Uncommon Problems in Phylogenetic Inference

Zur Erlangung des akademischen Grades eines

Doktors der Naturwissenshaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte
Dissertation

von

Ben Bettisworth
aus Juneau, Alaska

Tag der mündlichen Prüfung:	02.03.2023
Erster Gutachter:	Prof. Dr. Alexandros Stamatakis
Zweiter Gutachter:	Prof. Dr. Arndt von Haeseler

Hiermit erkläre ich, dass ich diese Arbeit selbständig angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht habe. Ich habe die Satzung des Karlsruher Institutes für Technologie (KIT) zur Sicherung guter wissenschaftlicher Praxis beachtet.

Heidelberg, 02.03.2023

. . . . . .

(**Ben Bettisworth**)

Zusammenfassung

Die Phylogenetik ist die Lehre der Entwicklung des Lebens auf der Erde. Das Aufdecken alter evolutionärer Beziehungen zwischen lebenden Arten ist von großem Wert, da sie zu wichtigen Entdeckungen in der Biologie führte, wie beispielsweise zur Entwicklung neuer Medikamente, zur Nachverfolgung der Dynamik einer globalen Pandemie sowie zu Erkenntnissen über den Ursprung der Menschheit. Heutzutage werden phylogenetische Analysen typischerweise mit Hilfe statistischer Modelle durchgeführt, wobei Sequenzdaten, in der Regel molekulare Sequenzen, als Eingabedaten verwendet werden. Basierend auf diesen statistischen Modellen wird die wahrscheinlichste Erklärung für die Eingabedaten berechnet. Das heißt, der (vermeintlich) korrekte phylogenetische Baum ist der Baum, der gemäß eines bestimmten Modells der Sequenzentwicklung am wahrscheinlichsten ist.

Die rasche Zunahme verfügbarer Daten in den letzten Jahren ermöglicht wesentlich kompliziertere phylogenetische Analysen. Paradoxerweise hat diese massive Zunahme der für die Analyse verfügbaren Daten nicht in allen Fällen zu einer endgültigen Schlussfolgerung geführt, d. h. das verwendete Modell ist unsicher bezüglich der wahrscheinlichsten Schlussfolgerung. Dies kann auf eine Vielzahl von Faktoren zurückzuführen sein, wie beispielsweise hochkomplexe Modelle, Rauschen in einigen oder allen Daten sowie physikalische Prozesse, die durch das Modell nicht angemessen berücksichtigt werden. Schwierigkeiten aufgrund von Ungewissheit sind weder in der Phylogenetik noch in der Wissenschaft im Allgemeinen neu, doch die Entwicklung komplizierterer Analysemethoden fordert neue Methoden zur Angabe, Analyse und Integration von Unsicherheiten.

Die vorliegende Arbeit präsentiert drei Beiträge zur Verbesserung der Unsicherheitsbewertung. Der erste Beitrag betrifft die Bestimmung der Wurzel von ungewurzelten phylogenetischen Bäumen. Phylogenetische Bäume sind entweder bezüglich der Zeit orientiert, in diesem Fall nennt man sie verwurzelt, oder sie haben keine Orientierung, in diesem Fall sind sie unverwurzelt. Die meisten Programme zur Bestimmung phylogenetischer Bäume erzeugen aus rechnerischen Gründen einen ungewurzelten phylogenetischen Baum. Ich habe das Open-Source-Softwaretool `RootDigger` entwickelt, das sowohl einen ungewurzelten phylogenetischen Baum, als auch eine Verteilung der wahrscheinlichen Wurzeln berechnet. Darüber hinaus verfügt `RootDigger` über ein Parallelisierungsschema mit verteiltem Speicher, welches auch die Analyse großer Datensätze erlaubt, wie beispielsweise die Bestimmung eines phylogenetischen Baumes aus 8736 SARS-CoV-2-Virussequenzen.

Mein zweiter Beitrag in der vorliegenden Arbeit ist das Open-Source-Softwaretool `Phylourny` zur Berechnung des wahrscheinlichsten Gewinners eines Knock-out-Turniers. Der Algorithmus in `Phylourny` ist angelehnt an den Felsenstein Pruning Algorithmus, einen dynamischen Programmierungsalgorithmus zur Berechnung der Wahrscheinlichkeit eines phylogenetischen Baums. Die Verwendung dieses Algorithmus erlaubt eine erhebliche Beschleunigung der Berechnung im Vergleich zu Standard-Turniersimulationen. Mit dieser beschleunigten Methode untersucht `Phylourny` auch den Parameterraum des Modells mit Hilfe einer MCMC-Methode, um Ergebnisse zu bewerten und zusammenzufassen, die eine ähnliche Wahrscheinlichkeit des Auftretens haben. Diese Ergebnisse weichen oft erheblich vom wahrscheinlichsten Ergebnis ab. In der vorliegenden Arbeit präsentiere ich die Performanz von `Phylourny` anhand zweier realer Fußball- und Basketballturniere.

Der finale Beitrag in dieser Arbeit ist die Neugestaltung und Neuimplementierung eines bekannten Tools für historische Biogeografie, mit dem sich Rückschlüsse auf die Verteilung der angestammten Verbreitungsgebiete ziehen lassen. Ein Hauptinteresse der Biogeographie besteht in der Bestimmung der Verbreitungsgebiete von Arten. Die historische Biogeografie befasst sich daher häufig mit der Ableitung des Verbreitungsgebiets der Vorfahren lebender Arten. Diese Verteilungen des Verbreitungsgebiets der Vorfahren sind ein häufiges Ergebnis von biogeografischen Studien, die oft mit einem Modell abgeleitet werden, das zahlreiche Ähnlichkeiten mit Modellen der Sequenzevolution aufweist. Meine neue Version, `Lagrange-NG`, berechnet die Ergebnisse bis zu 50 Mal schneller als die vorherige Version und bis zu zwei Größenordnungen schneller als das beliebte analoge Tool `BioGeoBEARS`. Darüber hinaus habe ich eine neue Abstandsmetrik entwickelt, die es erlaubt Ergebnisse alternativer Tools und Algorithmen zu vergleichen.

Abstract

Phylogenetics is the study of the evolution of life on earth. Uncovering the ancient evolutionary relationships between living species is valuable, as it has led to important discoveries in biology, such as new drug formulations, tracking the dynamics of a global pandemic and insights into the origin of humanity. Nowadays, a typical phylogenetic analysis will be performed using statistical models which use as their input sequence data, generally molecular sequences, which are then used to find the most likely explanation of the data. That is, the (putatively) correct phylogenetic tree is the tree which is the most likely under some particular model of sequence evolution.

In recent years, the ability to perform substantially more complicated phylogenetic analysis has been unlocked by the rapid increase in available data. Somewhat paradoxically, this massive increase of data available for analysis has not in all cases made it possible to arrive at a definitive conclusion, that is the model used is uncertain about the most likely conclusion. This can be due to a plethora of factors, including highly complex models, noise in some or all of the data, or physical processes which are not adequately accounted for by the model. Difficulties from uncertainty are not new to phylogenetics or to science in general, however the advent of more complicated analysis has fostered to the need for new methods of reporting, analyzing, and integrating uncertainty.

This thesis presents three contributions to improve uncertainty assessment. The first contribution concerns rooting previously unrooted phylogenetic trees. Phylogenetic trees are either oriented with respect to time, in which case they are called rooted, or they lack an orientation, in which case they are unrooted. For computational reasons, most tool which infer phylogenetic trees produce an unrooted phylogenetic tree. I developed the open source software tool `RootDigger`, which can both root an unrooted phylogenetic tree, or compute a distribution of likely roots. In addition, `RootDigger` is engineered with a distributed memory parallelization scheme, which allows it analyze large datasets, such as a tree built from 8736 SARS-CoV-2 virus sequences.

My second contribution in this thesis is the open source software tool `Phylourny`, which computes the most probable winner of a knock-out tournament. `Phylourny` uses an algorithm inspired by the Felsenstein Pruning Algorithm, a dynamic programming algorithm to compute the likelihood of a phylogenetic tree, to substantially accelerate the computation of results when compared to using standard tour-

nament simulations. Using this accelerated method, `Phylourny` also explores the parameter space of the model via an MCMC method to asses and summarize the outcomes which have similar likelihood of occurring. These outcomes often diverge substantially from than the most likely outcome. I present the performance of `Phylourny` using two real football and basketball tournaments.

My final contribution in this thesis is the redesign and reimplementation of a popular historical biogeography tool to infer ancestral range distributions. A primary interest in biogeography is to determine the range of species (i.e., where a species can be found). Naturally, historical biogeography is therefore often concerned with inferring the range of ancestral species for a set of living species. These ancestral range distributions are a common result of biogeography studies, which are often inferred using a model that exhibits numerous similarities to models of sequence evolution. My new version, `Lagrange-NG`, computes results up to 50 times faster than the old version, and up to 2 orders of magnitude faster than the popular analogous tool `BioGeoBEARS`. In addition, I also develop a novel distance metric which allows for the comparison of results from alternative tools and algorithms.

Acknowledgments

Foremost of all, I want to thank and express my sincere gratitude to my PhD supervisor Alexandros Stamatakis. Dr. Prof. Stamatakis has been very kind, patient, and trusting with regards to my work and research. I am especially thankful of his support in pursuing novel and unusual topics. Indeed, Alexis has always been a wonderful brainstorming partner, and some of the best ideas during my PhD were directly inspired by his comments.

I also would like to thank the fellow members of the Exelixis Lab during my time at the institute: Pierre Barbera, Lucas Czech, Alexey Kozlov, Benoit Morel, and Sarah Lutteropp, Julia Haag, Lukas Hübner, Dimitri Höhler, and Anastasis Togkousidis for their friendship and valuable advice in general. I would also like to thank by name some of the lab member who assisted in the development of coraxlib: Alexey Kozlov, Benoit Morel, Julia Haag, Lukas Hübner, Dimitri Höhler, Anastasis Togkousidis.

I would like to thank my collaborators who worked with me on Lagrange-NG and Phylourny, Stephen Smith and Alexander I. Jordan. Stephen has been supportive of the Lagrange-NG rewrite, and has been quick to answer any questions I had. Alexander I. Jordan assisted with improving rigor for the mathematics of Phylourny, as well as identifying some issues with early versions of the model. I would also like to thank Qihao Yuan for his work on the Krylov subspace based implementation of the matrix exponential.

Finally, I would like to thank my family for their support, including my mother Karen Perdue, and my father Charles Bettisworth.

None of my PhD would be possible without the funding and support of the European Union Marie-Curie ITN IGNITE and the Klaus Tschira Foundation.

Contents

List of Figures

List of Tables

List of Abbreviations

AA Amino Acid (sequence). 7, 21, 78

CLV Conditional Likelihood Vector. 10, 12

CTMC Continuous-Time Markov Chain. 8

DEC Dispersal-Extinction-Cladogenesis. 57, 58, 65

DNA Deoxyribonucleic Acid. 5–7, 11, 21, 78

GTR General Time Reversible (model). 21, 26, 30

L-BFGS-B Limited memory Broyden-Fletcher-Goldfarb-Shanno with Bounds. 14, 20, 21

LWR Likelihood Weight Ratio. 24–26, 29–33, 35

MAD Minimal Ancestral Deviation. 19, 22, 26, 27, 34

MCMC Markov Chain Monte Carlo. 2, 14

MPI Message Passing Interface. 24, 27, 77

MSA multiple sequence alignment. 2, 3, 6, 7, 9, 20, 25, 26

NCAA National Collegiate Athletic Association. xii, xv, xvii, 45–51, 53–55

OpenMP Open Multi-Processing. 24

UEFA Union of European Football Associations. xii, xv, xvii, 44–48, 50, 51, 53–55

WPV Win probability vector. xvii, 39, 40, 43–47, 50, 51, 53, 54

List of Terms

ancestral range distribution An inferred range distribution for a putative ancestral species. A primary result of historical biogeographical analysis. 3, 64, 74

bifurcating A condition on phylogenetic trees that every node has *either* two children or no children. Equivalently, all nodes have either degree 1 or 3. If the tree is rooted, then a single node with degree 2 is also required. xxii, 6

branch The edges of a phylogenetic tree. 6

branch length Edge weights on a phylogenetic tree, which represents some distance between species. Normally, the distance is represented in number of substitutions per site, but can also be some abstraction of time. 6, 10

Brent's method A root finding method, which is similar to the secant method and the bisection method [6]. 13, 21

Hadamard product A product of two vectors $\mathbf{v}$ and $\mathbf{w}$, which is defined as $\mathbf{z}_i = \mathbf{v}_i\mathbf{w}_i$. 10

inner node Nodes in a phylogenetic tree which have a degree greater than 1.. 6

knock-out tournament A tournament where the winner is decided by a series of rounds where each team faces another team, and is knocked out (or possibly moved to a lower bracket) based on the results. 2, 37, 38

likelihood The probability of observing the data, given a model. Explicitly $\mathcal{L}(M|D) = P(D|M)$. xvi, 8, 18, 64

multidimensional optimization An optimization problem where the function takes as input a vector in $\mathbb{R}^n$, rather than just $\mathbb{R}$. 14

multiple sequence alignment A matrix of sequences sampled from individuals such that each column is homologous.. xix, 6, 20

Newtons's method A root finding method, which finds roots by computing the tangent to function and extrapolating it forward. 13, 21

non-reversible model A character substitution model for which the detailed balance equation does not hold. 18, 19

phylogenetic tree A tree which describes the evolutionary relationship of a set of species. In this work, all trees are bifurcating. xxi, xxii, 2, 6, 9

Pulley Principle A principle that allows the root location to be ignored when inferring a phylogenetic tree under a reversible model [21]. 11, 17, 19

range distribution A vector of probabilities summing to 1 indicating the likelihood of a particular range state. Note that this vector is using states and not ranges, and therefore has 2^r states. xxi, 58

root A special node in a phylogenetic tree, which represents the most recent common ancestor of all species present in the phylogenetic tree. Formally, this node has no parents, and has degree 2. xxii, 2, 6

rooted A phylogenetic tree with a root. xxi, 6

tip Nodes in a phylogenetic tree with no children, or with degree 1.. 6

ultrametric In relations to trees, this means that the path length from all tips to the root are equal. xvi, 63

unrooted A phylogenetic tree which has no root. 2, 6, 17

win probability vector A vector containing the distribution of win probabilities for an ordered set of teams. xix, 39

1. Introduction

1.1 Motivation

Uncertainty in science can broadly be defined as a lack of conclusive evidence towards a specific conclusion when addressing specific scientific question. It comes in several forms, from measurement error, lack of sufficient data for the question at hand, or a shortcoming in the model such that establishing an answer with certainty is difficult. Uncertainty, by its nature, has always been difficult to handle, and uncertainty analysis has become more prominent for at least the past 200 years (e.g. the discovery of Ceres and the computation of its orbit by Gauss).

In modern times the amount of data available to researchers has increased exponentially, particularly in the area of Bioinformatics [62]. However this has paradoxically not reduced the relevance of uncertainty analysis. Instead, the frontier of science has advanced such that present day researchers are using highly complex mathematical and statistical models. Additionally, these complex models have largely been implemented in software tools, meaning that a large part of modern scientific work is computational in nature. As science has become more computational the need to integrate, tolerate, and report the uncertainty when using computational models has increased.

One such field is Bioinformatics, which has seen an explosion in the amount of data available for researchers since the advent of genetic sequencing in the 1970s. This has been further accelerated by the development of next generation sequencing technologies in the early 2000s [62]. Now, researchers are faced with the challenge of integrating large quantities of heterogeneous data along with computational and statistical models to test hypotheses [32, 71]. Unfortunately, many tools and models do not have the capacity to report uncertainty about their results, which may lead researchers to be overly confident in their conclusions.

In this thesis I focus on phylogenetics, a subfield of Bioinformatics, which is concerned with inferring the evolutionary relationships between species. For a more

thorough introduction into the specifics of phylogenetics, please see Chapter 2. However, for this section it is only important to know that phylogenetic tools generally require as input genetic sequences sampled from individuals, generally of difference species, which have been heavily preprocessed[1]. This preprocessing is not completely certain in its results [65, 90]. However, these tools which take these preprocessed sequences often assume that each preprocessing step produced results which are exactly correct. This assumption is, in general, not correct [46, 47, 81] and can affect the final result of phylogenetic tools [5, 35].

The reasons why uncertainty is under-disclosed and under-accounted for are complicated. However, they broadly fall into two categories: lack of resources (either financial or computational) or lack of tooling/formalism. In this thesis, I seek to address both issues by developing tools and methods which further enable researchers to understand and even utilize uncertainty to their advantage.

1.2 Contributions

My main contributions to this thesis are the open-source software tools `RootDigger`, `Phylourny`, and `Lagrange-NG`.

`RootDigger` is a software tool to root unrooted phylogenetic trees. For reasons that are discussed in the next chapter, phylogenetic trees are often inferred without a root. Put another way, unrooted phylogenetic trees lack an orientation with respect to time. `RootDigger` uses a character substitution model which yields a likelihood that varies with respect to the orientation of time. This allows the software to identify the most likely position for a root on a phylogenetic tree. To do this, `RootDigger` takes as input an unrooted phylogenetic tree, and an MSA, which the software then uses to compute the likelihood of various location for the root. An article describing `RootDigger` appeared in *BMC Bioinformatics* [4].

`Phylourny` is a tool to assist in predicting knock-out tournaments. Traditional methods of predicting knock-out tournaments generally fit a model to historical data, and then use that model to simulate a tournament numerous times. Instead, I propose to compute the probability of all possible outcomes via a dynamic programming algorithm, which is inspired by similar dynamic programming algorithm to compute the likelihood of a phylogenetic tree. This has the advantage of being substantially more computationally efficient, as well as computing the probabilities of outcomes exactly. I then use this more efficient algorithm to explore the parameter space around the most likely parameters, with the goal of assessing stability via an MCMC search. At the time of writing this thesis `Phylourny` is currently under review for publication in the journal *Statistics and Computing*.

[1] The preprocessing steps are not covered in detail in this thesis, as they are complex topics in their own right, and therefore outside the scope of this thesis. Nonetheless, in brief the steps are: gene sequencing; read assembly; (optionally) annotation; and finally sequence alignment. Each of these steps uses complex statistical and computational models to maximize the accuracy of the results.

I also redesigned and reimplemented an old, yet very popular, biogeographic tool for inferring ancestral range distributions of ancient species. The new version of the tool is named `Lagrange-NG`, and improves on the previous version of the tool in three important ways. First, the tool is now capable of multithreading, which accelerates the computation by using more resources when needed. Second, the software engineering quality is substantially improved, including fixing a rare but critical bug in the computation of results. Third, and most importantly, I made the software substantially more efficient. The new version is generally an order of magnitude faster than the old version for small datasets, and up to 2 orders of magnitude faster for larger datasets. In addition to providing an improved version of a well established and popular tool, I also describe a new metric for comparing results of analogous tools. Prior to this work, no formalized method of comparing results between competing tools in this field (ancestral range reconstruction) existed. A paper describing Lagrange-NG and the associated distance appeared in the journal *Systematic Biology* in 2023 [3].

I also have a number of minor contributions which have not been included in the thesis. I will mention them below in order of publication.

I was a part of an effort to investigate the genomes of SARS-CoV-2, the virus responsible for the COVID-19 pandemic. My role was to integrate RootDigger into the pipeline that fellow lab members had developed. Additionally, I was responsible for running and producing a distribution of likely rootings of the SARS-CoV-2 phylogeny using `RootDigger`. This effort culminated in a publication on the difficulty of analysing the SARS-CoV-2 genomes in a phlyogenetic context. This article appeared in the journal *Molecular Biology and Evolution* in 2020 [61].

I assisted a partner lab in investigating a method of accelerating phylogenetic tree inference by reducing the dataset size via sub-sampling the input MSA. I contributed to the project by modifying RAxML-NG [45] to output the required runtime information to train a model. This model will then reduce the size of the MSA via sampling, with the goal of reducing runtime. The article appeared in the journal *Bioinformatics* in 2022 [16].

I assisted a fellow lab member (Julia Haag) with an article concerning predicting the difficulty of datasets with regard to phylogenetic inference. The main contribution for this project is the tool Pythia, which assigns a difficulty score to a dataset. Pythia performs this task by computing several values describing the dataset (i.e. features) which are then provided to a machine learning model which computes the final score. My contribution to the work involved contribution of ideas (specifically, providing some of the features used in the final model), as well as general consulting on topics of academia. The article describing Pythia appeared in the journal *Molecular Biology and Evolution* in 2022 [28].

Finally, I led and participated in the development of a new version of the internal lab library (formally the Phylogenetic Likelihood Library or LibPLL-2 [23]), newly named `Coraxlib`. My contributions include: adding support for non-reversible models; building a testing framework; implementing a new tree parser for the Newick

tree format; documenting many critical but undocumented portions of the library; and organizing the efforts of volunteers to improve the quality of the library. Currently two of the Exelixis Lab's tools utilize `Coraxlib`: `RootDigger`, and GeneRax (a tool written by fellow lab member Benoit Morel).

1.3 Structure

The remainder of this thesis is structured as follows. Chapter 2 is an introduction into the models and methods which will be used in later chapters of the thesis. Chapter 3 contains a thorough discussion of the operation and performance of `RootDigger`. Chapter 4 explores the prediction of knock-out tournaments with `Phylourny`, along with two case studies involving real world sports tournaments in the sports (basketball and football). Chapter 5 discusses the technical improvements, the results of the improvements for `Lagrange-NG` as well as the formal details of the novel distance metric used to compare results between tools. And finally, Chapter 6 concludes the thesis, and gives directions for future work.

2. Background

2.1 Introduction to Phylogenetics and Biogeography

Phylogenetics is the study of the history of evolution. Therefore, the goal is to determine the evolutionary relationships between a given set of species. Historically, this analysis was performed by experts in a particular genus or family, and required substantial effort and many years of study of physical samples and their morphology [80]. However, computational methods were quickly adopted with the advent of computers and computational science, with the first computer inferred phylogeny being published in 1965 [8]. Early phylogenetic models were simplistic, mostly seeking to minimize the number of changes to a sequence when relationships were represented on a tree, using a criterion called Maximum Parsimony. Later, probabilistic models of sequence evolution were developed [21, 41], which alleviated some issues with Maximum Parsimony [20].

2.1.1 Fundamentals of Phylogenetics

Suppose that we wish to find a plausible explanation for the evolution of some set of species. That is, we wish to take a set of species, for which we are interested in the history of their evolution, and using attributes about them (commonly molecular data obtained from the organism, for example DNA) and produce a description of the evolutionary history of this group. This is the central problem of phylogenetics, and to enable this, we will make 2 assumptions.

The first assumption, that evolution is roughly treelike, is as old as Darwin's *On the Origin of Species* [11], which contains some of the first tree representations of phylogeny. These so-called phylogenetic trees are the main goal and result of phylogenetic analysis. While there are exceptions to evolution being treelike with events such as hybridization (when two previously physically separated population rejoin into a single population which can produce fertile offspring), and horizontal gene transfer (when genetic material is transferred between two species via means other

than producing offspring), the presence of trees as the major result of phylogenetic analysis remains.

To be specific, a phylogenetic tree is a graph structure satisfying the requirements of a tree, that is to say, that there exists only one path between any two nodes. Additionally, nodes on a phylogenetic tree are grouped into two categories: inner nodes and tips. Inner nodes (also known as internal nodes or hidden nodes, but they will be referred to as inner nodes only in this work) represent hypothetical ancestral species, which underwent a speciation process to produce 2 new species. Tips represent the observable, typically extant species which we can sample for information, such as DNA, to guide phylogenetic analysis.

The second assumption, that the speciation events are strictly binary, is an assumption born not out of the need to represent biology, but out of convenience. The particular benefits of this assumption will be discussed in detail in Section 2.1.3. But to summarize that discussion here: assuming that the tree is binary allows us to formulate the equations in a much simpler manner, and there is no loss of information by making this assumption.[2]

The edges of a phylogenetic tree are called branches. These branches may have associated branch lengths, which are a measure of the distance between two hypothetical species. This distance can be thought of in two ways, either as the amount of time elapsed or the amount of evolutionary events which have occurred.

A phylogenetic tree can be either rooted, in which case the tree has a root, or unrooted in which case the tree lacks a root. In this work, the root indicates the oldest species in the tree, which is the most recent common ancestor of the species present as tips of the tree.

Normally, phylogenetic trees are bifurcating, which is a realization of the assumption that speciation is a binary process. More formally, a bifurcating phylogenetic tree is a tree where all nodes in the tree have either degree 3 or degree 1. In the case of a rooted tree, there must be a single node with degree 2.

A multiple sequence alignment (MSA) is a matrix of sequences sampled from individuals such that each column represents an homologous site. Here, homologous can be interpreted to mean "from the same origin" that is that the site evolved from a common ancestral site, for all species in the MSA. Taking a given MSA and producing an accurate phylogeny (i.e. a phylogenetic tree which describes the history of evolution for the species present in the MSA) is the main challenge in phylogenetics.

Historically, phylogenies were produced by hand and the intuition of experts, as mentioned above. In modern times however, computational analysis of molecular sequences has almost entirely supplanted this method. In order to produce a phylogeny for publication, researchers now sample a sequence of Deoxyribonucleic Acid

[2] The reason why we can do this require understanding the statistical models of character evolution, which are discussed later. I can, however give an informal explanation. If we allow for branches with a length of zero, then we have a branch were no evolution occured. We can transform a tree which does not satsify the degree requirements, that is a non-binary tree, by inserting 0 length branches such that the tree eventually becomes binary.

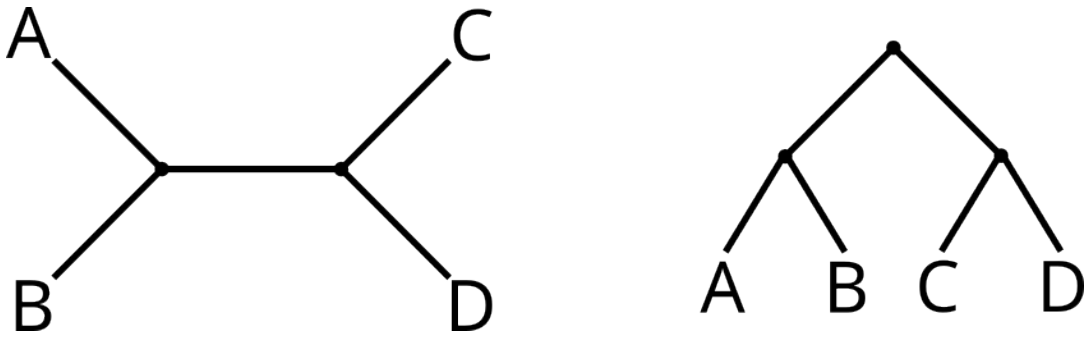

Figure 2.1: Two example trees, unrooted (left) and rooted (right).

(DNA) or Amino Acid (sequence) (AA) from an organism, prepare those sequences as an MSA in a long series of preparatory steps which will not be discussed here, and then provide that resulting MSA to a program which infers a phylogeny.

2.1.2 Models of Character Evolution

Suppose we have an MSA with two species, say species A and B, as well as a single site. The natural question we could ask is "given a putative phylogeny, what is the probability of observing the sequences obtained from species A and B". As there are only two species, there is only 1 possible topology, which is shown in Figure 2.2. If we imagine evolution as a process which changes character a at some instantaneous rate r over time t, then the probability of an event (i.e. a character substitution) is

$$1 - e^{-rt}. \tag{2.1}$$

Once an event occurs, we change the character to some other state from any of s states with uniform probability. Please note that the "null" change is allowed, that is a "becoming" a. Given this, the probability that character a remains as a at time t is

$$P_{a,a}(t) = e^{-rt} + \frac{1 - e^{-rt}}{s}. \tag{2.2}$$

That is, the probability of no change is the probability of no event $(1 - (1 - e^{-rt}))$ plus the probability of an event and selecting a again. In contrast, the probability that a changes to b, where $a \neq b$ is

$$P_{a,b}(t) = \frac{1 - e^{-rt}}{s}. \tag{2.3}$$

For convenience and to motivate Markovian models later, we can express the two equations above as a matrix, which we do so here for a 2 state model.

$$P(t) = \frac{1}{2}\begin{pmatrix} 1+e^{-rt} & 1-e^{-rt} \\ 1-e^{-rt} & 1+e^{-rt} \end{pmatrix} \tag{2.4}$$

where the probability of state a converting to state b is given by the matrix $P(t)$ entry at row a column b.

As elegant as the model in Eq. 2.4 is, it has severe limitations, the first of which is that it assumes equal rates between all state transitions. If we want to extend the model to account for different rates between states, we should instead look at the instantaneous change in the probability of observing a over some time dt. Let $\mathbf{p}$ be a vector with entries $(p_a(t), p_b(t))$, that is the probability of observing state a or b at time t. Then,

$$\frac{d}{dt}p_a(t) = q_{aa} \times p_a(t) + q_{ba} \times p_b(t) \tag{2.5}$$

and

$$\frac{d}{dt}p_b(t) = q_{ab} \times p_a(t) + q_{bb} \times p_b(t). \tag{2.6}$$

Where q_{ij} are the instantaneous rates of change from state i to j. We can use these relations to setup the following system of differential equation.

$$\frac{d}{dt}\mathbf{p}(t) = \mathbf{Q}\mathbf{p}(t) \tag{2.7}$$

where

$$\mathbf{Q} = \begin{pmatrix} q_{aa} & q_{ab} \\ q_{ba} & q_{bb} \end{pmatrix} \tag{2.8}$$

that is, the matrix of rates present in Eq. 2.5 and Eq. 2.6. If initial values are given as $\mathbf{p}(0)$, then

$$\mathbf{p}(t) = e^{\mathbf{Q}t}\mathbf{p}(0). \tag{2.9}$$

What we have defined here is a Continuous-Time Markov Chain (CTMC). Using a CTMC allows us to compute the probability of specific evolutionary events. A quick note about $\mathbf{p}(0)$, which is known as the prior distribution. This vector is important enough to be given its own notation, and is generally referred to as π.

