greedily pairs the most likely unpaired pair of sequences until all sequences are paired except for at most one. To implement this pairer, simply compute all pairwise distances among the sequences. Push all pairs of sequences into a heap, ordered by their distance. Then repeatedly pair and remove all unpaired pairs from the heap until the heap is empty or all sequences are paired.

For a given MSA, this algorithm runs in $O(n^2(l \log b + \log n))$ time and $O(n^2 l)$ space. In practice, this method runs substantially better than FastTree on many MSA's since this method has little to no overhead compared to FastTree.

5.3 Initial Site Rate Details

Algorithm 2 Estimate Site Rate

```
function EstimateSiteRate(MSA)
    mutations ← CountMutations(MSA)                    ▷ Count mutations at each site
    sorted_sites ← SortSites(mutations)                ▷ Sort sites by increasing mutations
    rates, weights ← ComputeRatesAndWeights            ▷ Compute rate categories and weights
    partitions ← PartitionSites(sorted_sites, weights) ▷ Partition sites into rate categories
    return partitions
end function
function CountMutations(MSA)
    mutations ← empty list
    for i ← 1 to length(MSA) do                        ▷ For each site
        count ← empty dictionary
        total ← 0
        for seq in MSA do                              ▷ Count amino acids at site i
            aa ← seq[i]
            if aa is not missing then
                count[aa] ← count[aa] + 1
                total ← total + 1
            end if
        end for
        site_mutations ← 0
        for aa, cnt in count do
            site_mutations ← site_mutations + cnt × (total − cnt)
        end for
        mutations.Append(site_mutations)
    end for
    return mutations
end function
function SortSites(mutations)
    sorted_sites ← Sort(mutations)                     ▷ Sort sites by increasing mutations
    return sorted_sites
end function
function ComputeRatesAndWeights
    rates ← list of r geometrically spaced points between 1/r and r
    midpoints ← list of geometric means between neighboring rates
    weights ← empty list
```

```
    shape ← 3, scale ← 1/3                          ▷ Parameters for gamma distribution
    for i ← 0 to length(rates) do
        if i = 0 then
            w ← GammaPDF(0, shape, scale)           ▷ Area between 0 and midpoints[0]
        else if i = length(rates) then
            w ← 1 − GammaCDF(midpoints[i − 1], shape, scale)   ▷ Area after midpoints[i − 1]
        else
            w ← GammaCDF(midpoints[i], shape, scale) − GammaCDF(midpoints[i − 1], shape, scale)
        end if
        weights.Append(w)
    end for
    return rates, weights
end function
function PartitionSites(sorted_sites, weights)
    partitions ← empty list
    start ← 0
    for i ← 0 to length(weights) do
        end ← start + ⌊weights[i] × length(sorted_sites)⌋
        partitions.Append(sorted_sites[start : end])
        start ← end
    end for
    return partitions
end function
```

5.4 Different Models of Site Rate Variation

Rather than using the pdf of a gamma distribution with shape=3 and scale=1/3 as our priors, we instead create a discrete gamma distribution with r discrete rates. We then use the likelihoods given by the discrete gamma distribution as our priors. These are identical to the priors used in the site rate initialization step described in Sections 3.3 and 5.3. We see that this method is slightly more statistically efficient and slightly less biased than our existing CherryML 2.0 whilst having the same runtime.

5.5 Extended Figures

We recreate the main figures of the paper here with the greedy and blossom based cherry estimators included. Additionally, we demonstrate the accuracy of estimating cherries based

