High-Performance approaches for Phylogenetic Placement, and its application to species and diversity quantification

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik

des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Pierre Barbera

Tag der mündlichen Prüfung:	25.10.2021
1. Referent:	Prof. Dr. Alexandros Stamatakis
2. Referent:	Prof. Dr. Thomas Ludwig

Für Oma Rosa

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, 01.05.2021

. . . .

(**Pierre Barbera**)

Zusammenfassung

In den letzten Jahren haben Fortschritte in der Hochdurchsatz-Genesequenzierung, in Verbindung mit dem anhaltenden exponentiellen Wachstum und der Verfügbarkeit von Rechenressourcen, zu fundamental neuen analytischen Ansätzen in der Biologie geführt. Es ist nun möglich den genetischen Inhalt ganzer Organismengemeinschaften anhand einzelner Umweltproben umfassend zu sequenzieren. Solche Methoden sind besonders für die Mikrobiologie relevant. Die Mikrobiologie war zuvor weitgehend auf die Untersuchung jener Mikroben beschränkt, welche im Labor (d.h., *in vitro*) kultiviert werden konnten, was jedoch lediglich einen kleinen Teil der in der Natur vorkommenden Diversität abdeckt. Im Gegensatz dazu ermöglicht die Hochdurchsatzsequenzierung nun die direkte Erfassung der genetischen Sequenzen eines Mikrobioms, wie es in seiner natürlichen Umgebung vorkommt (d.h., *in situ*).

Ein typisches Ziel von Mikrobiomstudien besteht in der taxonomischen Klassifizierung der in einer Probe enthaltenen Sequenzen (*Query*sequenzen). Üblicherweise werden phylogenetische Methoden eingesetzt, um detaillierte taxonomische Beziehungen zwischen Querysequenzen und vertrauenswürdigen *Referenz*sequenzen, die von bereits klassifizierten Organismen stammen, zu bestimmen. Aufgrund des hohen Volumens (10^6 bis 10^9) von Querysequenzen, die aus einer Mikrobiom-Probe mittels Hochdurchsatzsequenzierung generiert werden können, ist eine akkurate phylogenetische Baumrekonstruktion rechnerisch nicht mehr möglich. Darüber hinaus erzeugen derzeit üblicherweise verwendete Sequenzierungstechnologien vergleichsweise kurze Sequenzen, die ein begrenztes phylogenetisches Signal aufweisen, was zu einer Instabilität bei der Inferenz der Phylogenien aus diesen Sequenzen führt.

Ein weiteres typisches Ziel von Mikrobiomstudien besteht in der Quantifizierung der Diversität innerhalb einer Probe, bzw. zwischen mehreren Proben. Auch hierfür werden üblicherweise phylogenetische Methoden verwendet. Oftmals setzen diese Methoden die Inferenz eines phylogenetischen Baumes voraus, welcher entweder alle Sequenzen, oder eine geclusterte Teilmenge dieser Sequenzen, umfasst. Wie bei der taxonomischen Identifizierung können Analysen, die auf dieser Art von Bauminferenz basieren, zu ungenauen Ergebnissen führen und/oder rechnerisch nicht durchführbar sein.

Im Gegensatz zu einer umfassenden phylogenetischen Inferenz ist die phylogenetische Platzierung eine Methode, die den phylogenetischen Kontext einer Querysequenz innerhalb eines etablierten Referenzbaumes bestimmt. Dieses Verfahren betrachtet den Referenzbaum typischerweise als unveränderlich, d.h. der Referenzbaum wird vor, während oder nach der Platzierung einer Sequenz nicht geändert. Dies erlaubt die phylogenetische Platzierung einer Sequenz in linearer Zeit in Bezug auf die Größe des Referenzbaums durchzuführen. In Kombination mit taxonomischen Informationen über die Referenzsequenzen ermöglicht die phylogenetische Platzierung somit die taxonomische Identifizierung einer Sequenz. Darüber hinaus erlaubt eine phylogenetische Platzierung die Anwendung einer Vielzahl zusätzlicher Analyseverfahren, die beispielsweise die Zuordnung der Zusammensetzungen humaner Mikrobiome zu klinisch-diagnostischen Eigenschaften ermöglicht.

In dieser Dissertation präsentiere ich meine Arbeit bezüglich des Entwurfs, der Implementierung, und Verbesserung von EPA-ng, einer Hochleistungsimplementierung der phylogenetischen Platzierung anhand des Maximum-Likelihood Modells. EPA-ng wurde entwickelt um auf Milliarden von Querysequenzen zu skalieren und auf Tausenden von Kernen in Systemen mit gemeinsamem und verteiltem Speicher ausgeführt zu werden. EPA-ng beschleunigt auch die Verarbeitungsgeschwindigkeit auf einzelnen Kernen um das bis zu 30-fache, im Vergleich zu dessen direkten Konkurrenzprogrammen. Vor kurzem haben wir eine zusätzliche Methode für EPA-ng eingeführt, welche die Platzierung in wesentlich größeren Referenzbäumen ermöglicht. Hierfür verwenden wir einen aktiven Speicherverwaltungsansatz, bei dem reduzierter Speicherverbrauch gegen größere Ausführungszeiten eingetauscht wird.

Zusätzlich präsentiere ich einen massiv-parallelen Ansatz um die Diversität einer Probe zu quantifizieren, welcher auf den Ergebnissen phylogenetischer Platzierungen basiert. Diese Software, genannt SCRAPP, kombiniert aktuelle Methoden für die Maximum-Likelihood basierte phylogenetische Inferenz mit Methoden zur Abgrenzung molekularer Spezien. Daraus resultiert eine Verteilung der Artenanzahl auf den Kanten eines Referenzbaums für eine gegebene Probe. Darüber hinaus beschreibe ich einen neuartigen Ansatz zum Clustering von Platzierungsergebnissen, anhand dessen der Benutzer den Rechenaufwand reduzieren kann.

Abstract

In recent years, advances in high-throughput genetic sequencing, coupled with the ongoing exponential growth and availability of computational resources, have enabled entirely new approaches in the biological sciences. It is now possible to perform broad sequencing of the genetic content of entire communities of organisms from individual environmental samples. Such methods are particularly relevant to microbiology. The field was previously largely constrained to the study of those microbes that could be cultured in the laboratory (i.e., *in vitro*), which represents a small fraction of the diversity observed in nature. In contrast to this, high-throughput sequencing now enables the collection of genetic sequences directly from a microbiome in its natural environment (i.e., *in situ*).

A typical goal of microbiome studies is the taxonomic classification of the sequences contained in a sample (the *queries*). Phylogenetic methods are commonly used to determine detailed taxonomic relationships between queries and well-trusted *reference* sequences from previously classified organisms. However, due to the high volume (10^6 to 10^9) of query sequences produced by high-throughput sequencing based microbiome sampling, accurate phylogenetic tree reconstruction is computationally infeasible. Moreover, currently used sequencing technologies typically produce short query sequences that have limited phylogenetic signal, causing instability in the inference of comprehensive phylogenies.

Another common goal of microbiome studies is to quantify the diversity within a sample, as well as between multiple samples. Phylogenetic methods are commonly used for this task as well, typically involving the inference of a phylogenetic tree comprising all query sequences, or a clustered subset thereof. Again, as with taxonomic identification, analyses based on this kind of tree inference may result in inaccurate results, and/or be computationally prohibitive.

In contrast to comprehensive phylogenetic inference, phylogenetic placement is a method that identifies the phylogenetic context of a query sequence within a trusted reference tree. Such methods typically regard the reference tree as immutable, that is the reference tree is not altered before, during, or after the placement of a query. This allows for the phylogenetic placement of a query sequence in linear time with respect to the size of the reference tree. When combined with taxonomic information for the reference sequences, phylogenetic placement therefore allows to identify a query. Further, phylogenetic placement enables a wealth of additional post-analysis procedures, allowing for example the association of microbiome characteristics with clinical diagnostic properties.

In this thesis I present my work on designing, implementing, and improving EPA-NG, a high-performance implementation of maximum likelihood phylogenetic placement. EPA-NG is designed to scale to billions of input query sequences and to parallelize across thousands of cores in both shared, and distributed memory environments. It also improves the single-core processing speed by up to 30 times compared to its closest direct competitors. Recently, we have introduced an optional feature to EPA-NG that allows for placement into substantially larger reference trees, using an active memory management approach that trades memory for execution time.

Additionally, I present a massively parallel approach to quantify the diversity of a sample, based on phylogenetic placement results. the resulting tool, called SCRAPP, combines state-of-the-art methods for maximum likelihood phylogenetic tree inference and molecular species delimitation to infer a species count distribution on a reference tree for a given sample. Furthermore, it employs a novel approach for clustering placement results, allowing the user to reduce the computational effort.

Acknowledgments

As I look back on these past years, I can't overstate how lucky and privileged I feel. Working at HITS often felt like some fantasy come true, where there are very few barriers between a person and the pursuit of their interests. On top of this, on some days Heidelberg can seem like it underscores the magical element of the fantasy beautifully. However, of course, all this would be only an empty stage devoid of essence without the people that I have met, worked with, and who have shared their life with me.

First, I want to thank Prof. Dr. Alexandros Stamatakis, my friend and mentor. I sincerely think that you make the world of academia a better place, and I wish more professors had your level of dedication to their students.

In this spirit I also want to thank Prof. Dr. Thomas Ludwig who was kind enough to co-supervise my thesis over the years.

While working in the Exelixis Lab, I had the honor of meeting so many incredible people, each wonderful in their own way: Paschalia Kapli, Tomáš Flouri, Alexey Kozlov, Sarah Lutteropp, Diego Darriba, Rebecca Harris, Dora Serdari, Rudolf Biczok, Johanna Wegmann, Benoit Morel, Ben Bettisworth, Lukas Hübner, Dimitri Höhler and Julia Schmid. Other than that there is no one else in the lab I can think of, except maybe Lucas Czech who has been one of my closest friend throughout these years, and easily my second mentor. I can't even imagine how my time at HITS would have been without you Lucas, and I don't want to imagine how the future would be like without you.

I probably didn't use the opportunity of being able to interact with all the other fantastic people at HITS nearly enough, but I want to mention in particular Kashif Sadiq, Nikos Gianniotis, Dandan Xu, Kira Feldmann, Phillip Bongartz and Thomas Rasem for the plentiful thought provoking discussions. I also want to thank the rest of the administrative team for making life easy, and I want to thank the team in the kitchen for always keeping my particular needs in mind. Another important mention of course is Klaus Tschira, whose dream, I hope, us that work and have worked at HITS have brought to life. Certainly the Klaus Tschira Foundation has funded my work on this thesis, and thus enabled some of my dreams to come true.

Beyond Heidelberg I was able to meet so many more brilliant people (some of which I collaborated with) including Frederick A. Matsen, Cédric Berney, Colomban de

Vargas, Ewen Corre, Micah Dunthorn, Frédéric Mahe, Mahwash Jamy, and the Serratus team including Artem Babaian, Rayan Chikhi and Robert C. Edgar.

I also want to mention my awesome originally shared flat turned semi-professional pub-quiz group *“Kommune 0”*, whose friendship has endured over a decade of living, studying, partying, and quizzing together. Similarly, my time in Heidelberg would not have been the same without my amazing pen-and-paper group *“Die Willibalds”* with whom I've shared many adventures and tons of laughter over the last four years. In Heidelberg I was fortunate to find two roommates, Marius Felix and Sandra Hadenfeldt, with whom I shared many great evenings, meals, and cocktails, giving me a home away from home.

Natürlich wäre nichts von alle dem möglich gewesen ohne meine fantastischen Eltern Brigitte und Guiseppe Barbera, sowie meiner großartigen Schwester Partizia, deren Unterstützung und Liebe keine Grenzen kennt. Kein Kapitel, keine Seite, kein Satz, keine Formel, keine Zeile Code, kein commit, kein Release meiner Software, kein Diagramm, keine Tabelle, kein Paper, kein Wort meiner Arbeit hat nicht mindestens auch teilweise eure Handschrift.

Finalment, Laura, conèixer-te ha canviat completament la meva vida. Veure el teu somriure és el màxim motivador del planeta. Cada dia amb tu és especial d'alguna manera, i no puc imaginar una parella millor per a les aventures que ens esperen.

Contents

List of Figures

List of Tables

List of Acronyms

AA	Amino Acid
AMC	Actively Managed CLVs
AVX	Advanced Vector Extensions
BLO	Branch Length Optimization
BQP	Branch Query Phylogeny
BWPD	Balance Weighted Phylogenetic Diversity
CLV	Conditional Likelihood Vector
DAG	Directed Acyclic Graph
DNA	Deoxyribonucleic Acid
DP	Dynamic Programming
FPA	Felsenstein Pruning Algorithm
GMYC	General Mixed Yule Coalescent
GPGPU	General-Purpose Graphics Processing Unit
GTR	Generalised Time Reversible
HMM	Hidden Markov Model
I/O	Input/Output
KRD	Phylogenetic Kantorovich-Rubinstein Distance
LCA	Lowest Common Ancestor
LWR	Likelihood Weight Ratio
MAG	Metagenome-Assembled Genome
MC	Markov Chain
ML	Maximum Likelihood
MP	Maximum Parsimony
MPI	Message Passing Interface
mPTP	Multi-Rate PTP
MSA	Multiple Sequence Alignment
NGS	Next Generation Sequencing
NR	Newton-Raphson
OTU	Operational Taxonomic Unit
PD	Phylogenetic Diversity
PE	Parallel Efficiency
PLF	Phylogenetic Likelihood Function
PQS	Placement Locations of a Query Sequence
PSC	Placement Space Clustering
PTP	Poisson Tree Processes

QS	Query Sequence
RAM	Random Access Memory
RNA	Ribonucleic Acid
rRNA	Ribosomal Ribonucleic Acid
RT	Reference Tree
SP	Sum-of-Pairs

1. Introduction

1.1 Motivation and Background

Perhaps the greatest discovery of the last century has been that life operates on information that is encoded in chains of just four individual molecules. These molecular chains collectively comprise the *genetic information* of every living cell, and serve as the blueprint for practically all forms of life we observe. From single celled microbes to complex multicellular organisms such as ourselves, we find correlative and causative relationships between genetics and phenomena as disparate as human psychology [35] and the resilience of plants to drought [33].

It is therefore hard to overstate the significance of the recent emergence of two key technologies: inexpensive high throughput genetic sequencing, and abundant high-performance computing. The first, high throughput sequencing, allows us to collect genetic information in remarkable volumes. For example, the cost of sequencing the total genetic information (called *genome*) of an individual human has decreased dramatically since it was first achieved in 2001 (from ~$95,000,000 to ~$700) [113]. Meanwhile, the available computing power, for example as measured by the peak floating point operations per second that can be performed by the world's fastest supercomputer, is still growing exponentially [80]. Taken together, these technologies promise to help us in answering numerous questions about our health, our environment, and the impact we have on it.