We almost never use these models to compute a probability, but instead we use these models to compute the likelihood of the observed data. In general, the likelihood is defined as

$$\mathcal{L}(M|D) = P(D|M) \tag{2.10}$$

where D is some observations (i.e. data) and M comprises some model parameters. For our specific case, suppose we have a 2 state model like above, and a pair of observed states a and b, with a specified model $\mathbf{Q}$ and estimated time t. The likelihood then is

$$\mathcal{L}(\mathbf{Q}, t|a, b) = P(a, b|\mathbf{Q}, t) = P_{ab}(t). \tag{2.11}$$

As it can be seen above, computing the likelihood for a single pair of states is trivial. In the next section,we extend the models and scope of the likelihood calculations

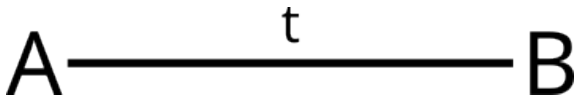

Figure 2.2: A simple tree with 2 taxa, and a single branch between them.

beyond this simple case. Finally, the math so far has been conducted assuming a single site. However, this is not very useful as there is simply not sufficient information contained in a single site to determine the evolutionary history of a large number of species. Instead, an assumption is made for most models (which is true for all models used in this work) that each site evolves independently, and identically to other sites. Specifically, we make an "independently and identically distributed" assumption about a set of sites, and typically perform our computations on a large number of sites, i.e. an MSA.

A small wrinkle with the i.i.d. assumption is the existence of segments of a genome which have evolved under different pressures. For example, a gene coding for a particularly important protein might evolve differently than a gene coding for a minor protein. In this case, it is convent to partition the MSA into sets of sites which evolved under the same model. In these cases we relax the i.i.d. assumption to be *within* a partition.

Therefore, the likelihood of a model given an MSA M with S sites is

$$\mathcal{L}(\mathbf{Q}, t|M) = \prod_{1 \leq i \leq S} \mathcal{L}(\mathbf{Q}, t|M_{a,i}, M_{b,i}) \tag{2.12}$$

where $M_{a,i}$ is the character for taxon a at site i.

An additional note about the $\mathbf{Q}$ in Eq. 2.8: in order for $e^{\mathbf{Q}t}$ to be a properly formed Markov matrix, the rows of $\mathbf{Q}$ must sum to zero. If this is not the case, the net rate of change for a state would either be negative or positive, that is the total number of sites would change over time. Given that we have a fixed number of sites in our observed data, we must impose this condition which conserves the number of sites over time. This means that we should rewrite Eq. 2.8 as

$$\mathbf{Q} = \begin{pmatrix} -q_{ab} & q_{ab} \\ q_{ba} & -q_{ba} \end{pmatrix}. \tag{2.13}$$

This reduces the number of free parameters to 2, and ensures that the result of $e^{\mathbf{Q}t}$ is well formed Markovian matrix.

2.1.3 Continuous-Time Markov Models and Phylogenetic Trees

Suppose we have a set of n species, an MSA for these species, and we wish to find the phylogenetic tree which best describes the evolutionary history of these species. If we suppose that the models of character evolution described in the previous section are an accurate description of the underlying evolutionary process, then we can simply

augment the likelihood in Equation 2.12 with two additional parameters: a binary tree T, and a set of branch lengths B. Doing this yields the expression

$$\mathcal{L}(\mathbf{Q}, T, B|M) = \prod_{i \leq S} \mathcal{L}(\mathbf{Q}, T, B|M_i) \tag{2.14}$$

where

$$\mathcal{L}(\mathbf{Q}, T, B|M_i) = P(M_i|\mathbf{Q}, T, B). \tag{2.15}$$

The specific expression to compute P depends on the particular structure of the tree parameter T. For example, suppose we want to use this expression to compute the likelihood of the tree in Fig. 2.3. We would then need to evaluate the expression

$$\sum_i \sum_j \sum_k \sum_l \pi_i \times P_1(j|i) \times P_2(k|j) \times P_3(l|j) \times P_4(m|i) \tag{2.16}$$

That is, the probability of transitioning from state i at the root to state j, and so on, summed over all possible states. If we rearrange the expression a little, we can notice some structure which we can exploit

$$\sum_i \pi_i \times \left(\sum_j \left(P_1(j|i) \times \left(\sum_k P_2(k|j) \times \sum_l P_3(l|j) \right) \right) \times \sum_m P_4(m|i) \right). \tag{2.17}$$

Notice the nesting structure of the tree is represented by the parentheses in this expression. We can further rewrite this expression to be more readable if we employ matrix notation and the Hadamard product

$$\pi^\intercal \left(\mathbf{P}_1 \left(\mathbf{P}_2 a \circ \mathbf{P}_3 b \right) \circ \mathbf{P}_3 c \right) \tag{2.18}$$

This observation, that the expression to compute the likelihood of a tree can be rearranged based on the topology of the tree, is known as the Felsenstein Pruning Algorithm [21], which itself is a realization of a technique known as Horner's method. This algorithm allows us to compute likelihoods efficiently, as it allows for partial results to be used in later computations, greatly improving the efficiency of the evaluation of likelihoods.

In practice, the result of a Hadamard product can be stored in a Conditional Likelihood Vector (CLV) and reused for later computation. This further simplifies the expression to compute the partial results corresponding to node i with children j and k

$$\mathbf{CLV}_i = (\mathbf{P}(t_j)\mathbf{CLV}_j) \circ (\mathbf{P}(t_k)\mathbf{CLV}_k) \tag{2.19}$$

where t_j and t_k are the branch lengths from nodes i to j and from i to k, respectively. We can compute the likelihood of some tree by recursively applying the Equation 2.19 to each inner node of the tree, finishing with the root which we will label r. Once that is complete, the likelihood is finally computed by evaluating

$$\mathcal{L} = \pi^\intercal \mathbf{CLV}_r. \tag{2.20}$$

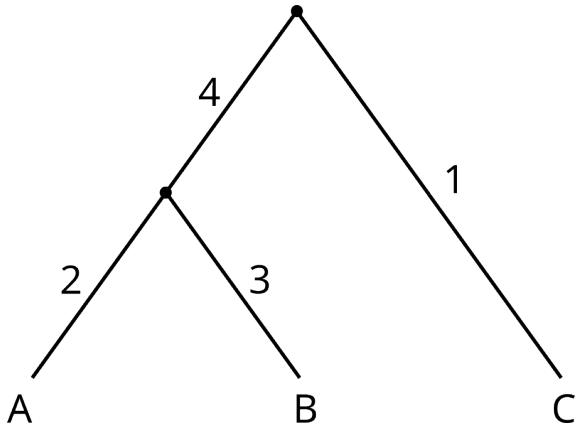

Figure 2.3: A three taxa rooted tree with labeled branches.

2.1.4 Reversible and Non-Reversible Markov Models

For this work the topic of reversibility of a Markov model is important. A Markov model is reversible if there exists a π such that the following equation holds for all i, j and $\mathbf{P}$

$$\pi_i \mathbf{P}_{ij} = \mathbf{P}_{ji} \pi_j. \tag{2.21}$$

If such a π exists, we call it a stationary distribution of the Markov model. In contrast, if such a π does not exist, then we call the model non-reversible. Traditionally, reversible models have been used in phylogenetic inference over non-reversible models due to the reduced computational complexity that a reversible model grants [21]. The major reduction in computational complexity comes from the so-called Pulley Principle, which points out that if a Markov model is reversible, then all root locations will yield the same likelihood. If a model is non-reversible, then different root locations will yield different likelihoods, which means that root placement must be included in the model optimization process.

In the context of evolution, reversibility is an assumption that is clearly false. If it were true that evolution followed a reversible process, we would expect the proportion of sites to be equal across the tree of life. However, we find instead that the GC content (the relative proportion of sites in a DNA sequence which are either a Guanine or Cytosine) varies substantially across the tree of life [30]. Nonetheless, many phylogenetic analyses are conducted with models which assume reversibility, due to the computational savings that such an assumption affords.

2.2 Computational Methods Used In This Work

2.2.1 Matrix Exponential

The matrix exponential is defined by the Taylor Series

$$e^{\mathbf{A}} = \frac{1}{0!}\mathbf{A}^0 + \frac{1}{1!}\mathbf{A}^1 + \frac{1}{2!}\mathbf{A}^2 + \dots \tag{2.22}$$

where $\mathbf{A}$ is a square matrix with (in our case) real entries, and $\mathbf{A}^0 = \mathbf{I}$. Computations using this formula are generally avoided, as it is both slow and extremely numerically unstable. Instead, specialized algorithms have been developed to calculate this quantity with varying degrees of accuracy and efficiency. Here, I only present only the algorithms used in this work, but the algorithms presented here are only a small subset of known algorithms for matrix exponentiation, which are discussed in detail in [59].

2.2.1.1 Matrix Exponential by Scaling and Squaring

There are several immediate optimizations that we can apply to Equation 2.22. The first is to use the Padé approximation, instead of the Taylor series. Padé approximations are a ratio of two polynomials which generally gives a better approximation with less terms when compared to a truncated Taylor Series. Additionally, one can notice that

$$e^{\mathbf{A}} = (e^{\mathbf{A}/m})^m \tag{2.23}$$

for some integer m. Therefore, by first scaling the matrix by m such that m is a power of 2, we can both reduce the number of terms in the Padé approximation as well as improve the numerical stability of the method. To this end, we can pick m to be a power of 2 such that we can obtain $(e^{\mathbf{A}/m})^m$ by repeated squaring.

2.2.1.2 Matrix Exponential by Krylov Subspaces

A Krylov subspace for matrix $\mathbf{A}$ and vector $\mathbf{b}$ is a space spanned by a series of vectors constructed as

$$\mathcal{K}_m = \{\mathbf{A}^0\mathbf{b}, \mathbf{A}^1\mathbf{b}, \mathbf{A}^2\mathbf{b}, \ldots, \mathbf{A}^m\mathbf{b}\}. \tag{2.24}$$

A property of a Krylov subspace is that it well approximates certain properties of $\mathbf{Ab}$, such as the largest m eigenvectors and eigenvalues. Generally, m is small compared to the size of $\mathbf{A}$, so if a Krylov subspace is used instead of $\mathbf{A}$, large gains in computational efficiency can be gained. To compute the matrix exponential with a Krylov subspace, Arnoldi Iterations are applied to form V_m, an orthonormal basis containing the basis vectors from the Krylov subspace for $\mathbf{Ab}$, and H_m, a partially Hessenberg reduced matrix which approximates the first m eigenvalues. From these matrices, $e^{\mathbf{A}}\mathbf{b}$ can be approximated with

$$e^{\mathbf{A}}\mathbf{b} \approx \mathbf{V}_m e_m^{\mathbf{H}}\mathbf{v}. \tag{2.25}$$

In our case, $\mathbf{b}$ will generally be the partial result stored in a CLV according to Felsenstein's Pruning Algorithm. For example, in Equation 2.19 every $\mathbf{P}(t_i)\mathbf{CLV}_i$ is in reality a matrix exponentiation followed by a matrix vector product, which is exactly the value being approximated in Equation 2.25. The choice of the number of basis vectors m is a bit arcane. The authors of [33] give an upper bound on the error of the approximation of $e^{\mathbf{A}}\mathbf{v}$ using m basis vectors as

$$\frac{2\|\mathbf{A}\|_2^m\|\mathbf{v}\|_2}{m!}\max(1, e^{\eta}) \tag{2.26}$$

where $\|\mathbf{A}\|_2$ is the l_2 norm, and η is the logarithmic norm of $\mathbf{A}$, which is equal to the largest eigenvalue of the symmetric part of $\mathbf{A}$. As $m!$ grows faster than a^m with respect to m, the maximum error from this approximation decreases rapidly as m increases.

This method can be significantly faster, as $\mathbf{H}_m$ is an m by m matrix, and m can be quite small when compared to the size of $\mathbf{A}$ while still being accurate. For more details, please see [7, 59].

2.2.2 Optimization

Optimization is a broad topic, which primarily concerns itself with finding a solution to the problem

$$\begin{aligned} \underset{\mathbf{x}}{\operatorname{argmin}} f(\mathbf{x}) \text{ s.t. } \mathbf{Ax} = \mathbf{0} \text{ and} \\ \mathbf{Bx} \geq \mathbf{0} \end{aligned} \tag{2.27}$$

where matrices $\mathbf{A}$ and $\mathbf{B}$ are constraints (in this case linear constraints) on the values of the vector $\mathbf{x}$. This problem constitutes a large area of research, and a full exploration of the topic is far beyond the scope of this work. Instead, we present a few of the methods used to solve this problem here, which are used in later sections of this work.

2.2.2.1 Newton's Method

Newtons's method is an iterative method of finding roots. This method proceeds by producing iteratively better guesses by starting with some guess x_i, and following the tangent line to the x axis, and using this as the new guess. Formally, the next guess

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{2.28}$$

where f' is the derivative of f.

In order to find optima with Newton's method, roots should be found in f' instead of f. That is, Newton's method finds points where the derivative is equal to zero. In the case of well-behaved one dimensional functions, these are guaranteed to be local optima. Therefore, by finding the roots of f', we can also find optima in f. However, this requires computing the second derivative of f, which can be either analytically, computationally, or numerically difficult.

2.2.2.2 Brent's Method

Brent's method is a root finding method which is a combination of several existing methods, and has applications similar to Newton's method. The main advantage of Brent's method over Newton's method is that Brent's method does not require the computation of a derivative, either numerically or analytically, which can be either computationally expensive, analytically intractable, or impossible depending on the function. In particular, if the method is used to optimize a function value (that

is, perform root finding on the derivative of a function), then a second derivative needs to be computed, which can be difficult to compute numerically, as higher order derivatives require more function evaluations in order to be accurate.

Additional details for Brent's method may be found in [6], however the method works by using

1. Inverse Quadratic Interpolation,
2. The Secant Method, and
3. Bisection

roughly in this order of priority in order to find a function root.

2.2.2.3 L-BFGS-B

Limited memory Broyden-Fletcher-Goldfarb-Shanno with Bounds (L-BFGS-B) is a general purpose multidimensional optimization routine which seeks to outperform gradient descent, while also avoiding the computation of a second derivative [97]. Again, additional details can be found in [97], or in most textbooks on numerical optimizations, but a summary is presented here. L-BFGS-B operates by emulating Newton's method, but instead of computing the second derivative (here known as the Hessian) either analytically or numerically, it instead builds an approximation of the Hessian via a series of updates computed from the gradient and the steps taken each iteration of the algorithm. As the algorithm proceeds, the estimate of the Hessian becomes increasingly accurate, allowing for rapid convergence to an optimum.

2.2.2.4 Linear Programming

Linear programming is an optimization method applied to problems of the form

$$\begin{aligned} \underset{\mathbf{x}}{\operatorname{argmin}}\, \mathbf{a}^\intercal \mathbf{x} \text{ s.t. } & \mathbf{A}\mathbf{x} = \mathbf{0} \text{ and} \\ & \mathbf{B}\mathbf{x} \geq \mathbf{0} \end{aligned} \tag{2.29}$$

where $\mathbf{A}$ and $\mathbf{B}$ are a set of linear constraints represented as a matrix, $\mathbf{a}$ is a set of weights for the linear optimization problem. In contrast to other optimization problems, linear programming problems are relatively easy to solve, and have favorable properties. In particular, solutions to linear programming problems are always global optima, if they exist.

2.2.3 Markov Chain Monte Carlo

Markov Chain Monte Carlo (MCMC) methods are a class of techniques used to sample from a probability distribution. While there are several algorithms which can be classified as MCMC methods, such as Gibbs Sampling and Hamiltonian Monte Carlo, this work only focuses on MCMC via the Metropolis-Hastings Algorithm [57].

In phylogenetics, and also in this work, the probability distribution we wish to sample from is the posterior distribution, i.e.

$$P(M|D) = \frac{P(D|M)P(M)}{P(D)}. \tag{2.30}$$

Where M is some model and D is some observed data. In this equation, the expression on the right is generally impossible to compute directly, as $P(D)$, is simply not known for the real world. To see why, consider that $P(D)$ is the marginalization of $P(D|M)$ over all *possible* M. This is a monumental task as even defining all possible models is, at the very least, difficult. Therefore instead of dealing with $P(D)$ directly, the Metropolis-Hastings algorithm avoids computing this term by instead only computing the ratio of two posteriors:

$$\frac{P(M_1|D)}{P(M_2|D)} = \frac{P(D|M_1)P(M_1)}{P(D)} \times \frac{P(D)}{P(D|M_2)P(M_2)} = \frac{P(D|M_1)P(M_1)}{P(D|M_2)P(M_2)} \tag{2.31}$$

This term on the right then becomes the acceptance ratio in the Metropolis-Hastings algorithm, and allows us to characterize the posterior. In this way, we have turned an impossible problem into merely a computationally expensive problem, as adequately exploring the posterior via the Metropolis-Hastings algorithm can require many samples, and therefore many computations. Nonetheless, Metropolis-Hastings is a very powerful tool, as it allows us to fully characterize probability distributions which are difficult or impossible to characterize analytically.

3. RootDigger

This chapter is based on the following peer-reviewed software article:

> **Ben Bettisworth**, Alexandros Stamatakis. "Root Digger: a root placement program for phylogenetic trees" *BMC Bioinformatics*, Volume 22, Issue 1, pp. 1-20, https://doi.org/10.1186/s12859-021-03956-5

3.1 Introduction

In standard phylogenetic inference, most tools [64, 77] yield unrooted trees. This is because they typically implement time-reversible nucleotide substitution models [21] as they yield the phylogenetic inference problem more computationally tractable. However, time-reversible models are incapable of identifying the root, as they disregard the direction of evolution. This is the result of the so-called Pulley Principle (for more information, please see Section 2.1.4) [21]. Nevertheless, a rooted phylogeny is often required for downstream analyses and interpretation of results as it can resolve long standing disputes regarding the placement of large clades on the tree of life for example [15]. In many cases, researchers will have to use a dedicated tool or include additional information in the analysis to recover the root of an inferred *unrooted* phylogenetic tree.

To root a tree when the primary phylogenetic inference is performed via a reversible model, researchers typically deploy one of the two following methods: including a set of outgroup taxa in the analysis, or using some form of molecular clock analysis. Unfortunately, both approaches exhibit their own challenges and pitfalls [2]. Including outgroup taxa in the analysis increases the amount of work that must be conducted in order to infer a tree, which primarily comes in the form of addtional research and possibly sequencing on the part of the researchers. More importantly,

adding an outgroup can also affect the ingroup topology in unexpected ways [34] (See the next section for more details). Molecular clock analysis can be complicated by the need to calibrate the molecular clock, as this often requires appropriate and a sufficient number of fossil records that are related to the organisms under study [1, 2].

Alternatively, one can use a non-reversible model as, under such a model, the root placement *does* affect the likelihood of the tree [95]. Examples of non-reversible models include gene tree species tree reconciliation methods that account for gene duplication, loss, and transfer [60], or non-reversible Markov substitution processes of character evolution. It is the latter process that `RootDigger` uses to root an existing phylogeny. This allows `RootDigger` to circumvent the compute-intensive step of inferring a tree under a non-reversible model, and instead only use a non-reversible model to root the inferred tree in a final step. By doing in this manner, one can combine the advantages of both: fast tree inference under reversible models; and rooting the tree under a non-reversible model.

The rest of this paper is organized as follows. First, we provide some more background on the theory and operation of `RootDigger`, as well as a justification for our method. Then, we describe the operation of `RootDigger` in detail. Next, we outline the methodology used to experimentally verify `RootDigger` and present the respective results. Finally, we discuss the effectiveness of `RootDigger` with respect to other rooting methods which are applied after tree inference, specifically `IQ-TREE` 2 and MAD.

3.2 Background

Methods that use additional topological information take advantage of prior knowledge about the world, which is not present in the, generally molecular, data that is used to infer the tree. In particular, knowledge about specific species which are not too distantly related to the species in question can be included as a so-called outgroup. This outgroup can then be used to place the root on the tree, as the most recent common ancestor of the ingroup and the outgroup should be the root of the overall tree.

There are challenges to including an outgroup to an analysis. Gatsey *et. al.* [24] showed how adding a single taxon to an analysis can substantially change the resulting tree topology, even for the taxa which were already present in the analysis (i.e., the ingroup). Holland [34] investigates this phenomenon in simulations, and finds that outgroups affecting or altering the ingroup topology are common.

Alternatively, molecular clock analysis can be used to place a root without prior topological knowledge [92]. The molecular clock hypothesis assumes that the substitution process exchanges bases (i.e., “ticks”) at a stochastically constant rate. Using this supposition a likely location can be inferred for the root on an existing phylogenetic tree. A simple version of this is midpoint rooting, which relies on a constant molecular clock assumption in order to produce a phylogenetically meaningful

rooting. However it can be applied to any binary tree, regardless of whether it is ultrametric or not. Other methods, such as Minimal Ancestral Deviation (MAD) [82] and MinVar [54] also rely on the molecular clock assumption. They attempt to solve the potentially poor performance of midpoint rooting in the presence of a violation of the strict molecular clock by allowing for variation in the rate of the molecular clock.

Molecular clock analyses exhibit their own difficulties. In particular, the clock does not generally "tick" at a constant rate over the tree [51, 78]. Relaxed clock models exist which can alleviate this problem, but are not always successful at correctly identifying the root as shown in [2] and come with their own set of inference errors and methodological challenges.

The final method that can place a root on a tree is to perform the phylogenetic analysis under a non-reversible model of evolution. When using a non-reversible model, the direction of time affects the likelihood of the tree [95]. Using this property, the most likely location on the tree for the root can be found, so long as the model has an appropriate fit. Indeed, early results suggested that some non-reversible models (particularly those based on character substitution) are inappropriate for the purposes of rooting a tree [36]. However, in this work we find that these concerns appear to be mostly overstated (see Results). Several software packages are able to infer or score a phylogenetic tree under a non-reversible model, and as a by-product also identify a root [64, 75].

Non-reversible model for phylogenetic trees come in many forms. For example, accounting for duplication, transfer, and loss events yields a non-reversible model [60]. In particular, duplication events have been used for rooting trees [18]. Another method, the one primarily used in this work, is to eliminate the reversibility assumption of standard character (e.g., nucleotide or amino acid) substitution models. Unfortunately, eliminating this assumption significantly increases the computational effort required to find a good (high likelihood) phylogenetic tree. This is due to the resulting inapplicability of the Pulley Principle [21], which allows phylogenetic inference tools to ignore root placement during tree inference. Therefore, by adopting a non-reversible model, the location of the root on a phylogenetic tree affects the likelihood of that tree.

As the location of the root affects the likelihood of the tree, when using standard tree search techniques all possible rootings would need to be evaluated for each tree considered in order to find the rooting with the highest likelihood. In the worst case, this increases the work *per tree* being visited during the tree search by a factor of $\mathcal{O}(n)$ where n is the number of taxa in the dataset. Therefore, eliminating the reversibility assumption drastically increases the computational effort required to infer a tree. Hence, standard inference tools choose to adopt the reversibility assumption, as phylogenetic tree inference would be computationally significantly more intensive otherwise.

As an alternative to the computationally expensive process of inferring a tree with a root, an unrooted tree which has already been inferred under a reversible model can

be evaluated a posteriori for possible root locations under a non-reversible model. This requires less computational effort, as it skips the expensive step of identifying "good" rootings in intermediate trees during the tree search. With this method, we can find the most likely root location for a given phylogenetic tree. Even this approach still faces numerical challenges, as previous research suggests that the likelihood function for rooting a phylogenetic tree may exhibit several local maxima [36], although we did not find this to be a major issue in our experiments (see discussion).

We implemented the open source software tool `RootDigger` which uses a non-reversible model of character substitution to infer a root on an already inferred, given tree. The inputs to our tool are a multiple sequence alignment (MSA) and an unrooted phylogenetic tree. `RootDigger` then returns a rooted tree. `RootDigger` implements fast and a slow root finding modes, called Search mode and Exhaustive mode, respectively. The search mode simply finds the most likely root quickly via appropriate heuristics, and is intended for users who simply intend to root the tree. For a more through exploration of the possible roots, we designed the exhaustive mode, which thoroughly evaluates the likelihood of placing the root into every branch of the given tree, and reports the likelihood weight ratio [79] for placing a root on that branch for every branch on the tree. In other words, the exhaustive mode allows to quantify root placement uncertainty over the branches of the tree.

Additionally, `RootDigger` supports both thread and process level parallelism, over a potential data partition of phylogenomic alignments and over distinct search starting locations (i.e., parallelization of the root search procedure), respectively. Finally, to support root inferences on extremely large datasets using compute clusters, we have implemented checkpointing in `RootDigger`, which allows for the search to be halted and resumed at a later point in time in case of hardware failures or when the job time limit has been exceeded.

3.3 The Software

Usage of `RootDigger` is straight-forward. All that is required is a tree in newick format, and a MSA in either PHYLIP or FASTA format. `RootDigger` is open source, released under the MIT license, and written in C++, and is targeted at the Linux platform. The code, documentation, test suite, as well as any modifications to existing libraries can be found at github.com/computations/root_digger.

In order to implement both, likelihood computations, and non-reversible models, `RootDigger` has two major dependencies: `Coraxlib` (the successor to the Phylogenetic Likelihood Library [LibPLL] [23]), and L-BFGS-B [97]. `Coraxlib` is used for efficient likelihood calculations and non-symmetric model computation, and L-BFGS-B is used for substitution rate optimization.

3.4 Methods

The input to `RootDigger` is an MSA and a phylogenetic tree with branch lengths in expected mean substitutions per site. `RootDigger` then uses the tree and branch

lengths to find the most likely root location by calculating the likelihood of a root location under a non-reversible model of DNA substitution[3](specifically, UNREST [94] with a user specified number of discrete Γ rate categories, and optionally a proportion of invariant sites, i.e., UNREST+Γ+I). The UNREST model is used because numerous other models (including models which are in the Lie group detailed in![91]) have been derived from this model. The optimal position of the root along a specific branch of length t is calculated by splitting the given branch in two with resulting branch lengths βt and $(1-\beta)t$, with $0 \leq \beta \leq 1.0$. We then find the maximum likelihood value of β, and report the likelihood for the given branch as the likelihood of the root location on that branch. By formulating the problem this way, we can use single parameter optimization techniques such as Brent's method [6], which are computationally more efficient compared to multi-parameter optimization routines such as the L-BFGS-B algorithm (named for its creators: Broyden, Fletcher, Goldfarb, and Shanno). Note that we specifically selected Brent's method instead of Newtons's method, because it does not require the calculation of the second derivative to optimize the function. While an analytical computation of the second derivative could be implemented, initial estimates showed that the savings were insufficient to justify the increased complexity and potential numerical issues. Nonetheless, in principle, the computation of the second derivative of the likelihood is feasible and could be implemented.

A potential problem of Brent's and analogous methods is that they find extrema by identifying roots for the derivative of the objective function. In order to find maxima, though, it is required that the objective function's value is also determined, as a root of the derivative could correspond to a minimum. In addition, Brent's method will fail to find all extrema. To alleviate this, we need to search for bracketing windows that can be used to safely find extrema. Unfortunately, we are not aware of a general method for finding such bracketing windows, so a recursive method is employed, were the search range is bisected and adequately searched for appropriate windows. Appropriate here means that the sign of the function in question has opposite signs at the respective endpoints of the window.

As already mentioned, `RootDigger` offers two modes of operation. These modes will be discussed individually, starting with the search mode:

1. Initialize numerical model parameters:
    - α-shape parameter for discrete Γ rates to 1.0 (if applicable),
    - Character substitution rates to $\frac{1}{4(4-1)} = \frac{1}{12}$

[3] Typically tree inference which uses AA data does not allow for using the fully unrestricted General Time Reversible (model) (GTR) rate matrix, instead picking from one of several precomputed empirical substitution matrices. This substantially limits the number of free parameters, thereby reducing the risk for over fitting. In contrast, the fully unrestricted reversible AA rate matrix would have 380 free parameters. Therefore, we choose to limit `RootDigger` to DNA data because the equivalent model for AA data would have far too many parameters to reliably optimize.

- Base frequencies to $\frac{1}{4}$

2. Generate starting roots according to one of the following strategies (default 1% of possible root positions)
   - Modified MAD (Default) or,
   - Randomly.
3. For each starting root:
   a Optimize model parameters
      - α-shape parameter for Γ distributed rates (if applicable, and only every 10 iterations of root placement updates),
      - Character substitution rates,
      - Base Frequencies.

   b Find the best root location for the current model
      i. Create a list of high likelihood candidate root locations evaluated at the midpoint of every branch.
      ii. For the top root candidates (default 1%), optimize the root location along their specific branch.

   c Repeat from 3(a) until a stopping condition is met:
      - The difference between likelihoods between the current iteration and the previous iteration is sufficiently small (below user defined parameter `atol`, by default 1×10^{-4}),
      - If early stopping is enabled, the new root location is sufficiently close to the old root location by distance along the branch (below user defined parameter `brtol`, by default 1×10^{-12}) or,
      - More than 500 iterations have been exectuted.
4. Report the best found root candidate, along with its log-likelihood

In order to select the starting branches in search mode, we have developed two strategies: modified MAD and random selection. When using modified MAD, we compute the approximate MAD ranking for each branch via a simplified version of the MAD algorithm for the purposes of computational efficiency. This approximate metric is used to rank branches for selection as initial root positions. There is a possibility that this option will bias the results, so we also provide a random branch strategy for these cases.

During the search, we re-estimate the base frequencies in every iteration to sufficiently optimize the likelihood, and because the cost of optimizing these parameters is small (approximately 10% of overall run time). Furthermore, because we use a non-reversible substitution matrix, the base frequencies might not be stable across every branch of the tree. Therefore, to ensure a good fit, we need to re-optimize the

base frequencies every time. The algorithm for the exhaustive mode is analogous; the core optimization routines are the same as in search mode. The major difference is that now, all branches are being considered as starting branches:

1. For every branch on the tree:
    a Place root at current branch.
    b Initialize numerical parameters:
        - α-shape parameter for discrete Γ rates to 1.0 (if applicable),
        - Character substitution rates to $\frac{1}{4(4-1)} = \frac{1}{12}$
        - Base frequencies to $\frac{1}{4}$
    c Optimize model parameters
        - α-shape parameter for Γ (if applicable, and only every 10 iterations of root search),
        - Character substitution rates,
        - Base Frequencies.
    d Repeat from $1(c)$ until a stopping condition is met:
        - The difference between likelihoods between this iteration and the previous iteration is sufficiently small (below `atol`) or,
        - If early stopping is enabled, the new root location is sufficiently close to the old root location by distance along the branch (below `brtol`).
        - More than 500 iterations have passed.
2. Report the tree in newick format with NHX annotations for every branch:
    - The root position along the branch,
    - The log-likelihood,
    - and the Likelihood Weight Ratio [79].