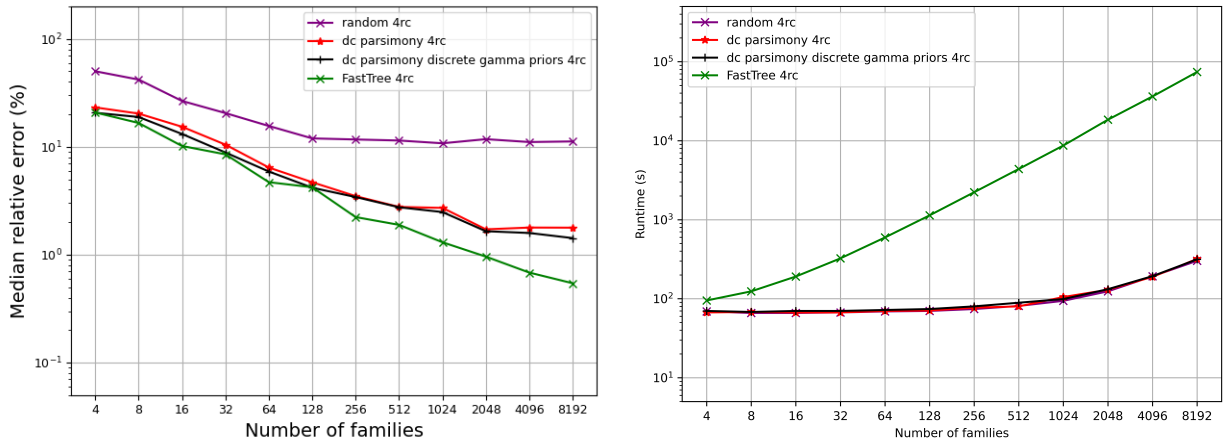


Figure 5.1: Reproduces Figure 2.1 and includes a new cherry estimator in black. This estimator uses discrete gamma priors rather than gamma priors to estimate site rates.

on the likelihoods of the cherries as described in section 3.6. See Figures 5.2, 5.4, 5.5, and 5.6.

We observe that the greedy and blossom pairer are comparable in accuracy our method, albeit with a slight impact to the performance and poorer scalability for large MSA's. Similarly, we observe that pairing with log likelihood results in comparable accuracy to parsimony whilst being a few times slower. Additionally, although likelihood based pairing returns less biased cherries, in practice they don't result in more accurate rate matrices.

5.6 CherryML 2.0 Under WAG

In the absence of site rate variation (i.e. the WAG model), it is preferable to estimate cherries using the likelihood of cherries rather than parsimony (figures 5.9, 5.8). We suspect the cause for this difference is that site rate variation significantly increases the degrees of freedom of the model, hence making the pairing step less important. In the absence of site rate variation, estimating cherries with likelihood results in significantly better cherries, which improves the estimated rate matrix. This is especially important in the presence of missing data, as likelihood based pairing can perform significantly better. This can be observed in their relative performance on the insect dataset, which is known for having high amounts of missing data (Figure 5.7).