Of particular relevance for this thesis is the study of the genetic content of entire microbial communities, or *microbiomes*. Microbial communities are ubiquitous in the biosphere, and fulfill a wide variety of functions in their environments. The microbial community in the human gut, for example, has been shown to be incredibly diverse [90], and also to perform varied and important functions in the digestive process [31]. Perhaps unsurprisingly, there are strong associations between the composition of the human microbiome, and human diseases as well as disorders [13, 105, 119].

If we broaden our view, we observe a widespread repetition of this pattern of the presence of microbial communities and their connection to the larger environment. For instance, a diverse group of microscopic organisms called plankton (from the ancient Greek word πλαγκτός, meaning "wandering") form the basis of the marine food chain [62]. In addition, the planktonic ecosystem constitutes a major carbon sink [42], and thus monitoring its state is of high relevance to the ongoing efforts to reduce the atmospheric carbon concentration.

Finally, microbial communities have been found to be an integral part of soil, for instance in natural forests [70] and agricultural lands [6]. For example, some plants are able to form symbiotic relationships with nitrogen-fixing microbes, where the host plant creates an anaerobic environment for the microbes, while benefiting from the nitrogen the microbes produce [86]. If this symbiosis can be extended to other plants such as food crops, it may open up a novel path to agriculture that is less dependent on synthetic fertilizer [85].

Consequently, considerable research effort has been dedicated to catalogue such microbial communities. For the human body, the Human Microbiome Project comprises an extensive database of microbiome samples [47, 111]. For the oceans, the Tara Oceans expeditions have collected Terabases of genetic information from marine water samples [54, 107]. More recently, the Plankton Planet project has collected globally distributed marine microbiome samples using volunteer sailors (affectionately dubbed *planktonauts*) [24].

Common to microbiome studies is that their sample data is compositional, that is a sample represents a collection of organisms. Thus, a common goal is to determine the composition of a sample and in doing so, attempt to characterize the environment that was sampled. In practice, we do so by identifying the relation of each individual component of a sample (called a *query*) against a larger set of known organisms (referred to as the *references*) using computational methods.

One straight-forward way to assign an identity to a query is through similarity search against a database of organisms [1]. Such approaches typically do not require much preprocessing, and yield an answer that is simple to interpret: a percent similarity measure, and a measure of confidence in the result. A drawback of this methodology is that it only provides limited indication about the relation of the query to other organisms, especially if the reference data is incomplete/sparse [58].

In contrast to this, *phylogenetic* methods allow us to find the closest relatives of a query from an evolutionary perspective. Phylogenetic trees represent the evolutionary history of a group of organisms, based on their (genetic) data. When we add a query to an existing tree, we do not only gain information about the closest reference but we also determine its evolutionary relationship with respect to all other references. Through this, we are able to more confidently assess the identity of a query and its novelty in relation to the reference data.

One limiting factor of phylogenetic methods is the computational complexity of searching for the tree topology that best explains the data. In part, this is due to the exceedingly high number of possible ways to resolve the branching pattern: for a tree comprising n organisms there are $\mathcal{O}(n!)$ possible topologies [37].

For the specific problem of adding (or *placing*) a query onto a tree of references (or *reference tree*), we can circumvent the need to conduct a full tree search by keeping the original topology fixed. This allows us to evaluate the possible insertion locations of a query in $\mathcal{O}(n)$ operations, where n is the number of organisms in the tree. This makes the problem tractable for typical input sizes, which can comprise millions of individual queries [70]. This approach is called *Phylogenetic Placement*, and it is the main focus of the work presented in this thesis.

In addition to query identification, we can use phylogenetic methods to quantify the diversity of a sample [34]. In other words, the distance between two organisms in a phylogenetic tree implies a measure of relatedness. If we include multiple individuals from a single population in the tree, we can expect them to be grouped closely together, as it is unlikely that substantial genetic variation will be present among individuals. In contrast, if we include organisms that are highly dissimilar (such as avians and mammals) we generally expect the distances in the tree to be large in relation to within-population distances. This reasoning also allows us to use phylogenetics to determine a boundary between groups of organisms, or *species* [41, 118].

1.2 Scientific Contribution

The scientific contributions of this thesis are centered around the concept of Maximum Likelihood Phylogenetic Placement: its implementation, its enhancement, enabling its application to ever larger datasets, and broadening its use to new areas of study.

During my work I wrote and evaluated the EPA-NG (Evolutionary Placement Algorithm - Next Generation) software (covered in Chapter 3). It combines features from the previously existing phylogenetic placement softwares EPA [8] and PPLACER [75], while greatly improving the single-core speed (up to 30 times faster), and for the first time enabling phylogenetic placement to be natively parallelized on distributed memory systems, such as supercomputing systems. EPA-NG was written using the LIBPLL-2 library, which offers highly optimized and efficient core routines to calculate the phylogenetic likelihood function [40]. During my work I actively contributed to LIBPLL-2, and whenever possible I abstracted code into the library so as to increase the re-usability of my efforts.

This was the case with my most recent publication (covered in Chapter 4), which added an active memory management approach to EPA-NG. Here, I added the underlying functionality to the LIBPLL-2 library, enabling other users and future phylogenetic softwares to re-use it. This active memory management enables allows to compute the phylogenetic likelihood of a tree, while using only a fraction ($\sim log(n)$) of the typical memory footprint. For EPA-NG, enabling this functionality has implications on the performance of the program, as it creates a trade-off between memory savings and execution time. This new functionality will enable users to perform EPA-ng-based phylogenetic placement analyses in cases where it was previously not possible, such as when the main memory available to the user is limited, or when the reference tree is particularly large.

For my third first author publication I have implemented and evaluated SCRAPP (covered in Chapter 5). SCRAPP is a pipeline that takes results from EPA-ng and combines them with the phylogenetic tree inference tools ParGenes [82] and RAxML-ng [60], as well as the molecular species delimitation tool mPTP [53]. Similar to EPA-ng, it can also run in parallel on distributed memory systems. The output of the pipeline is a detailed statistic of the distribution of species counts on the reference tree, as well as a novel application of an existing phylogenetic diversity metric. I present a novel approach to reducing the dimensionality of placement results, called *Placement Space Clustering* (PSC), which allows the user of the pipeline to decrease the computational effort.

More recently, I was the co-first author in a publication that reviewed the difficulties of inferring reliable phylogenetic trees for the SARS-CoV-2 virus [83]. To this end, we constructed a number phylogenies using a range of methodological choices, and evaluated them using statistical methods. We concluded that the data is extremely challenging and that any phylogeny of this virus should be interpreted with caution. We also investigated the possibility of rooting the SARS-CoV-2 phylogenies we generated using Bat and Pangolin sequences via two different methods, one of which being phylogenetic placement. However we were unable to conclusively do so, as we did not observe a definitive signal when attempting to place these outgroup sequences onto our evaluated phylogenies. In addition, we attempted to apply the mPTP molecular species delimitation method to this phylogeny, again without success.

In addition to my first author work, I collaborated on several publications. In [16] we presented a method which enables the automatic, taxonomy driven construction of a reference tree. We achieve this by grouping the sequences of a large corpus of references together, each of which represents a possible subtree. The main reference tree is then this *backbone* tree, against which query sequences can be placed. To more accurately resolve the placement of a query, a *multi-level* placement can be performed by recursing into the subtree. The publication also introduces our method for taxonomic assignment of placement results, and its implementation in our gappa tool.

A subsequent publication covered the gappa tool, and its underlying library genesis, in greater detail [17]. The genesis C++ library covers functionality that is highly useful in handling a wide variety of phylogenetic data, with a particular focus on phylogenetic placement data. The gappa tool represents a command line interface to the most common functionality included in the library, as well as more advanced use-cases related to data analysis, visualization, data management, and data generation.

Another collaboration resulted in a publication that investigated the efficacy of using long-read metabarcoding sequences to study microbiomes [50]. In particular, this involved directly comparing phylogenetic placement with fully resolving a tree that includes both query and reference sequences. Additionally, this study included another application of the aforementioned taxonomic assignment approach, as well as a more bespoke version thereof that could be applied to the fully resolved tree that combined query and reference sequences.

A more recent collaboration is my involvement in the Serratus open science project [30]. Here the goal was to systematically explore one of the largest raw sequence databases using methods from sequence assembly and phylogenetic/taxonomic identification. This was enabled by the use of cloud computing infrastructure, which allowed us to process ~10 petabases of data and identify ~10^5 novel RNA viruses, which expands the total number of known viruses by an order of magnitude.

1.3 Structure of this Thesis

In Chapter 2 I will introduce the foundational concepts and methods that are necessary to understand the remainder of the thesis.

The three following chapters are largely based on the three main first-author publications that constitute the body of my thesis. The subject of Chapter 3 is the original publication that accompanied the release of EPA-ng and comprises a detailed description of the implementation of the algorithms, the parallelization schemes employed, as well as the efforts we undertook to evaluate the software. In Chapter 4, I present my most recent enhancements to the memory management of EPA-ng, which allows the user to set an upper limit to the memory consumption. I investigated the impact of this mode on the execution time for a number of datasets, and investigate its implications on the core parallelization employed in the software. Chapter 5 introduces SCRAPP, a pipeline that uses phylogenetic tree inference and phylogenetic species delimitation tools to compute species counts and a measure of diversity for a given phylogenetic placement result.

Finally, I conclude the thesis with Chapter 6 where I provide a brief summary of the thesis, and discuss possible future directions of research. In addition, I include some supporting information in Appendix A which is beyond the scope and format of the thesis, such as file formats.

2. Foundations

In this chapter I outline the fundamental concepts and approaches that are relevant to this thesis. These include background on the data and its initial handling, the concepts behind phylogenetics and how they relate to phylogenetic placement, and finally some background on related topics such as quantifying biological diversity and species delimitation.

2.1 Molecular Sequence Data

We obtain molecular data through the process of *sequencing*, which reads the genomic (or proteomic) information of a cell, tissue sample, organism, or microbial environment. We can then transform this information into a human and/or machine readable textual representation.

The predominant sequencing technology (dubbed *Next Generation Sequencing* (NGS)) produces an exceedingly high number (on the order of 10^6) of relatively short (50-400 characters) contiguous strings called *reads* [68, 74]. Currently, a new generation of *long-read* sequencing technologies are emerging that produce orders of magnitude longer read lengths [3, 49].

More generally, we represent molecular data as a string, or *sequence* of characters from a given alphabet. The most common alphabets are the *Deoxyribonucleic Acid* (DNA) alphabet (`A`, `C`, `G`, and `T`), the *Ribonucleic Acid* (RNA) alphabet (swap `T` for `U`), and the *Amino Acid* (AA) alphabet (20 characters coding for the standard amino acids). These alphabets can be extended to include ambiguous characters. For example, when it is unclear whether a character should be `A` *or* `G` we represent it as `R` (Adenine and Guanine are both Pu**r**ine bases). Thus, in practice we may have to represent DNA using 16 characters covering all possible ambiguous states.

Due to the large volume of sequences produced by NGS, a common first step is to perform a dimensionality reduction through sequence clustering. Typically, we then select one sequence of each cluster as the representative of the cluster. This

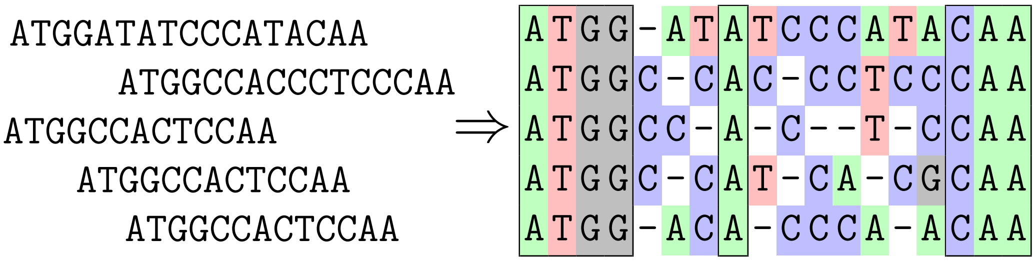


Figure 2.1: Illustration of Multiple Sequence Alignment process. On the left are five DNA sequences of similar length. The process of *Multiple Sequence Alignment* (MSA) groups similar characters together in *columns* (shown on the right), with the goal of a column containing characters that have descended from a common ancestor (i.e., establishing a relationship of *homology*). Doing so typically involves adding *gap characters* such as '-', which represent the insertion or deletion of a character in the sequence during its evolutionary history. In this example, the highlighted columns were especially stable, or *conserved*.

approach is also called *Operational Taxonomic Unit* (OTU) clustering, as each cluster is expected to represent sequences of either the same, or related organisms. We use the more abstract term of 'taxonomic unit', as the level of relatedness in terms of taxonomy can vary due to a number of factors, such as the clustering parameters, and the rate of evolution of the organisms. The most common approaches to OTU clustering use sequence similarity thresholds [29, 97, 103], although more sophisticated approaches do exist [71, 72].

2.1.1 Multiple Sequence Alignment

A first step in investigating the possible shared evolutionary history of a set of sequences is to perform *sequence alignment*. Specifically, we call the process of aligning more than two sequences against each other *Multiple Sequence Alignment* (MSA). The goal of alignment is to group the characters of the different sequences together into alignment *columns* in such a way that a column represents a shared evolutionary history, such as descent from a common ancestor. We call this similarity due to shared descent *homology* [23], and more specifically in this case *sequence homology* [57].

Over generations, characters of a sequence may be lost (called a *deletion*) or added (called an *insertion*). The alignment process can account for this by inserting *gap* characters (usually denoted by '-') into the sequences. Note that, when a column of an alignment consists entirely of gap characters, it does not convey any meaningful information and is thus typically ignored by downstream applications.

I illustrate the alignment process using a toy example in Figure 2.1. The resulting alignment columns can be seen as the components of a character matrix, with each row representing a sequence/organism.

There are two major approaches to align two sequences with each other (called *pairwise* alignment). *Global* alignment approaches attempt to align every character of both sequences, and are best suited for sequences of approximately equal length and with an expectation of a high number of homologous sites across the entire width of both sequences.

In contrast to global alignment, a *local* alignment approach attempts to align a sequence to a subregion of the other sequence, flanking the smaller alignment region with contiguous gap sections. Local alignment is best suited for aligning a substantially shorter sequence against some larger sequence, where the larger sequence often serves as a reference. One common use of this is sequence similarity search, such as implemented in BLAST [1], which allows finding similar regions in a substantially larger reference corpus/database.