We set to default the initial model parameters in every iteration (from (3) in search mode and (1) in exhaustive mode) to avoid the numerous local minima, as discussed in [36]. In both modes, there is an upper limit to the number of iterations of 500. In empirical and simulated datasets this limit has never been reached, and only exists to ensure that the program will eventually halt.

In addition to the two search modes, there is an optional early stop mode, which can be combined with either of the root search modes. In this early stop mode, the search will terminate if the root placement is nearly the same twice in a row. This is to say, if the location of the best root position is on the same branch as in the previous iteration *and* the value inferred for the root position along that branch is sufficiently close to the position in the previous iteration, the program will terminate. While the early stop optimization does improve rooting times substantially (approximately

by a factor of 1.7 on some empirical datasets), the likelihood of each root placement will not be fully optimized. In practice, this does not substantially affect the final root placement, but it does render comparison of the likelihood with results from other tools invalid.

We utilize both OpenMP [67] and MPI to parallelize parts of the computation. First, we use the thread level parallelism of OpenMP to optimize each partition (sections of the alignment which are given their own independent model parameters) independently. If there are too few partitions present in the dataset to achieve 'good' parallel efficiency, we also parallelize the transition matrix calculations over the branches. We use process level parallelism to parallelize searches over the initial search locations. This is most efficient in exhaustive mode, where there are many independent searches that can be carried out in parallel. To synchronize the processes, the results from each independent search are written to an append only binary log file. By using an append only file, synchronization of file locations is handled by the underlying filesystem, simplifying multinode checkpointing. At the end of the search, the results (root locations and their associated log-likelihoods, as well as the associated model parameters) in the checkpoint filed are reviewed, and the final step of finding the best root is performed by the master node. Using this strategy, we are able to (with sufficient independent searches) achieve a 'good' parallel efficiency of 0.58 (see figure 3.12). Furthermore, by using this append only logging method, we can also implement checkpointing for the search. If the computations are interrupted during the search, when the search is re-started, the previous results are taken into account, and the search continues where it left off. In order to ensure that no write corruption has occurred during writes to disk and that all writes are complete, a checksum is computed. To compute the checksum, we use the Alder-32 algorithm, which is implemented as a part of `zlib` [55]. To avoid a dependency on zlib for the checksum `RootDigger` includes the algorithm in its own code base.

3.5 Results

To validate `RootDigger`, we conducted several experiments on both simulated and empirical data. Furthermore, we also used the Likelihood Weight Ratio (LWR) [79] to asses the confidence of root placements on empirical datasets. Finally, we investigated the effects of the early stop mode on the final results.

3.5.1 Experimental Design

In the following sections we will describe the experimental setup for both simulated data and empirical data. Here, we will describe how we measured and computed the error for each of the methods. For simulations and empirical data, we computed the *topological* distance from the estimated root (by both `IQ-TREE` and `RootDigger` in search mode) to the true root, and normalized it by the number of nodes in the tree (both internal nodes as well as tips). If the correct root is picked, the distance is zero. For empirical data, the true root was taken to be the root indicated by the outgroup.

Evaluating the exhaustive search mode is difficult, since to our knowledge there are no other tools which perform the same task. Instead, we show the LWR distributions of empirical data which have been annotated onto trees. Additionally, these trees have the true root (again, as indicated by the outgroup) indicated.

3.5.1.1 Simulations

Tests with simulated data were conducted to both, validate the software, and to compare against `IQ-TREE` version 2.0.4 [58] which also implements the non-reversible UNREST model. We created a pipeline to

1. Generate a random rooted tree with ETE3 [37] and random model parameters.
    - Substitution parameters for INDELible were generated by drawing uniformly between 0.01 and 1.01.
    - Frequency parameters for INDELible were generated by an exponential distribution and then normalizing the parameters so that the frequency parameters sum to 1.
    - Otherwise, options for INDELible were left to defaults.
    - Branch lengths were generated via an exponential distribution using a scale parameter of 0.5
2. Simulate an MSA with indelible [22] using uniformly distributed substitution rates from 0.0 to 1.0 with 0.1 added to the result to prevent pathological substitution rates
3. Execute `RootDigger` and `IQ-TREE` [58] with the simulated MSA, given the generated random tree.
4. Repeat from (2) for a total of 100 iterations
5. Compute comparisons
    a Calculated rooted RF distance with ETE3 [73]
    b Mapped root placement onto original tree with the true root.

Both `IQ-TREE` and `RootDigger` were given the same model options for all runs. `RootDigger` was executed with the arguments.

```
rd --msa <MSA FILE> --tree <TREE FILE>
```

By default `RootDigger` uses no discrete Γ rate categories, and currently only supports the UNREST model [94]. `IQ-TREE` was executed with the arguments

```
iqtree2 -m 12.12 -s <MSA FILE> -te <TREE FILE>
```

The `-m 12.12` argument to `IQ-TREE` specifies that the UNREST model should be used [91] and the `-te <TREE FILE>` option constrains the tree search to the given user tree. When given a fully resolved unrooted tree, this has the effect of rooting the tree. We used this option to simulate the operation of `RootDigger`. For all runs, the UNREST model was used. Furthermore, we vary two additional parameters to control dataset size: the number of MSA sites and taxa. In total, we ran 9 simulated trials with MSA sizes of 1000, 4000, and 8000 sites as well as tree sizes of 10, 50, and 100 taxa. The results from these experiments, as well as the execution times, are shown in Figure 3.1.

3.5.1.2 Empirical Data

In addition to simulated data, we conducted tests with empirical data using `IQ-TREE` and additionally MAD [82]. The datasets used are described in Table 3.1 with additional statistics about these datasets provided in Table 3.2. The empirical datasets were chosen from TreeBASE [69, 88] and helpfully provided by fellow researchers [86] to include an existing, strongly supported outgroup. For each of the empirical datasets, we ran `RootDigger` in exhaustive mode to calculate the Likelihood Weight Ratio (LWR) for each branch. We ran the experiments on the datasets with the outgroup included, as well as with the outgroup excluded.

We also performed some preprocessing. In order ensure that all branch lengths in all trees used were specified in substitutions per site, the branch lengths were reoptimized using RAxML-NG [45] version `0.9.0git`. The original model was used when known, otherwise the branch lengths were optimized under GTR+Γ4.

Annotations are suppressed for branches with a small LWR (less than 0.0001). The trees with annotated LWR are shown in Figures 3.2 - 3.10. The analysis errors are summarized in Table 3.3 and runtimes for each method are summarized in Table 3.4.

3.5.2 Effect of early stopping on result

Finally, we investigated the effect of the early stopping criterion on the final LWR results. To do this, we ran `RootDigger` in exhaustive mode on all empirical datasets with early stopping enabled and disabled. For most runs, the results with and without early stopping showed no meaningful difference (difference in LWR less than 0.000001). The dataset that showed the largest difference in LWR is shown in Figure 3.11. In exchange, the runtime for this dataset with early stopping enabled is about 1.7 times faster.

Run time improvements for early stopping in search mode are less pronounced. We were not able to measure any large differences in results or speed in search mode between early stopping enabled and disabled. We suspect that this is because the speed gain from early stopping in exhaustive mode is primarily due to it "skipping" low likelihood branches, which do not contribute significantly to the LWR.

3.5.3 Parallel efficiency

Finally, we also evaluated the parallel efficiency of `RootDigger`. Figure 3.12 plots the speedup (how many times faster than 1 node) vs perfect efficiency for dataset DS7. We choose DS7 because it is one of the larger datasets at hand, and therefore is ideal for displaying the strengths and weaknesses of `RootDigger`'s parallelization strategy. Results were computed on a cluster, using MPI to communicate between nodes with `RootDigger`'s exhaustive mode. The parallel efficiency ranges from 0.94 on 2 nodes, to 0.50 on 32 nodes, each with 16 cores.

3.6 Discussion

Compared to `IQ-TREE`, `RootDigger` performs competitively, as can be seen in both sides of Figure 3.1. The results on simulations are mixed, with `IQ-TREE` performing slightly better in terms of root placement under all simulated scenarios. `RootDigger` is faster than `IQ-TREE` on all datasets we tested. When analysing empirical data, `RootDigger` also performed well, though not as well as `IQ-TREE` or MAD for most datasets, yet produced minimal errors in most cases. A notable exception is dataset DS3, for which `RootDigger` obtained a better result than either MAD or `IQ-TREE`. Examining the dataset with `RootDigger`'s exhaustive mode (see Figure 3.8), we see that there is a number of branches with good likelihood weight support for a root placement. This suggests that there is conflicting signal as to the root location for this dataset, which naturally leads to confusion in generally reliable methods like MAD.

In general, the exhaustive mode is more successful at identifying the correct root location (see Figures 3.2, 3.3, 3.6, 3.7, 3.8, and 3.9). This is to be expected, since the exhaustive mode performs a substantially more thorough search for the best root location. Nonetheless, this shows that `RootDigger` is successful not only at identifying the correct root location, but also at identifying any uncertainty or ambiguous signal for the dataset at hand.

Parallel Efficiency of `RootDigger` is acceptable, but could be further improved. Currently, it seems that losses in efficiency are largely due to the fact that different initial search locations require different amounts of time to complete. When this happens, some of the nodes finish early, and must wait for the remaining nodes to complete their computations. Due to this behavior, the parallel efficiency of `RootDigger` is dataset dependent. Fortunately, this behavior generally only manifests itself when each node has a small number of initial starting positions assigned to it. When this is the case, small variations in runtime are not given a chance to "average out" over many initial starting positions. In contrast, when a dataset is large with respect to the number of taxa, the number of initial starting positions increases and consequently the average time to complete computational work per node converges to an average amount. Nonetheless, `RootDigger` could benefit from a heuristic method to intelligently assign initial search locations to nodes. For example, a strategy that could work is to compute initial likelihoods for each branch, and then assign each core a roughly similar amount of total likelihood. The hope here is similar initial

likelihoods are correlated with runtime, though this would need to be examined before implementing such a heuristic.

3.7 Conclusions

In Huelsenbeck [36], it was shown that the prior probability of a root placement on a sample tree did not have a strong signal when using a non-reversible model of character substitution. While performing our verification of `RootDigger` using empirical data, we found that this was often not the case. For example on the AngiospermsCDS12 dataset (see figure 3.6), we found a clear signal for the root placement, both with and without the outgroup.

Even in cases when the signal was not as strong, for example SpidersMitochondrial (see figure 3.4), there is a substantially stronger signal for root placement than the results in Huelsenbeck [36] would suggest we should obtain with this kind of analysis (which is to say, analysis using a non-reversible model). Those results in Huelsenbeck would suggest that we would essentially not be able to recover *any signal at all*. Instead, the signal appears to be moderately strong, at least most of the time, as `RootDigger` managed to obtain the correct root in 3 out of 8 empirical datasets 100% of the time while using search mode. It is interesting to note the ficus dataset, which showed at least marginal support for the root on nearly all branches of the tree, but `RootDigger` nonetheless managed to correctly identify the root using the exhaustive mode. We suspect that this is due to Huelsenbeck performing the analysis on a 4 taxon tree with the distantly related taxa frog, bird, mouse, and human. By only using 4 distantly related taxa, the rate matrix is less constrained by the data present, which may lead to over-fitting. In contrast, for "localized clades" we believe that we have shown that the methods presented here will typically produce a clear signal for the rooting of a tree, and when they do not we can identify such situations with the use of `RootDigger`'s exhaustive search mode.

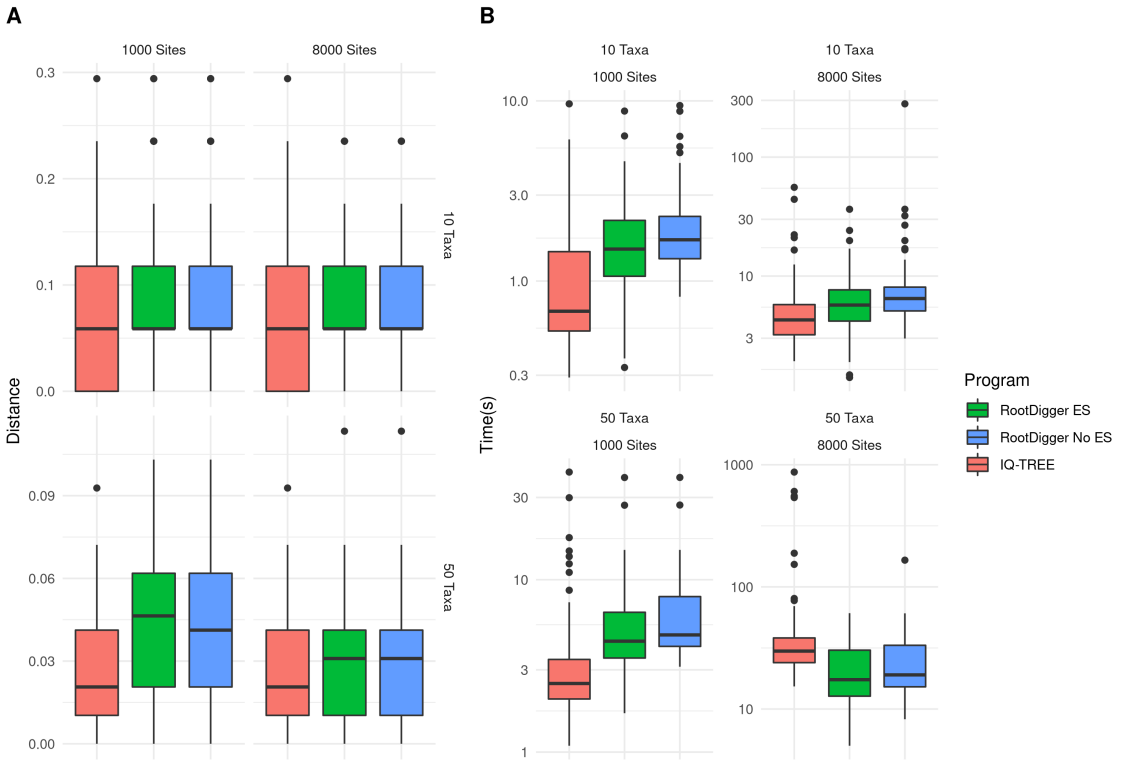

Figure 3.1: Box plot of results and execution times for `IQ-TREE` and `RootDigger` on simulated data with and without early stopping enabled.

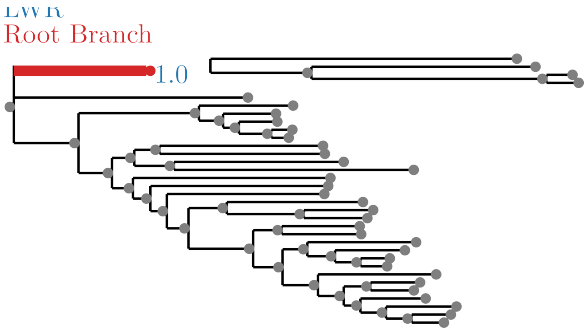

Figure 3.2: SpidersMissingSpecies dataset analyzed without an outgroup. LWR is the Likelihood weight ratio of placing a root on the branch. The true root branch is indicated in red.

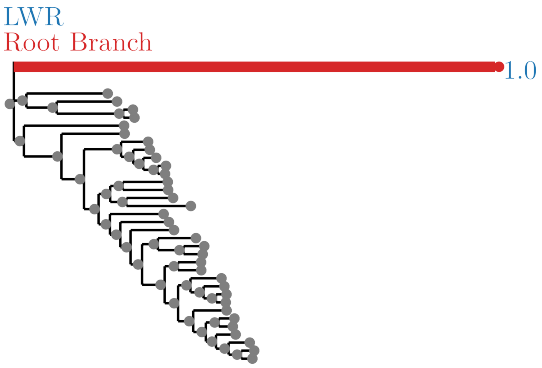

Figure 3.3: SpidersMissingSpecies dataset analyzed with an outgroup. LWR is the Likelihood weight ratio of placing a root on the branch. The true root branch is indicated in red.

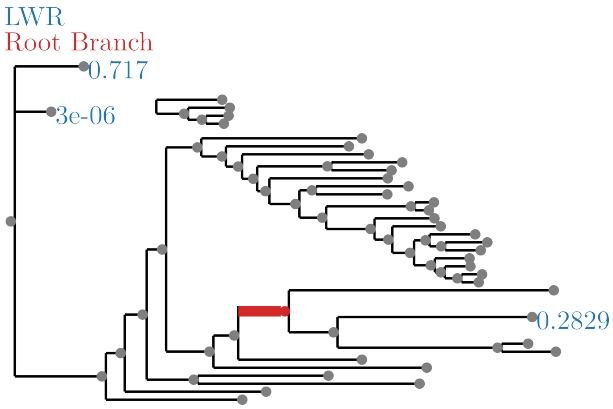

Figure 3.4: SpidersMitocondrial dataset analyzed without an outgroup. LWR is the Likelihood weight ratio of placing a root on the branch. The true root branch is indicated in red.

Table 3.1: Table of empirical datasets used for validation.

Name	Dataset	Original Model	Model Used	Source
DS1	AngiospermsCDS12	GTR+Γ	UNREST + Γ4	[70]
DS2	AngiospermsCDS	GTR+Γ	UNREST + Γ4	[70]
DS3	Grasses	GTR+ G4 + I	UNREST + Γ4	[9]
DS4	Ficus	GTR+ G	UNREST + Γ4	[10]
DS5	SpidersMissingSpecies	NA[a]	UNREST + Γ4	[49]
DS6	SpidersMitocondrial	NA[a]	UNREST + Γ4	[49]
DS7	Beetles	GTR+G4	UNREST + Γ4	[86]
DS8	BeetlesHomogeneous	GTR+G4	UNREST + Γ4	[86]

[a] The paper states that PartitionFinder was used, but the results were not provided.
[b] The dataset is partitioned, and the partition file was provided. UNREST was used instead of any substitution matrices, but invariant sites and rate categories was preserved.

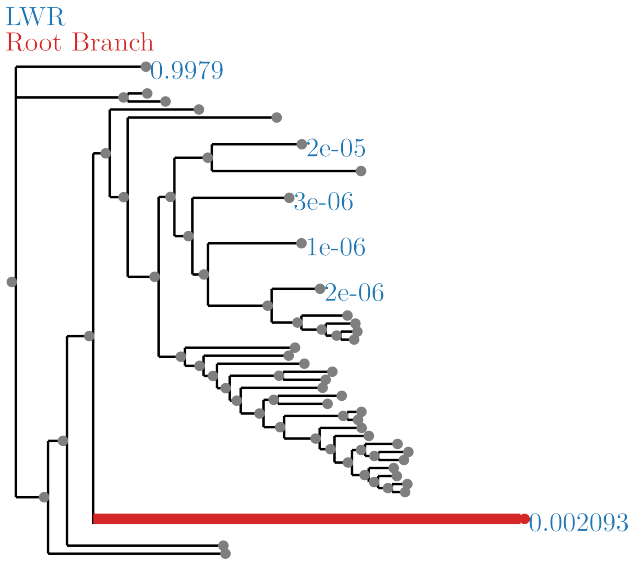

Figure 3.5: SpidersMitocondrial dataset analyzed with an outgroup. LWR is the Likelihood weight ratio of placing a root on the branch. The true root branch is indicated in red.

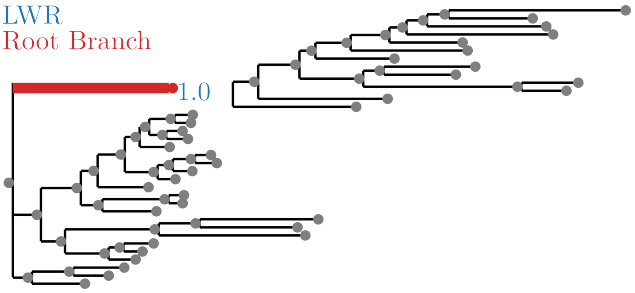

Figure 3.6: AngiospermsCDS12 dataset analyzed without an outgroup. LWR is the Likelihood weight ratio of placing a root on the branch. The true root branch is indicated in red.

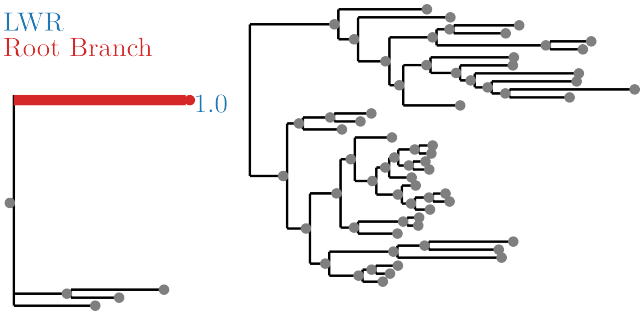

Figure 3.7: AngiospermsCDS12 dataset analyzed with an outgroup. LWR is the Likelihood weight ratio of placing a root on the branch. The true root branch is indicated in red.

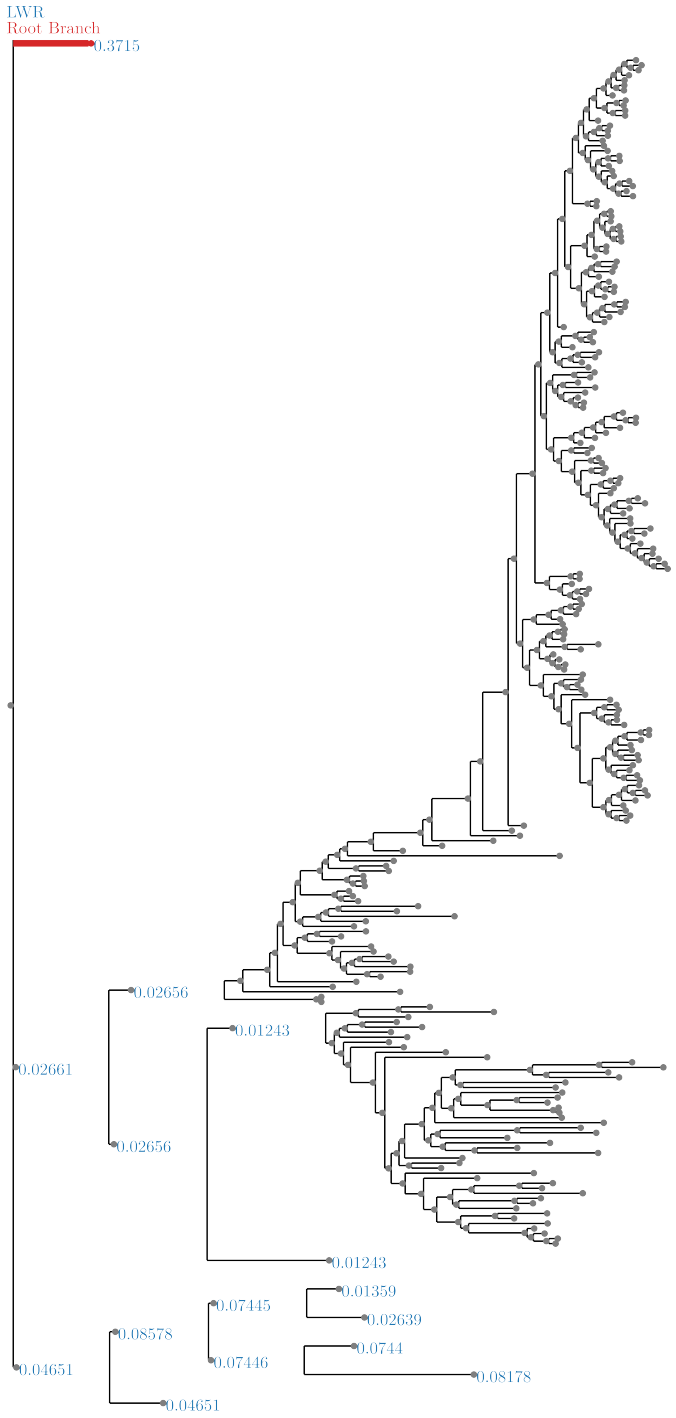

Figure 3.8: Grasses dataset analyzed without an outgroup. LWR is the Likelihood weight ratio of placing a root on the branch. The true root branch is indicated in red.

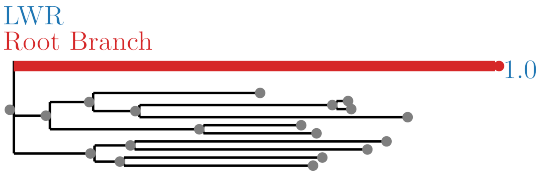

Figure 3.9: Beetles dataset analyzed without an outgroup. LWR is the Likelihood weight ratio of placing a root on the branch. The true root branch is indicated in red.

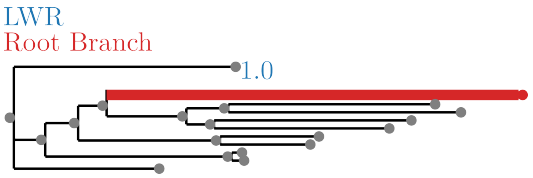

Figure 3.10: BeetlesHomogeneous dataset analyzed without an outgroup. LWR is the Likelihood weight ratio of placing a root on the branch. The true root branch is indicated in red.

Table 3.2: Table of statistics for the emperical datasets.

Name	Tree Diameter[a]	Root Branch Length[b]	Ratio[c]	#Genes	#Taxa	#Sites
DS1	1.1204	0.2276	0.203	1308	35	864,029
DS2	0.5236	0.1089	0.208	1308	35	1,296,043
DS3	0.6657	0.0856	0.129	3	245	4,973
DS4	0.0985	0.0316	0.320	5	200	5,552
DS5	0.0628	0.0099	0.158	1019	33	1,097,842
DS6	0.4189	0.0283	0.068	15	34	12,479
DS7	2.5334	0.0842	0.033	2948	14	4,098,894
DS8	1.6601	0.0539	0.032	101	14	186,499

[a] Defined here to be the longest path between two taxa.
[b] The length of the root branch if the tree was *unrooted*.
[c] Root Branch Length over Tree Diameter.

Table 3.3: Table of empirical datasets used for validation and results. RD Distance and IQ Distance are the average topological distances over 100 runs from the inferred root to the true root normalized by the number of nodes (both tips and internal nodes). Similarly for MAD the distance is also normalized by the number of nodes but only 1 iteration was performed.

Dataset	RD Distance[a b c]	IQ Distance[a]	MAD Distance
DS1	0.000	0.000	0.000
DS2	0.075	0.000	0.000
DS3	0.002	0.004	0.025
DS4	0.038	0.005	0.003
DS5	0.000	0.000	0.000
DS6	0.031	0.015	0.000
DS7	0.000[d]	0.000	0.000
DS8	0.158	0.000	0.000

[a] Averaged over 100 independent executions
[b] In early stop mode
[c] In search mode
[d] Results obtained using UNREST (without rate categories)

Table 3.4: Table of empirical datasets used for validation and results. Search and Exhaustive times are for the respective modes of `RootDigger`.

Dataset	Search Time[a b]	`IQ-TREE` Time[a c]	Exhaustive Time	MAD Time
DS1	8.1m	48m	340m	0.00m
DS2	24m	114m	554m	0.00m
DS3	2.5m	1.5m	123m	0.02m
DS4	0.4m	0.4m	45m	0.00m
DS5	6.8m	25m	162m	0.00m
DS6	0.2m	1.5m	7m	0.00m
DS7	167m[d]	327m	441m	0.00m
DS8	4.8m	19.2m	81m	0.00m

[a] Averaged over 100 independent executions
[b] In early stop mode
[c] In search mode
[d] Time obtained with UNREST+G4 (with rate categories)

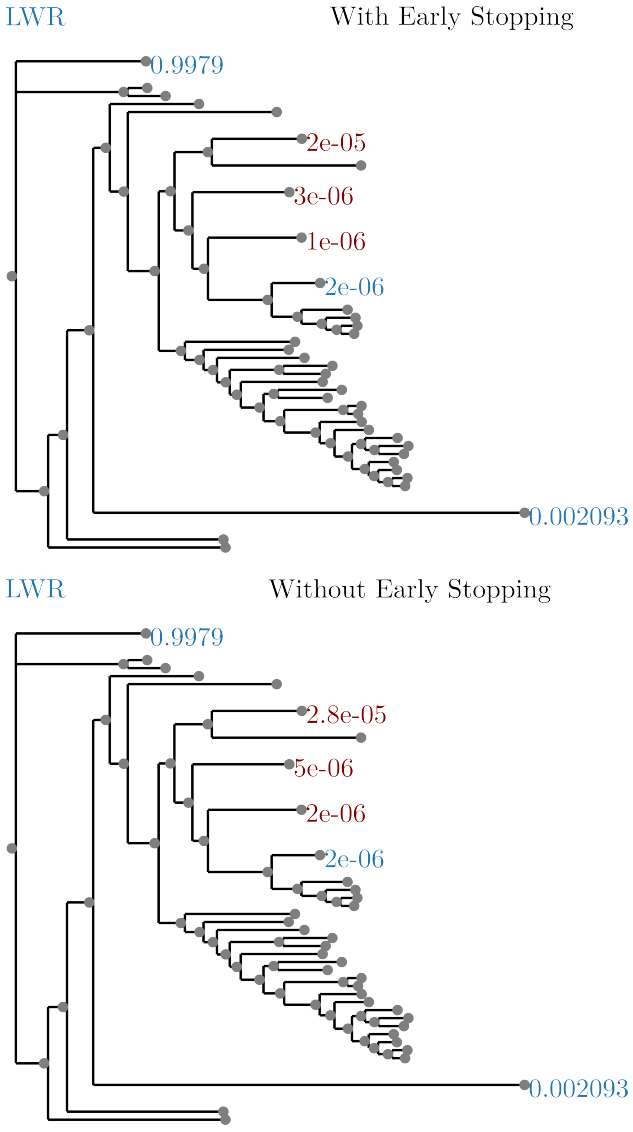

Figure 3.11: Effect of early stopping on results. Dataset is SpidersMitocondrial and has the largest observed difference of LWR between with and without early stopping.