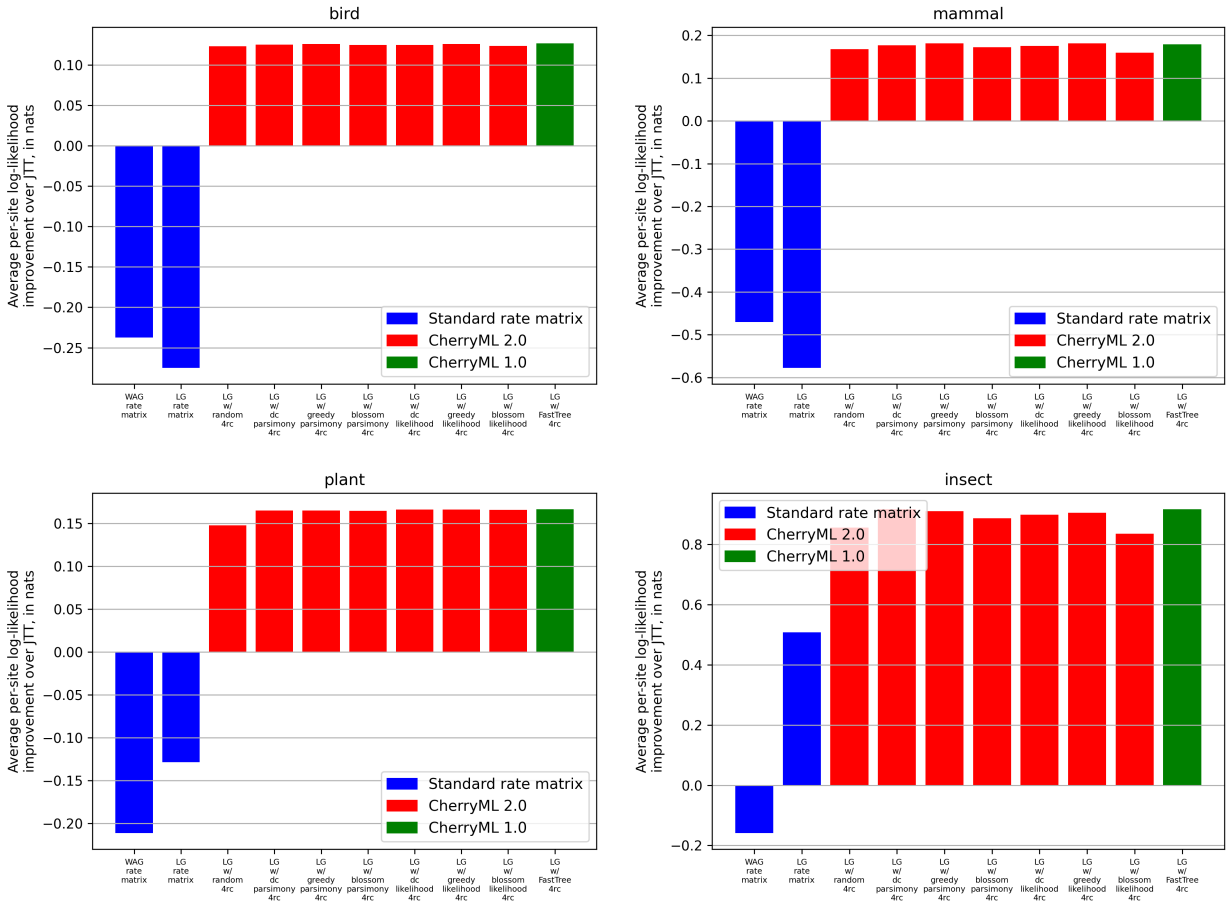


Figure 5.2: Reproduces Figure 2.3 with the other combinations of pairing with divide and conquer, greedy, and blossom optimizing for parsimony and likelihood.

5.7 Quality of Estimated Site Rates

We additionally made heat maps comparing the site rates derived by CherryML 2.0 against the ground truth site rates on simulated data in Figure 5.10. For these plots, data was simulated with 4 rate categories and CherryML 2.0 also used 4 rate categories. We see that a vast majority of the rates lie on or near the line $y = x$ showing that our method also accurately estimates site rates.