We can evaluate a given MSA using the *Sum-of-Pairs* (SP) score [88]. For a given site, we first calculate the sum of all pairwise distances between the sequences in the MSA, using a distance function such as the edit distance [104]. We can then calculate the overall SP score as the sum of these per-site distances.

Pairwise alignment algorithms have polynomial-time complexity, however computing an SP-optimal MSA is NP-Complete [112]. Fundamentally, this is because alignment algorithms rely on *Dynamic Programming* (DP), where each sequence represents a dimension of the DP matrix. Extending this basic approach to n sequences would result in a DP matrix of n dimensions. In practice, greedy algorithms and heuristics are used to overcome this issue, as it would otherwise yield aligning thousands of sequences as computationally infeasible [109].

2.1.2 Alignment against a Reference

A more specialized use-case of MSA is aligning a number of sequences against an already established *reference* MSA. This scenario is particularly relevant for microbiome data, as sequences from such studies are typically short, numerous, and exhibit a relatively high sequencing error rate [102]. Here, we call such sequences *Query Sequence*s (QSs), as they are typically used to *query* a larger corpus of data to answer questions such as “Which known organism does the sequence belong to?”. In this example, the reference MSA would be a collection of curated, evolutionarily related sequences that cover the diversity of the microbial environment under study.

Alignment against a reference simplifies the problem of MSA, as the alignment of the reference sequences remains fixed (i.e., it is not changed by including the QS). For example, using HMMER [28] we can build a *Hidden Markov Model* (HMM) profile of a set of aligned reference sequences. The resulting profile can be understood as a single entity, or meta-sequence, against which we can align our QSs.

2.2 Phylogenetic Tree Terminology

Using graph theory, we represent a phylogenetic tree as a *Directed Acyclic Graph* (DAG), comprising a set of nodes which are connected via a set of edges. I illustrate the related terminology in Figure 2.2. We call the edges of this graph the *branches* of the tree. The *tip* or *leaf* nodes (i.e., nodes with a degree of one) of the graph

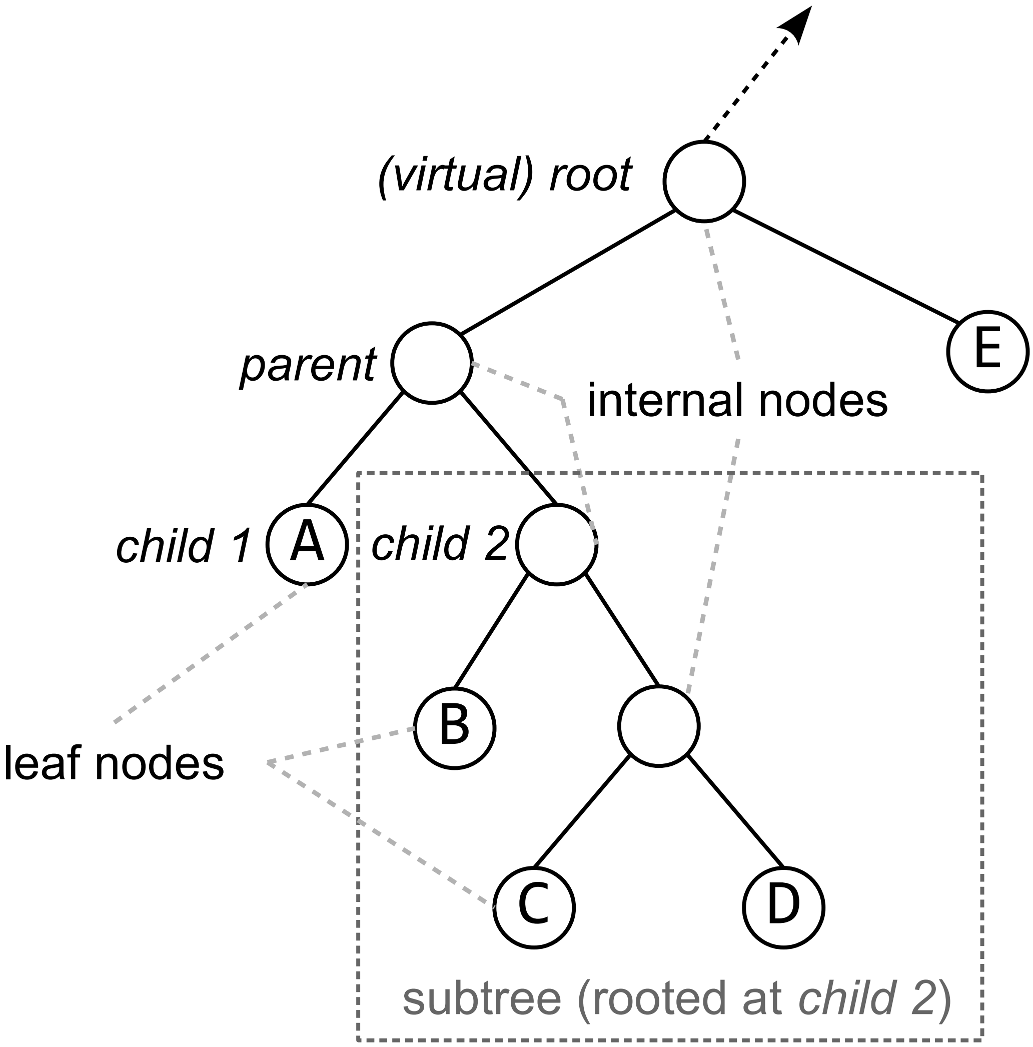


Figure 2.2: Basic Phylogenetic Tree terminology. Displayed is a tree with 5 leaves (A through E), connected via branches to 4 internal nodes. The dashed connection represents the possibility of further nodes in that direction and is purely for the purpose of demonstration: if we assume that the tree ends there, the figure serves as an example of a rooted tree (with the root being the node labeled as such). Otherwise, if the tree continues, the figure is an example of an unrooted tree with the root node being the arbitrarily chosen *virtual* root. Each internal node has 3 connections, one in the direction of the root, and two in in the opposite direction, also referred to as *below* the node. The nodes below an internal node represent its descendants. Thus in its immediate context we call the internal node the parent, and its two direct descendant nodes its *child* nodes. Each internal node induces a subtree, which includes the internal node and all its descendant nodes.

represent the sampled organisms of the tree. If the tree is rooted, the graph includes an explicit *root* node with a degree of two. All other nodes in the graph are called *internal* nodes and represent evolutionary events such as speciation or within-species diversification. If the tree is *unrooted*, meaning there is no explicitly designated root node, we may choose any internal node as the *virtual* root of the tree. If not otherwise specified, we will henceforth refer to the virtual root of an unrooted tree simply as its root.

For the purposes of this work we define a phylogenetic tree to be *strictly bifurcating*, meaning that each internal node is connected through branches to exactly two *descendant* nodes (called its *child* nodes), as well as being connected to exactly one *ancestor* node (called its *parent* node) in the direction of the root of the tree. This ancestor/descendant relationship extends from an internal node to all nodes *below* (i.e., in the direction away from the root) the internal node.

We call a subset of a tree including an internal node, all its descendant nodes, and the respective branches, a *subtree*. The internal node that defines a subtree is called the root of the subtree, and represents the *Lowest Common Ancestor* (LCA) of all nodes in the subtree. We can also interpret a subtree as a *lineage* of its constituent organisms.

2.2.1 Tree Traversal

It is often necessary to navigate through a tree via its nodes an edges, usually visiting them all, in a process called *tree traversal*. Typically starting from the root, a number of traversal orders are possible. Most commonly, we use depth-first-type traversals, as they follow the branching pattern of the tree and can be expressed in a recursive statement: the traversal of a parent node comprises the full traversal of the first child node, and subsequently the full traversal of the second child node. If the current node does not have children (i.e., it is a leaf), the recursion terminates. We further distinguish traversals by the point in time when a node is considered to have been visited: the first time the recursion passes the node (making it a *pre-order* traversal), or the last time the recursion passes the node (yielding a *post-order* traversal). Additionally, we define a traversal to be *in-order* if it performs a traversal of one child, then visits the parent node, and finally performs a traversal of the other child.

2.2.2 Branch Lengths and Ultrametric Trees

The branches of a tree may have a length associated with them. For the methods used in this work, branch lengths represent the mean expected number of character substitutions per site in the underlying MSA (also called *evolutionary time*).

Evolutionary time may differ from actual time as rates of nucleotide substitution can vary between lineages for a variety of factors, such as differing times between subsequent generations of the organisms, or increased environmental pressures causing higher diversification rates. For this reason the distance from the leaves of a tree to the root will generally not be equal for all leaves.

When the opposite is true, that is the path lengths from the root to each leaf are equal, the tree is *ultrametric*.

2.3 Phylogenetic Likelihood

A phylogenetic tree represents a hypothesis about the evolutionary relationships of the organisms it comprises, which are themselves represented by (in most cases molecular) data. The central goal of phylogenetics is to find the tree that best reflects reality. However, as we only have limited data (i.e., we are, with rare exceptions, unable to obtain data from extinct organisms and ancestral species), we hope to approximate reality by trying to find the tree that yields the best possible explanation, given the data. To do so, we require a function that allows us to calculate a score for a specified tree.

There are several established approaches that score a tree based on the data. For example, *Maximum Parsimony* (MP) methods sum the required character changes needed to explain the topology of the tree, given the data. The best possible tree under the MP criterion is the one that minimizes the score, yielding the *simplest* explanation (hence the name Maximum Parsimony).

For the purposes of this work, the relevant scoring method is the *Phylogenetic Likelihood Function* (PLF), which is the basis of *Maximum Likelihood* (ML) phylogenetic inference. It relies on statistical models of evolution, and operates on the tree using the *Felsenstein Pruning Algorithm* (FPA). In the following sections I elaborate on these concepts.

2.3.1 Statistical models of Nucleotide Evolution

Through inheritance, the genetic information of an organism is passed to subsequent generations. This process relies on the replication of the genome of the organism, which is an imperfect process that introduces errors. When such incorrectly copied sites persist, we call them *mutations*. Mutations can, in the best case, contribute to the ability of an organism or species to adapt to its environment, and thus selective environmental pressures will influence the DNA sequence of the organism. For example, when competition for food sources is high, an organism that had a mutation that allowed it to access a new food source will have an advantage. If such a new trait is passed down to its direct descendants, they too will have an advantage, and thereby adapt better to the environment. This was most famously observed in the differentiation of Finch beaks on the Galápagos Islands, allowing them to specialize to eat different nuts [23]. It is this type of change and diversification over time that a phylogenetic tree displays.

The rate at which a character in the sequence changes from some state x to some state y is expressed by the *transition rate* $\lambda_{x,y}$. The $\lambda_{x,y}$ rates express all possible transitions between the states of the alphabet, and thereby form a *Markov Chain* (MC). We illustrate this in Figure 2.3 for the DNA alphabet.

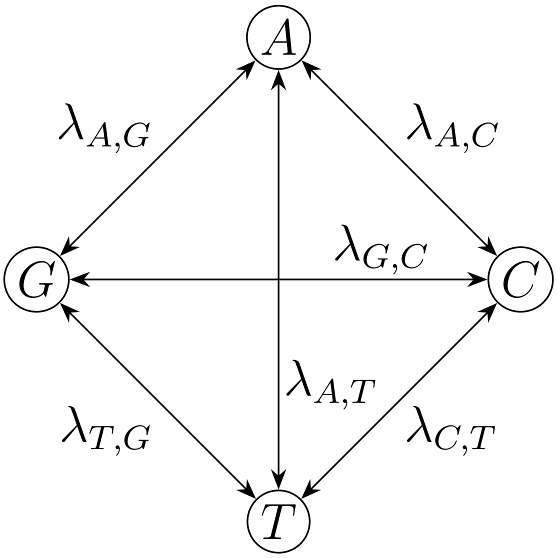


Figure 2.3: The Markov chain process of nucleotide state change. Displayed are the four states (`A`, `C`, `G`, `T`) of the DNA alphabet, and the transitions between them. Each transition between two states x and y has an associated rate $\lambda_{x,y}$. Here we show the transitions and their rates as being symmetric, that is the edges are bidirectional and $\lambda_{x,y} = \lambda_{y,x}$.

Additionally the Markov process requires a prior probability π_x of starting in state x, which we call the *stationary* or *base frequency*. The values of π must sum to one:

$$\sum_{x \in X} \pi_x = 1 \tag{2.1}$$

More commonly, we define the MC by its Q-matrix (in this case for the DNA alphabet)

$$Q = \begin{pmatrix} q_{\mathtt{A}} & \pi_{\mathtt{C}}\lambda_{\mathtt{C,A}} & \pi_{\mathtt{G}}\lambda_{\mathtt{G,A}} & \pi_{\mathtt{T}}\lambda_{\mathtt{T,A}} \\ \pi_{\mathtt{A}}\lambda_{\mathtt{A,C}} & q_{\mathtt{C}} & \pi_{\mathtt{G}}\lambda_{\mathtt{G,C}} & \pi_{\mathtt{T}}\lambda_{\mathtt{T,C}} \\ \pi_{\mathtt{A}}\lambda_{\mathtt{A,G}} & \pi_{\mathtt{C}}\lambda_{\mathtt{C,G}} & q_{\mathtt{G}} & \pi_{\mathtt{T}}\lambda_{\mathtt{T,G}} \\ \pi_{\mathtt{A}}\lambda_{\mathtt{A,T}} & \pi_{\mathtt{C}}\lambda_{\mathtt{C,T}} & \pi_{\mathtt{G}}\lambda_{\mathtt{G,T}} & q_{\mathtt{T}} \end{pmatrix} \tag{2.2}$$

where we define q_x such that the row of the matrix sums to 0.

We further assume that this Markov process is *time reversible*, meaning that $\pi_x\lambda_{x,y} = \pi_y\lambda_{y,x}$. Notably, using a time reversible model makes the choice of direction of the ancestral relationship arbitrary, as the transition rates between character states are symmetric. In other words, the choice of the root of a tree can be arbitrary, at least from the perspective of the evolutionary model.

Constraining the number of $\lambda_{x,y}$ parameters allows to define different models of character evolution (also called *substitution models*). For the DNA alphabet, common choices include setting all transition rates to the same value [38, 52], or differentiating between transitioning between or within nucleotide types (i.e., transitions and transversions) [43, 56]. Further, a commonly applied constraint is to set the base frequencies to be uniform (i.e., each to $\frac{1}{4}$) [52, 56]. When we constrain these choices, we limit the number of free parameters in the model. In contrast, the *Generalised Time Reversible* (GTR) model imposes no constraints on λ or π, and therefore allows for the maximum number of free parameters [108]. Choosing a model for a given dataset is often done via model selection algorithms [21, 81].