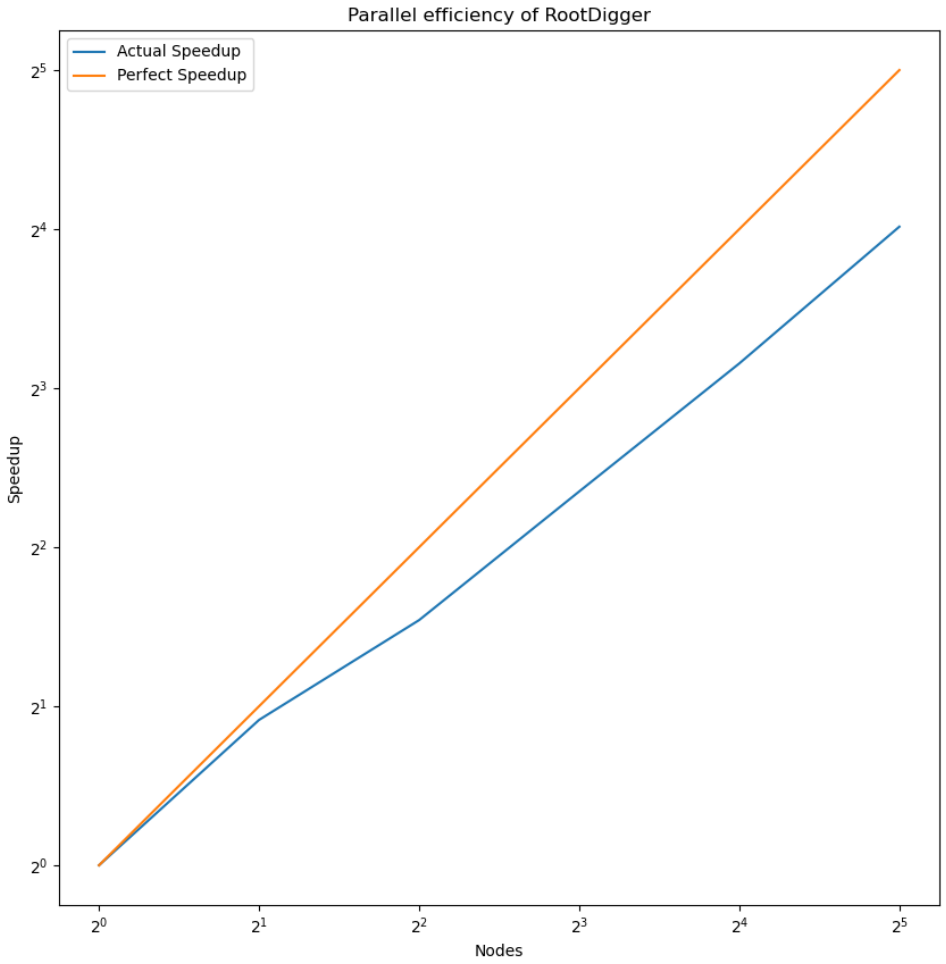

Figure 3.12: Plot depicting parallel efficiency, which is log nodes vs log speedup. Trials were run on 1, 2, 4, 8, 16, and 32 nodes with 16 threads per node using DS7 in exhaustive mode with early stopping turned off. The parallel efficiency ranges from 0.94 on 2 nodes to 0.50 on 32 nodes.

4. Phylourny

This chapter is based on the following peer-reviewed software article:

> **Ben Bettisworth**, Alexander I. Jordan, Alexandros Stamatakis. "Phylourny: Efficently Calculating Elimination Tournament Win Probabilities via Phylogenetic Methods" *Statistics and Computing*, Volume 33, Issue 4, pp. 80, https://doi.org/10.1007/s11222-023-10246-y

> Amongst all unimportant subjects, football is by far the most important.
>
> Pope John Paul II

4.1 Introduction

Predicting the per-team win probabilities of a knock-out tournament (alternatively bracket-based or elimination tournament) given a pairwise win probability matrix P, can become computationally expensive if a high degree of numerical accuracy shall be attained. In some cases the prediction will need to be computed thousands or even millions of times, for instance, to quantify the impact of slight perturbations of the pairwise win probability matrix P on the per-team tournament win probability. Given a tournament with n teams, one needs to evaluate a polynomial with $\approx 2^n$ terms to fully and exactly calculate the tournament win probability for a specific team via a naïve implementation (see the Section 4.2.1 for details). To calculate this tournament win probability for every team, an additional n such polynomials must be evaluated. Alternatively, one typically deploys stochastic simulations (again

given a pairwise win probability matrix P), over the tournament tree to approximate the per-team win probabilities. Typically, this is computationally more efficient than computing the aforementioned polynomial, but comes at the cost of reduced numerical precision of the results [12, 17].

Prior work for predicting knock-out tournaments has generally focused on producing accurate outcomes, and not on the efficiency of the simulations per se. Consequently, these works generally deploy a statistical model of pairwise match win probabilities to predict match winners, such as the Bradley-Terry model, or an Independent Poisson Model which is also used in this work. Using such models, parameters are inferred from historic matches, and these parameters are subsequently used to predict the outcome of individual tournament matches [27, 50]. Alternatively, researchers have attempted to devise models for directly predicting the final ranking of teams in a tournament without taking into account the tournament (tree) structure [83]. These models generally only infer a few sets of parameters, that is, only the most likely outcome is used to generate a prediction.

In the following, we propose a novel algorithm to efficiently ($O(n^2)$ which translates to a runtime improvement by 2-4 orders of magnitude) *and* exactly compute win probabilities for single elimination tournaments, given a square pairwise win probability matrix P. Our method was inspired by an observation [93] that the Felsenstein Pruning Algorithm [21] can more generally be interpreted as an efficient way to compute polynomials of a high degree. We implement and make available our new method in an open source software tool named `Phylourny` (the name is a pun, on the words phylogeny and tournament). We experimentally demonstrate the order(s) of magnitude runtime improvement of `Phylourny` over stochastic tournament simulations and naïve evaluations of the polynomials. We also experimentally determine the differences in numerical accuracy between `Phylourny` and the stochastic simulation approach.

Finally, we showcase the new predictive possibilities that emerge through this increase in computational efficiency. By example of two recent tournaments, one with a large amount of data and one with a small amount of data (a basketball and football tournament respectively), we show how slight yet reasonable perturbations of P affect prediction uncertainty by calculating millions of tournament win probabilities within hours on a standard laptop. The main contribution of this paper is the substantially more computationally efficient approach to computing tournament win probabilities given a pairwise win probability matrix P. To this end, in our case studies we deploy a simplified version of a standard model from [50] to compute P but do not propose improved approaches for computing P. Instead, we show to which extent slight alterations of P affect tournament win probabilities. Such studies are now feasible in acceptable times with `Phylourny`.

4.2 Methods

We initially describe our algorithm for exactly and efficiently calculating the tournament win probabilities in Section 4.2.1 and provide software implementation details

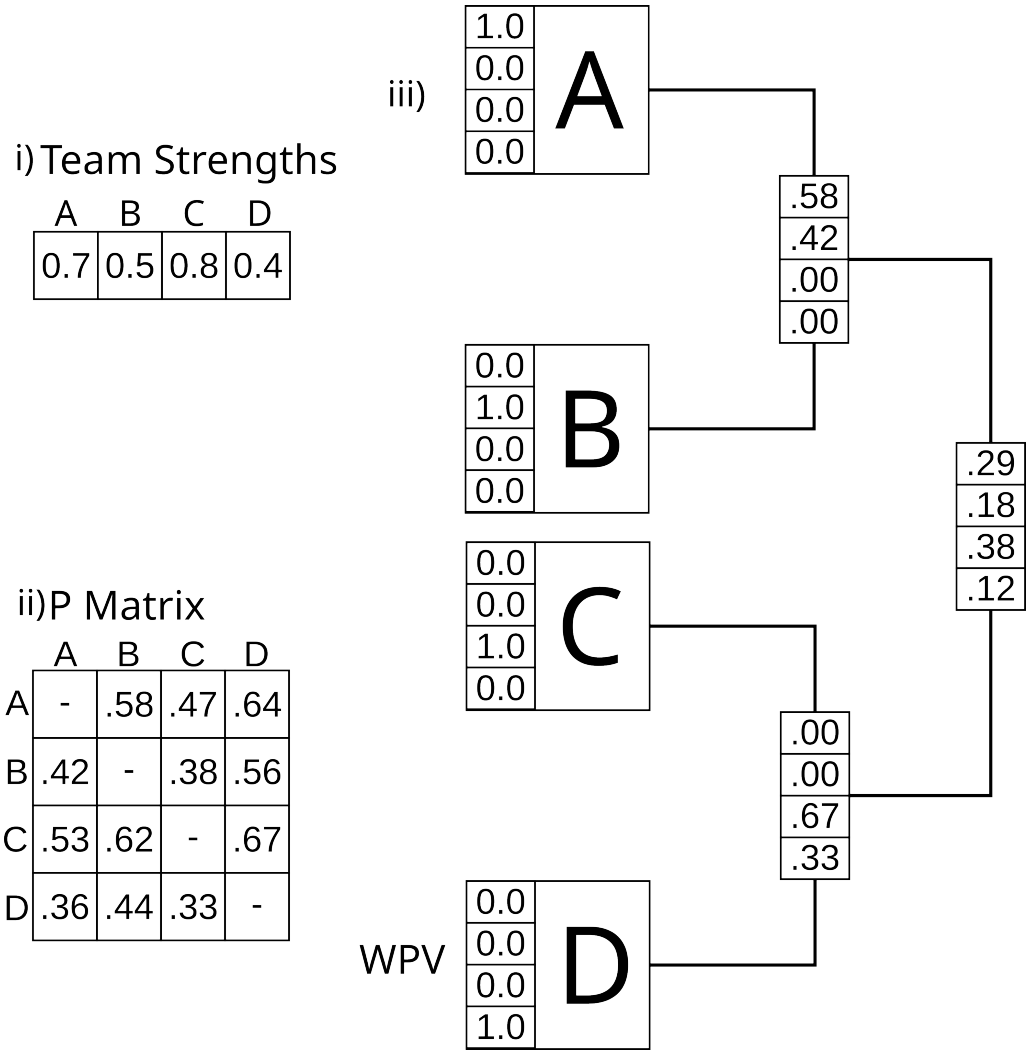

Figure 4.1: A single elimination tournament with $n := 4$ teams.i) A set of example team strength parameters. ii) The P matrix created from the team strength parameters using a simplified likelihood model, where win probabilities are equal to $r_a/(r_a + r_b)$. iii) A tournament with computed Win probability vectors (WPVs).

in Section 4.2.2. Thereafter, we describe our simple models for calculating reasonable P matrices in Section 4.2.3 and outline how we deploy Markov Chain Monte Carlo (MCMC) sampling in 4.2.4 to quantify the prediction uncertainty induced by slight alterations of P.

4.2.1 The `Phylourny` Algorithm

We initially provide some definitions and introduce some notation.

The Win probability vector (WPV) for a given node in the tournament tree is a vector containing the probabilities of observing a specific team at that node, denoted by $R \in [0, 1]^n$, where n is the number of teams. Evidently, all tournament tree nodes below the tournament final, that is, the root of the tree, will comprise some entries

that are equal to zero with the leaves being represented by the canonical unit vectors. For an illustration, see panel iii) in Figure 4.1.

Let $P_{a\vdash b} \in [0, 1]$ denote the pairwise probability of team a winning over team b in a single match, that is, the probability that "team a beats team b". By convention, we define $P_{a\vdash a} = 0$ for any team a. In the simplest tournament with only two teams, a and b, there is only a single match. The pairwise win probability matrix is given by

$$P = \begin{pmatrix} P_{a\vdash a} & P_{a\vdash b} \\ P_{b\vdash a} & P_{b\vdash b} \end{pmatrix},$$

and the WPV for the single node in this tournament is

$$R = (R_a, R_b) = (P_{a\vdash b}, P_{b\vdash a}). \tag{4.1}$$

Because this constitutes a trivial case, the calculation is straight-forward. To recursively extend this to larger trees, we rewrite the above expression by also using the respective child nodes. First, we introduce the child WPVs as $V = (1, 0)$ and $W = (0, 1)$ leading to the expression

$$R_a = (P_{a\vdash a} \times W_a + P_{a\vdash b} \times W_b) \times V_a \tag{4.2}$$

for the win probability R_a of team a, assuming the team can only enter the match via the first child of the node as indicated by $V_a = 1$ and $W_a = 0$. As such, $P_{a\vdash a} \times W_a$ vanishes regardless of the value of $P_{a\vdash a}$, and Eq 4.2 reduces as given in Eq 4.1.

In general, for any number of teams n, the WPV R at any given node can be calculated from the respective child node WPVs V and W and the pairwise win probability matrix P as

$$R = V \odot \left(WP^\top\right) + W \odot \left(VP^\top\right), \tag{4.3}$$

where $\odot$ denotes the element-wise product. Please note that Eq 4.3 is a generalized restatement of Eq 4.2 using matrix and vector notation, and accounts Thereby, we account for any team entering the match via either child, but for single elimination tournaments at most one of V_a and W_a can be positive as each team only has one route to the finals (and thereby to this node). The WPV at the root can be efficiently computed via a post-order traversal of the tournament tree, that is, by computing WPVs at the nodes bottom-up from the tips/leaves toward the final/root. Figure 4.1 depicts a simple example with the pairwise win probabilities calculated by normalizing relative team strengths.

In some tournaments, $P_{a\vdash b}$ will correspond to a "best of k" series of play-off matches instead of a single match, as for example in the National Basketball Association (NBA) playoffs. Further, this k can vary over the duration of the tournament since early matches are often "best of 1" with $k := 1$, whereas later matches might be "best of 5" with $k := 5$. We can seamlessly account for this by introducing $P(k)$, a node-dependent pairwise win probability matrix for a "best of k" series.

4.2.2 The `Phylourny` Software

The open-source `C++` implementation of our algorithm is available via GitHub at https://github.com/computations/phylourny under GNU GPL version 3.0. The software only requires `CMake` to build and `git` to download. `Phylourny` also implements stochastic (that is, simulation based) as well as naïve polynomial tournament win probability calculations for the sake of conducting run time and numerical precision comparisons. Finally, it also offers the simple models for devising reasonable P matrices and conducting Markov Chain Monte Carlo (MCMC) sampling presented in the following Sections 4.2.3 and 4.2.4. Finally, `Phylourny` has a software quality score of 7.7 as rated by the software quality analysis tool SoftWipe [96], which places `Phylourny` in the top 10% of scientific software tools included in the SoftWipe benchmark. Version `v1.2.1` was used to perform the uncertainty analyses presented here.

Computing the P matrix based on the Poisson likelihood model (which is discussed in the next section) is comparatively computationally expensive. Therefore, to expedite these computations, we parallelized the computation of the likelihood over the historic matches using OpenMP [68]. Despite this parallelization, the computation of the likelihood score still accounts for approximately 90% of the overall run time of the Poisson model based MCMC analysis.

To perform an analysis with `Phylourny`, a list of teams who will participate in the elimination tournament needs to be provided as an input file. `Phylourny` can then compute a win probability when given a probability matrix P, which must be provided as a CSV file. Alternatively, `Phylourny` can conduct an MCMC search of the parameter space of the Independent Poisson Likelihood Model (discussed in Section 4.2.3). In this case, a list of historical matches needs to be provided in a CSV file. The results from the MCMC search will be summarized in 3 output files with three different summaries: the maximum likelihood prediction (MLP) which is the prediction using the parameters with the highest likelihood; the maximum marginal posterior prediction (MMPP) which is the prediction averaged over all posterior samples; and the list of samples taken from the posterior during the MCMC search. The MLP and MMPP are discussed in more detail in Section 4.2.4.

4.2.3 The Independent Poisson Likelihood Model

The success of a tournament prediction heavily relies on the P matrix, that is, the methods used to calculate and also the data used to evaluate its likelihood. Thus, improved methods for obtaining this matrix constitute an active area of research. Improving upon them is beyond the scope of this thesis [31, 38, 42, 52]. As a simple yet effective reference model, we adapt the "Independent Poisson Model" from [50] to model the pairwise win probabilities based on historical match data. In a nutshell, two competing teams are assumed to independently score points under respective Poisson distributions, with parameters driven mainly by the difference of the teams' strengths. The win probability of a team is the probability to score more points than the opponent as given by the Skellam distribution that describes the difference between two independent Poisson random variables.

Our version of the Independent Poisson Model is a straightforward implementation of the model described in [50], slightly modified by removing the constraint that the team strength parameters need to sum to zero. During our MCMC search, we constrain the strength parameters to be between 0.0 and 1.0, which has a similar effect. Additionally, we remove the distinction between home and away games to further simplify the model. A home advantage parameter could be integrated into the model in a future version of `Phylourny`. Let M denote a series of historical matches (a, b, g_a, g_b), where a and b are the teams and g_a and g_b are the goals scored by each team, respectively. Then, the likelihood of the Independent Poisson Model is given by

$$L(R, \rho) = \prod_{(a,b,g_a,g_b)\in M} \left(\frac{\lambda_{a\vdash b}^{g_a}}{g_a!} e^{-\lambda_{a\vdash b}} \times \frac{\lambda_{b\vdash a}^{g_b}}{g_b!} e^{-\lambda_{b\vdash a}} \right), \tag{4.4}$$

where $R = (r_a, r_b, \dots) \in [0, 1]^n$ is the parameter vector of team strengths that reflect the skill levels of each team, and $\rho \in \mathbb{R}$ represents an "average" skill level among all teams in the Poisson parameter

$$\lambda_{a\vdash b} = e^{r_a - r_b + \rho}.$$

The expression in Equation 4.4 is useful to describe the model. However, it is unsuitable for computation in general as many sports have score counts which are substantially larger than that of football. For example, basketball scores are generally 10-80 times higher. The issue is that when scores are large, some terms in the computation simultaneously become very large (for example $g_a!$) and very small (for example $e^{-\lambda_{a\vdash b}}$). This introduces substantial numerical deviations, which can potentially be amplified by the MCMC search, as it might sample numerical error under unfavorable conditions. If numerical deviations yield likelihoods scores that are better than the exact analytical likelihood scores, the MCMC search will preferably sample points in parameter space that maximize the numerical error. While a strong prior can prevent this in many cases, it is preferable to devise more numerically stable computations, to prevent this type of potential error a priori. To alleviate this, we deploy the standard solution to reduce numerical error by computing the log-likelihood instead. As we show, this provides sufficient numerical stability to also apply this model to basketball.

Additionally, the particular model we use for the sake of the example, might likely not be correct for many sports, including basketball. This is because a Poisson distribution always has a mean equal to its variance. However, this assumption does likely not hold for sports such as basketball, where the score variance is generally much smaller than the score mean. For example, in the dataset for the basketball tournament we analyze later in this work, the mean score is ≈ 70 and the standard deviation is ≈ 12. Nonetheless, we choose to use the Independent Poisson Model as it strikes a good balance between realism and simplicity to substantiate our claims that novel types of statistical analyses are feasible because of the computational savings of `Phylourny`.

While we do present and implement as open-source code the Independent Poisson Model here and use this model for MCMC analyses (see below), `Phylourny` does by

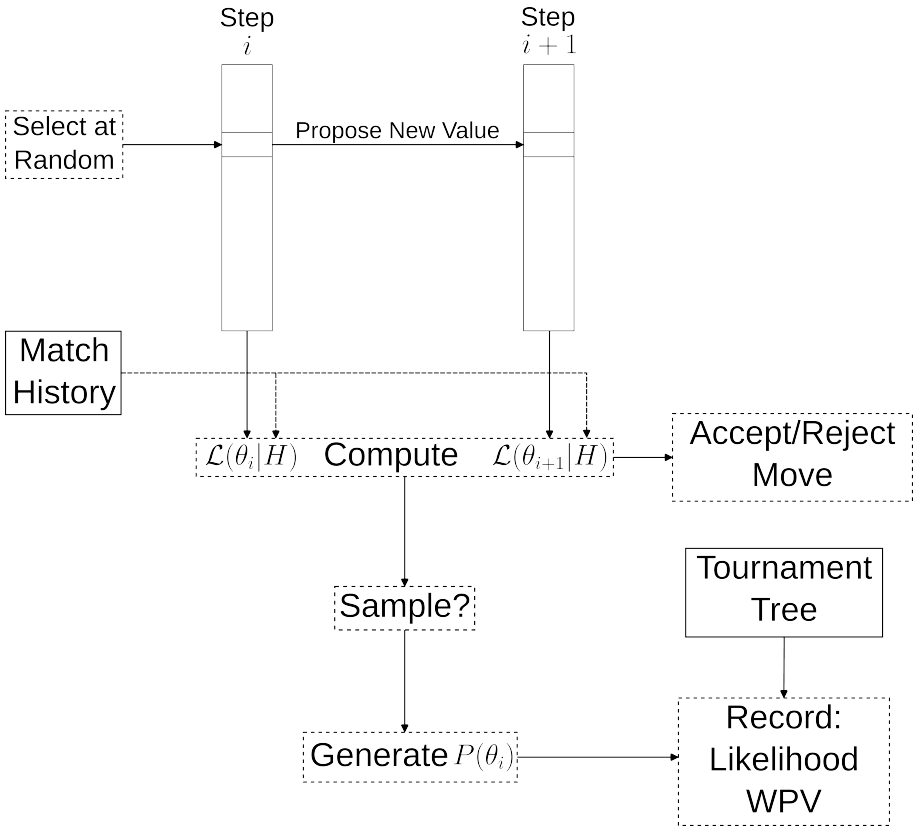

Figure 4.2: Diagram showing an MCMC step for `Phylourny`. The function $\mathcal{L}(\cdot|H)$ denotes the likelihood function and θ_i denotes the parameters of the model for the matrix P. For the Independent Poisson Model, θ comprises R and ρ and $\mathcal{L}(\theta|H) = L(R, \rho)$ from Eq 4.4. The decision whether to accept θ_{i+1} depends on the prior (and in the case of a Hasting correction the proposal distribution) in addition to the likelihood. Solid borders represent inputs to the algorithm, while dashed borders steps of the algorithm.

no means rely on this particular model. In fact, any model which can compute a pairwise win probability matrix P can be used. Furthermore, any parametric model can be used to perform the MCMC analyses we describe next.

4.2.4 Sampling the P Matrix via MCMC

To generate a sample of reasonable P matrices that accurately reflect a given match history and to quantify the uncertainty of the tournament win probabilities at the WPV of the final, we deploy MCMC sampling via the Metropolis-Hastings algorithm [57]. A diagram showing an example sampling step is given in Figure 4.2.

At each MCMC step, a new set of model parameters for the Independent Poisson Model is proposed yielding a new pairwise win probability matrix P', and the likelihood of these parameters is computed, i.e. $L(R, \rho)$. If the proposed model parameters are accepted, then the WPV of the tournament is computed under P' and recorded as a sample together with the likelihood of the corresponding model. Op-

tionally, the sample set can be thinned for saving disk space by taking a sample only every n generations.

At the start of the MCMC chain, all parameters are initialized (that is, the team strengths r_i and the scale parameter ρ) to 0.5. In each MCMC step, a parameter is selected at random with equal probability. If a team strength r is selected, then a new strength r_i' is proposed according to a Beta distribution with $\alpha := \beta := 1.5$. The corresponding density function is denoted by $b_{1.5}$. If the average strength ρ is selected, then a new average strength ρ is proposed by adding a value drawn from a Normal distribution with $\mu := 0.0$ and $\sigma := 0.1$, or simply $\rho' \sim \mathcal{N}(\rho, 0.1^2)$. Both proposal functions have no particular meaning, and could be replaced with other proposals so long as they satisfied a few requirements. First, the average skill level ρ must be allowed to vary to any value in $\mathbb{R}$. Second, the strength parameters can be shifted, as a group, by some constant and still have the same likelihood. Therefore, to improve convergence of the chain, we found it best to constrain the strength parameters by using a proposal with a single mode, which has the effect of preferring an average relative strength of 0.5. We could have implemented this as a prior with the same requirements on the strength parameters, however we found it simpler to satisfy these requirements with the proposal than to implement these requirements as a prior distribution.

The proposal process is symmetric in the average strength, but non-symmetric in any team strength, so we calculate the Hasting's ratio as 1 and $b_{1.5}(r)/b_{1.5}(r')$, respectively. The acceptance ratio is then computed as the product of the likelihood ratio, the prior ratio, and the Hasting's ratio. A value for the acceptance ratio larger than 1 is reduced to 1. We accept a proposed new team strength with a probability that is equal to the acceptance ratio. We implemented and tested several priors including a Normal distribution, a Beta distribution, and an Uniform Distribution. None of the priors had a strong effect on the results, so we chose to use a Uniform prior, for the sake of simplicity. Finally, we sample the chain every 100 generations in order to thin the samples. Thinning is performed so that some result files, particularly the file containing the samples, do not become excessively large.

The MCMC sampling procedure should be continued until the chain has reached "apparent convergence" as true convergence can only be attained if the MCMC sampling is executed infinitely. Further, only the lack of convergence can be assessed via appropriate diagnosis tools. Hence, as assessing the convergence of MCMC is impossible, in our experiments, we only draw a fixed number of samples. However, computing the WPV of a single sample using `Phylourny` is computationally inexpensive. Therefore, we are able to compute an extremely large number of samples within an acceptable amount of time. For a football tournament with $n := 16$ teams (the UEFA 2020 knock-out stage), we can evaluate 10 million proposals under the Independent Poisson Model which result in exactly 100 thousand WPV samples after thinning, within approximately 51 seconds using a standard laptop. This corresponds to approximately 1961 exact calculations of the tournament final WPV and $196,078$ likelihood evaluations per second. We believe that using 100 thousand samples is justified, as the state space for *this* specific tournament is not excessively

large, and should be sufficiently sampled with this number of samples, particularly since we explore the parameter space for ≈ 10 million generations.

We discard the first 10 thousand samples (10% of samples) as burn in to compute summary statistics. Once we have obtained all sampled WPVs from the MCMC procedure, we can compute two predictions: the maximum likelihood prediction (MLP), or the maximum marginal posterior prediction (MMPP). The MLP is simply the prediction given by the P matrix that yielded the highest likelihood score, whereas the MMPP is the average prediction over all samples. Because an MCMC procedure will sample the posterior with a probability distribution hopefully approximating the true posterior, the average over all samples is approximately the average of the posterior. The difference between these two predictions is one of philosophy rather than mathematics, as they encapsulate distinct interpretations about what "really" matters. The school of thought advocating the MLP, claims that the only thing that matters is the *most likely* outcome, regardless of the underlying distribution, whereas the school of thought supporting the MMPP claims that the *totality of evidence* is what matters.

4.3 Case Studies, Experimental Setup, and Hardware

We showcase and assess the runtime and the numerical performance of our method on two historical tournaments. We apply `Phylourny` to the 2020 UEFA European Football Championship (UEFA 2020) and the 2022 NCAA Division I Men's Basketball Tournament (NCAA 2022) to perform an uncertainty analysis on the tournament results. As input to `Phylourny` we use historical match data from games played prior to the elimination phase to conduct MCMC searches, as described in Section 4.2.4.

We chose the UEFA 2020 and NCAA 2022 tournaments for several reasons which we think best allow us to showcase our method. First, UEFA 2020 and NCAA 2022 cover different sports, which allows us to show that `Phylourny` is not dependent on a specific sport. Second, UEFA 2020 and NCAA 2022 have very different sizes, as UEFA tournaments have a small number of competitors (typically 16 teams) while NCAA tournaments have a large number (64 teams). Finally, the amount of historic match data available for NCAA tournaments is generally much more extensive than that of UEFA tournaments as NCAA has an extensive playoff season with teams which are permanently established. Therefore, UEFA 2020 is the "small" case and NCAA 2020 is the "large" case. These two cases represent the extremes of tournament configuration in terms of size and matches before the tournament. Therefore they allow us to explore the entire range of `Phylourny`'s performance.

All input data and relevant output files of `Phylourny` for the experiments that we describe in more detail below are available at https://github.com/computations/phylourny.

4.3.1 UEFA 2020 and NCAA 2022 historical match input data

We used the group stage matches for UEFA 2020 to perform our analysis. These matches are played in order to determine the "seeding" for the knockout round. In

order to support UEFA 2020 representing the "small" case, we elected to *not* include qualifying round data, which are the matches played in order to determine who will enter the group stages. There were a total of 37 games, including games played to break ties which arose during the group stage, which we included as historical match data in our analysis.

Because there is a more extensive pre-season to what is colloquially referred to as "March Madness" in the U.S. when compared to qualifying rounds for football tournaments, there is a more extensive dataset we can use for likelihood calculations. Therefore, a total of 1795 matches were eligible, that is involving at least one team which participated in the NCAA 2022 tournament, for use in our uncertainty analysis.

4.3.2 MCMC Analyses

As described in Section 4.2.4, the search was conducted via the Metropolis-Hastings algorithm [57]. For the UEFA 2020 and NCAA 2022 uncertainty analyses, 100,000 samples were collected with thinning enabled. We present summary statistics for the most likely of these samples (the 99.9%-ile) for the UEFA 2020 and NCAA 2022 analyses in Figures 4.3 and 4.4, respectively.

4.3.3 Hardware used and Build Parameters

We used a Intel i7 CPU with 4 cores clocked at 2.8 GHz with 16 GiB of memory for all computational experiments. We used GCC version `12.1.1` [25] and CMake version `3.23.3` to build `Phylourny`. `Phylourny` was built using and was built as well as executed for the purposes of analysis on Linux `5.18.16`.

4.3.4 Numerical Error Assessment

We also investigate the numerical error when using simulations to compute a WPV. To this end, we produced a sample of 1000 P matrices from an MCMC search for each of the two tournaments. The P matrices were sampled uniformly from the respective MCMC chains, after discarding the first 10% of samples as burn-in. For each sampled P matrix, we compute both the exact WPV using `Phylourny`, as well as an estimate using one hundred, one thousand, ten thousand, one hundred thousand, and one million simulations. For these estimates, we report both the relative error, which is

$$\text{Mean}\left(\left\|\frac{\text{WPV}_{\text{sim},i} - \text{WPV}_{\text{phy},i}}{\text{WPV}_{\text{phy},i}}\right\|\right)$$

where WPV_{sim} is the WPV computed using simulations and WPV_{phy} is the WPV computed using `Phylourny`. We also report the norm error, which is

$$\frac{\|\text{WPV}_{\text{sim}} - \text{WPV}_{\text{phy}}\|}{\|\text{WPV}_{\text{phy}}\|}.$$

The results from these analyses are summarized in Table 4.3.