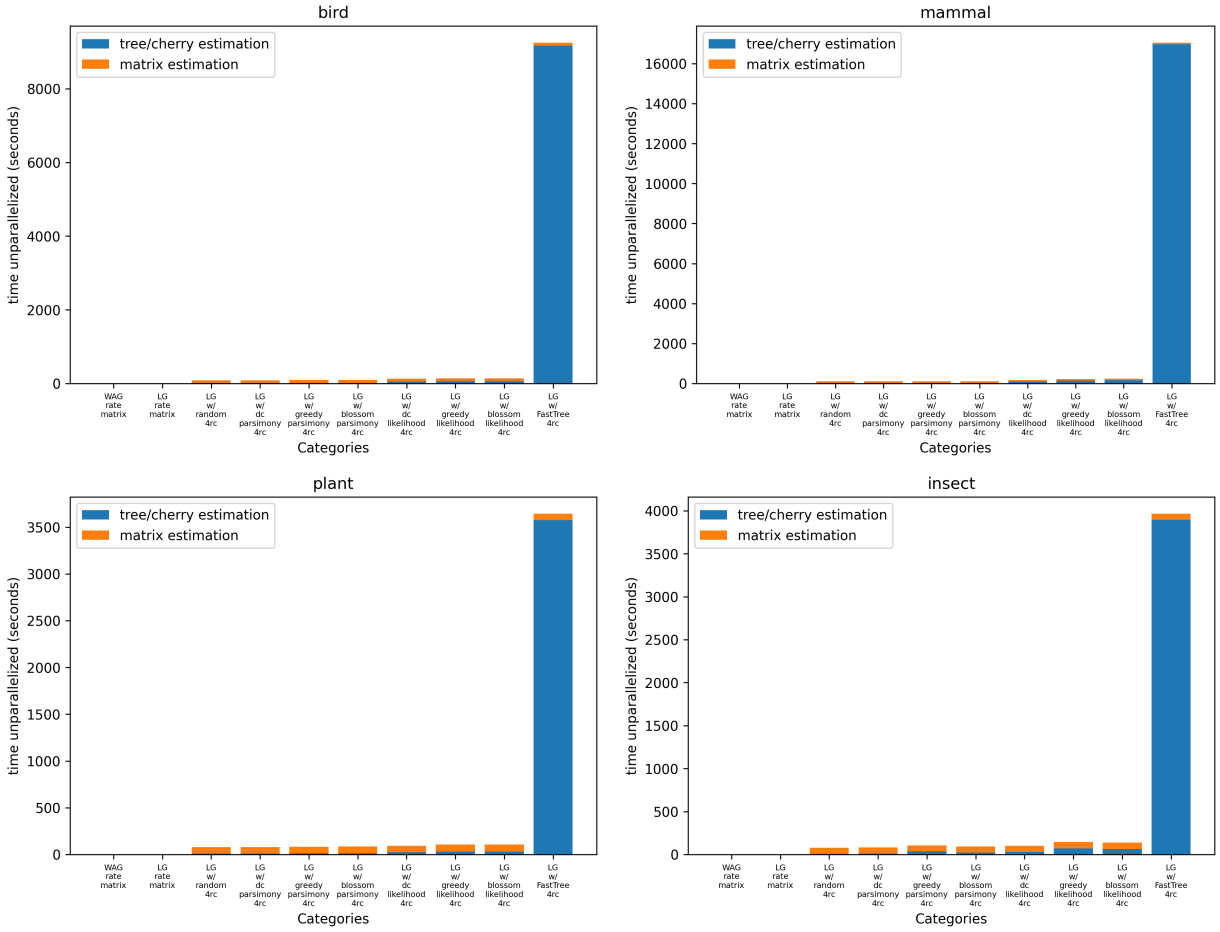


Figure 5.3: Runtimes for all methods benchmarked in Figure 5.2.

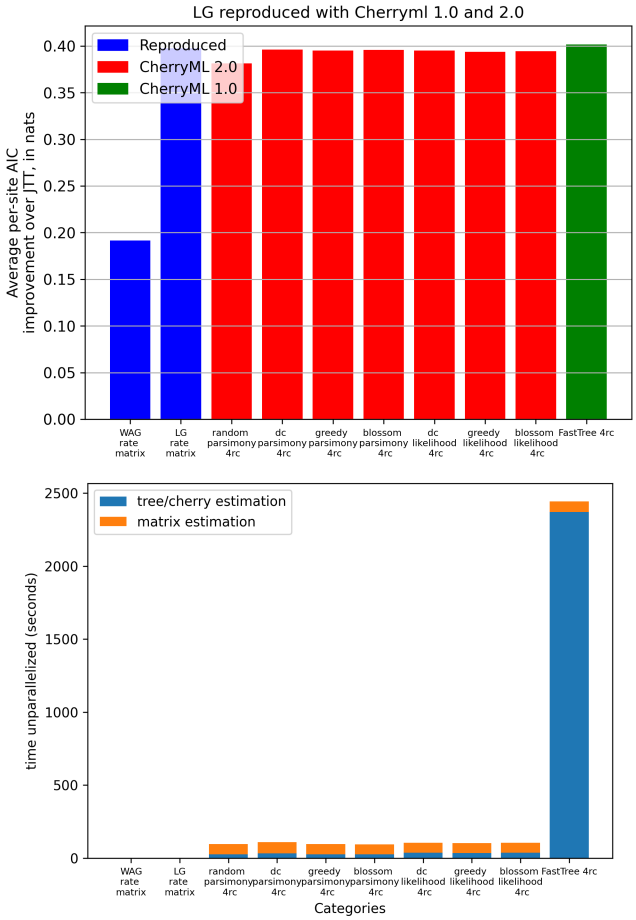


Figure 5.4: Reproduces Figure 2.2 with the other combinations of pairing with divide and conquer, greedy, and blossom optimizing for parsimony and likelihood.