A priori, substitution models make the basic assumption that sites evolve at the same rate (i.e., they assume *rate homogeneity*). In reality, rates may vary across sites due to their relative importance in altering the function or structure of the resulting protein or RNA molecule [114]. For example, the different sites of a codon, the triplet of DNA characters that code for an amino acid, affect the resulting amino acid differently. Multiple nucleotides in the third position may code for the same amino acid, given that the remaining two positions are unchanged. Thus, a site in the alignment belonging to the third position of a codon can evolve more freely (i.e., at a higher rate).

We can account for *rate heterogeneity*, while limiting the additional number of free parameters, by modeling it using a fixed number of rates. Most commonly this is done using the Γ model of rate heterogeneity [114], in which rate variation is modeled using a Γ distribution, from which we obtain a set of j discrete rates $\vec{\gamma} = \gamma_1, ..., \gamma_j$. The Γ rates are not part of the substitution model, and instead are applied in the calculation of the *phylogenetic likelihood*, which is the subject of the following section.

2.3.2 Phylogenetic Likelihood Function

Recall that our goal is to use the just described models to score a tree, given some data. The nodes of the tree are connected by branches, which represent the transition between the possible states at the nodes. Thus, we have to take into account the branch lengths when calculating probabilities on the tree.

To do so, for a branch length b, we calculate the P-matrix

$$P(b) = e^{Qb} \tag{2.3}$$

giving us the transition probabilities $p_{x,y}(b)$ (i.e., the entries of the matrix).

We can now formulate the likelihood of a tree. Consider the example tree T displayed in Figure 2.4, with a parent node k and its two descendant internal nodes m and l. In turn, m and l have descendant leaf nodes, for which we only display the character states. In this example, we will only consider a single site s_i of the underlying alignment, displayed at each internal node with a possible state $x \in X$, X being the state alphabet, and at the leaf nodes by the actual state characters. For the sake of simplicity assume that k is the root of the tree. We can calculate the likelihood as the sum over all possible values for the different x, and their associated transition probabilities along the tree:

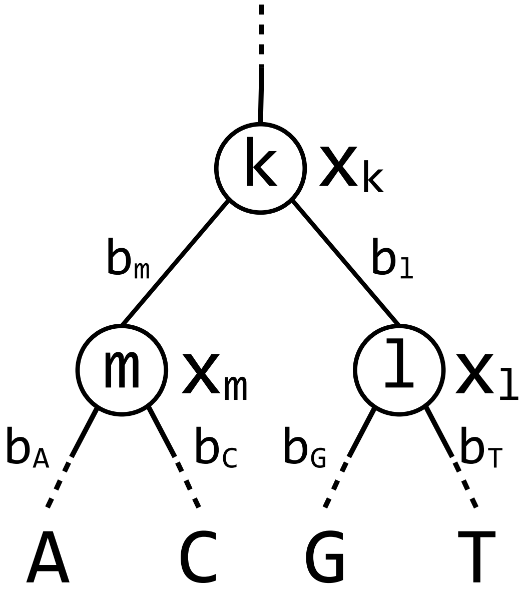


Figure 2.4: Subtree illustrating tip and inner node states. Tip nodes ($N_t = \{\mathtt{A,\ C,\ G,\ T}\}$) are represented by their nucleotide states, inner nodes are labeled arbitrarily ($N_i = \{k, l, m\}$). Nodes are connected via edges with some given branch lengths $b_i, i \in N_t \cup N_i$. Possible inner node character states are labeled $x_i, i \in N_i$.

$$P(s_i|T) = \sum_{x_m \in X} \sum_{x_l \in X} \sum_{x_k \in X} \pi_{x_k} p_{x_m,x_k}(b_m) p_{x_l,x_k}(b_l) p_{\mathtt{A},x_m}(b_{\mathtt{A}}) p_{\mathtt{C},x_m}(b_{\mathtt{C}}) p_{\mathtt{G},x_l}(b_{\mathtt{G}}) p_{\mathtt{T},x_l}(b_{\mathtt{T}}) \tag{2.4}$$

To obtain the likelihood of the tree for the entire alignment, we calculate the product over the per-site probabilities:

$$\mathcal{L}(T|D) = \prod_i P(s_i|T) \tag{2.5}$$

In practice, $P(s_i|T)$ quickly becomes very small. To avoid numerical issues, we calculate its logarithm $log(P(s_i|T))$, and thus report the *log-Likelihood* instead

$$log\mathcal{L}(T|D) = \sum_i log(P(s_i|T)) \tag{2.6}$$

2.3.3 Felsenstein Pruning Algorithm

We can rearrange Equation 2.4 to group together computations belonging to an inner node and its descendants:

$$P(s_i|T) = \sum_{x_k \in X} (\pi_{x_k} \sum_{x_m \in X} (p_{x_m,x_k}(b_m) p_{\mathtt{A},x_m}(b_{\mathtt{A}}) p_{\mathtt{C},x_m}(b_{\mathtt{C}})) \sum_{x_l \in X} (p_{x_l,x_k}(b_l) p_{\mathtt{G},x_l}(b_{\mathtt{G}}) p_{\mathtt{T},x_l}(b_{\mathtt{T}}))) \tag{2.7}$$

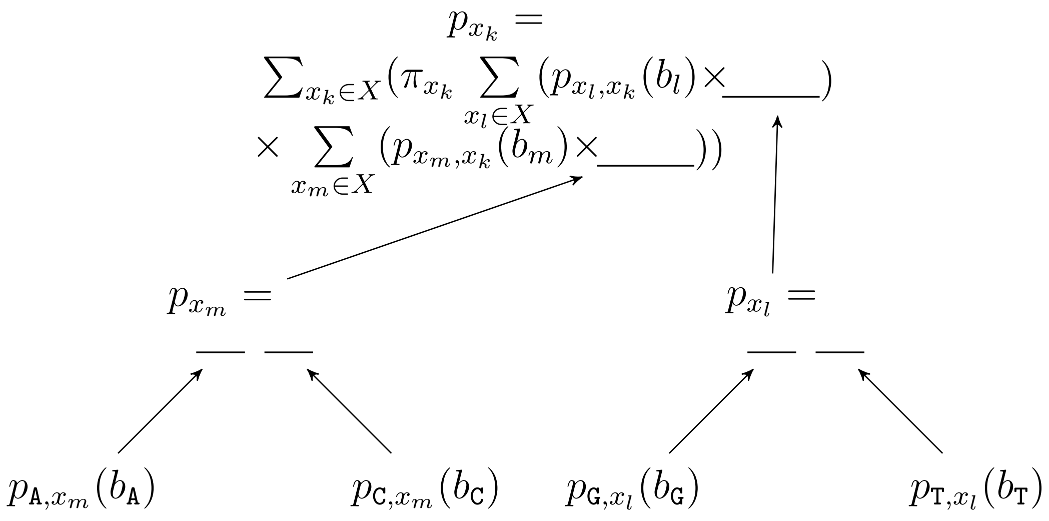

$p_{x_k} =$
$\sum_{x_k \in X}(\pi_{x_k} \sum_{x_l \in X} (p_{x_l,x_k}(b_l) \times \_\_\_\_\_)$
$\times \sum_{x_m \in X} (p_{x_m,x_k}(b_m) \times \_\_\_\_\_))$
$p_{x_m} =$
$p_{x_l} =$
$p_{\mathtt{A},x_m}(b_{\mathtt{A}})$
$p_{\mathtt{C},x_m}(b_{\mathtt{C}})$
$p_{\mathtt{G},x_l}(b_{\mathtt{G}})$
$p_{\mathtt{T},x_l}(b_{\mathtt{T}})$

Figure 2.5: The phylogenetic likelihood, as calculated in Equation 2.7, factored to show its relation to the tree structure. This figure reflects the tree displayed in Figure 2.4, with inner nodes $N_i = \{k, l, m\}$. The tree has four unnamed leaf nodes, each of which observes one of the character states of the state alphabet $X = \{\mathtt{A,\ C,\ G,\ T}\}$.

Doing so allows us to divide up the necessary computations *exactly* along the branching pattern of the tree. I illustrate this in Figure 2.5. Here again I show the situation for a single site in the alignment. We denote the per-state probabilities for node k as p_{x_k}, which we calculate based on the per-state probabilities p_{x_m} and p_{x_l} of its two direct descendants m and l. In turn, we calculate p_{x_m} and p_{x_l} from their respective direct descendants, which in this case are leaf nodes with distinct state characters.

This approach allows us to compute partial results of the overall tree likelihood for each subtree, independently. In practice, we store such partial, or *conditional*, likelihood results in a *Conditional Likelihood Vector* (CLV). A CLV at node k stores p_{x_k} for each state $x_k \in X$, for each site in the alignment. This is compounded when we use the Γ model of rate heterogeneity (Section 2.3.1), as such a model requires us to calculate p_{x_k} for each *combination* of discrete Γ rate and possible character state.

Consequently, the size of a CLV is the number of sites, times the number of states in the alphabet, times the number of discrete Γ rates. In software, we store each such value using a double precision floating point number (i.e., requiring 8 bytes each). Consider that, for phylogenetic inference software such as RAxML-NG [60], we store one CLV at each internal node. Thus, for either large trees (e.g., $> 10,000$ leaves) and/or wide alignments (e.g., $> 10,000$ sites), the memory requirements for storing CLVs can become a limiting factor.

2.3.4 Branch Length Optimization

The PLF allows us to score a given tree with a given set of branch lengths. In ML phylogenetic methods, we obtain the values for the branch lengths through the process of *Branch Length Optimization* (BLO), which attempts to find values for the set of branch lengths that maximize the overall tree likelihood, while keeping the tree topology fixed. In practice, we perform BLO using the *Newton-Raphson* (NR) method [96, 116], which iteratively finds values x for which a function $f(x)$ becomes successively closer to 0. The iteration is defined as

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{2.8}$$

In our case, as we want to maximize the likelihood $\mathcal{L}$ of the tree, we are interested in finding the zero-values of its first derivative $\mathcal{L}'$. Thus, for a branch length b the NR formulation becomes

$$b_{n+1} = b_n - \frac{\mathcal{L}'}{\mathcal{L}''} \tag{2.9}$$

which we perform until convergence (i.e., the difference between b_n and b_{n+1} falls below some threshold). We repeat the procedure for every branch in the tree. In ML phylogenetic inference, we optimize each branch by visiting it via a tree traversal (typically a post-order traversal), and perform the overall tree traversal multiple times, as the optimal length of one branch may change depending on the values of the other branch lengths in the tree. In turn, this tree-wide branch length optimization converges when the change overall tree likelihood between subsequent tree traversals falls below some given threshold.

2.4 Phylogenetic Placement

Phylogenetic placement is the main focus of this work. I include a short introduction of its main concepts and terminology in the following section, and elaborate on details in later chapters. This section attempts to be sufficiently general such as to cover the underlying ideas of most phylogenetic placement approaches.

2.4.1 Terminology

In phylogenetic placement, our goal is to determine the branches of a given *Reference Tree* (RT) to which a *Query Sequence* (QS) belongs. Doing so typically involves extending each branch of the RT by a novel branch leading to the QS. I illustrate this basic scenario in Figure 2.6. The branch on which we carry out the extension, called *insertion branch*, initially comprises two nodes which belong to the reference tree. We identify these nodes by their relation to the root of the RT, calling the node that is closer to the root the *proximal* node and its opposite counterpart the *distal* node. To attach the new branch we insert a new node, called the *basal* node, into the branch. This segments the branch into two parts, which we call the *proximal* and *distal* branches, again reflecting their relation to the root of the tree. Finally, we attach a node representing the QS via a branch, called the *pendant* branch, to the

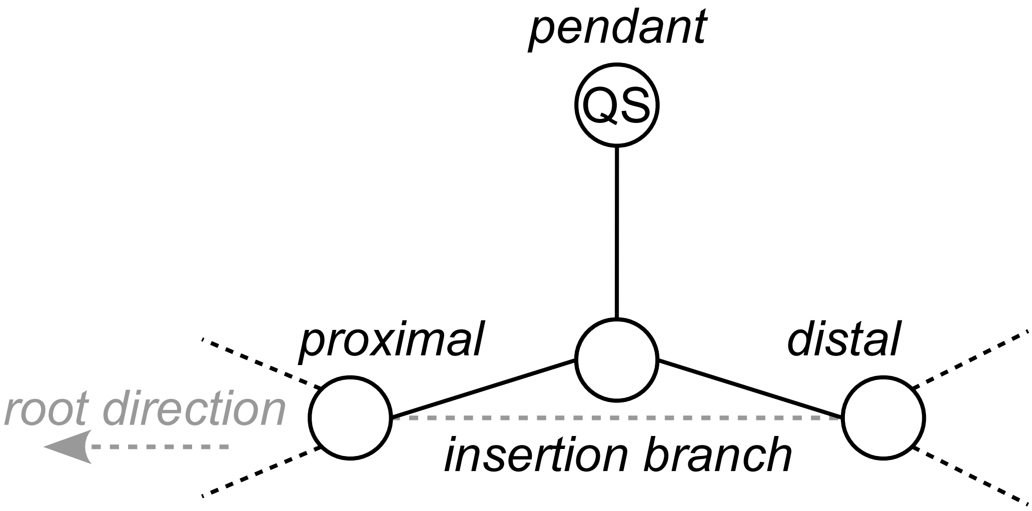


Figure 2.6: Illustration of placement terminology. The situation we display is the placement of a QS onto an existing branch of the *Reference Tree* (RT), called the *insertion branch*. We represent the QS by a new leaf node called the *pendant* node, which attaches to the original tree via a new internal node (here shown in the middle) called the *basal* node. The basal node connects to two nodes of the underlying RT, called the *proximal* and *distal* nodes. The naming reflects their relation to the root of the tree, with the proximal node being the closer one to the root. Dashed lines show the original branches of the RT.

basal node. For brevity, we will refer to the length of the proximal/distal/pendant branch as the proximal/distal/pendant length.

In the case of ML phylogenetic placement, we score each branch/QS combination by calculating the likelihood of the RT extended by the QS at that branch. To simplify this operation, we assume that the CLVs for the distal and proximal nodes of the insertion branch have already been calculated. As a CLV summarizes the rest of the tree *under* it, it implies that the root is located in the opposite, *above* direction. Without loss of generality we can assume that the root is located on the pendant branch, which means that each placement comprises 1) the calculation of the CLV at the basal node (i.e., summarizing the pendant and distal CLVs, given the pendant/distal lengths), 2) setting the tip states according to the QS at the pendant node, and 3) calculating the overall likelihood of the extended tree at the pendant branch. We denote the placement likelihood of a QS on a reference tree T at a insertion branch b, given the reference MSA D, as $\mathcal{L}(T, d|D, QS)$.