Additionally, we also conduct the same uncertainty analysis as described in Section 4.2.4 for both the UEFA 2020 and NCAA 2022 tournaments, but using simulations to estimate the WPVs instead of `Phylourny`. Results from these analyses are presented in two plots in the Figs 4.6 and 4.7.

4.3.5 Run time comparison

Finally, we also compare the runtimes of `Phylourny` with other methods (simulations and naïve computation) for computing the tournament WPV. Using the sample of 1000 P matrices produced in Section 4.3.4 we also recorded the execution time for each method: `Phylourny`, Naïve, and Simulations. For comparison we only use 1000 simulations, which corresponds to a norm error of > 0.1 on the UEFA 2020 dataset (see Table 4.3). While 1000 simulations are fewer simulations than one would utilize in a rigorous analysis, it is an appropriate choice as even this inaccurate level of simulation is less time efficient than `Phylourny`. The results from these runtime experiments are summarized in Figure 4.8.

4.4 Results

The analysis of the UEFA 2020 tournament with 16 teams required 51 seconds for 100,000 samples (generated via 10,000,000 MCMC steps), by executing the parallelized Poisson likelihood model using 4 cores on our test hardware system. The likelihood model calculations accounted for 90% of overall runtime. The analysis of the NCAA 2022 tournament required $\approx$ 1.5 hours for 100,000 samples (generated via 10,000,000 MCMC steps) and also using 4 cores. The difference in runtime is due to the substantially larger amount of data ($\approx$ 48 times more historical match data when compared to the UEFA 2020 analysis) used to compute likelihoods for NCAA 2022. Approximately 80% of the runtime increase can be attributed to the larger historical match dataset used. In addition, there are 64 instead of 16 teams in the NCAA tournament, which increases the time required to compute tournament WPVs and the P matrix. The NCAA 2022 MCMC search achieved an acceptance ratio of $\approx .17$, while the UEFA 2020 MCMC search achieved an acceptance ratio of $\approx .67$.

In Figures 4.3 and 4.4 we summarize the results of the uncertainty analyses using the thinned samples from the MCMC search. We plot the results from their respective analyses, restricted to the top 0.1% (i.e., top 100 WPVs by log- likelihood) of samples by likelihood for UEFA 2020 and NCAA 2022. Additional summary statistics for these top 0.1% samples are shown in Table 4.1. Summary statistics for the entire thinned sample from the MCMC search set are presented in the Table 4.2.

Notable results include the correct identification of the winner for UEFA 2020 (Italy) and the high ranking for NCAA 2022 winner (Kansas) tournaments, who both receive a high median win probability in the uncertainty analysis. This is shown in Fig 4.3 for UEFA 2020 and in Fig 4.4 for NCAA 2022 probabilities.

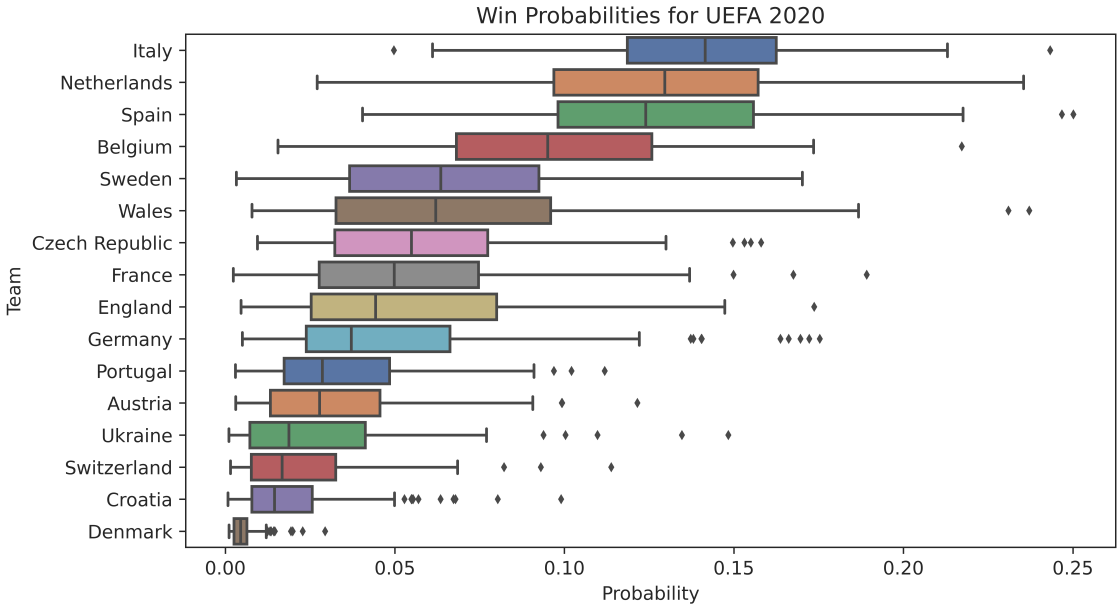

Figure 4.3: Probabilities for each team winning the UEFA 2020 tournament. The samples summarized are the top 0.1% percent of samples by likelihood from our 100,000 MCMC samples.

	Mean	STD	Min	Median	Max
UEFA	-98.92	0.27	-99.22	-98.99	-97.91
NCAA	-13800.04	1.99	-13802.38	-13800.75	-13793.86

Table 4.1: Summary statistics for the samples from the uncertainty analysis. Values shown are Log-Likelihoods of samples taken during the MCMC search for the UEFA 2020 and NCAA 2022 uncertainty analysis which have been restricted to the 99.9%-ile.

	Mean	STD	Min	Median	Max	Samples
UEFA	-105.18	2.45	-119.04	-104.99	-97.91	90,000
NCAA	-13842.15	14.00	-13901.66	-13841.72	-13793.86	90,000

Table 4.2: Summary statistics for the full uncertainty samples. Values shown are Log-Likelihoods of samples taken during the MCMC search for the UEFA 2020 and NCAA 2021 uncertainty analysis.

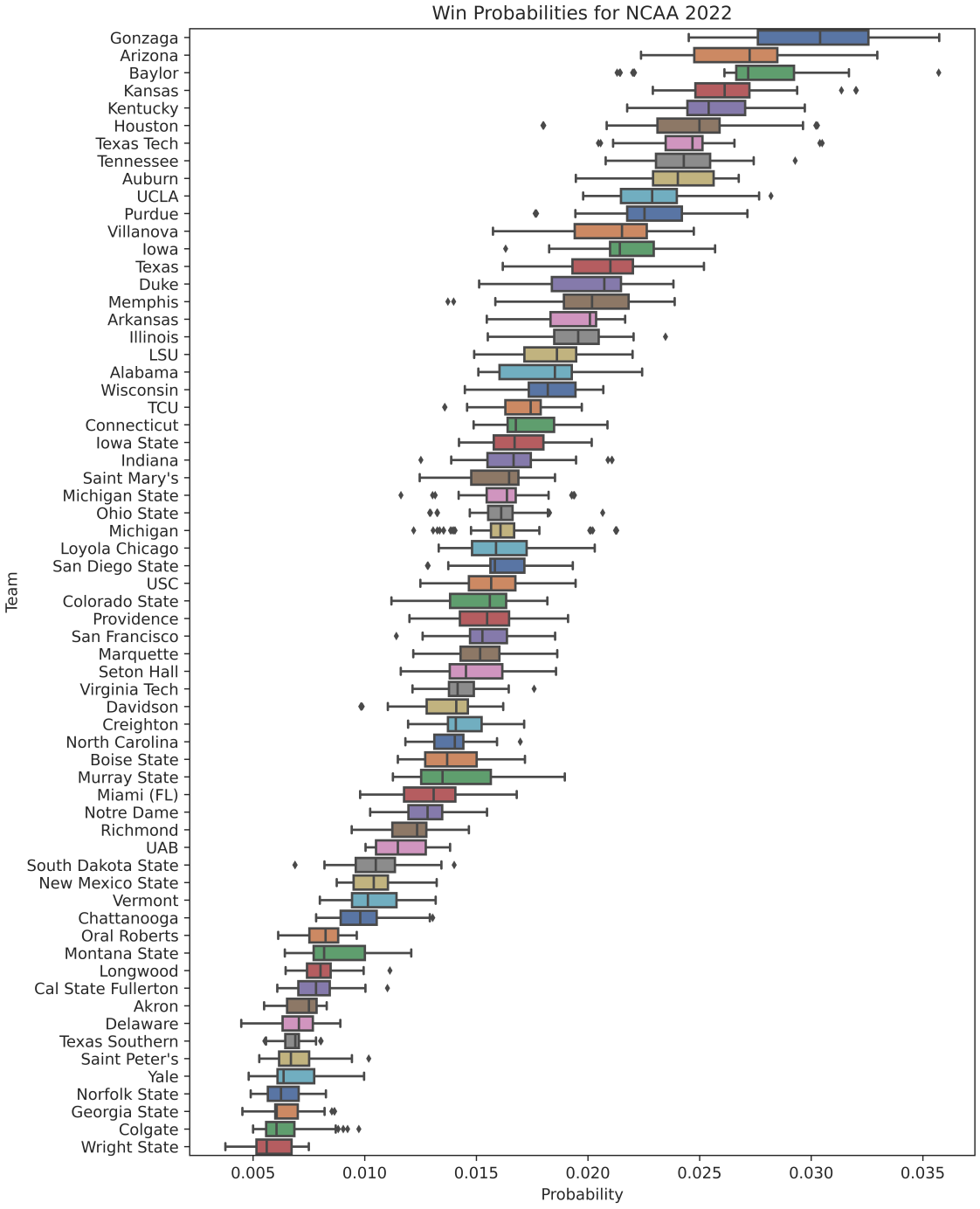

Figure 4.4: Probabilities for teams winning the NCAA 2022 tournament. The samples summarized are the top 0.1% percent of samples by likelihood from our 100,000 MCMC samples.

Dataset	Simulation Samples	Median Relative Error	Norm Error
NCAA 2022	100	0.727	0.691
	1,000	0.232	0.216
	10,000	0.073	0.069
	100,000	0.023	0.022
	1,000,000	0.007	0.007
UEFA 2020	100	0.211	0.273
	1,000	0.066	0.086
	10,000	0.020	0.027
	100,000	0.007	0.009
	1,000,000	0.002	0.003

Table 4.3: Simulation error for computing a WPV with an increasing number of samples for NCAA 2022 and UEFA 2020. We used a sample of 1000 P matrices from an MCMC search for each tournament. Matrices were randomly sampled at uniform from an MCMC chain after discarding the first 10% as burn-in. In this table we report the mean of the 1000 samples.

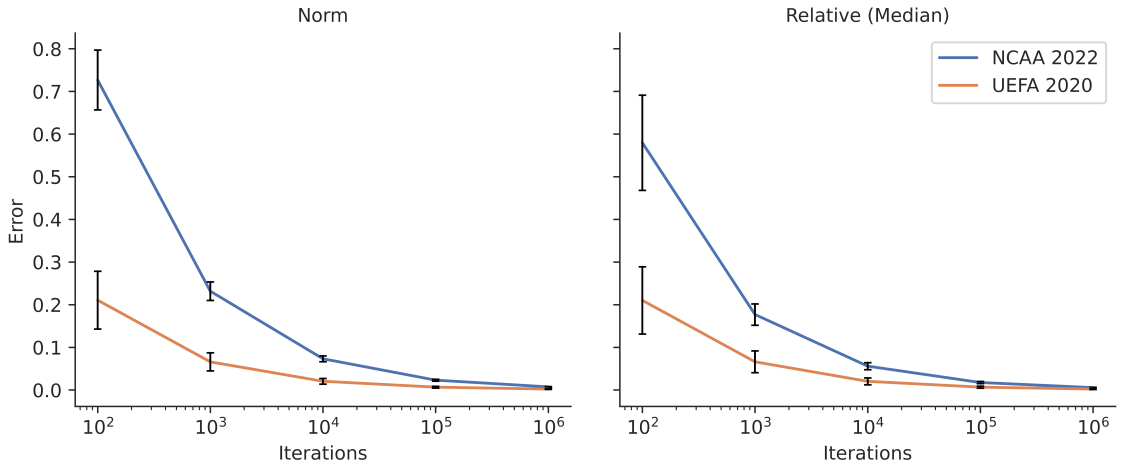

Figure 4.5: Plot of simulation errors with respect to number of simulations conducted for NCAA 2022 and UEFA 2020. We sampled 1000 P matrices from an MCMC search for each tournament. Error bars represent 1 standard deviation.

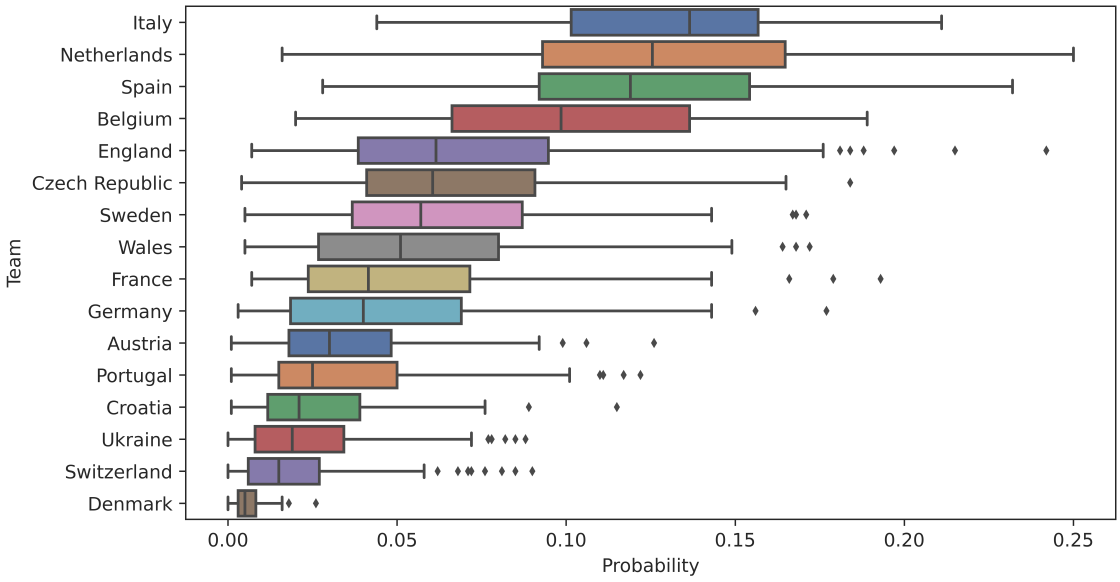

Figure 4.6: Win probabilities for the top 99.9% of samples by likelihood for UEFA 2020, computed with simulations

We present the run times of three methods of computing WPVs in Figure 4.8. Our method, `Phylourny`, performs the best (mean runtime 15 μs and 599 μs for UEFA 2020 and NCAA 2022 respectively). For smaller tournaments like UEFA 2020 we also found that it was faster to evaluate win probabilities naïvely (1413 μs) rather than conduct 1000 simulations (2111 μs). However, this does not hold for larger tournaments like NCAA 2022, where we were unable to obtain a result for the naïve computation, even after 2 hours ($\approx 7.2 \times 10^9 \mu$s) of run time, whereas conducting 1000 simulations was feasible (15010 μs). To obtain these runtimes, we conducted 1000 simulations, which corresponds to a median relative error of $\approx 7\%$ in the case of small tournaments like UEFA 2020. Of all the methods tested here, `Phylourny` remains the least computationally expensive by ≈ 2 orders of magnitude.

4.5 Discussion

We have shown that the problem of predicting tournament winners is sufficiently similar to phylogenetic likelihood calculations such that analogous computational techniques can be applied. We have demonstrated this by developing methods inspired by computational phylogenetics to predict tournaments, and that applying these methods yields substantial computational speedups. In addition, we can calculate the final WPV of a tournament *exactly*, instead of using simulations to approximate it. This also allows, for instance, for a seamless deployment of MCMC methods as illustrated by our uncertainty analysis examples for the UEFA 2020 and NCAA 2022 tournaments.

Finding the appropriate method to infer an accurate pairwise win probability matrix P remains a challenge. Modeling sports in a way that will accurately determine the

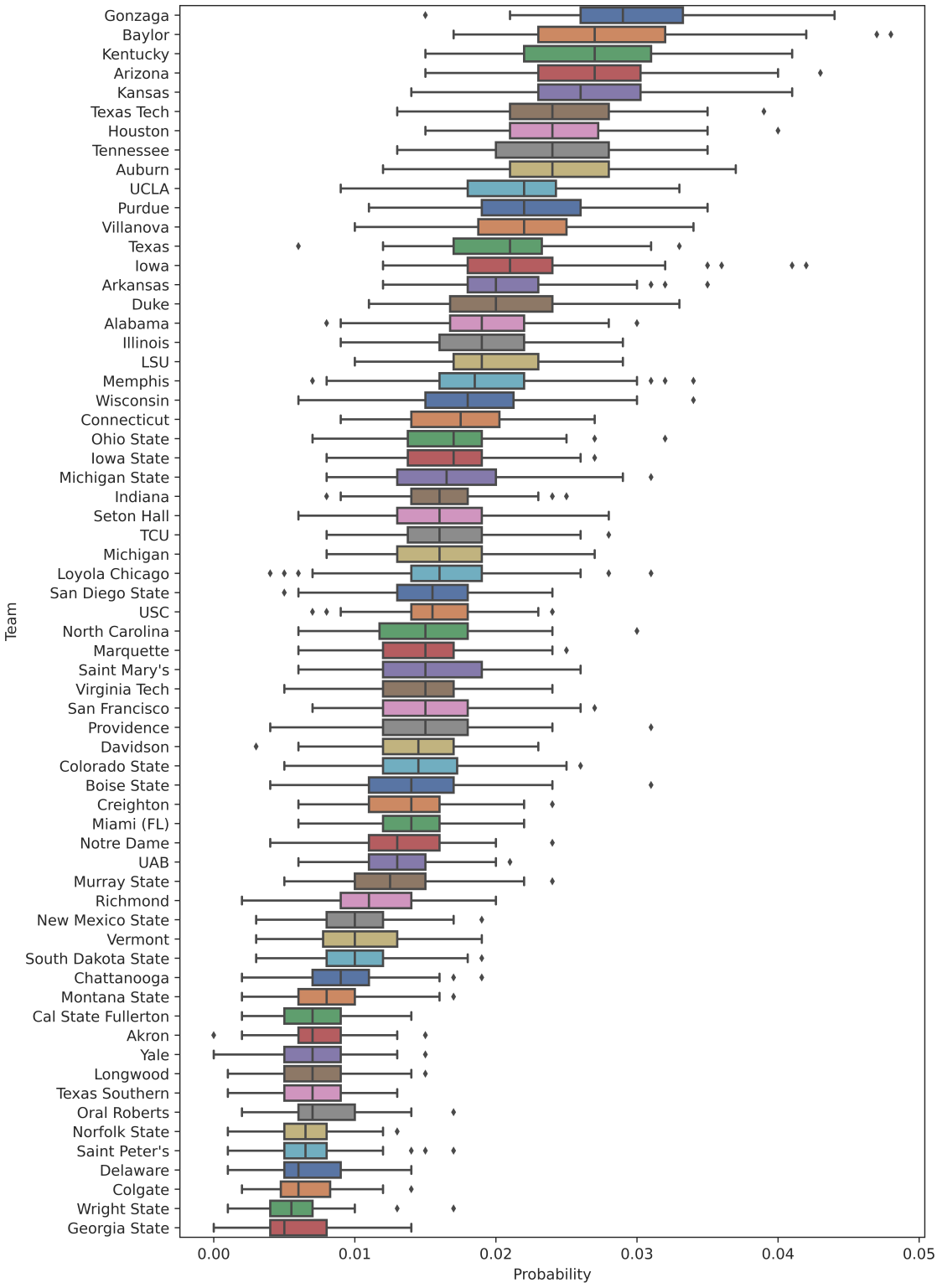

Figure 4.7: Win probabilities for the top 99.9% of samples by likelihood for NCAA 2022, computed with simulations. 1000 simulations per sample was utilized to produce an estimate of the win probability of each team.

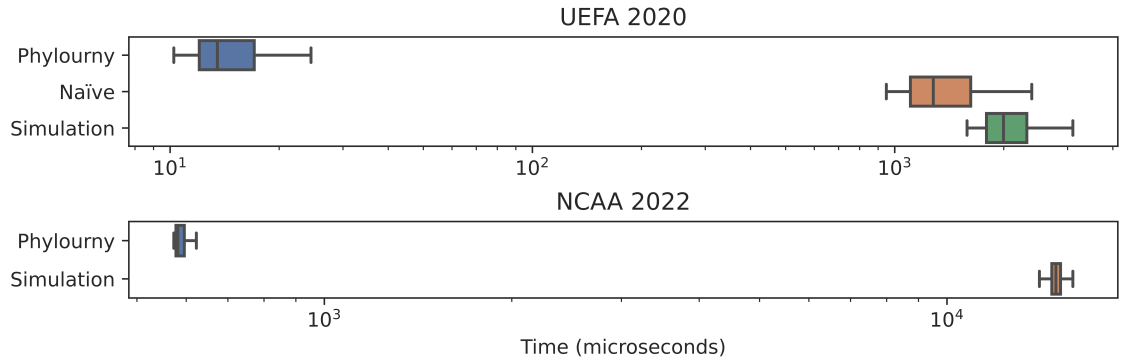

Figure 4.8: Tournament evaluation times for the different computation methods for 1000 sampled P matrices. Matrices were sampled according to the procedure described in Section 4.3.4. Times are reported as μs with a log scale. UEFA 2022 Simulation mean: 2111 μs, Naïve mean: 1413 μs, `Phylourny` mean: 15 μs. NCAA 2022 Simulation mean: 15010 μs, `Phylourny` mean: 599 μs. We did not obtain a time for the Naïve mode using NCAA 2022 as the time required to compute even a single evaluation was prohibitive. 1,000 simulations were conducted for these run time measurements.

probability of a specific outcome is difficult. Private industry (bookmakers) as well as academic researchers have invested considerable effort into methods to predict the outcome of sports matches [43, 53]. These investigations are beyond the scope of our work, and we intentionally do not address more complicated pairwise win models. Instead, we have showcased that comparatively simple models, such as the Independent Poisson Model which was further simplified from its form in [50], perform well when the uncertainty of the estimated model parameters is taken into account.

One may also argue that using the same P matrix through all stages of the tournament constitutes a simplification. In reality, the probability of a team beating another team most likely does not remain constant in the course of a tournament. Additionally, win probabilities might not remain constant for all matches in a "best of k" series. For some sports, particularly in the rising field of e-sports, adapted strategies will develop over the course of a series of repeated matches between two teams.

Despite these two (over-)simplifications, the ability to compute a WPV for a tournament both exactly and efficiently is highly useful, as advanced methods of analysis normally will require an exact result in order to be applicable. For example, when sampling from a posterior using an MCMC search using a complex model, it is desirable to have an accurate result for each sample, as this reduces the number of samples required to produce an accurate estimate of the posterior. While a sufficient degree of accuracy can be obtained via an appropriately large number of simulations, this approach is computationally expensive and will eventually become prohibitive.

Case in point, using the execution times measured in Fig 4.8, the uncertainty analysis for NCAA 2022 which took approximately 1.5 hours with `Phylourny` would have taken approximately 5 hours using 1000 simulations per sample. However, in this case, 1000 simulations would correspond to a median relative error of $\approx 22\%$. To achieve a more acceptable error level, 10,000 simulations could be used per sample, but this would increase the expected runtime to ≈ 40 hours. We demonstrate that we can efficiently conduct such an analysis by implementing our own comparatively simple MCMC analysis of the UEFA EURO 2020 football tournament.

Additionally, `Phylourny` is model agnostic, which allows for more complicated models to be implemented. An example is to add a time element to the Independent Poisson Model, which increases the likelihood contribution of more recent matches when compared to older matches. In fact, this time element might be an accidental reason why our prediction of the UEFA 2020 is accurate, as we only include the group stage matches where Italy performed surprisingly well, as opposed the extended match history including the qualifying round matches. Of course, this was not intentional but was an incidental result of limiting the historical data. Nonetheless, it shows how a likelihood model which incorporates match time could be advantageous. The likelihood model can also be augmented by including match locations, which models a home game advantage and incorporates this advantage into the likelihood. A match location augmented likelihood model also has implications for WPV computation, as the inclusion of location information for each match in the knockout round might improve results.

The main contribution of our work consists in the introduction of the *computational* method, which accelerates the exact computation of final win probabilities, given an estimate P of pairwise win probabilities, and the surprising connection between two seemingly unrelated branches of science.

We further demonstrate the efficiency and utility of `Phylourny` by implementing our own uncertainty analysis for the UEFA 2020 tournament as well as for the NCAA 2022 tournament. As can be seen in Figs 4.3 and 4.4, there is a remarkable diversity of predicted outcomes. This is *despite* the likelihoods for these samples being essentially equivalent, as can be seen in Table 4.1. For example the difference between the minimum log-likelihood and the maximum log-likelihood for the 99.9%-ile samples for the UEFA 2020 analysis only amounts to 1.30 log-likelihood units. We interpret this as the sample containing predictions with *essentially* the same amount of support from the data. Despite this, the range of outcomes predicted is comparatively diverse, and generally contradictory. Furthermore, in the UEFA 2020 uncertainty analysis, the win probabilities for Italy range between less than 0.05 to greater than 0.2. Likewise, the top 5 teams by median win probability in NCAA 2022 have mostly overlapping ranges for estimated win probabilities. In other words, there is a high uncertainty as to which team is the most likely to win. However, the forecast is clearly more certain for the NCAA 2022 sample when compared to the UEFA 2020, despite the smaller number of teams in UEFA 2020. This is due to the NCAA 2022 analysis using substantially more historical matches (on the order of ≈ 40 times more historical matches), and the sport of basketball being less a product

of random chance, due to the large score size. However, NCAA tournaments are an exception, with a large number of teams in each group, and therefore a large number of matches in the lead up to "March Madness". Furthermore, the amount of data in the UEFA 2020 analysis was intentionally reduced for this work.

From this example, lessons can be learned for the practice of phylogenetics. In particular, care should be taken when analysing a single result from phylogenetic inference, as a given dataset might provide support for a large range of conflicting explanations. For example COVID-19 phylogenies are difficult to estimate for this precise reason [61], and placing stock in any single result runs the risk of ignoring other plausible explanations. Therefore, this work is yet another reminder to incorporate uncertainty, particularly of parameter estimates, when performing either phylogenetic or any model based analysis.

5. Lagrange-NG: The next generation of Lagrange

This chapter is based on the following peer-reviewed software article:

> **Ben Bettisworth**, Stephen Smith, Alexandros Stamatakis. "Lagrange-NG: The Next Generation of Lagrange " *Systematic Biology*, 2023, https://doi.org/10.1093/sysbio/syad002

The Dispersal-Extinction-Cladogenesis (DEC) model [72] is widely used to analyze biogeographical data. However, computing likelihoods under this model is computationally challenging, as (i) the geographical regions are splayed into $2^r = s$ states and (ii) the computation of the respective transition matrix has a time complexity of $\mathcal{O}(s^3)$. Thus, computing a single likelihood of the DEC model requires $\mathcal{O}((2^r)^3) = \mathcal{O}(2^{3r})$ time. In other words, the likelihood computation is exponential with respect to the number of regions under study. Therefore, the scalability of data analyses under the DEC model is limited be the number of regions, that is, only a small number of 6 to 10 regions can be analyzed in a reasonable amount of time [56].

The most expensive inference step is the computation of the transition matrix that often accounts for 80% or more of overall runtime. As in standard likelihood-based phylogenetics, the transition matrix is computed via a matrix exponential, albeit on a substantially larger matrix. Substantial research effort has been invested into finding the best way to compute the matrix exponential [59], but it still remains a challenge to compute it efficiently and accurately. Additionally, unlike in standard phylogenetics, the DEC model is non-reversible. The relevant implication of a non-reversible is that the matrix to be exponentiated is generally non-symmetric, which limits the number of applicable numerical methods for computing the matrix exponential, typically to less precise ones. In the following, we present the `Lagrange-NG`

(Lagrange-Next Generation) software, an almost complete rewrite of the popular and widely used `Lagrange` software [72]. As the primary challenge to computing the likelihood under the DEC model is to efficiently calculate the matrix exponential, `Lagrange-NG` relies on a relatively recent method of computing a matrix exponential based on Krylov Subspaces [59] for a moderate to large number of regions (6 and more) in its default mode of operation.

Alongside the improvements to the matrix exponential, many so-called "micro-optimizations" (for example, passing function arguments by reference instead of value, using more efficient data structures to store regions, or eliminating unnecessary computation) have been implemented that further accelerate computations. We have also implemented a task-based hybrid multi-threading approach, which increases the rate of analyses by up to a factor of 8 for datasets exceeding 200 taxa. Furthermore, we improve upon the numerical stability compared to the original software, and fix a major bug which we discovered during development. Finally, to verify that `Lagrange-NG` produces analogous results as the original implementation, we devised a novel method of comparing range distributions on trees, which is based on the Earth mover's distance metric. A similar application of the Earth mover's distance has been successfully applied to phylogenetic placement, though this method and application is distinct [19].

5.1 Background

`Lagrange-NG` implements the Dispersal-Extinction-Cladogenesis (DEC) model of geographic range evolution [72]. A geographic range, in this context, describes the broadly defined distribution of the habitat of a particular species. The evolution of this range is assumed to follow the phylogeny, or the biological evolution of a species or clade. The DEC model takes as input, at a minimum, a phylogenetic tree, and a set of regions. The phylogenetic tree is assumed to be the true phylogeny of the included species, and the regions are the generalized areas of potential habitation for the species in question. The DEC model constructs a list of states based on the valid set of regions that a particular species could inhabit. With these components, the DEC model constructs a transition matrix between states using two parameters, an extinction parameter and a dispersion parameter. Using this transition matrix, a likelihood of the model parameters can be computed and used to optimize the model parameters. Once the optimal model parameters have been found, the most likely ancestral ranges can be obtained by computing the model "backwards".

Computation of the likelihood of a particular set of parameters under the DEC model proceeds in a fashion similar to the standard Felsenstein pruning algorithm [21]. In this algorithm, the computation starts from the tips and moves towards the root, storing intermediate results in buffers called conditional likelihood vectors. Please see Chapter 2.1.3 for a more detailed explanation, including a detailed discussion about the savings involved with such a scheme. What is relevant for this discussion is that the Felsenstein pruning algorithm avoids excess computation by noticing that, at certain points in the computation of a likelihood on a tree, the only relevant quantity is the likelihood *conditioned on the current state*.