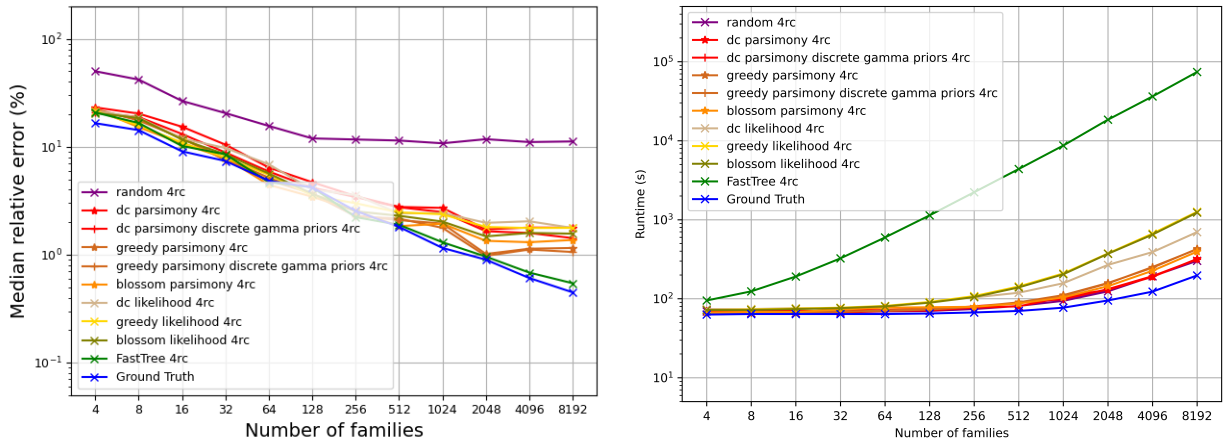


Figure 5.5: Reproduces Figure 2.1 with the other combinations of pairing with divide and conquer, greedy, and blossom optimizing for parsimony and likelihood.

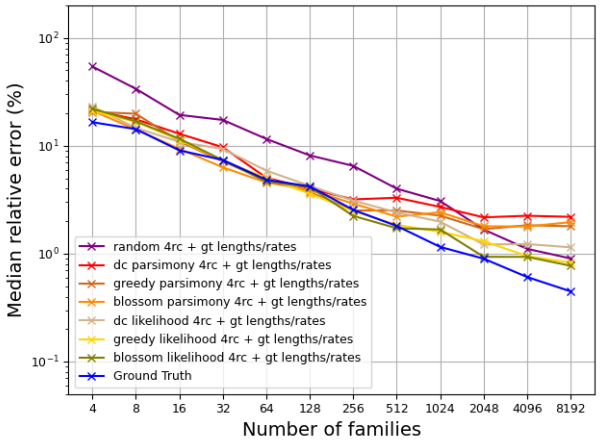


Figure 5.6: Reproduces Figure 2.5 with the other combinations of pairing with divide and conquer, greedy, and blossom optimizing for parsimony and likelihood.