2.4.2 Result Organization

A common misconception is that placement is an iterative process that extends the RT, QS by QS. Instead, the RT remains fixed, and we evaluate each insertion of a QS into each RT branch independently. Consequently, placement generates a large number of intermediate results. Here we call a given set (which may not be exhaustive) of scored insertion branches of a QS the *Placement Locations of a Query Sequence* (PQS).

The minority of these results will be "interesting", as a QS will typically tend to fit well onto a small number of branches. Alternatively the QS might not fit the RT well enough for any branch to have a high score relative to the others. Thus, filtering these results constitutes a main component of software implementing phylogenetic placement.

For ML placement, we filter by normalizing the likelihoods of all $x \in PQS$ via their *Likelihood Weight Ratio*s (LWRs):

$$LWR(x) = \frac{\mathcal{L}(T, x|D, QS)}{\sum_{p \in PQS} \mathcal{L}(T, p|D, QS)} \tag{2.10}$$

Based on the LWR, we can identify placement locations of a given QS that are worth reporting. The simplest selection is to report the highest LWR, sometimes called the *best hit*. As this would represent overly aggressive filtering, other approaches such as reporting the top placement locations whose collective LWR sum to some threshold are used [5].

We can also interpret the LWRs of a set of PQS as a distribution on the RT. Doing so can help us to distinguish between some common placement scenarios. For example, a QS may appear to not fit the RT well due to a low LWR value for the best hit. Examining the complete distribution may indicate that the LWR is spread across a subtree, which typically indicates that the RT may contain too many closely related reference sequences [105].

Across most tools that implement phylogenetic placement, the common output format is the JPLACE file format [76]. A JPLACE file contains a specification of the reference tree (using the NEWICK format [39]), as well as a hierarchical list of all reported placements. For each placed QS, this list contains its corresponding PQS. Each PQS entry represents the data for an individual placement, containing the branch ID, the likelihood score, the LWR, the distal length, and the pendant length. Additionally, the file contains fields to specify metadata, such as the command line call used to generate the file.

2.4.3 Intersample Distance

We can combine the PQS of multiple QS into a larger result which we call a *sample*. If we normalize all LWRs of a sample, we obtain a multimodal distribution on the tree, called the *placement distribution*, which characterizes the sample. This approach is of particular use for analyzing and visualizing environmental microbial data [17, 70, 105].

In particular, we can compare samples by calculating a distance between their respective distributions on the same underlying RT. A simple approach to compare two distributions is the Earth-Movers distance, so called due to its guiding analogy of representing positive values of the distribution as piles of earth, or *mass*. Transforming a distribution in this analogy requires moving the mass along some distance, thus incurring *work*. The Earth-Movers distance between two distributions is defined as the minimum amount of work required to transform one distribution into the other.

The *Phylogenetic Kantorovich-Rubinstein Distance* (KRD) [32] extends this approach to distributions on a tree. When applied to placement distributions, we view the LWR as the mass, and the RT as the space through which we can move the mass. Thus, using the KRD, we calculate the distance between two samples as the minimum work required to equalize their distributions.

Note that when we desire to compare two placement distributions using the KRD, their underlying trees have to be identical.

2.5 Measuring diversity

One way of using phylogenies is for quantifying the diversity of a set of organisms (or for a set of tips/leaves in the tree). We distinguish between diversity *within* a set of leaves, called the α-diversity, and diversity *between* sets of leaves, called the β-diversity.

One example for an α-diversity metric is the *Phylogenetic Diversity* (PD) [34] (also called *Faith's* PD). For a subset of leaves of a tree, we calculate the PD by first determining the minimum set of branches that connect all leaves of the subset (called the minimum spanning tree). We can then calculate the PD as the sum over the branch lengths of that set.

The PD can be extended to incorporate weights, such as abundance information [7]. McCoy and Matsen [78] build on this and define a one parameter function, called the *Balance Weighted Phylogenetic Diversity* (BWPD), which is able to interpolate between the classical PD and its weighted version. The BWPD is especially relevant to this thesis as we can directly apply it to phylogenetic placement results.

2.6 Species Delimitation

A related problem to diversity quantification for a set of organisms is their delimitation into individual species. Defining what constitutes a species, and whether two organisms belong to the same or different species, is the source of some contention in the literature [22, 77, 92].

Here, we will focus on the *phylogenetic species concept*, meaning that we broadly view a species as a group of organisms that share an evolutionary history and a common ancestor. The phylogenetic species concept allows us to delimit species by only using a phylogenetic tree. I illustrate the phylogenetic species concept in Figure 2.7.

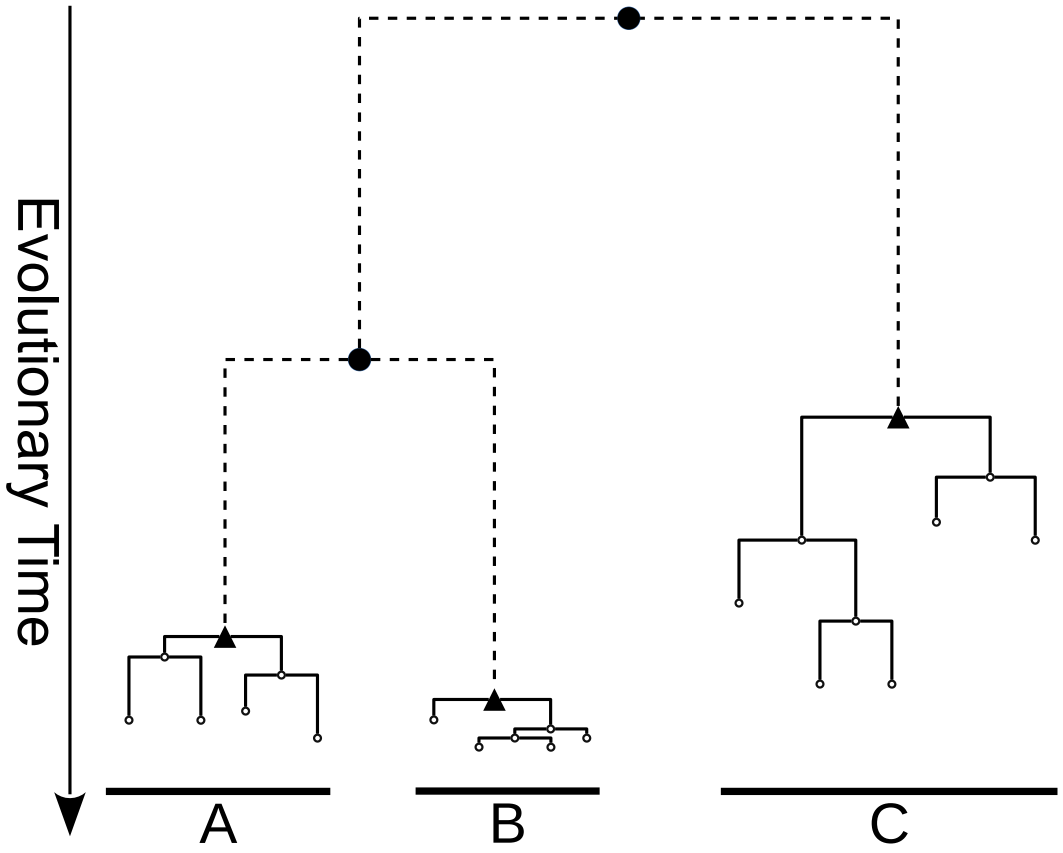


Figure 2.7: Illustration of the phylogenetic species concept. Dashed lines and filled circles indicate the branching pattern due to *inter*-specific (or *among* species) diversification. Subtrees drawn using solid lines indicate the branching pattern due to *intra*-specific (or *within* species) diversification, and represent cohesive species clusters. Nodes drawn as triangles represent the LCA (Section 2.2) of the respective species clusters. In this example, three distinct species were delimited (A through C). We observe that species, or lineages, may vary in their rate of diversification (here most prominently shown between B and C).

Multiple methodologies to produce phylogenetic species delimitations have been proposed. The *General Mixed Yule Coalescent* (GMYC) [41], for example, explores the space of possible delimitations of a tree by segmenting it into a between-species part (typically the deep interior branches) and m subtrees, each of which represents a species population (i.e., the individuals of a species). To achieve this, the GMYC-model calculates the likelihood that a given time-point (i.e., an internal node) represents a transition from the process driving speciation (*inter*-specific branching), to the process driving within-species variation (*intra*-specific branching). Note that the GMYC-model requires the underlying tree to be time-calibrated (i.e., to be ultrametric).

Similarly to the GMYC-model, the *Poisson Tree Processes* (PTP) [118] models both inter-, and intra-specific branching processes and evaluates their fit to a given tree using a likelihood model. In contrast to the GMYC-model however, the two processes are modeled as two exponential distributions based on the number of substitutions, and can thus be calculated directly using ML branch length estimates without requiring the tree to be ultrametric.

Kapli *et al.* [53] further improve on the PTP by extending it to accommodate multiple distinct intra-specific branching processes, such that every cluster of species is modeled at its own rate of diversification. Consequently, their model is called the *Multi-Rate PTP* (mPTP).

3. EPA-ng: Massively Parallel Evolutionary Placement of Genetic Sequences

This chapter is based on the peer-reviewed open-access publication:

P. Barbera, A. Kozlov, L. Czech, B. Morel, D. Darriba, T. Flouri, and A. Stamatakis. “EPA-ng: Massively Parallel Evolutionary Placement of Genetic Sequences.” *Systematic Biology*, 2019, Volume 68, Issue 2, Pages 365–369.

accessible online at: `https://doi.org/10.1093/sysbio/syy054`

3.1 Introduction

As stated in Chapter 1, phylogenetic placement is a method for identifying the phylogenetic context of genetic sequences. The currently most common application of placement is the characterization of microbial samples, which comprise thousands to millions of sequences each.

The sheer volume of such metagenomic sequences/samples presented the primary scalability challenge in phylogenetic placement when I first approached the subject. The two tools that had previously implemented ML phylogenetic placement, PPLACER and RAxML-EPA, had substantial throughput and scalability limitations that hindered their applicability to increasingly larger datasets.

To alleviate these scalability challenges, we designed EPA-NG, a re-implementation of ML phylogenetic placement which combines features from RAxML-EPA and

PPLACER. EPA-NG builds upon LIBPLL-2 [40], a state-of-the-art library for phylogenetic likelihood computations, and achieves a high degree of parallelism by combining OpenMP and *Message Passing Interface* (MPI) to efficiently run on distributed memory systems.

3.2 Methods

As briefly described in Section 2.4, the ML phylogenetic placement procedure determines a set of branches on a RT to which a given QS belongs with the highest relative likelihoods. We formalize the basic placement algorithm Algorithm 3.1.

Let SQS be the set of QSs, and let RT be the reference tree. The LWR is calculated as defined in Equation 2.10.

Algorithm 3.1 Placement Procedure

```
for each sequence $q \in SQS$ do
    for each branch $b \in RT$ do
        insert the sequence $q$ into branch $b$
        compute the likelihood $\mathcal{L}$ of the resulting tree
        record $\mathcal{L}$ and the location of $q$ in its PQS
        remove sequence $q$ from branch $b$
    end for
    calculate the LWR across all PQS entries
end for
```

Note that, each query sequence is placed into the reference tree independently of all other query sequences. In part, this is possible due to the way we store partial likelihood results of the RT. As discussed in Section 2.3.3, partial likelihood results are stored in so-called CLVs at the internal nodes of the tree, summarizing the results of the respective subtree rooted by (or *below*) an internal node. Thus, a CLV implies a direction toward a root, which must be located somewhere *above* the given internal node. However, as described in Section 2.4.1, the ML placement procedure assumes that the root is located somewhere on the newly inserted pendant branch. Consequently, we calculate the CLVs for each possible rooting of the RT and store them in memory, such that for each placement task the relevant CLV buffers can be accessed independently and concurrently. For a RT with n leaves this results in $3 \times (n - 2)$ CLVs that we store in memory.

3.2.1 Heuristics

By default, EPA-NG executes two successive variants of the placement procedure to save runtime via appropriate heuristics. The first variant, called *preplacement*, simply inserts the query sequence into the midpoint of the insertion branch, with a default length for the branch leading to the new tip. The likelihood of the resulting tree is then calculated without re-optimizing any branch lengths.

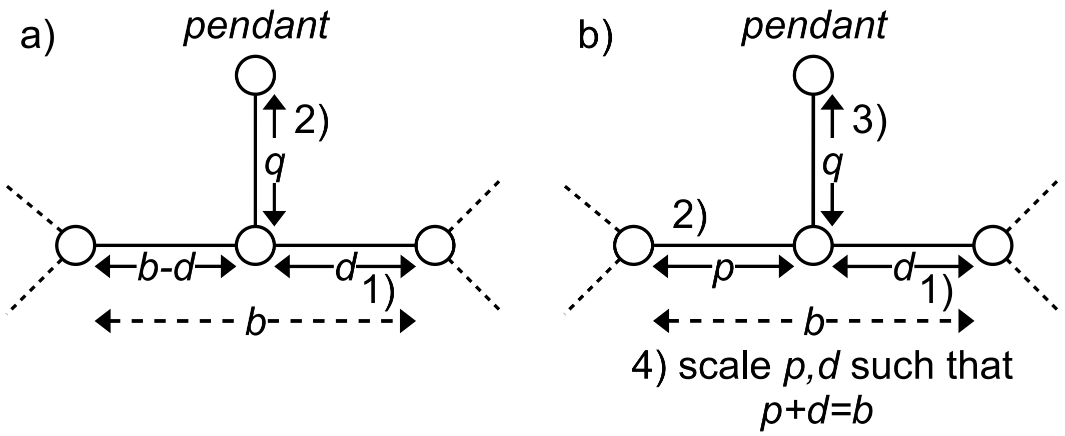


Figure 3.1: Illustration of the two BLO modes implemented in EPA-NG. The default BLO mode (subfigure **a)**), which optimizes the distal branch length d such that it cannot exceed the original insertion branch length b. The proximal branch length p directly results from this step and is calculated as $p = b - d$. Finally, the pendant branch length q is optimized independently from the other branches.
The second BLO mode (subfigure **b)**) independently optimizes the branch lengths d, p, and q, without constraints. In a final step, p and d are rescaled such that $p + d = b$. This enables the distal length d to be interpreted as a position along the original insertion branch.

The second variant, called *thorough placement*, calculates a more precise maximum likelihood score. It does so by optimizing the two branch lengths, obtained by splitting the insertion branch b in two, as well as the branch length of the newly inserted sequence. Further, EPA-NG offers two variations of branch length optimization, which are illustrated in Figure 3.1. By default, the total length of the insertion branch is considered to be fixed and only the position of the QS branch along this insertion branch is being optimized. This is analogous to the BLO as implemented in PPLACER. Alternatively, the user can choose to not constrain the insertion BLO in this way (using the `--raxml-blo` option), which will, in most cases, result in a new total length of the insertion branch that differs from the original branch length in the RT. In such a case, the distal length (see Section 2.4.1) as reported in the placement result is nonetheless calculated to be proportional to the insertion location of the query along the branch.