5.2 Software Description

`Lagrange-NG` constitutes an nearly complete rewrite of the original (unpublished) C++ version of `Lagrange`. Of the 4600 lines of code, only 5% remain from the original code base. This redesign retains the complete functionality of `Lagrange`, but is computationally more efficient, and implements a parallelization of DEC calculations. `Lagrange-NG` implements four major improvements to `Lagrange`. First, it supports parallelism via a hybrid task based parallelization scheme which utilizes both coarse and fine grained parallelism. Second, it deploys more efficient numerical methods and algorithms which were developed relatively recently to compute the matrix exponential, for example, an algorithm based on Krylov subspaces which we use in `Lagrange-NG`. Third, it introduces general improvements and optimizations, that is, micro-optimizations, which individually do not notably increase efficiency but that altogether yield a substantial improvement. Finally, the fourth improvement is a substantial increase in coding standards adherence and hence, software quality, as measured by the coding standards adherence evaluation tool and benchmark SoftWipe [96]. The SoftWipe score of the original `Lagrange` software is 5.5, while our nearly complete rewrite increases this to a score of 7.8. While the original score of 5.5 is fairly average, the new score of 7.8 places `Lagrange-NG` 3rd in the list of 51 scientific software tools written in C or C++ that are contained in the SoftWipe benchmark.

Importantly, during the process of improving the code quality, a potentially serious bug was discovered. In order to correct numerical instabilities, the transition matrix was normalized such that the rows summed to 1.0 after the matrix exponential computation. During normal computation, this operation will have little effect on the results. However, if the rate matrix is sufficiently ill-conditioned, the computation exhibits extreme numerical instabilities such that any results produced are meaningless. If the matrix is then normalized at this point, then results produced with this matrix are made to appear sensible. Therefore, any error in the computational process is hidden from the user, and the results of the computation will be perceived as plausible. Fortunately, as long as the matrix remains unnormalized, these errors are easy to detect, as several analytical conditions are no longer met (such as the rows no longer summing to 1.0). We are not aware of any approaches to recover from these errors, but at least the user is not misled into thinking that meaningless results are plausible. This normalization error is exceedingly rare, as the authors never observed it in the thousands of datasets analyzed for this paper. Despite this, the error *can* occur, and *will* by Murphy's law. As such, we are convinced that in the event of this bug, the user should be appropriately informed. Therefore, in this case, `Lagrange-NG` simply fails, and alerts the user to what occurred.

Additionally, we identified and corrected a configuration error in the process of building `Lagrange`, where important compiler optimization options were not properly utilized. Fixing this configuration error alone increased the computational efficiency of the original `Lagrange` by up to 10x. While this error is easy to overlook, yet trivial to fix, we assume that many past `Lagrange` analyses were conducted using

the unoptimized code. Nonetheless, in this work when we perform benchmarks with `Lagrange`, we conduct them with this configuration error fixed.

`Lagrange-NG` can be downloaded from GitHub at https://github.com/computations/lagrange-ng. To build the software, the only requirements are a C++ compiler, and CMake. Optionally, `Lagrange-NG` can be built with the respective system versions of the Intel Math Kernel Library (MKL) [13] and NLOpt [40, 63]. If a system version of MKL is not present, `Lagrange-NG` will build with OpenBLAS [66] instead.

5.3 Methods and Algorithms

`Lagrange-NG` utilizes a task based parallelization scheme in which each node of the tree is assigned as a task. In order to compute the results for a generic node of the tree results for its two children must first have been computed. This involves computing

1. The right and left instantaneous rate matrices: Q_r, Q_l,
2. The right and left transition probability matrices: $P_r = e^{Q_r t_r}$ and $P_l = e^{Q_l t_l}$ respectively,
    - When using the Krylov based matrix exponentiation, we do not compute P_r and P_l and instead compute w_r and w_l (the result of the next step) directly.
3. The result of the Markov process along the left and right branches: $w_r = P_r v_r$ and $w_l = P_l v_l$ respectively,
4. The weighted combination of w_r and w_l, v_t.

Together, these operations make up a single task for a worker. However, as it can be seen above, the task can be further subdivided into smaller parts, which we will call operations. For the purposes of this paper, we will label each of the operations as

1. Make Rate Matrix Operation,
2. Expm Operation,
3. Dispersion Operation and,
4. Split Operation.

In order to more easily support parallel computation on other platforms, such as GPUs, we have separated the operations from the memory buffers which are required to store the intermediate results needed for likelihood computation. For example, in Figure 5.2 we show a generic node and its associated operations. For each operation,

there are a set of indices which indicate the location of the assigned memory buffer. Additionally, they store the last execution clock point, details and purpose of which are discussed later.

Operations are typically fast enough that they cannot be separated into parallel tasks. However, if the model parameters and branch lengths used in the computation are equal, the results of these operations can be shared between two tasks. For example, consider the tree topology in Figure 5.1. Here, two of the branches have the same branch length, 1.0. If they also share a rate matrix (which they almost always will), then the result of the two matrix exponential operations will be identical. Therefore, the likelihood computation on this tree can be accelerated by computing e^{Qt} only once, and saving the result. An operation can be shared when the model parameters (the extinction and dispersion rates) and the branch lengths are the same between two branches.

In order to avoid such redundant computations, `Lagrange-NG` can share operations between tasks. However, when computing with multiple threads, this introduces the possibility of conducting computations with inconsistent values from dependant operations when performing successive evaluations of the likelihood with model parameters that are altered by optimization routines.

To avoid race conditions, i.e. cases when one thread is reading data that is not ready, we use a clock based method to enforce a partial ordering on the computation of operations. Readers familiar with vector clocks will recognize this as a vector clock with the number of distributed elements equal to one[4]. After the evaluation of each operation, a "time" is recorded in the evaluated operation, and the clock incremented. This time is not a true time, but instead a virtual time that is incremented every time an operation is completed. To determine if an operation can be carried out, a thread only needs to check if the clocks of its immediate dependant operations show a larger value. If this is the case, then the dependant operations have already been evaluated, and the operation can be performed on consistent input data.

By carefully dividing the tasks into operations, merging identical operations between tasks, and enforcing a partial order on the computation of operations, we can implement an effective task-based parallelization scheme. However, this scheme does have an unavoidable bottleneck. Since the dependencies for the operations are based on the topology of the tree, and the tree has fewer branches near the root than near the tips, the threads will necessarily be work starved near the end of a round of computation. Therefore, our parallel efficiency is limited, and in fact dependant on, the topology of the tree.

As noted briefly above, when computing w_r and w_l using a Krylov based method, we can omit the computation of P_r and P_l. Therefore, the technique of combining branches with the same branch lengths and model parameters in order to omit redundant computation is no longer viable, as the results also depend on v_r and v_l. However, when there are a sufficient number of regions (6 regions and above, for

[4] In fact, the vector clock scheme was picked for the option of increasing the number of distributed elements to allow for a cluster based version of `Lagrange-NG` in the future.

example) to make the Krylov based method faster, the increase in computational efficiency is so large that the trade-off is worth it.

However, we retain the same graph structure as shown in Figure 5.3, for the ability to fallback to a more reliable method of computing results in the case of serious numerical errors, as the matrix exponential operation contains information about the branch length and rate matrix for the particular branch. Instead, the result of $P_r v_r$ is computed in the dispersion operation, and the matrix exponential operation is used just to store information about the branch.

Unfortunately, computing matrix exponentials via a Krylov subspace can be very numerically unstable, in our experience. To compensate for this, we implemented an adaptive mode in `Lagrange-NG`, which combines both the Krylov method and the scaling and squaring method. If the number of regions is small (less than 6), then scaling and squaring is used. Otherwise, the Krylov method is used to compute the transition matrix. The Dispersal Operation is conducted, and we check to see if the following conditions are true for the resulting vector:

1. All entries are less than or equal to 1.0 or,
2. All entries are greater than or equal to 0.0 or,
3. None of the entries are NaNs.

If any of these conditions are not true, we trigger a safety fallback to the scaling and squaring method for the *specific* Expm Operation which will be used for the rest of the analysis. By using this adaptive approach, we are able to avoid the major issues with numerical stability that are the result of the Krylov method, but still retain much of the speedup from using the Krylov method. In general, the recovered speedup, that is the speed retained from enabling fallback, is dataset dependant, and can be as low as 0%. However, in our experience, the recovered speedup is much closer to 95% for almost all datasets.

5.3.1 Coarse and Fine Grained Parallelization

Most linear algebra libraries offer some form of BLAS level parallelization, which is what `Lagrange-NG` uses for its "fine-grained" parallelism. However, in early testing we found that the parallel efficiency of this mode of parallelism was quite poor for our purposes, often making the run slower for a moderate number of threads and a small number of regions (for example, 6 threads and 5 regions). So, we elected to implement a coarse-grained parallelization method, which parallelized over the available operations. To distinguish these two modes of parallelism, we named the coarse-grained threads "workers" and the fine-grained parallelism "threads per workers".

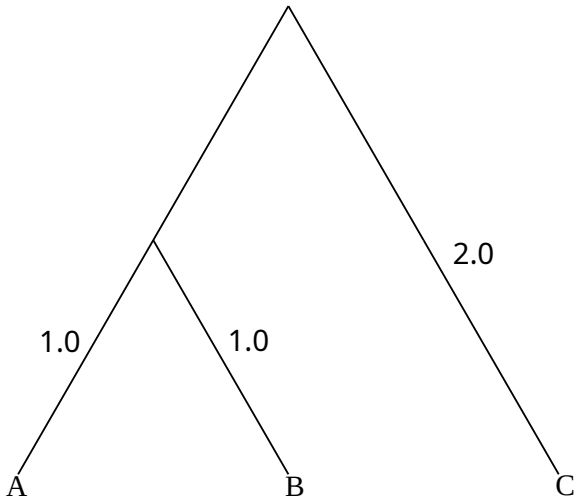

Figure 5.1: A simple ultrametric tree.

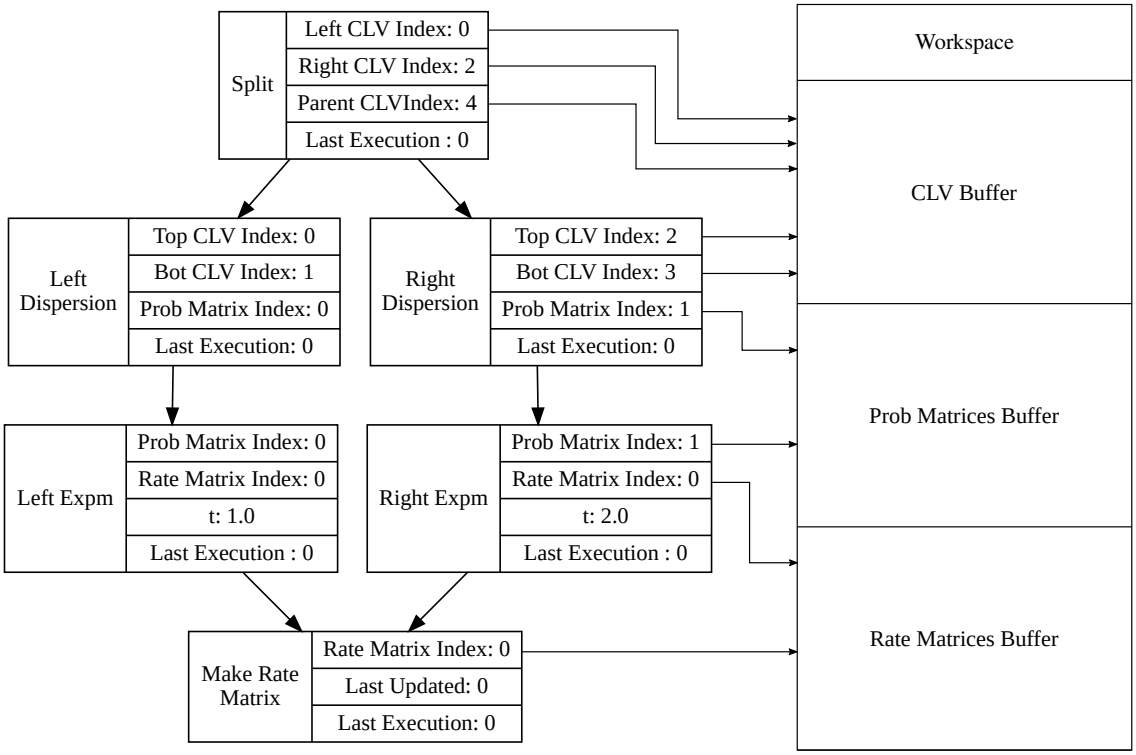

Figure 5.2: An example set of operations for a generic unspecified node.

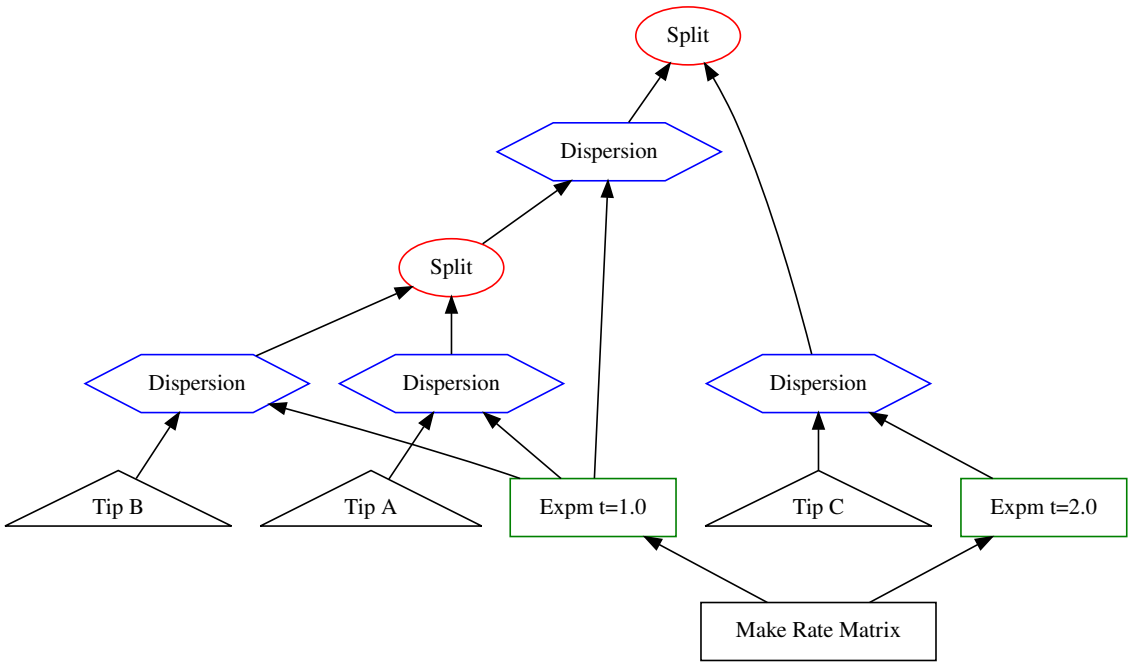

Figure 5.3: Tree in Figure 5.1 decomposed into the operations used to compute the likelihood.

5.3.2 Comparing Distributions on Trees

When comparing the results of `Lagrange-NG` and `Lagrange`, it is difficult to assess if the numerical results are "close enough" to be considered equivalent. By way of example, scientific software will typically, instead of asking if two floating point numbers are exactly identical to one another, ask if they are closer than some small number (generally labeled ϵ). This is due to the limitations in the IEEE 754 floating point number format, which is the format used to encode real numbers in nearly all modern computers. A more complete explanation of this phenomenon is outside the scope of this paper, and is extensively discussed in most textbooks on numerical computing. For our purposes it is sufficient to say that differences in the order of associative mathematical operations will generally yield different results.

`Lagrange-NG` does not escape this limitation of the IEEE 754 standard. So, in order to appropriately compare results, we must take this into account. One can examine, by eye, each distribution individually and compare the distributions by hand, but this is time consuming, subjective, and error prone. Therefore, it is desirable to construct a metric between the node base probability distributions on trees in order to automatically compare the results.

Consider an example, where we have two sets of ancestral range distributions computed by differing methods using the same tree. Let us call those distributions d_1 and d_2. The distribution for node n is then represented by the notation $d_i(n)$. Since the tree topology is identical for the two distributions, we can match the node level distributions in a one-to-one mapping between the two sets of distributions. We will index the individual elements of the distribution either by the list of region names or

the binary notation for regions. For instance, if we have a distribution over regions A, B, and C, then the entry for the distribution AB for node n would be indexed as $d_1(n, AB)$. Equivalently, we can use a binary notation to write $d_1(n, 110)$. In this case A stands for the most significant bit, B the second most significant bit, and C the least significant bit.

In this example, the first approach would be to treat $d_1(n)$ and $d_2(n)$ as vectors, and simply compute the cosine distance between distributions. This will indeed produce a metric, but it has some undesirable properties. For example, suppose we have a distribution over 5 regions: A, B, C, D, and E where $d_1(n, AB) = 1.0$. If we use the cosine distance to compute the distance between $d_1(n)$ and $d_2(n)$ where $d_2(n, ABC) = 1.0$, then the resulting distance will be 1.0, as the vectors are orthogonal. However, this is the same distance as if we had $d_2(n, CDE) = 1.0$ instead which is also orthogonal to $d_1(n)$. But, the prediction AB is much closer to ABC than CDE. In the first case, the predicted ranges differ by only *one* region, whereas in the second case, the predicted ranges differ by *every* region.

Since the cosine distance does not account for the valid transitions between states in the DEC model, we should pick a distance that is aware of these transitions. To accomplish this, we first embed the two distributions into a hypercube graph. A hypercube is graph with 2^n nodes and each node is connected to n other nodes (see Figure 5.4 for an example). Importantly, the edges of a hypercube graph correspond to the valid transitions between states in the DEC model[5].

Once the distributions are embedded in a hypercube graph, we can compute the distance between the distributions as the "amount of effort required to turn one distribution into the other". This is the Wasserstein metric, also known as the Earth mover's distance, and is what we base our distance on. Suppose we have the distributions $D1$ and $D2$ from Figure 5.5. In order to transform $D1$ into $D2$, we need to find a way to move 0.25 "earth" from node `10` to node `01`. Two possible example transformations can be seen in Figure 5.6. While the distance computation in Figure 5.6 is straightforward, in general, finding the minimum transformation distance requires the use of an optimization routine.

To find the minimum distance, we formulate the problem as a linear programming problem. Specifically, we solve

$$\begin{aligned} &\min_x \textstyle\sum_i x_i \\ &\text{such that } Ax = b \\ &x_i \geq 0 \end{aligned} \tag{5.1}$$

[5] Some readers might have noticed in Figure 5.4 that the edges as shown don't distinguish a direction of transitions, which means that transitions *out* of the extinct state are valid. While conceptually this is a problem, for the purposes of computing a distance, it will not affect the results, as taking the path through the extinction state is equivalent to taking any other path of equal distance. Due to the nature of the hypercube, this second path must exist.

Where A is the matrix representation of the hypercube as a graph, also known as an incidence matrix. A will act as the constraint matrix for the linear programming problem, and $b = d_1(n) - d_2(n)$, that is, the difference between the two distributions. If we have a distribution with s states then we can produce A by creating $s-1$ rows and $s(2^s-1)$ columns. The rows represent the nodes of the graph, and the columns represent the edges of the graph, split in two for each direction of flow. The entries of A are defined as:

$$A(n,e) = \begin{cases} 1 & \text{If } e \text{ points to } n \\ -1 & \text{If } e \text{ points away from } n \\ 0 & \text{Otherwise.} \end{cases} \tag{5.2}$$

Please note that there are only $n-1$ rows. This is because the final row can be expressed as a linear combination of the previous rows, and will therefore induce no additional constraints to the problem. Additionally, we choose to suppress edges leading away from the extinct state, to be consistent with the model. By suppressing these edges, we remove s columns from the matrix as well.

As an example, the distance between the distributions in Figure 5.5 can be computed with the matrix

$$A = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & -1 & 0 & 1 \end{pmatrix}$$

and the vector

$$b = \begin{pmatrix} 0 \\ 0.25 \\ 0.25 \end{pmatrix}.$$

To actually compute the solution, we can turn to any of a number of linear programming solvers, for example the one in SciPy [87] which the solver we use for this thesis. Finally, in order to normalize the distance, we divide the result by the maximum possible path length, which is simply the number of *regions*.

5.3.2.1 Metric Performance

To demonstrate the performance of the Wassserstein metric, we use a case study with 3 regions: A, B, and C. These 3 regions induce a state space with 8 states, with the states being "Empty", "A", "B", "C", "AB", "AC", "BC", and "ABC". For each of these states, we generated a "basis" distribution, which is a distribution vector containing 1.0 in the entry corresponding to the state, and 0.0 everywhere else. For example, the basis distribution representing "A" would be

$$(0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)$$

We then computed the pairwise distance between all of the basis vectors. The results of this computation is summarized in Figure 5.7. As it can be seen from this summary, our Wasserstein metric behaves as intended, which is to say, states which have fewer regions in common with each other are farther away than states with more regions in common.

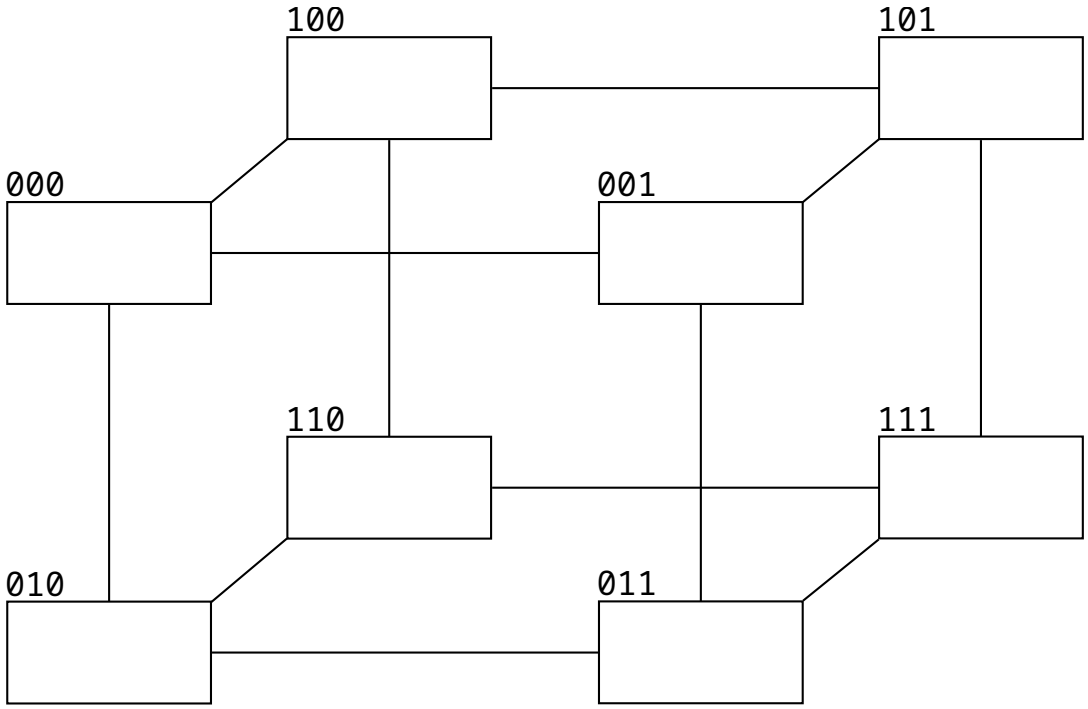

Figure 5.4: An example of a 3-dimensional hypercube with associated region names in binary notation.

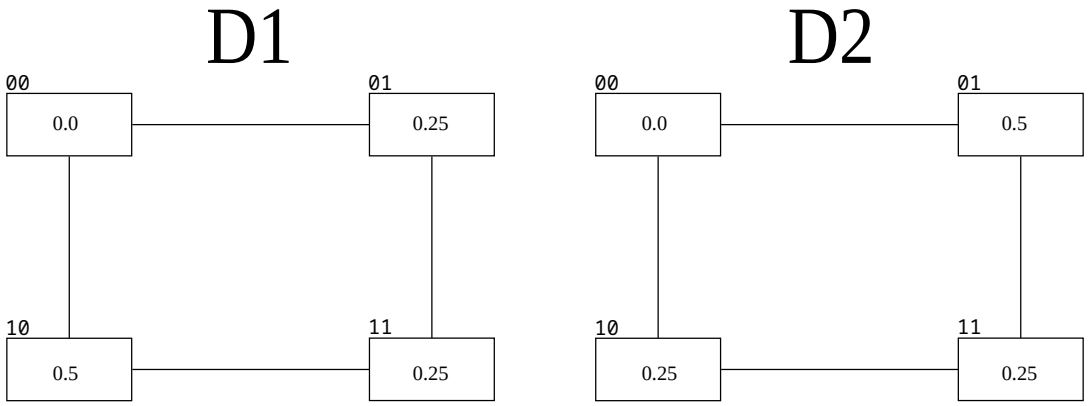

Figure 5.5: Two example distributions, displayed as 2d-hypercubes (squares).

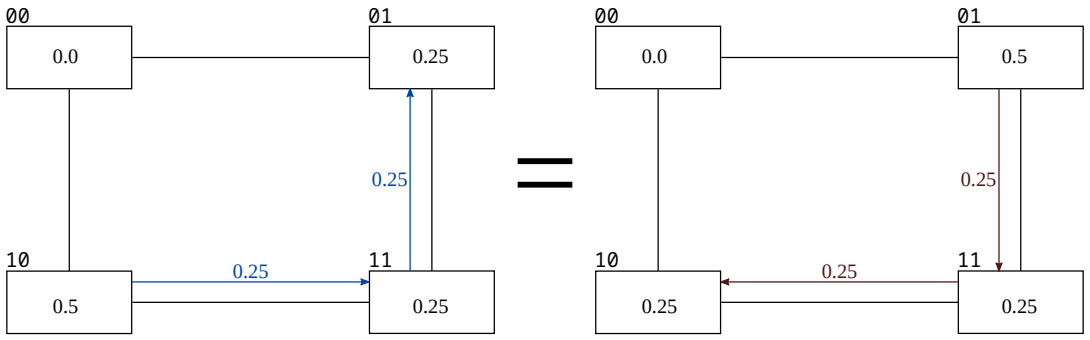

Figure 5.6: Distributions from Figure 5.5 with the Wasserstein metric applied. On the left, we choose to move 0.25 units of "mass" from node `10` to node `01`. This requires 2 transitions, shown in blue, for a total distance of 0.5. On the right, we choose to transfer mass from node `01` to node `10`, which yields the same result.

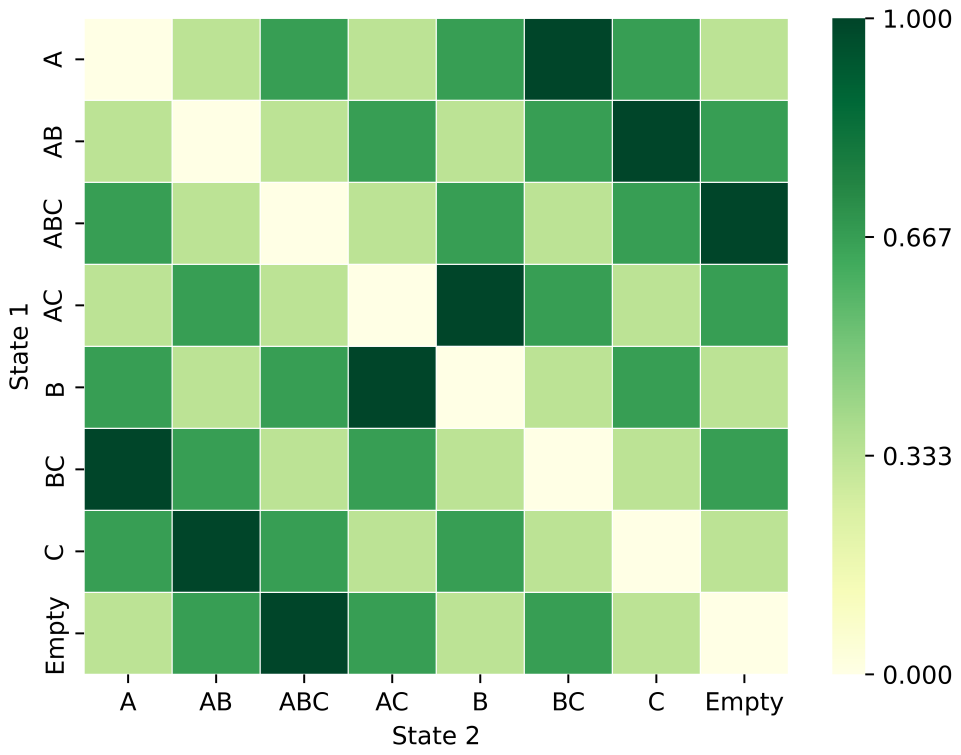

Figure 5.7: Plot showing the pairwise Wasserstein metric between the "basis" distributions on three regions: A, B, and C. Here, the "basis" distributions are a set of distributions, one for each state, with 1.0 in the corresponding entry for that state. Distributions are labeled by which entry contains the 1.0. States are ordered by a Grey code for aethetic reasons only.

5.4 Performance

To assess the performance of `Lagrange-NG` relative to the original implementation, we randomly generated a large number of synthetic datasets with a varying number of regions and executed `Lagrange` and `Lagrange-NG` to record the respective runtimes. We generated 100 random datasets with either 5, 6, or 7 regions, and 100 taxa, to obtain a total of 300 datasets. Furthermore, we ran an additional series of parallel performance evaluations on `Lagrange-NG` with 8 threads assigned to coarse grained parallelization using the same datasets. The results of this performance assessment are shown in Figure 5.12. Additionally, we assessed the performance of only `Lagrange-NG` on datasets with 8, 9, 10, 11, or 12 regions when using 8 workers. For the experiments with 8 regions, we generated 100 datasets, for the experiments with 9 or 10 regions, we generated 30 datasets, and for the experiments with 11 or 12 regions, we generated 10 datasets. Less datasets were generated for experiments with larger region numbers to limit the total time spent running experiments. Results from this performance assessment are shown in Figure 5.13.