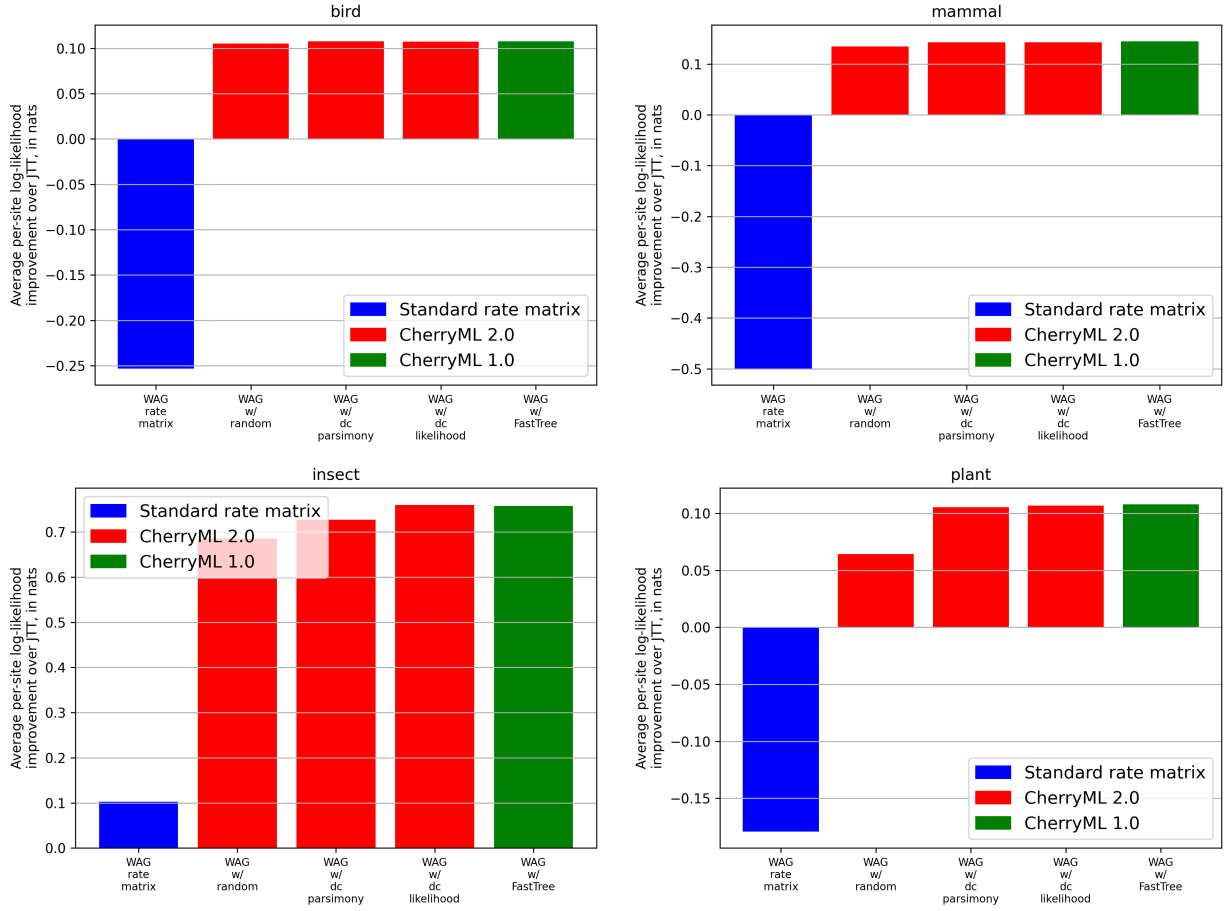


Figure 5.7: Reproduces Figure 2.3 under the WAG model. This is done by using only 1 rate category.

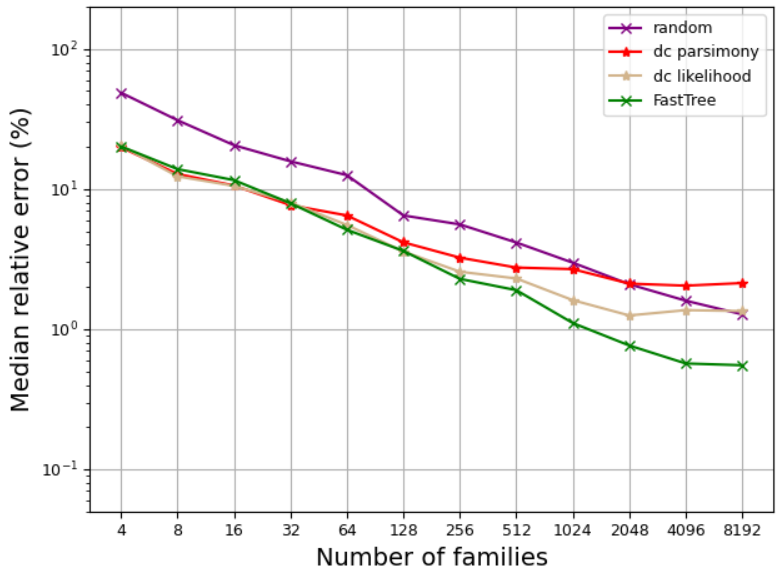


Figure 5.8: Reproduces Figure 2.1 under the WAG model. This is done by using only 1 rate category.