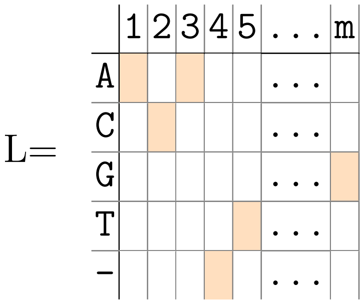
For obtaining maximum accuracy, one can perform such a thorough placement for every branch of the RT and every QS. However, as such optimizations, and thus thorough placements, are very compute-intensive, the two variants are combined into a heuristic algorithm. Preplacement selects a set of promising candidate branches for each QS, for which thorough placements are subsequently computed. In the original implementation [8], this set was determined by simply using the top $x\%$ of branches (for some user-defined value of x), sorted by their LWRs. By default, EPA-NG uses a more adaptive heuristic, that adds branches to the set of promising branches until a user-specified accumulated LWR threshold (0.99999 by default) is reached. This heuristic has greater flexibility to accommodate variation in the placement signal of an individual QS, as placements may be spread across a large portion of the RT, or be highly concentrated in a specific part of the RT.

Like other phylogenetic placement software, EPA-NG operates in two phases: it first quickly determines a set of promising candidate branches for each QS (*preplacement*), and subsequently evaluates the maximum placement likelihood of the QS into this set of candidate branches more thoroughly using numerical optimization routines for the branch lengths (*thorough placement*). The user can choose to treat every branch of the tree as a candidate branch, however this induces a significantly higher computational cost. Consequently, by default, EPA-NG dynamically selects a small subset of the available branches via preplacement. Using preplacement heuristics typically reduces the number of thoroughly evaluated branches from thousands (depending on the RT size) to often less than ten (depending on the query and reference data).

To further accelerate the preplacement phase, we make use of the fact that every QS preplacement is static with regards to the tree, meaning that the distal/proximal and pendant lengths are the same across all QSs for a given branch. For a given state alphabet X, we construct $|X| + 1$ artificial sequences, with each sequence consisting of m sites of the same state $x \in X$, m being the width of the reference alignment. Additionally, we construct a sequence which consists entirely of gap characters. This set of sequences is then placed onto a given branch using the preplacement approach, and we store the per-site likelihoods in a combined matrix with dimensions $(|X| + 1) \times m$ which we call the *preplacement lookup-table*. We illustrate this lookup-table in Figure 3.2.

Subsequently, when we perform preplacement of a QS, we sum over the per-site likelihoods in the lookup-table that correspond to each site, and character at that site, of the QS. Note that these preplacement lookup-tables can incur a significant memory overhead ($sites \times (states+1) \times branches \times 8$ bytes). However, the substantial execution time savings justify this design choice (this will be further explored and discussed in Chapter 4).

EPA-NG also offers a second heuristic called *masking* that is similar to the premasking feature in PPLACER. It effectively strips the input MSAs of all sites that are unlikely to contribute substantially to the placement likelihood score. Such sites consist entirely of gaps either in the reference or in the query alignment. Additionally, for each individual QS, only the *core* part of the alignment is used to compute the likelihood of a placement. The core of an aligned QS is the sequence with all leading, and trailing gaps discarded. Note that PPLACER also discards all gap sites within an individual sequence, including gaps in the core. We opted not to implement this, as our experiments showed that computing these per-site likelihoods, instead of omitting these computations, was more efficient in our implementation.



$$\text{QS} = \texttt{A C A - T ... G}$$

$$\downarrow$$

$$\mathcal{L} = L_{1,\texttt{A}} + L_{2,\texttt{C}} + L_{3,\texttt{A}} + L_{4,\texttt{-}} + L_{5,\texttt{T}} \cdots + L_{m,\texttt{G}}$$

Figure 3.2: Illustration of a preplacement lookup-table. For a given RT branch, a preplacement lookup-table L can be constructed by performing preplacement for a set of sequences that each consist entirely of one of the characters in the given state alphabet, plus the gap character '-'. We construct each sequence to span the entirety of the reference alignment with m sites. Each combination of sequence and branch results in a vector of per-site likelihoods, which comprise the rows of L. Consequently we can calculate the preplacement likelihood $\mathcal{L}$ of a given QS by looking up the per-site log-likelihoods for each site in the QS (here illustrated by the coloring of the cells of L) and calculating the sum over the individual log-likelihoods. Note that this is a simplified illustration, as we have to account for ambiguous states in the actual implementation, resulting in 16 rows in the DNA case.

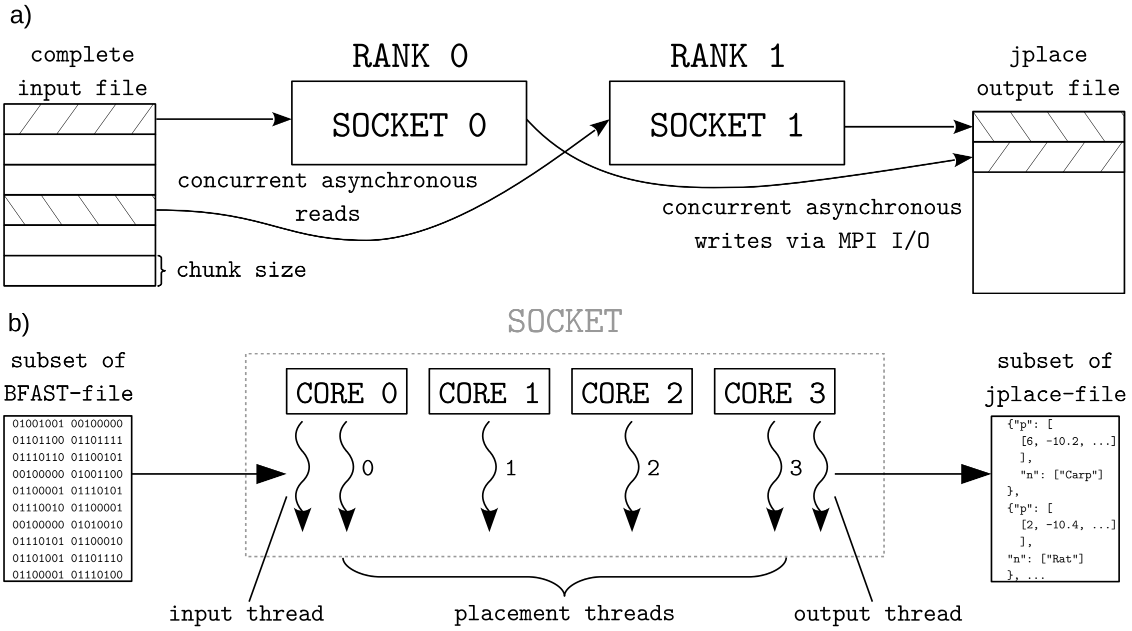


Figure 3.3: Illustration of the hybrid parallelization scheme implemented in EPA-NG. **a)** shows the parallelization strategy at the level of multiple MPI-ranks. In this case each rank is assigned to a socket of a node. Each MPI-rank processes a distinct subset of QS from the input file, and does so in chunks of a given size. When a chunk of QS has been successfully placed the result is written to a global JPLACE [76] output file, using collective MPI File *Input/Output* (I/O) write operations. **b)** shows the parallelization strategy within each MPI-rank (in this case: one complete CPU socket). The given subset of the binary input file is read asynchronously by a dedicated input thread, which allows prefetching of one chunk during computation of another. All actual placement work is then split across as many OpenMP worker threads as the user specified (in this case as many as there are physical cores on the socket). Finally, a dedicated output thread writes the per-chunk results to a file, which again allows overlapping computation with *Input/Output* (I/O).

3.2.2 Parallelization

EPA-NG offers two levels of parallelism: MPI to split the overall work between the available compute nodes, and OpenMP to parallelize computations within the compute nodes. Such *hybrid* parallelization approaches typically reduce MPI related overheads and yield improved data locality [94].

Figure 3.3 illustrates how EPA-NG utilizes hybrid parallelism. In hybrid mode, EPA-NG splits the input set of QSs into parts of equal size, such that each MPI-rank has an equal number of QSs to place on the tree. No synchronization is required to achieve this, as each rank computes which part of the data it should process from its rank number and the overall input size.

For within-node parallelization, we use OpenMP. Here, each thread works on a subset of QS and branches.

3.2.2.1 File Preparation

To reduce overhead and accelerate parallel *Input/Output* (I/O), we developed an internal binary *bfast* file format that replaces the FASTA format for storing sequences. Its main feature is its header, which contains information about the total number of sequences and their individual offsets in the file. Optionally, the header may include a binary mask, specifying which sites of a sequences should be read. The sequences are stored in a 4-bit encoded format for each nucleotide, covering all four DNA bases as well as their ambiguities. Consequently, as of now, this bfast format is only applicable to DNA data. These features yield a randomly accessible, binary encoded sequence file that is well suited for use on compute clusters. Additionally, bfast files only have about half the size of the corresponding FASTA-files.

3.2.2.2 Parallel I/O

To allow EPA-NG to scale to thousands of cores and compute nodes, efficient parallel I/O was of central importance. To this end, we employ two major concepts: overlapping I/O with computation, and fully parallelized write operations.

To enable the former, we utilized C++ STL threading to perform I/O operations in separate, asynchronous, threads. For reading the query sequence input, an explicit request for the next *chunk* of sequences returns such a chunk, but also spawns an asynchronous task that already prefetches the next chunk. After such a chunk has been fully processed, the placement result data is passed to another asynchronous task that is responsible for writing the data to a single output file. To guard against increasing memory utilization in the event that chunk computation is faster than writing the output, we limit the number of asynchronous tasks for each of these two phases to one at a time.

To enable fully parallel writing of results, we utilize collective MPI File write operations to write result chunks of known sizes, from each MPI rank, to one global output file. As this relies on collective MPI operations, each rank must block until all other ranks are ready to collaborate on its execution. However, this blocking time is again hidden via the aforementioned use of asynchronous I/O tasks. Using these dedicated MPI file I/O methods increased the *Parallel Efficiency* (PE) in the fully heuristic test (Section 3.4.5) for $2,048$ cores from ~20% to ~60%.

3.2.2.3 Additional Optimizations

We deploy several additional optimizations to further increase parallel efficiency. On machines that offer multiple hardware threads per physical core (*hyper-threading*), these additional hardware threads can be used for executing the asynchronous I/O tasks mentioned above.

Further, we observed greatest parallel efficiency when configuring the parallel environment to spawn one MPI rank per *socket* instead of one per core or one per node. This is almost certainly due to better data locality, as data that is shared between OpenMP threads within a MPI rank will be placed in the same NUMA locality domain. Of course, this optimization is only applicable to machines that have multiple CPU sockets per node.

We additionally observed a modest $2 - 4\%$ increase in PE when executing EPA-NG with lockless concurrent implementations of MALLOC, such as JEMALLOC. Note that, this is the only one of the listed optimizations that we did not include in our PE results.

3.2.2.4 Possible Enhancements

One might need to adapt the parallelization scheme for extremely large RTs as storing the CLVs of the RT might exceed main memory limits. To address this, the program could cyclically free branch related data (mainly CLVs) when they are not needed any more to calculate placements and load branch related data that will be needed in the near future (i.e., data prefetching). We explore such an approach in Chapter 4.

3.3 Recent Related Work

Since the release of EPA-NG, multiple competing phylogenetic placement methods have emerged, which I will describe in this section.

In contrast to ML methods, RAPPAS [66] does not rely on QSs that are aligned to the reference MSA. Instead, it first calculates a per-branch matrix similar to our preplacement lookup-table (Section 3.2.1 and Figure 3.2), with the distinction that the values it calculates are not ML per-site likelihoods, but rather likelihood values based on ancestral states that are reconstructed for the inner nodes of the RT. Based on this table, RAPPAS identifies k-mers in the table that have a probability above some defined threshold, which they call *phylogenetically informative k-mers*. These k-mers are subsequently stored in a database, which records the most probable insertion branches along with the insertion probability, for each k-mer. Finally, RAPPAS places a set of QSs by splitting each QS into its component k-mers and performing a lookup in the database for each k-mer. It then forms a consensus over the results of all k-mers to determine a set of likely placements on the tree. The idea behind the two-step approach of RAPPAS is to perform the more compute intensive database construction once. A user can then perform placement of unaligned QSs by simply using the precomputed database. In their evaluation of RAPPAS, Linard *et al.* [66] include EPA-NG in their accuracy and performance comparisons. This also constitutes an independent verification of EPA-NG.

Another recently developed phylogenetic placement method is APPLES [4]. It uses the distance-based least-squares tree building method to extend an existing phylogeny by a given QS. This method comprises two distance measures: the pairwise path distances between tips in a tree (also called *patristic distance*), and the pairwise sequences distances between the organisms of the tree. The least-squares optimal tree is defined as the tree that minimizes the squared differences between the two distance measures. One of the advantages of such an approach is that the sequence data does not need to be aligned or even *assembled*, as the only requirement for the data is that we can calculate pairwise dissimilarities for it. This applies to both, the reference and the query data, enabling the use of less conventional data such as genome skims, which are sequences that result from shallow, random sampling of genomic DNA [26, 99]. Another advantage of this distance based approach is the ability to scale to reference trees that exceed $100,000$ organisms, which still presents a significant challenge to ML based placement methods. EPA-NG is included in the evaluation of APPLES as well. The authors find that ML placement methods are the most accurate methods, when applicable (i.e., when the size of the RT is not too large).

In a very recent preprint, Jiang *et al.* [51] have combined APPLES with a deep neural network based sequence distance to perform placement of single-locus data on a multi-locus species tree. This approach attempts to account for the discordance between a gene tree (to which a given QS belongs with high confidence) and the corresponding species tree (which comprises data from multiple gene trees). For such applications, the authors find that their approach significantly outperforms the accuracy of ML placement approaches.

The most recent novel approach to placement is App-SpaM (available as a preprint, [11]). Like APPLES, App-SpaM is a distance-based approach that does not require the reference/query data to be aligned or assembled. App-SpaM applies the Leimeister *et al.* [63] *Filtered Spaced-Word Matches* distance to the phylogenetic placement problem by calculating pairwise distances between a given QS and all reference sequences. The authors describe multiple approaches to calculate the placement location for a given QS. The approach that they find to work best is to assign the QS to the branch leading to the LCA of the two closest references.