5.4.1 Further Experiments

Tests and tooling for experiments in this paper are written in Python 3 [76], with additional support from Numpy [29], SciPy [87], and seaborn [89].

5.4.1.1 Investigating the Optimal Threading Configuration

Because `Lagrange-NG` implements both fine- and coarse-grained parallelization, we need to investigate the optimal threading configuration, that is, the number of fine-grained threads per task to be used, for a given dataset. To this end, we generated datasets with 50 or 100 taxa and 5, 6, 7, or 8 regions. This yielded a total of 8 distinct parameter sets. As the fine grained parallelization scheme is nested inside of coarse grained threads/cores, the total number of threads used by `Lagrange-NG` is the product of fine grained threads and coarse grained threads. So, in addition to these 8 parameter sets, we also generated the 6 threading configurations with a total number of 32 threads each, as 32 can be factored with: 1 and 32; 2 and 16; 4 and 8. Since order matters, the configuration of 2 workers and 16 threads per worker is different than the configuration of 16 workers and 2 threads per worker. For each of these parameter set and threading configurations, we ran 100 trials and recorded the times. The results from these runs can be seen in Figure 5.8.

5.4.1.2 Determining the parallel efficiency of `Lagrange-NG`

Given the results from the previous experiment to determine the optimal threading configuration, we choose to determine the parallel efficiency of `Lagrange-NG` using only workers. This is to say, we only increased the number of threads allocated to the coarse grained tasks. To this end we tested `Lagrange-NG` with 1, 4, 8, 16, and 32 threads by generating 100 datasets for each threading configuration. We did this with datasets with 6 regions and 100 and 500 taxa. We computed the mean of the execution times for the runs with a single thread, and used this value to compute the respective speedups.

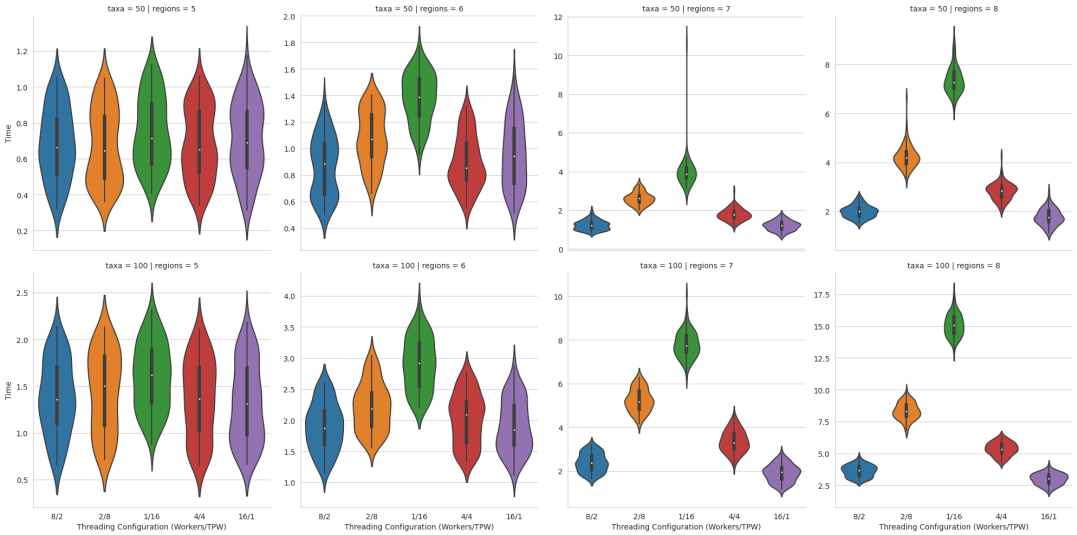

Figure 5.8: Plot of the threading configurations on various dataset sizes. 100 datasets were generated for each Taxa, Region, and Threading Configuration. Each dataset was generated randomly, similar to how datasets are constructed in the rest of this work.

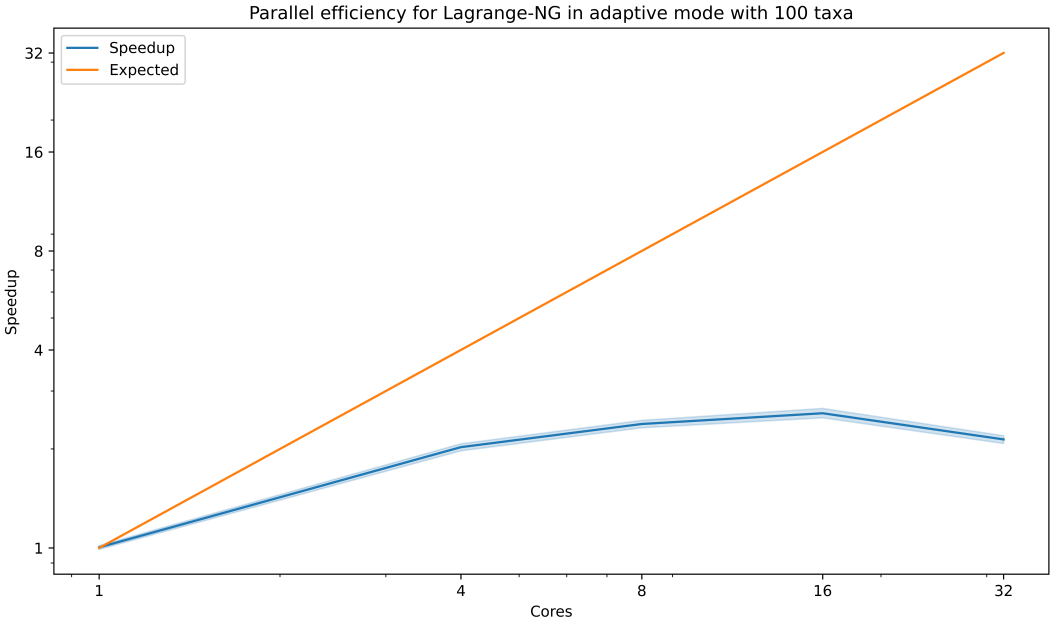

Figure 5.9: Parallel efficiency plot for a datasets with 100 taxa and 6 regions. Please notice the log-log scaling. The actual values plotted are 2.0, 2.4 , 2.6, 2.1 for 4, 8, 16, 32 threads, respectively. The ratio of the realized speedup to the optimal speedup is 0.51, 0.30, 0.16, 0.07 for 4, 8, 16 and 32 threads respectively.

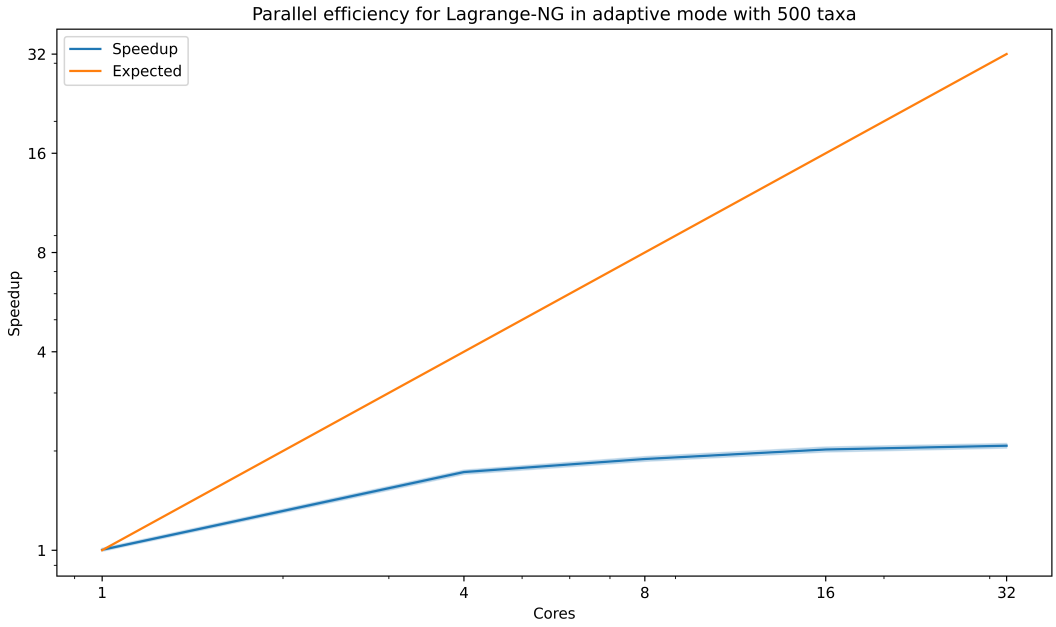

Figure 5.10: Parallel efficiency plot for a datasets with 500 taxa and 6 regions. Please notice the log-log scaling. The actual values plotted are 1.7, 1.8, 2.0, 2.1 for 4, 8, 16, and 32 threads, respectively. The ratio of the realized speedup to the optimal speedup is 0.43, 0.24, 0.13, 0.06 for 4, 8, 16, and 32 threads, respectively.

5.4.2 Biological Examples

While we conducted extensive tests on simulated data, we also verify that `Lagrange-NG` behaves correctly on empirical datasets. To this end, we reproduced the results from a previous study on sloths from [84]. Additionally, we took the opportunity to reproduce the results using the tools specified in the paper as this gave us an opportunity to compare with `BioGeoBEARS` [56], a similar tool written in R which performs optimization via simulated annealing, and is very feature rich when compared to `Lagrange` or `Lagrange-NG`.

In order to reproduce results, we downloaded the supplementary data from the Dryad repository associated with the publication. To run the analyses with `BioGeoBEARS` and `Lagrange-NG`, we had to slightly modify the data. This involved correcting some taxon names so that they matched between the tree and the region data, and also removing the outgroup from the tree as there was no region data included for the outgroup. These modifications appear to be in line with what the original authors must have done, because the results from both `BioGeoBEARS` and `Lagrange-NG` match the results reported in the paper. Both `BioGeoBEARS` and `Lagrange-NG` were run with the same dataset on the same computer. Despite the fact that the original study limited the number of regions to 5, we decided to also measure `Lagrange-NG`'s performance with no region limit, to show that `Lagrange-NG` can analyse large empirical datasets without a region limit.

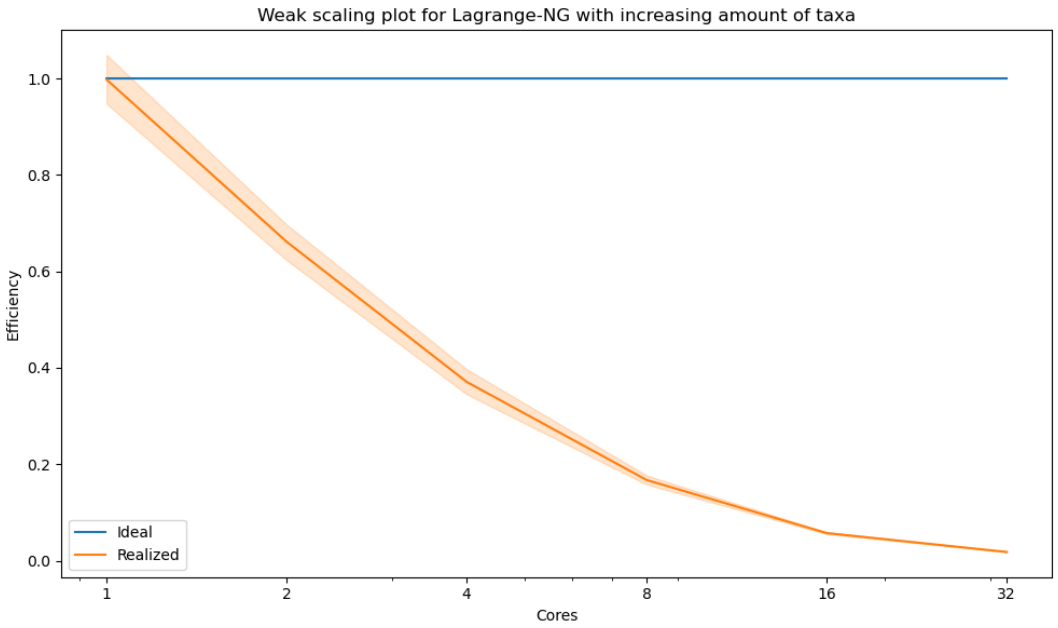

Figure 5.11: Weak parallel scaling plot for `Lagrange-NG`. We ran `Lagrange-NG` 30 times with 50, 100, 200, 400, 800 and 1600 taxa and 7 regions with 1, 2, 4, 8, 16, and 32 workers. Plotted is the efficiency, computed by taking the ratio of the time to execute with 1 worker and the time to execute with n workers. The line labeled "Ideal" is the plotted efficiency if `Lagrange-NG` had perfect scaling with respect to workers.

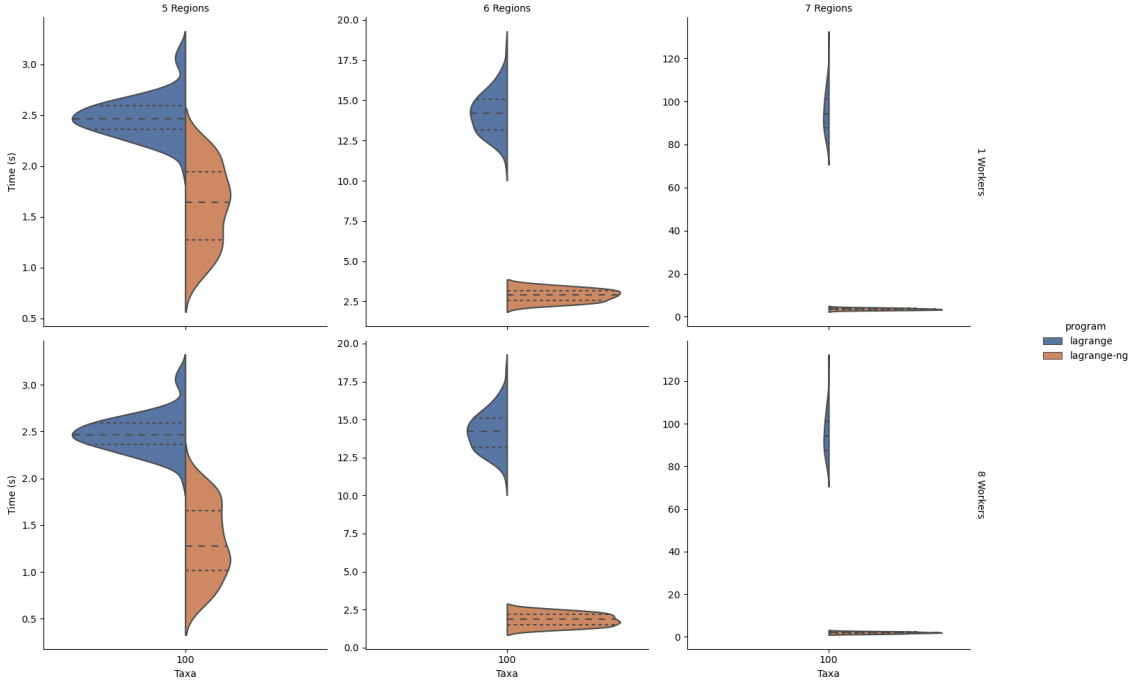

Figure 5.12: Comparison of runtimes between `Lagrange` (left) and `Lagrange-NG` (right) with sequential `Lagrange-NG` (top) and parallel `Lagrange-NG` using 8 workers (bot). Results were obtained by generating 100 random datasets. Note that the original `Lagrange` was not run with any multi-threading, as it does not support it. Instead, the data has been replicated for comparison's sake. Times are in seconds. The figure was generated using seaborn [89]

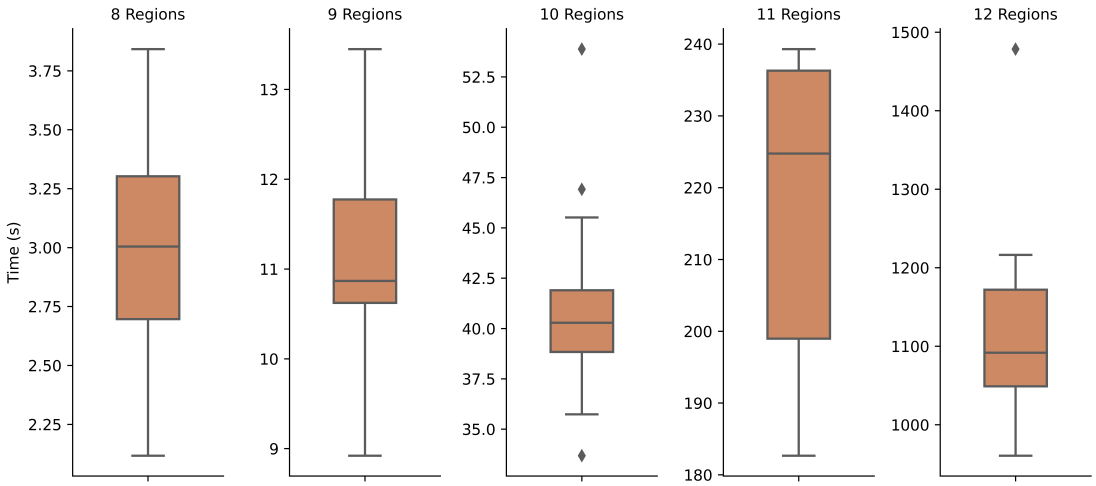

Figure 5.13: Runtimes for `Lagrange-NG` on a larger number of regions when using 8 workers. Results were obtained by generated random datasets with 100 taxa and 8, 9, 10, 11, or 12 regions. We generated 100 random datasets for the 8 region case; for the 9 and 10 region cases we generated 30 datasets; and for the 11 and 12 region cases we generated 10 datasets. Runs were conducted with 8 workers for all cases, and 1 thread per worker in all cases.

5.5 Validation

`Lagrange-NG` re-implements core numerical routines of `Lagrange`. Such changes in numerical routines are often associated with difficult and subtle bugs as well as slight numerical deviations. We sought to ensure that `Lagrange-NG` and `Lagrange` produced the same results. To this end, we developed a pipeline to (i) generate random datasets, (ii) run both, `Lagrange`, and `Lagrange-NG`, and (iii) compare the results of the two programs. To compare results, we developed a measure to evaluate the distance between ancestral range distributions on trees based on the Wasserstein metric [85]. We have already provided the detail of this metric in Section 5.3.2.

We ran this comparison for 100 iterations on datasets comprising 10, 50, and 100 taxa, and a number of regions between 2 and 6. This yielded 15 parameter sets, for a total of 1,500 tests.

5.6 Results

The validation of `Lagrange-NG` with respect to `Lagrange`, was surprisingly successful, despite substantial modifications of nearly all critical code paths and numerical routines. Of the 1,500 tests, 0 produced results with differences over the tolerance of 1×10^{-4} when computed using our novel distance method, indicating that the results are equivalent between the two tools.

The mean sequential speedup between `Lagrange-NG` over `Lagrange` on one core for 5, 6, and 7 regions is 1.54, 4.93, and 26.63, respectively. The overall time-to-solution

speedup of Lagrange-NG with 8 cores over sequential Lagrange on one core for 5, 6, and 7 regions is 1.88, 7.75, and 49.2, respectively. For datasets with larger regions, Lagrange-NG analyzed these datasets with a mean time of 3.00s, 11.12s, 40.75s, 217.01s, and 1130.60s for 8, 9, 10, 11, and 12 regions, respectively.

In addition, Lagrange-NG is substantially faster than BioGeoBEARS. On the empirical dataset, Lagrange-NG computed the result in about 7 seconds using 8 cores, while BioGeoBEARS required about 14 minutes to analyze the data using 80 cores. BioGeoBEARS and Lagrange-NG inferred different optima for model parameters, with BioGeoBEARS achieving a slightly better log-likelihood score (-216.127 vs -224.396). It is unclear if these likelihoods are directly comparable. Nonetheless, this does not affect the respective qualitative results as BioGeoBEARS and Lagrange-NG agree on the most likely distribution for every node.

The analysis of this dataset with no region limit using Lagrange-NG produced similar results to the analysis with the 5 region limit, albeit with a better likelihood (-217.023). The time for this analysis was about 2 seconds using 8 cores.

Comparing runtimes with RevBayes is difficult, as RevBayes and Lagrange-NG produce different kinds of results. RevBayes is a Bayesian analysis software, and as such produces a distribution of parameter values as the posterior. On the other hand, Lagrange-NG only finds the parameter values with the highest likelihood. Nonetheless, the limited range of available tools to compare with necessitates us to use RevBayes as a comparison. We let RevBayes run for $\approx$ 7 hours on the sloth dataset, and in that time RevBayes managed to perform $\approx$ 30 iterations, which is $\approx$ 14 minutes per iteration. Using 3000 iterations as the default number of iterations, a number of iterations suggested by the tutorial for DEC analysis using RevBayes, the full analysis would take $\approx$ 29 days. This is far too simple of a stopping criterion for a realistic Bayesian analysis, as normally a researcher would conduct convergence analysis in order to determine that the distribution of samples has apparently converged to the posterior. Therefore, the actual number of samples required to perform an analysis might be less than 3000. Nonetheless, we report the estimated total analysis time here using 3000 iterations so that readers can have a rough idea of the comparative cost of Lagrange-NG vs RevBayes.

5.7 Discussion

We have shown that computation of likelihood-based biogeographical models can be greatly accelerated without sacrificing result quality. An 2 to 26 fold increase in speed over the original implementation, and over a 100 fold increase in speed over BioGeoBEARS represents a step forward, especially when taking the time complexity of the matrix exponential into account. Additionally, we retain this speed even on datasets with a large number of regions and no region limit, enabling for more fine grained as well as exploratory analyses of biogeographical data.

Readers might wonder why the execution time for the analysis of the empirical dataset with a maximum number of regions is the slower than the analysis without

a maximum number of regions. Ostensibly, a smaller number of regions should lead to a faster execution, but the runtimes shown contradict this. However, for this specific dataset, when using a a maximum number of regions `Lagrange-NG`'s adaptive mode detected numerical issues on nearly all results involving the matrix exponential, and therefore had to fall back to the slower, but safer, method of computing results. Indeed, if we force `Lagrange-NG` to use the faster, but unsafe, mode for computing the matrix exponential, the numerical errors are so excessive that a final result cannot be computed. As a happy accident, this showcases the utility of `Lagrange-NG`'s adaptive mode, where it was able pick the best method of computation without intervention from the user.

Regarding the optimal threading configuration, Figure 5.8 shows that allocating all the available cores to coarse-grained threads is typically optimal. Occasionally, allocating 2 fine threads per worker is slightly faster. This is slightly surprising, and might indicate that the linear algebra library used has potential issues with lock contention. If this is the case, then changing the library might improve results. However, there are still sequential parts of the likelihood computation which do not benefit from the fine grained parallelization, so improving lock contention will have decreasing marginal returns by Amdahl's law.

The threading efficiency of `Lagrange-NG`'s coarse grained parallelization ranges between 0.51 and 0.06. We feel that this is expected, as the method of parallelization is based on the tree topology. Children nodes must be evaluated before parent nodes can be evaluated, which leads to dependencies which prevent optimal parallel efficiency from being achieved. This means that, for every likelihood evaluation, there is a phase in optimal the computation where there is less work available than threads. In this case the excess threads idle. However, we expect that as the number of taxa grows, the efficiency of this method should increase, as the proportion of time spent in this "work starved" phase near the root of the tree is smaller.

5.8 Availability

The software, tools, and data used for this paper are available online at https://github.com/computations/lagrange-ng.

6. Conclusions and Future Work

6.1 Conclusions

In this thesis, I have outlined some of my contributions to computational science, particularly those in phylogenetics. I have presented the tool `RootDigger` which assists in assessing the possible root locations for an unrooted phylogenetic tree. In addition, I presented the tool and method `Phylourny`, which utilizes an efficient method to compute the probable outcomes of knock-out tournaments, and also performs uncertainty analysis for knock out tournaments. In turn, this allows for an improved and more detailed summary of possible outcomes for a given tournament. Finally, I presented `Lagrange-NG`, a tool to infer ancestral ranges using biogeographical models as well as a metric to verify that the results between tools are consistent and comparable.

`RootDigger`, in its simplest mode, allows for a user to automatically root a previously unrooted phylogenetic tree using the most likely root location. In the more involved mode, it allows for a user to explore the relative likelihood of all possible root locations. In addition, `RootDigger` is implemented with Message Passing Interface (MPI) support, which allows for distributed memory parallelization, and to reduce the time-to-solution. This acceleration allows for larger datasets to be analyzed, such as a tree built from 8700 SARS-CoV-2 sequences [61], which enables researchers using the tool to perform more extensive analysis.

Next, I developed `Phylourny`, a tool which predicts the winner of a knockout tournament given historical match data. By using a dynamic programming algorithm we can substantially accelerate the computation of win probabilities for a set of model parameters. This substantially more efficient method also allows us to deploy more costly models which enables us to better explore the total space of outcomes. This in turn allows us to summarize possible outcomes more accurately, which can even help to improve upon the accuracy of inadequate or inappropriate models. Using `Phylourny`, I also performed analysis on two historical tournaments, and analyzed their results. I found them to be accurate, even in the presence of limited data.

Finally, I developed `Lagrange-NG`, re-design and substantial improvement of an older tool which computes ancestral range distributions. This update improves the wall time to solution by a factor of 2 for datasets with a small number of regions, and by a factor of 20 for datasets with a large number of regions. Additionally, I implemented a task based concurrency scheme that further improves upon the time to solution for datasets with a large number of regions by a factor of 50. Furthermore, when compared to a competing tool (`BioGeoBEARS`), `Lagrange-NG` is about two orders of magnitude more efficient. In addition, I devised a novel way to compare ancestral range distributions by developing an appropriate metric. This metric better accounts for the underlying structure of the model, and can therefore more accurately describe the difference or similarity between two results.

6.2 Future Work

Extension to the work presented in this thesis can proceed in several ways, which I will outline in the following sections.

6.2.1 `RootDigger`

First, `RootDigger` can be extended to utilize additional models. These additional models could be the non-reversible versions of models such as JC [41] or K2P [44]. These models have, maybe paradoxically, less parameters than the UNREST model that is currently supported by `RootDigger`. However, having too many parameters can lead to over-fitting of the data which in turn induces inference errors. One option which might be conceptually simpler than manually specifying non-reversible versions of common models is to implement the Lie-group family of non-reversible models described in Woodhams [91], of which these common models are a member of.

In addition to integrating more models, other data types could be supported, in particular Amino Acid (sequence) (AA) data. In this work, we decided to not use AA data as it would increase the number of free parameters from 12 for DNA data to 380. Given this number, we suspect that it is too prone to over-fitting to be useful, but this has never been investigated.

Finally, there are a few parameters that are not part of the model but that could be heuristically set in a less naïve way. These parameters include the number of initial candidate roots in the search mode and the number of roots to fully optimize during each step of the search mode. In this work our default parameters were performing well on simulations, but better results could possibly be obtained via an adaptive strategy, such as conditionally performing more searches based on the results of a fast initial search.

As mentioned in Chapter 3, the parallel efficiency of `RootDigger` could be improved using either of two techniques: Heuristically assigning initial search locations to nodes; or scheduling of initial search locations to compute nodes. In the first technique, we would attempt to estimate how long each root position would take in to

compute relative terms, and then assign the initial search locations in such a way as to better balance the computational load. Traditionally, this can be quite difficult to do effectively, as the heuristic will often need to be finely tuned, which can cause degraded performance on atypical datasets. Alternatively, the initial search locations can be assigned dynamically. In this case, the initial search locations are assigned on demand in a dispatcher-worker scheme, when a node has no computational work to conduct. From this point, it is not clear which method would perform better, and both should be investigated.

6.2.2 `Phylourny`

Given that `Phylourny` is not just an algorithm, but also a set of methods to characterize the uncertainty of knock-out tournaments, the possible extensions to the work are numerous. However, they fall into two broad categories: additional and more complex sports models; or more efficient sampling of the posterior.

At present, `Phylourny` only implements two models: the simplified Independent Poisson Model, and a simple internal test model which was not discussed in Chapter 4. While the Independent Poisson Model performs well, we would expect even better performance from more elaborate models, some of which are also described in Ley et. al. [50]. Of particular note are models which take into account match dates, giving matches which are more recent a higher weight. Given the accuracy of the predictions inferred by `Phylourny` for the UEFA 2020 tournament, which is in part due to very recent data being used for the prediction, it seems that models which incorporate temporal information might be even more successful once implemented in `Phylourny`.

An alternative way that the performance of `Phylourny` could be improved is via a different method of sampling the posterior. I implemented the Metropolis-Hastings algorithm in `Phylourny`, but there are other methods which might perform better, such as Gibbs Sampling [26] or Hamiltonian Monte Carlo [14], either of which might require fewer iterations to converge to the posterior.

6.2.3 `Lagrange-NG`

Future work on `Lagrange-NG` includes extending the range of models that can be computed by the DIVA/DIVALIKE and BAYAREA family of models [48, 74]. Additionally, the current models can be further optimized in three areas, although we expect unspectacular performance improvements.

We produced a version of `Lagrange-NG` that utilized GPU acceleration for the matrix computations. Unfortunately, this method failed to produce acceptable speedups even for large datasets (10-11 regions). This is in line with the performance results of previous attempts to accelerate likelihood computations for phylogenetic tree inference on GPUs [39]. The fundamental difficulty is that the tree-based nature of the computation that induces a decreasing degree of parallelism as we approach the root, leaves many computational units starved for work, as is the case with the existing CPU-based course-grained parallelization of Lagrange-NG. It is possible

that further development would produce better results, but we believe that by the time that datasets become large enough to observe large speedups, the analysis will simply be infeasible due to the exponential nature of the problem.

For remaining improvements to the computational efficiency of `Lagrange-NG`, the first is to further refine the matrix exponential routine. While the current implementation is extremely fast, the implementation in `Lagrange-NG` has not been thoroughly optimized for this particular use case. In particular, the size of the Krylov space has not been thoroughly examined to asses if significant runtime and accuracy trade-offs can be obtained. Additionally, one could further refine the load distribution for the coarse grained parallelization approach. The current method of assigning tasks is straight-forward, and can be improved upon by becoming aware of which nodes are "most blocking" of other tasks. It might be possible to devise an algorithm that can minimize the "task starved" period of computation, either via a clever assignment method, or via a so-called "work stealing" scheme.