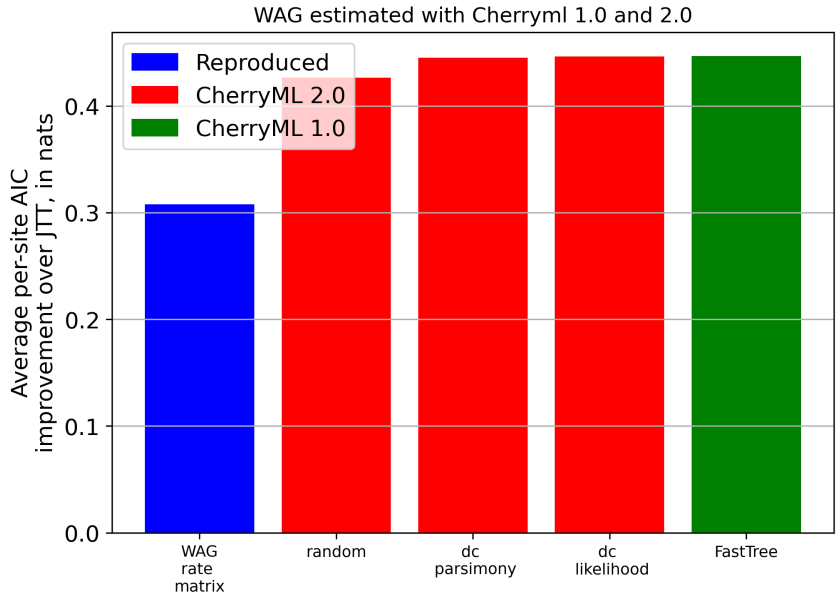


Figure 5.9: Reproduces 2.2 under the WAG model. This is done by using only 1 rate category.

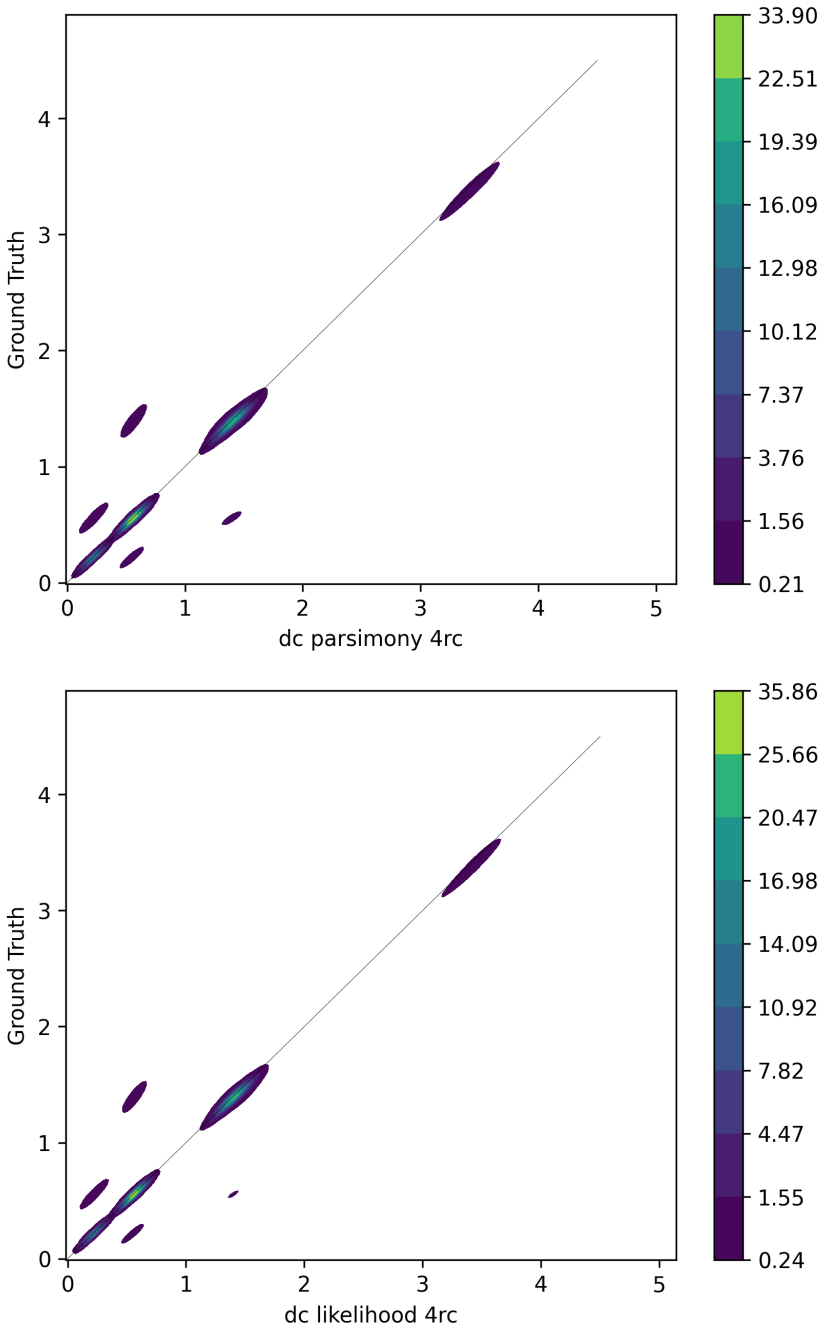


Figure 5.10: Compares the estimated site rates of CherryML 2.0 with parsimony against ground truth rates and CherryML 2.0 with likelihood against ground truth rates.