Aside from competing placement methods, there has been ongoing work by Linard *et al.* [67] to establish an extensible testing framework to compare all major placement tools. Such a framework may facilitate further development by providing a more standardized accuracy testing and performance benchmarking environment. We utilize this framework (called PEWO) in Chapter 4.

3.4 Evaluation

3.4.1 Datasets

We used three empirical datasets to evaluate and verify EPA-NG. In the following, we describe the characteristics of these datasets as well as the pre-processing steps.

3.4.1.1 Neotrop data

This DNA dataset includes a 512 taxa RT and a 16S *Ribosomal Ribonucleic Acid* (rRNA) reference MSA with $4,686$ sites [70]. The corresponding QSs with which we worked are one million dereplicated amplicon sequences. We sub-sampled QSs to perform smaller tests; the exact number of QSs used is specified in the individual tests. In the following, we will refer to this as the *neotrop* dataset.

3.4.1.2 BV data

This DNA dataset consists of a RT with 797 taxa, a 16S rRNA reference MSA with 2, 763 sites, and 15, 060 QSs [105]. We refer to this as the *bv* dataset.

3.4.1.3 Tara Oceans data

This DNA dataset consists of a RT with 3, 748 taxa, and a 16S rRNA reference MSA with 3, 374 sites [54]. The QS are metagenomic reads, filtered to the 16S rRNA locus. From this dataset, we sub-sampled 10, 000 QS. We refer to this as the *tara* dataset.

3.4.2 Test Settings

We compared EPA-NG against PPLACER and RAxML-EPA under different settings: with/without masking (not implemented in RAxML-EPA), with/without preplacement.

3.4.2.1 preplacement

The preplacement heuristic, as described in Section 3.2.1. To enable preplacement: PPLACER enabled by default (`--strike-box 3.0, --max-strikes 6, --max-pitches 40`), EPA-NG enabled by default (`--dyn-heur 0.99999`), and RAxML-EPA with `-G 0.01`. To suppress preplacement: PPLACER with `--max-strikes 0`, EPA-NG with `--no-heur`, and RAxML-EPA with default settings.

3.4.2.2 premasking

The premasking heuristic, as described in Section 3.2.1. Note that RAxML-EPA does not support premasking.

To enable premasking for PPLACER and EPA-NG, no special arguments are required. To suppress premasking, both PPLACER and EPA-NG use the flag `--no-pre-mask`.

3.4.2.3 thorough mode

This mode disables both previously mentioned heuristics and performs placement with full BLO on all branches for all sequences.

3.4.3 Verification

In [8], and [75], the authors verify the placement accuracy of their algorithms via simulation studies and leave-one-out tests on empirical data. As there already exist two highly similar and well-tested evolutionary placement tools, we compare the results of EPA-NG to the RAxML-EPA and PPLACER results via the KRD metric (Section 2.4.3) to verify that our implementation works correctly.

We computed the pairwise median KRD between the the results of the three programs PPLACER, RAxML-EPA, and EPA-NG, for three distinct datasets in two different modes. Both, the *bv*, and *tara* datasets were used as described in Section 3.4.1, with $15,060$ and $10,000$ QS respectively. For the *neotrop* dataset, we sub-sampled $10,000$ QS to roughly match the other two dataset sizes. Sub-sampling allowed us to directly compare the runtimes of the different modes, as the non-heuristic settings can increase the runtime by up to three orders of magnitude.

Two different modes were tested: the *thorough* setting with all heuristics disabled, and the *preplacement* setting, with heuristics enabled, but *premasking* disabled for better comparability with RAxML-EPA which does not implement *premasking*.

Mode	Data	EPA-ng to RAxML	EPA-ng to pplacer	RAxML to pplacer	Δ
preplacement	neotrop	0.162	0.045	0.115	1.11
	bv	0.009	< 0.001	0.010	2.11
	tara	0.064	0.013	0.072	1.87
thorough	neotrop	0.113	0.024	0.108	1.57
	bv	0.010	0.002	0.011	1.83
	tara	0.060	0.013	0.070	1.92

Table 3.1: Median KRD between placement implementations. The ratio by which the distance between the two older implementations RAxML-EPA and PPLACER is greater than the average distance between either of them to EPA-NG is denoted by Δ.

The results are shown in Table 3.1. They include a measure denoted by Δ that summarizes the KRD-values for a given dataset and mode. Δ is the ratio by which the distance between RAxML-EPA and PPLACER is greater than the average distance between EPA-NG to either of them. In other words, Δ quantifies the proximity of our results to those of RAxML-EPA and PPLACER. We observe an average Δ-value of 1.7 for the preplacement mode, and of 1.77 for the thorough mode. This means, that under comparable settings, EPA-NG produces results that are on average $70 - 77\%$ closer to RAxML-EPA and PPLACER, than RAxML-EPA and PPLACER are to each other. Overall, the absolute KRD values are very small. Thus, we are confident that our implementation is correct and yields qualitatively and quantitatively highly similar results to existing placement tools.

3.4.4 Sequential Performance

We compared the sequential runtimes of EPA-NG, RAxML-EPA, and PPLACER, under two settings. Firstly, with or without the preplacement heuristic. Secondly, with or without the masking heuristic. The combination of these settings results in four distinct comparisons (see Figure 3.4). We used $50,000$ aligned QSs from the *neotrop* dataset, as well as the accompanying reference tree and alignment for this test. The *preplacement* and *masking* settings are as specified in Section 3.4.2. Note that RAxML-EPA does not implement masking and therefore respective results are missing.

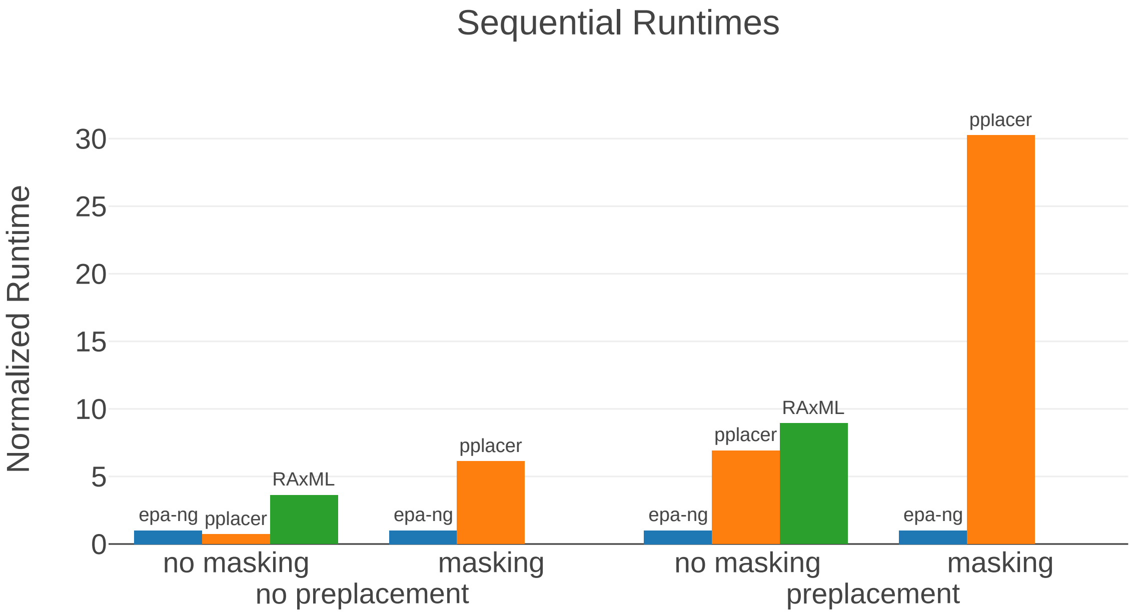


Figure 3.4: Comparison of sequential runtimes of the three programs EPA-NG, PPLACER, and RAxML-EPA, under four different configurations. The y-axis represents the runtime, normalized by the runtime of EPA-NG for each distinct configuration (reads as: *program was x-times slower than* EPA-NG). The absence of data for RAxML-EPA for the masking setting is due to the absence of such a heuristic in RAxML-EPA.

Under most configurations, EPA-NG substantially outperforms the competing programs. The only exception is the case where all heuristics are disabled. In this case we observe a runtime that is ~30% *slower* than for PPLACER, while still performing ~3.5-times faster than RAxML-EPA. However, runs with all heuristics disabled do not represent the typical use case. In the configuration using both heuristics, we observe a ~30-fold performance improvement for EPA-NG over PPLACER.

The *Sequential performance* tests were conducted on a machine with a single Intel® Core ™ i5-6600 Sky Lake CPU, and 16GB of main memory. This CPU supports *Advanced Vector Extensions* (AVX) vector instructions, which LIBPLL-2 [40] (used in our implementation) uses extensively to accelerate likelihood calculations.

3.4.5 Distributed Memory Parallel Performance

We tested the scalability of EPA-NG under three configurations. First, with preplacement and masking heuristics disabled (*thorough* test). Secondly, with only the preplacement heuristic enabled. Lastly, we tested masking in conjunction with preplacement. This corresponds to the default settings (*default* test). Please note that, as RAxML-EPA does not support masking, the respective results are missing.

As runs under these configurations exhibit large absolute runtime differences, we used three distinct input sizes (number of QS) for each of them. The smallest input size for each configuration was selected, such that a respective sequential run terminates within 24 hours. We chose subsequent sizes to be 10, and 100 times, larger, representing medium and large input sizes for each configuration.

All scalability tests were based on a set of one million (1M) aligned QSs from the *neotrop* dataset. To ensure better runtime homogeneity between parts of the 1M set, the order of the set was randomized. This is needed to ensure better comparability between runtimes based on subsets of this set. It should be noted that such a randomization also improves overall PE, as higher data-runtime homogeneity typically makes scheduling of worker threads easier. To obtain the desired input sizes, we either sub-sampled (10k, 100k) or replicated (10M, 100M, 1B) the original set of 1M sequences.

Furthermore, we pre-processed all QS files by converting them into our binary bfast-format with EPA-NG, as this improves I/O performance.

We executed the scalability tests on a infiniband-connected compute cluster. Each compute node has two Intel® Xeon® E5-2630 v3 (Haswell) CPUs (16 physical cores, 32 hyper-threads) and 64GB of *Random Access Memory* (RAM). We assigned two MPI ranks per node (one per socket), splitting the available physical cores per node equally between them. We further utilized C++ threading capabilities to asynchronously perform disk and network I/O, by oversubscribing some of the physical cores with more than one thread.

Each individual run (heuristic configuration, size, number of cores) was executed five times to minimize cluster interferences (e.g., heavy I/O activity by other users). We took the respective minima of these series of five runs, as parallel speedup and efficiency are primarily concerned with the fastest observed time.

We compute the parallel efficiency $E(N)$ as

$$E(N) = \frac{S(N)}{N} \quad \text{, with} \quad S(N) = \frac{T_1}{T_N} \tag{3.1}$$

where N is the number of cores used, S is the parallel speedup, T_N is the execution time of the program using N cores, and T_1 is the fastest sequential execution time.

As the parallel speedup and the parallel efficiency are calculated based on the fastest sequential execution time, we performed a separate run using the sequential version of EPA-NG (see: *Sequential Performance*). For each configuration, we performed a sequential run for the *small* input volume. As the larger input volumes could not be analyzed sequentially within reasonable times, we multiplied the sequential runtime by 10 and 100, for the medium and large input sizes, respectively.

The results are displayed in Figure 3.5. We observe that the *thorough* test preserves the single-node efficiency (16 cores, ~80% PE) consistently for all core counts and input data sizes. The *preplace* test behaves similarly, but parallel efficiency tends to decrease with increasing core count. This is because the response times are becoming so short, that overheads (e.g., MPI initialization and some pre-computations) start dominating the overall runtime according to Amdahl's law. This is most pronounced

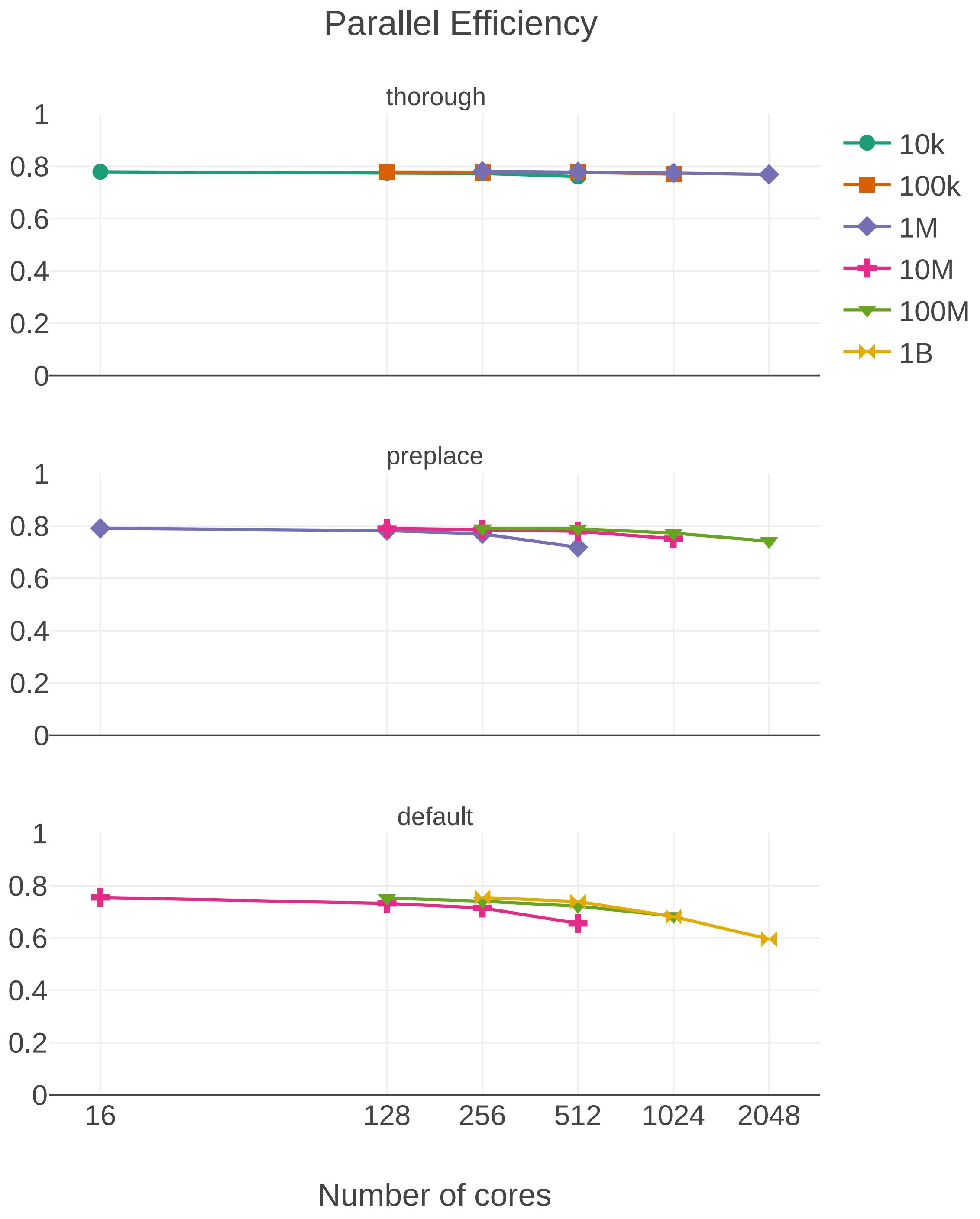


Figure 3.5: Weak Scaling Parallel Efficiency plot of EPA-ng on a medium-sized cluster. Input files with sizes ranging from ten thousand (10K) to one billion (1B) query sequences. Three different configurations are shown: *thorough*, meaning no preplacement of masking heuristic was employed, *preplace* where only the preplacement heuristic was used, and *default* where both masking and preplacement were employed.