Bibliography

[1] Guy Baele et al. "Accurate Model Selection of Relaxed Molecular Clocks in Bayesian Phylogenetics". en. In: *Molecular Biology and Evolution* 30.2 (Feb. 2012), pp. 239–243. ISSN: 0737-4038, 1537-1719. DOI: 10.1093/molbev/mss243. URL: https://academic.oup.com/mbe/article/30/2/239/1018357 (visited on 07/16/2020).

[2] Fabia U. Battistuzzi et al. "Performance of Relaxed-Clock Methods in Estimating Evolutionary Divergence Times and Their Credibility Intervals". en. In: *Molecular Biology and Evolution* 27.6 (June 2010), pp. 1289–1300. ISSN: 0737-4038. DOI: 10.1093/molbev/msq014. URL: https://academic.oup.com/mbe/article/27/6/1289/1111317 (visited on 06/28/2019).

[3] Ben Bettisworth, Stephen A Smith, and Alexandros Stamatakis. "Lagrange-NG: The next generation of Lagrange". In: *Systematic Biology* (Jan. 2023), syad002. ISSN: 1063-5157. DOI: 10.1093/sysbio/syad002. URL: https://doi.org/10.1093/sysbio/syad002 (visited on 02/13/2023).

[4] Ben Bettisworth and Alexandros Stamatakis. "Root Digger: a root placement program for phylogenetic trees". In: *BMC Bioinformatics* 22.1 (May 2021), p. 225. ISSN: 1471-2105. DOI: 10.1186/s12859-021-03956-5. URL: https://doi.org/10.1186/s12859-021-03956-5 (visited on 02/13/2023).

[5] Benjamin P. Blackburne and Simon Whelan. "Class of Multiple Sequence Alignment Algorithm Affects Genomic Analysis". In: *Molecular Biology and Evolution* 30.3 (Mar. 2013), pp. 642–653. ISSN: 0737-4038. DOI: 10.1093/molbev/mss256. URL: https://doi.org/10.1093/molbev/mss256 (visited on 02/09/2023).

[6] R. P. Brent. "An algorithm with guaranteed convergence for finding a zero of a function". In: *The Computer Journal* 14.4 (Jan. 1971), pp. 422–425. ISSN: 0010-4620. DOI: 10.1093/comjnl/14.4.422. URL: https://doi.org/10.1093/comjnl/14.4.422 (visited on 12/13/2022).

[7] Peter N. Brown and Youcef Saad. "Hybrid Krylov Methods for Nonlinear Systems of Equations". In: *SIAM Journal on Scientific and Statistical Computing* 11.3 (May 1990). Publisher: Society for Industrial and Applied Mathematics, pp. 450–481. ISSN: 0196-5204. DOI: 10.1137/0911026. URL: https://epubs.siam.org/doi/abs/10.1137/0911026 (visited on 12/13/2022).

[8] Joseph H. Camin and Robert R. Sokal. "A Method for Deducing Branching Sequences in Phylogeny". In: *Evolution* 19.3 (Sept. 1965), p. 311. ISSN: 00143820. DOI: 10.2307/2406441. URL: https://www.jstor.org/stable/2406441?origin=crossref (visited on 11/14/2022).

[9] Pascal-Antoine Christin et al. "Molecular Dating, Evolutionary Rates, and the Age of the Grasses". en. In: *Systematic Biology* 63.2 (Mar. 2014), pp. 153–165. ISSN: 1063-5157. DOI: 10.1093/sysbio/syt072. URL: https://academic.oup.com/sysbio/article/63/2/153/1643272 (visited on 11/14/2019).

[10] Astrid Cruaud et al. "An Extreme Case of Plant–Insect Codiversification: Figs and Fig-Pollinating Wasps". en. In: *Systematic Biology* 61.6 (Dec. 2012), pp. 1029–1047. ISSN: 1063-5157. DOI: 10.1093/sysbio/sys068. URL: https://academic.oup.com/sysbio/article/61/6/1029/1667297 (visited on 12/05/2019).

[11] Charles Darwin. *On the origin of species, 1859*. Routledge, 2004.

[12] Richard Demsyn-Jones. "Misadventures in Monte Carlo". en. In: *Journal of Sports Analytics* 5.1 (Mar. 2019), pp. 1–15. ISSN: 2215020X, 22150218. (Visited on 06/15/2021).

[13] *Developer Reference for Intel® oneAPI Math Kernel Library - C*. en. URL: https://www.intel.com/content/www/us/en/develop/documentation/onemkl-developer-reference-c/top.html (visited on 03/15/2022).

[14] Simon Duane et al. "Hybrid Monte Carlo". en. In: *Physics Letters B* 195.2 (Sept. 1987), pp. 216–222. ISSN: 0370-2693. DOI: 10.1016/0370-2693(87)91197-X. URL: https://www.sciencedirect.com/science/article/pii/037026938791197X (visited on 03/02/2023).

[15] Casey W. Dunn et al. "Animal Phylogeny and Its Evolutionary Implications". en. In: *Annual Review of Ecology, Evolution, and Systematics* 45.1 (Nov. 2014), pp. 371–395. ISSN: 1543-592X, 1545-2069. DOI: 10.1146/annurev-ecolsys-120213-091627. URL: http://www.annualreviews.org/doi/10.1146/annurev-ecolsys-120213-091627 (visited on 11/04/2019).

[16] Noa Ecker et al. "A LASSO-based approach to sample sites for phylogenetic tree search". In: *Bioinformatics* 38.Supplement_1 (July 2022), pp. i118–i124. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btac252. URL: https://doi.org/10.1093/bioinformatics/btac252 (visited on 02/13/2023).

[17] Claus Thorn Ekstrøm et al. "Evaluating one-shot tournament predictions". en. In: *Journal of Sports Analytics* 7.1 (Apr. 2021), pp. 37–46. ISSN: 2215020X, 22150218. (Visited on 06/15/2021).

[18] David M. Emms and Steven Kelly. "STRIDE: Species Tree Root Inference from Gene Duplication Events". en. In: *Molecular Biology and Evolution* 34.12 (Dec. 2017), pp. 3267–3278. ISSN: 0737-4038. DOI: 10.1093/molbev/msx259. URL: https://academic.oup.com/mbe/article/34/12/3267/4259048 (visited on 12/04/2019).

[19] Steven N. Evans and Frederick A. Matsen. "The phylogenetic Kantorovich–Rubinstein metric for environmental sequence samples". en. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 74.3 (2012), pp. 569–592. ISSN: 1467-9868. DOI: 10.1111/j.1467-9868.2011.01018.x. (Visited on 04/06/2022).

[20] Joseph Felsenstein. "Cases in which Parsimony or Compatibility Methods will be Positively Misleading". In: *Systematic Biology* 27.4 (Dec. 1978), pp. 401–410. ISSN: 1063-5157. DOI: 10.1093/sysbio/27.4.401. eprint: https://academic.oup.com/sysbio/article-pdf/27/4/401/4828731/27-4-401.pdf. URL: https://doi.org/10.1093/sysbio/27.4.401.

[21] Joseph Felsenstein. "Evolutionary trees from DNA sequences: A maximum likelihood approach". en. In: *Journal of Molecular Evolution* 17.6 (Nov. 1981), pp. 368–376. ISSN: 1432-1432. DOI: 10.1007/BF01734359. URL: https://doi.org/10.1007/BF01734359 (visited on 11/14/2022).

[22] William Fletcher and Ziheng Yang. "INDELible: A Flexible Simulator of Biological Sequence Evolution". en. In: *Molecular Biology and Evolution* 26.8 (Aug. 2009), pp. 1879–1888. ISSN: 0737-4038. DOI: 10.1093/molbev/msp098. URL: https://academic.oup.com/mbe/article/26/8/1879/980884 (visited on 07/02/2019).

[23] T. Flouri et al. "The Phylogenetic Likelihood Library". en. In: *Systematic Biology* 64.2 (Mar. 2015), pp. 356–362. ISSN: 1063-5157. DOI: 10.1093/sysbio/syu084. URL: https://academic.oup.com/sysbio/article/64/2/356/1630375 (visited on 07/03/2019).

[24] John Gatesy, Rob DeSalle, and Niklas Wahlberg. "How Many Genes Should a Systematist Sample? Conflicting Insights from a Phylogenomic Matrix Characterized by Replicated Incongruence". en. In: *Systematic Biology* 56.2 (Apr. 2007), pp. 355–363. ISSN: 1063-5157. DOI: 10.1080/10635150701294733. URL: https://academic.oup.com/sysbio/article/56/2/355/1691194 (visited on 06/27/2019).

[25] GCC team. *GCC online documentation - GNU Project*. Accessed on 12 August 2022. 2022. URL: https://gcc.gnu.org/onlinedocs/ (visited on 08/12/2022).

[26] Stuart Geman and Donald Geman. "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-6.6 (Nov. 1984). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 721–741. ISSN: 1939-3539. DOI: 10.1109/TPAMI.1984.4767596.

[27] Andreas Groll et al. *Hybrid Machine Learning Forecasts for the FIFA Women's World Cup 2019*. 2019. arXiv: 1906.01131 [stat.ML].

[28] Julia Haag et al. "From Easy to Hopeless—Predicting the Difficulty of Phylogenetic Analyses". In: *Molecular Biology and Evolution* 39.12 (Dec. 2022), msac254. ISSN: 1537-1719. DOI: 10.1093/molbev/msac254. URL: https://doi.org/10.1093/molbev/msac254 (visited on 02/13/2023).

[29] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[30] Falk Hildebrand, Axel Meyer, and Adam Eyre-Walker. “Evidence of Selection upon Genomic GC-Content in Bacteria”. en. In: *PLOS Genetics* 6.9 (Sept. 2010). Publisher: Public Library of Science, e1001107. ISSN: 1553-7404. DOI: 10.1371/journal.pgen.1001107. URL: https://journals.plos.org/plosgenetics/article?id=10.1371/journal.pgen.1001107 (visited on 01/03/2023).

[31] S. E. Hill. “In-game win probability models for Canadian football”. In: *Journal of Business Analytics* 0 (Dec. 2021). published online. ISSN: 2573-234X. DOI: 10.1080/2573234X.2021.2015252. (Visited on 07/26/2022).

[32] Oliver Hobert. “The Impact of Whole Genome Sequencing on Model System Genetics: Get Ready for the Ride”. In: *Genetics* 184.2 (Feb. 2010), pp. 317–319. ISSN: 1943-2631. DOI: 10.1534/genetics.109.112938. URL: https://doi.org/10.1534/genetics.109.112938 (visited on 02/09/2023).

[33] Marlis Hochbruck and Christian Lubich. “On Krylov Subspace Approximations to the Matrix Exponential Operator”. In: *SIAM Journal on Numerical Analysis* 34.5 (Oct. 1997). Publisher: Society for Industrial and Applied Mathematics, pp. 1911–1925. ISSN: 0036-1429. DOI: 10.1137/S0036142995280572. URL: https://epubs.siam.org/doi/abs/10.1137/S0036142995280572 (visited on 01/03/2023).

[34] B. R. Holland, D. Penny, and M. D. Hendy. “Outgroup Misplacement and Phylogenetic Inaccuracy Under a Molecular Clock—A Simulation Study”. en. In: *Systematic Biology* 52.2 (Apr. 2003), pp. 229–238. ISSN: 1063-5157. DOI: 10.1080/10635150390192771. URL: https://academic.oup.com/sysbio/article/52/2/229/1634389 (visited on 06/27/2019).

[35] A. S. Mukarram Hossain et al. “Evidence of Statistical Inconsistency of Phylogenetic Methods in the Presence of Multiple Sequence Alignment Uncertainty”. In: *Genome Biology and Evolution* 7.8 (Aug. 2015), pp. 2102–2116. ISSN: 1759-6653. DOI: 10.1093/gbe/evv127. URL: https://doi.org/10.1093/gbe/evv127 (visited on 02/09/2023).

[36] John P. Huelsenbeck, Jonathan P. Bollback, and Amy M. Levine. “Inferring the Root of a Phylogenetic Tree”. en. In: *Systematic Biology* 51.1 (Jan. 2002), pp. 32–43. ISSN: 1063-5157. DOI: 10.1080/106351502753475862. URL: https://academic.oup.com/sysbio/article/51/1/32/1631340 (visited on 06/27/2019).

[37] Jaime Huerta-Cepas, François Serra, and Peer Bork. “ETE 3: Reconstruction, Analysis, and Visualization of Phylogenomic Data”. en. In: *Molecular Biology and Evolution* 33.6 (June 2016), pp. 1635–1638. ISSN: 0737-4038. DOI: 10.1093/molbev/msw046. URL: https://academic.oup.com/mbe/article/33/6/1635/2579822 (visited on 07/02/2019).

[38] Lars Magnus Hvattum and Halvard Arntzen. “Using ELO ratings for match result prediction in association football”. en. In: *International Journal of Forecasting*. Sports Forecasting 26.3 (July 2010), pp. 460–470. ISSN: 0169-2070.

[39] Fernando Izquierdo-Carrasco et al. "A Generic Vectorization Scheme and a GPU Kernel for the Phylogenetic Likelihood Library". In: *2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum*. May 2013, pp. 530–538. DOI: 10.1109/IPDPSW.2013.103.

[40] Steven G. Johnson. *stevengj/nlopt*. original-date: 2013-08-27T16:59:11Z. Nov. 2021. URL: https://github.com/stevengj/nlopt (visited on 11/30/2021).

[41] Thomas H. Jukes and Charles R. Cantor. "CHAPTER 24 - Evolution of Protein Molecules". en. In: *Mammalian Protein Metabolism*. Ed. by H. N. Munro. Academic Press, Jan. 1969, pp. 21–132. ISBN: 978-148-323-2-1-1-9. DOI: 10.1016/B978-1-4832-3211-9.50009-7. URL: https://www.sciencedirect.com/science/article/pii/B9781483232119500097 (visited on 11/14/2022).

[42] Edward H. Kaplan, Kevin Mongeon, and John T. Ryan. "A Markov model for hockey: Manpower differential and win probability added". In: *INFOR: Information Systems and Operational Research* 52.2 (May 2014), pp. 39–50. ISSN: 0315-5986. (Visited on 07/26/2022).

[43] Lisandro Kaunitz, Shenjun Zhong, and Javier Kreiner. "Beating the bookies with their own numbers - and how the online sports betting market is rigged". In: *arXiv:1710.02824 [cs, stat]* (Nov. 2017).

[44] Motoo Kimura. "A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences". en. In: *Journal of Molecular Evolution* 16.2 (June 1980), pp. 111–120. ISSN: 1432-1432. DOI: 10.1007/BF01731581. URL: https://doi.org/10.1007/BF01731581 (visited on 02/28/2023).

[45] Alexey M Kozlov et al. "RAxML-NG: a fast, scalable and user-friendly tool for maximum likelihood phylogenetic inference". In: *Bioinformatics* 35.21 (Nov. 2019), pp. 4453–4455. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btz305. URL: https://doi.org/10.1093/bioinformatics/btz305 (visited on 02/13/2023).

[46] James A Lake. "The order of sequence alignment can bias the selection of tree topology." In: *Molecular biology and evolution* 8.3 (1991), pp. 378–385.

[47] Giddy Landan and Dan Graur. "Characterization of pairwise and multiple sequence alignment errors". en. In: *Gene*. Phylogenomics and its Future: Devoted to Masami Hasegawa 441.1 (July 2009), pp. 141–147. ISSN: 0378-1119. DOI: 10.1016/j.gene.2008.05.016. URL: https://www.sciencedirect.com/science/article/pii/S0378111908002229 (visited on 02/09/2023).

[48] Michael J. Landis et al. "Bayesian Analysis of Biogeography when the Number of Areas is Large". In: *Systematic Biology* 62.6 (Nov. 2013), pp. 789–804. ISSN: 1063-5157. DOI: 10.1093/sysbio/syt040. (Visited on 04/14/2022).

[49] Geneviève Leduc-Robert and Wayne P. Maddison. "Phylogeny with introgression in Habronattus jumping spiders (Araneae: Salticidae)". In: *BMC Evolutionary Biology* 18.1 (Feb. 2018), p. 24. ISSN: 1471-2148. DOI: 10.1186/s12862-018-1137-x. URL: https://doi.org/10.1186/s12862-018-1137-x (visited on 11/14/2019).

[50] Christophe Ley, Tom Van de Wiele, and Hans Van Eetvelde. "Ranking soccer teams on the basis of their current strength: A comparison of maximum likelihood approaches". en. In: *Statistical Modelling* 19.1 (Feb. 2019). Publisher: SAGE Publications India, pp. 55–73. ISSN: 1471-082X.

[51] Wen-Hsiung Li and Masako Tanimura. "The molecular clock runs more slowly in man than in apes and monkeys". En. In: *Nature* 326.6108 (Mar. 1987), p. 93. ISSN: 1476-4687. DOI: 10.1038/326093a0. URL: https://www.nature.com/articles/326093a0 (visited on 06/28/2019).

[52] Dennis Lock and Dan Nettleton. "Using random forests to estimate win probability before each play of an NFL game". en. In: *Journal of Quantitative Analysis in Sports* 10.2 (June 2014), pp. 197–205. ISSN: 1559-0410. (Visited on 07/26/2022).

[53] Michael J. Lopez, Gregory J. Matthews, and Benjamin S. Baumer. "How often does the best team win? A unified approach to understanding randomness in North American sport". In: *The Annals of Applied Statistics* 12.4 (2018). Publisher: Institute of Mathematical Statistics, pp. 2483–2516. ISSN: 1932-6157.

[54] Uyen Mai, Erfan Sayyari, and Siavash Mirarab. "Minimum variance rooting of phylogenetic trees and implications for species tree reconstruction". en. In: *PLOS ONE* 12.8 (Aug. 2017). Publisher: Public Library of Science, e0182238. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0182238. URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0182238 (visited on 11/11/2020).

[55] Mark Adler. *A Massively Spiffy Yet Delicately Unobtrusive Compression Library (Also Free, Not to Mention Unencumbered by Patents).*

[56] Nicholas J. Matzke. *BioGeoBEARS: BioGeography with Bayesian (and Likelihood) Evolutionary Analysis in R Scripts*. University of California, Berkeley. Berkeley, CA, 2013. URL: https://CRAN.R-project.org/package=BioGeoBEARS.

[57] Nicholas Metropolis et al. "Equation of State Calculations by Fast Computing Machines". In: *The Journal of Chemical Physics* 21.6 (June 1953). Publisher: American Institute of Physics, pp. 1087–1092. ISSN: 0021-9606.

[58] Bui Quang Minh et al. "IQ-TREE 2: New models and efficient methods for phylogenetic inference in the genomic era". en. In: *bioRxiv* (Nov. 2019), p. 849372. DOI: 10.1101/849372. URL: https://www.biorxiv.org/content/10.1101/849372v1 (visited on 01/15/2020).

[59] Cleve Moler and Charles Van Loan. "Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later". en. In: *SIAM Review* 45.1 (2003), pp. 3–49.

[60] Benoit Morel et al. "GeneRax: A tool for species tree-aware maximum likelihood based gene tree inference under gene duplication, transfer, and loss". en. In: *bioRxiv* (Sept. 2019), p. 779066. DOI: 10.1101/779066. URL: https://www.biorxiv.org/content/10.1101/779066v1 (visited on 12/30/2019).

[61] Benoit Morel et al. "Phylogenetic analysis of SARS-CoV-2 data is difficult". In: *Molecular Biology and Evolution* 38.5 (May 2021), pp. 1777–1791. ISSN: 0737-4038.

[62] Paul Muir et al. "The real cost of sequencing: scaling computation to keep pace with data generation". In: *Genome Biology* 17.1 (Mar. 2016), p. 53. ISSN: 1474-760X. DOI: 10.1186/s13059-016-0917-0. URL: https://doi.org/10.1186/s13059-016-0917-0 (visited on 02/13/2023).

[63] J. A. Nelder and R. Mead. "A Simplex Method for Function Minimization". In: *The Computer Journal* 7.4 (Jan. 1965), pp. 308–313. ISSN: 0010-4620. DOI: 10.1093/comjnl/7.4.308. (Visited on 11/30/2021).

[64] Lam-Tung Nguyen et al. "IQ-TREE: A Fast and Effective Stochastic Algorithm for Estimating Maximum-Likelihood Phylogenies". en. In: *Molecular Biology and Evolution* 32.1 (Jan. 2015), pp. 268–274. ISSN: 0737-4038. DOI: 10.1093/molbev/msu300. URL: https://academic.oup.com/mbe/article/32/1/268/2925592 (visited on 04/11/2019).

[65] Jason A. O'Rawe, Scott Ferson, and Gholson J. Lyon. "Accounting for uncertainty in DNA sequencing data". en. In: *Trends in Genetics* 31.2 (Feb. 2015), pp. 61–66. ISSN: 0168-9525. DOI: 10.1016/j.tig.2014.12.002. URL: https://www.sciencedirect.com/science/article/pii/S0168952514002091 (visited on 02/13/2023).

[66] *OpenBLAS : An optimized BLAS library*. URL: http://www.openblas.net/ (visited on 11/30/2021).

[67] OpenMP Architecture Review Board. *OpenMP Application Program Interface Version 4.5*. Nov. 2015. URL: https://www.openmp.org/wp-content/uploads/openmp-4.5.pdf.

[68] OpenMP Architecture Review Board. *OpenMP Application Program Interface Version 5.0*. 2020. URL: https://www.openmp.org/spec-html/5.0/openmp.html.

[69] WH Piel et al. *Treebase v. 2: A database of phylogenetic knowledge. e-Biosphere*. 2009.

[70] Jin-Hua Ran et al. "Phylogenomics resolves the deep phylogeny of seed plants and indicates partial convergent or homoplastic evolution between Gnetales and angiosperms". In: *Proceedings of the Royal Society B: Biological Sciences* 285.1881 (June 2018), p. 20181012. DOI: 10.1098/rspb.2018.1012. URL: https://royalsocietypublishing.org/doi/10.1098/rspb.2018.1012 (visited on 11/14/2019).

[71] Thiago F. Rangel et al. "Phylogenetic uncertainty revisited: Implications for ecological analyses". In: *Evolution* 69.5 (May 2015), pp. 1301–1312. ISSN: 0014-3820. DOI: 10.1111/evo.12644. URL: https://doi.org/10.1111/evo.12644 (visited on 02/09/2023).

[72] Richard H. Ree et al. "A Likelihood Framework for Inferring the Evolution of Geographic Range on Phylogenetic Trees". en. In: *Evolution* 59.11 (2005), pp. 2299–2311. ISSN: 1558-5646. DOI: 10.1111/j.0014-3820.2005.tb00940.x. (Visited on 06/10/2020).

[73] D. F. Robinson and L. R. Foulds. "Comparison of phylogenetic trees". In: *Mathematical Biosciences* 53.1 (Feb. 1981), pp. 131–147. ISSN: 0025-5564. DOI: 10.1016/0025-5564(81)90043-2. URL: http://www.sciencedirect.com/science/article/pii/0025556481900432 (visited on 07/02/2019).

[74] F Ronquist. "DIVA version 1.2. Computer program for MacOS and Win32". In: *Evolutionary Biology Centre, Uppsala University. Available at http://www. ebc. uu. se/systzoo/research/diva/diva. html* (2001).

[75] Fredrik Ronquist and John P. Huelsenbeck. "MrBayes 3: Bayesian phylogenetic inference under mixed models". en. In: *Bioinformatics* 19.12 (Aug. 2003), pp. 1572–1574. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btg180. URL: https://academic.oup.com/bioinformatics/article/19/12/1572/257621 (visited on 06/28/2019).

[76] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1-4414-1269-7.

[77] Alexandros Stamatakis. "RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies". en. In: *Bioinformatics* 30.9 (May 2014), pp. 1312–1313. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btu033. URL: https://academic.oup.com/bioinformatics/article/30/9/1312/238053 (visited on 04/11/2019).

[78] Michael E. Steiper and Nathan M. Young. "Primate molecular divergence dates". In: *Molecular Phylogenetics and Evolution* 41.2 (Nov. 2006), pp. 384–394. ISSN: 1055-7903. DOI: 10.1016/j.ympev.2006.05.021. URL: http://www.sciencedirect.com/science/article/pii/S1055790306001953 (visited on 06/28/2019).

[79] Korbinian Strimmer and Andrew Rambaut. "Inferring confidence sets of possibly misspecified gene trees". In: *Proceedings of the Royal Society of London. Series B: Biological Sciences* 269.1487 (Jan. 2002), pp. 137–142. DOI: 10.1098/rspb.2001.1862. URL: https://royalsocietypublishing.org/doi/abs/10.1098/rspb.2001.1862 (visited on 11/29/2019).

[80] R. J. Tillyard. "A New Classification Of The Order Perlaria". en. In: *The Canadian Entomologist* 53.2 (Feb. 1921). Publisher: Cambridge University Press, pp. 35–43. ISSN: 1918-3240, 0008-347X. DOI: 10.4039/Ent5335-2. URL: https://www.cambridge.org/core/journals/canadian-entomologist/article/abs/new-classification-of-the-order-perlaria/19A45868AC05A1048D88466C6210608D (visited on 03/08/2023).

[81] Ole K Tørresen et al. "Tandem repeats lead to sequence assembly errors and impose multi-level challenges for genome and protein databases". In: *Nucleic Acids Research* 47.21 (Dec. 2019), pp. 10994–11006. ISSN: 0305-1048. DOI: 10.1093/nar/gkz841. URL: https://doi.org/10.1093/nar/gkz841 (visited on 02/09/2023).

[82] Fernando Domingues Kümmel Tria, Giddy Landan, and Tal Dagan. "Phylogenetic rooting using minimal ancestor deviation". en. In: *Nature Ecology & Evolution* 1.1 (June 2017). Number: 1 Publisher: Nature Publishing Group, pp. 1–7. ISSN: 2397-334X. DOI: 10.1038/s41559-017-0193. URL: https://www.nature.com/articles/s41559-017-0193 (visited on 04/14/2020).

[83] Alkeos Tsokos et al. "Modeling outcomes of soccer matches". en. In: *Machine Learning* 108.1 (Jan. 2019), pp. 77–95. ISSN: 1573-0565. DOI: 10.1007/s10994-018-5741-1. URL: https://doi.org/10.1007/s10994-018-5741-1 (visited on 02/26/2023).

[84] Luciano Varela et al. "Phylogeny, Macroevolutionary Trends and Historical Biogeography of Sloths: Insights From a Bayesian Morphological Clock Analysis". In: *Systematic Biology* 68.2 (Mar. 2019), pp. 204–218. ISSN: 1063-5157. DOI: 10.1093/sysbio/syy058. (Visited on 12/17/2021).

[85] L N Vaserstein. "Markov Processes over Denumerable Products of Spaces, Describing Large Systems of Automata". ru. In: (1969), p. 10.

[86] Alexandros Vasilikopoulos et al. "Phylogenomics of the superfamily Dytiscoidea (Coleoptera: Adephaga) with an evaluation of phylogenetic conflict and systematic error". en. In: *Molecular Phylogenetics and Evolution* 135 (June 2019), pp. 270–285. ISSN: 1055-7903. DOI: 10.1016/j.ympev.2019.02.022. URL: http://www.sciencedirect.com/science/article/pii/S1055790318306493 (visited on 09/11/2020).

[87] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[88] Rutger A. Vos et al. "NeXML: Rich, Extensible, and Verifiable Representation of Comparative Data and Metadata". en. In: *Systematic Biology* 61.4 (July 2012), pp. 675–689. ISSN: 1063-5157. DOI: 10.1093/sysbio/sys025. URL: https://academic.oup.com/sysbio/article/61/4/675/1640188 (visited on 01/22/2020).

[89] Michael L. Waskom. "seaborn: statistical data visualization". In: *Journal of Open Source Software* 6.60 (2021), p. 3021. DOI: 10.21105/joss.03021. URL: https://doi.org/10.21105/joss.03021.

[90] Karen M. Wong, Marc A. Suchard, and John P. Huelsenbeck. "Alignment Uncertainty and Genomic Analysis". In: *Science* 319.5862 (Jan. 2008). Publisher: American Association for the Advancement of Science, pp. 473–476. DOI: 10.1126/science.1151532. URL: https://www.science.org/doi/abs/10.1126/science.1151532 (visited on 02/13/2023).

[91] Michael D. Woodhams, Jesús Fernández-Sánchez, and Jeremy G. Sumner. "A New Hierarchy of Phylogenetic Models Consistent with Heterogeneous Substitution Rates". en. In: *Systematic Biology* 64.4 (July 2015), pp. 638–650. ISSN: 1063-5157. DOI: 10.1093/sysbio/syv021. URL: https://academic.oup.com/sysbio/article/64/4/638/1650486 (visited on 07/04/2019).

[92] Ziheng Yang. *Computational Molecular Evolution*. en. OUP Oxford, Oct. 2006. ISBN: 978-019-856-6-9-9-1.

[93] Ziheng Yang et al. *Computational molecular evolution*. Vol. 284. Oxford University Press Oxford, 2006.

[94] Ziheng Yang. "Estimating the pattern of nucleotide substitution". en. In: *Journal of Molecular Evolution* 39.1 (July 1994), pp. 105–111. ISSN: 1432-1432. DOI: 10.1007/BF00178256. URL: https://doi.org/10.1007/BF00178256 (visited on 01/16/2020).

[95] Von Bing Yap and Terry Speed. "Rooting a phylogenetic tree with nonreversible substitution models". In: *BMC Evolutionary Biology* 5.1 (Jan. 2005), p. 2. ISSN: 1471-2148. DOI: 10.1186/1471-2148-5-2. URL: https://doi.org/10.1186/1471-2148-5-2 (visited on 06/17/2019).

[96] Adrian Zapletal et al. "The SoftWipe tool and benchmark for assessing coding standards adherence of scientific software". en. In: *Scientific Reports* 11.1 (May 2021), p. 10015. ISSN: 2045-2322. DOI: 10.1038/s41598-021-89495-8. (Visited on 03/11/2022).

[97] Ciyou Zhu et al. "Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization". In: *ACM Transactions on Mathematical Software (TOMS)* 23.4 (Dec. 1997), pp. 550–560. ISSN: 0098-3500. DOI: 10.1145/279232.279236. URL: https://doi.org/10.1145/279232.279236 (visited on 01/16/2020).