Bibliography

[1] Daniel A. Schult Aric A. Hagberg and Pieter J. Swart. "Exploring network structure, dynamics, and function using NetworkX". In: *Proceedings of the 7th Python in Science Conference (SciPy2008)* (2008).

[2] M. O. Dayhoff and R. M. Schwartz. "Chapter 22: A model of evolutionary change in proteins". In: *in Atlas of Protein Sequence and Structure*. 1978.

[3] Joseph Felsenstein. "Maximum Likelihood and Minimum-Steps Methods for Estimating Evolutionary Trees from Data on Discrete Characters". In: *Systematic Biology* 22.3 (Sept. 1973), pp. 240–249. ISSN: 1063-5157. DOI: 10.1093/sysbio/22.3.240. eprint: https://academic.oup.com/sysbio/article-pdf/22/3/240/4741566/22-3-240.pdf. URL: https://doi.org/10.1093/sysbio/22.3.240.

[4] Stéphane Guindon et al. "New Algorithms and Methods to Estimate Maximum-Likelihood Phylogenies: Assessing the Performance of PhyML 3.0". In: *Systematic Biology* 59.3 (May 2010), pp. 307–321. ISSN: 1063-5157. DOI: 10.1093/sysbio/syq010. eprint: https://academic.oup.com/sysbio/article-pdf/59/3/307/24207259/syq010.pdf. URL: https://doi.org/10.1093/sysbio/syq010.

[5] David T. Jones, William R. Taylor, and Janet M. Thornton. "The rapid generation of mutation data matrices from protein sequences." In: *Comput. Appl. Biosci.* 8.3 (1992), pp. 275–282. URL: http://dblp.uni-trier.de/db/journals/bioinformatics/bioinformatics8.html#JonesTT92.

[6] Vladimir Kolmogorov. "Blossom V: A new implementation of a minimum cost perfect matching algorithm". In: *Math. Prog. Comp* (2009). DOI: https://doi.org/10.1007/s12532-009-0002-8.

[7] Si Quang Le and Olivier Gascuel. "An Improved General Amino Acid Replacement Matrix". In: *Molecular Biology and Evolution* 25.7 (Mar. 2008), pp. 1307–1320. ISSN: 0737-4038. DOI: 10.1093/molbev/msn067. eprint: https://academic.oup.com/mbe/article-pdf/25/7/1307/3520981/msn067.pdf. URL: https://doi.org/10.1093/molbev/msn067.

[8] Bui Quang Minh et al. "QMaker: Fast and Accurate Method to Estimate Empirical Models of Protein Evolution". In: *Systematic Biology* 70.5 (Feb. 2021), pp. 1046–1060. ISSN: 1063-5157. DOI: 10.1093/sysbio/syab010. eprint: https://academic.oup.com/sysbio/article-pdf/70/5/1046/39679725/syab010.pdf. URL: https://doi.org/10.1093/sysbio/syab010.

[9] Cleve Moler and Charles Van Loan. "Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later". In: *SIAM Review* 45.1 (2003), pp. 3–49. DOI: 10.1137/S00361445024180. eprint: https://doi.org/10.1137/S00361445024180. URL: https://doi.org/10.1137/S00361445024180.

[10] Morgan N. Price, Paramvir S. Dehal, and Adam P. Arkin. "FastTree 2 – Approximately Maximum-Likelihood Trees for Large Alignments". In: *PLoS ONE* 5.3 (Mar. 2010). Ed. by Art F. Y. Poon, e9490. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0009490. URL: https://dx.plos.org/10.1371/journal.pone.0009490.

[11] Sebastian Prillo et al. "CherryML: scalable maximum likelihood estimation of phylogenetic models". In: *Nature Methods* 20.8 (2023), pp. 1232–1236. ISSN: 1548-7105. DOI: 10.1038/s41592-023-01917-9. URL: https://doi.org/10.1038/s41592-023-01917-9.

[12] Simon Whelan and Nick Goldman. "A General Empirical Model of Protein Evolution Derived from Multiple Protein Families Using a Maximum-Likelihood Approach". In: *Molecular Biology and Evolution* 18.5 (May 2001), pp. 691–699.