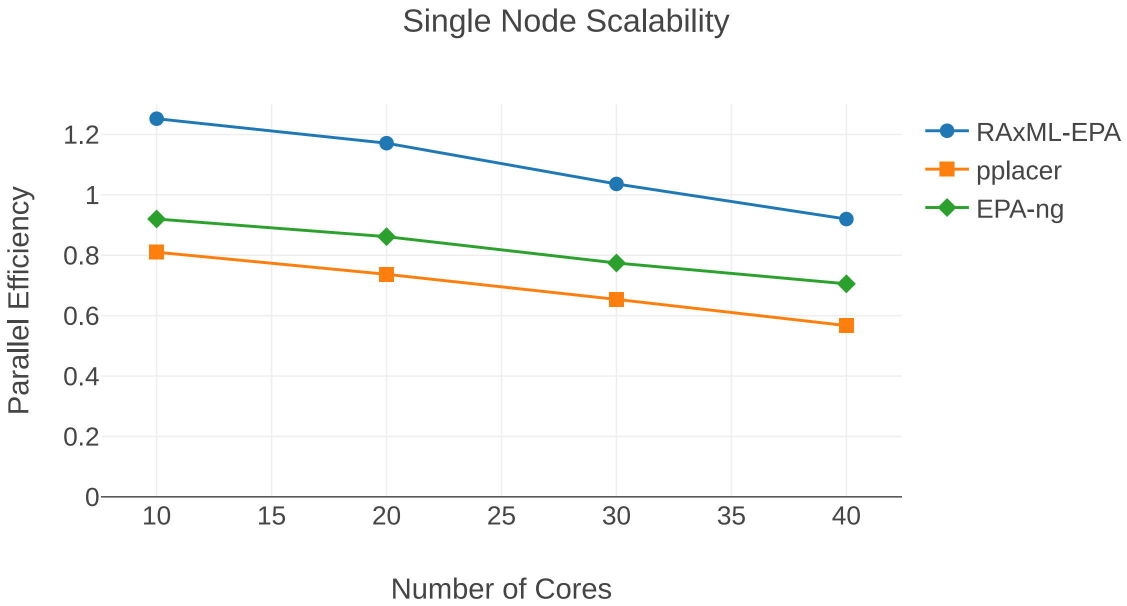


Figure 3.6: Parallel efficiency comparison of the three programs RAxML-EPA, PPLACER, and EPA-NG. The data used in these tests were 1, 000 query sequences and the reference tree from the *neotrop* dataset. All heuristics were disabled. The tests ran on a machine with two Xeon® Gold 6148 (Skylake-SP) CPUs, totaling 40 physical cores.

for the 1M QS / 512 cores data point, where PE noticeably declines. The response time in this case was only 83 s, compared to 30 542 s of the corresponding sequential run.

These effects become even more prominent in the *default* run, which shows a PE of ~60% on 2, 048 cores. This is primarily due to the increased processing speed when using masking that accelerates preplacement by an additional factor of ~7. As a consequence, operations such as I/O, MPI startup costs, or data pre-processing functions have a more pronounced impact on PE.

3.4.6 Shared Memory Parallel Performance

We performed an additional test to compare single node parallel efficiency between the three programs, RAxML-EPA, PPLACER, and EPA-NG. The results can be seen in Figure 3.6. The tests ran on a machine with two Xeon® Gold 6148 (Skylake-SP) CPUs, totaling 40 physical cores. We used the *neotrop* dataset to place 1, 000 query sequences against the 512 taxa tree, without the use of any heuristics (called *thorough mode* above). Each program and core count combination was run 5 times, and the data points shown are again the minima of this series of 5. For the sequential data points based on which the PE was calculated, we restricted the programs to purely sequential mode (through compilation of sequential executables for RAxML-EPA and EPA-NG, and through specifying `-j 0` when running PPLACER).

We observe that parallel efficiency decreases at a similar rate, although RAxML-EPA exhibits a somewhat higher PE. It should be noted that the absolute run time of RAxML-EPA was consistently ~2.3 – 2.4 times longer than the run time of EPA-NG. In turn, the run times of EPA-NG were between ~30% – 40% longer than the run times of PPLACER.

3.4.7 Real-World Showcase

Finally, we performed two tests to showcase the improved throughput of EPA-NG and to demonstrate how this enables larger analyses in less time.

First, we performed phylogenetic placement of one billion reads (pre-filtered to the 16S rRNA region) against a 3,748 taxa reference tree. Using 2,048 cores (128 compute nodes), we were able to complete this analysis in under 7 hours.

Second, we extrapolated the total reduction in analysis time of [70]. We used a representative sample of the *neotrop* dataset to obtain runtimes for both, EPA-NG, and RAxML-EPA, using the same settings as in the original study. With this runtime data, we extrapolated the total placement time of the study for both programs. We find that EPA-NG would have required less than half the overall CPU time (RAxML-EPA: 2,173 core days, EPA-NG: 864 core days) under the same heuristic settings (no heuristics). Further, using EPA-NG's novel heuristics, the placement could have been completed within ~14 core hours (roughly a 3,700-fold runtime reduction).

Our distributed parallelization also improves usability. That is, the user does not have to manually split up the query data (i.e., split the data into smaller chunks which can complete within say 24 hours on a single node) for circumventing common cluster wall time limitations.

3.5 Summary

In this chapter I presented EPA-NG, a highly scalable tool for phylogenetic placement. The evaluation showed that EPA-NG is up to 30 times faster than its direct competitors when executed sequentially, while yielding qualitatively highly similar results. EPA-NG includes several performance optimizations to achieve this, such as utilizing a memoization technique (preplacement lookup-table), and a more adaptive approach to selecting the top candidate placement branches. EPA-NG is written in modern C++, and is available as free open-source software at `https://github.com/Pbdas/epa-ng`.

Moreover, EPA-NG is the first ML phylogenetic placement implementation that can natively parallelize over multiple compute nodes of a cluster, thereby enabling analyses of extremely large query datasets, while achieving high parallel efficiency and short response times.

In a showcase test, we utilized 2,048 CPU cores to place 1 billion QSs from the Tara Oceans project, on a RT with 3,748 taxa, requiring a total runtime of under 7 hours.

4. Efficient Memory Management in Likelihood-based Phylogenetic Placement

This chapter is based on the peer-reviewed open-access publication:

> **P. Barbera**, and A. Stamatakis. "Efficient Memory Management in Likelihood-based Phylogenetic Placement." *HiCOMB 21*, 2021, *accepted, in press*

4.1 Introduction

In this work so far, optimizing phylogenetic placement focused on scalability and throughput with regards to the number of QSs. However, depending on the resources available, the memory requirements for a given analysis can also constitute a limiting factor. Furthermore, high memory requirements can yield the deployment of hardware accelerators, such as *General-Purpose Graphics Processing Unit*s (GPGPUs), challenging as such devices typically have less RAM available than the host system. Even when access to high performance computing resources is available, the memory requirements of specific analyses can still be prohibitive. For example, in the future, third generation/long-read sequencing (Section 2.1) is expected to further increase the number of available high length and high quality sequence assemblies, up to and including whole genomes.

For likelihood-based phylogenetic inference (ML and Bayesian inference), improving memory efficiency represents a particular challenge due to the way we compute the likelihood on a phylogenetic tree. At each internal node of the tree, we calculate a conditional likelihood for each possible character state (typically DNA or protein data) of a site in the underlying input MSA and typically store it as a hard-to-compress floating point value. Realistic models of nucleotide substitution (e.g., the Γ model of rate heterogeneity [114] or other mixture models) further increase the memory requirements as we need to calculate and store conditional likelihood values for several rates of evolution per state, internal node, *and* MSA site. The data structure holding these conditional likelihoods at each inner node of the tree is called *Conditional Likelihood Vector* (CLV) (see Section 2.3.3).

Apart from general phylogenetic tree inference, memory efficiency becomes even more performance-critical in the special case of likelihood-based ML phylogenetic placement as implemented in EPA-NG. To accelerate its computations, EPA-NG calculates and stores CLVs for all possible directions (i.e., all three outgoing branches) at each internal node of the tree in memory. Note that most tree search tools typically only store one CLV per node due to the distinct pattern of likelihood calculations they conduct. This memory organization in EPA-NG incurs a substantial memory overhead, making placement infeasible for large reference trees [4, 11] containing on the order of $100,000$ references.

Previous work has shown that the likelihood of a tree with n leaves can be calculated using a minimum of $log_2(n) + 2$ CLVs [48]. Based on this property, we conceived strategies to reduce the overall memory requirements of ML likelihood calculations at the cost of additional computations. However, this approach was never integrated into our production-level tools.

In this chapter, I show how this memory saving technique can be applied to placement and I outline its integration into our production-level placement software EPA-NG. More specifically, I describe the necessary adaptations to the parallelization approach used in EPA-NG. In Section 4.5 we experimentally assess the respective memory versus inference time trade-offs incured by using this approach.

4.2 Methods

As outlined in Section 2.3.3, we compute the likelihood of a strictly binary tree via a post order traversal, starting at the leaves of the tree and moving toward the virtual root via the FPA [36]. The virtual root of the tree can be placed into any position of any branch of the tree for the sake of defining a traversal order. Its placement does not alter the likelihood score of the tree as long as the substitution process is time-reversible, which is the case for all standard statistical models of evolution. Each step of the FPA operates on a small subset of the overall tree, where the CLV of a parent node is computed by accessing the CLVs of the two child nodes. The parent CLV summarizes the data/signal contained in the subtree it roots. When this recursive algorithm terminates at the virtual root of the tree, the entries of the CLV(s) at the virtual root are used to calculate the overall likelihood of the tree. We illustrate one such step in the FPA recursion in Figure 4.1.

In standard phylogenetic likelihood implementations, memory is allocated for all CLVs visited during the FPA, that is, at all inner nodes of the tree. This memory

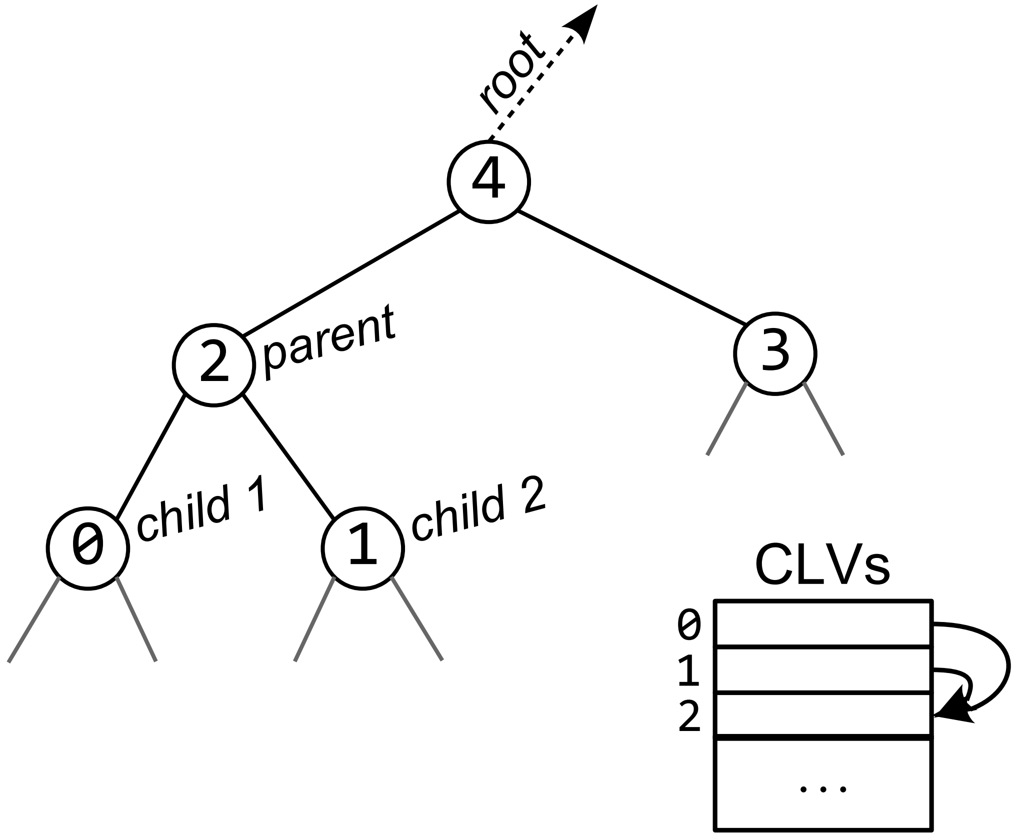


Figure 4.1: We illustrate a subtree of the phylogenetic tree during a recursion step of the FPA. The CLV index number of the nodes is displayed inside the nodes. In the FPA step shown here, we combine the CLV information from the two nodes labeled *child 1* and *child 2* in the *parent* node. After this step, the two child node CLVs become obsolete. We repeat this recursion until we reach and calculate the CLV of the root node (not shown; direction of root indicated by an arrow), that is subsequently used to calculate the overall likelihood of the tree.

allocation scheme yields 'good' computational efficiency, as a large fraction of CLVs can typically be reused in subsequent likelihood calculations, for instance, when only a part of the tree topology has changed. However, this comes at the expense of an increased memory footprint. In general, this trade-off is justified when taking into account the input data sizes of typical phylogenetic analyses and the increasing amount of main memory available in modern computers. However, as already mentioned, for certain likelihood based tools there is a pressing need to offer alternative solutions that allow to explicitly limit the amount of memory used by likelihood calculations. This is particularly the case in EPA-NG which stores $3 * (n - 2)$ CLVs, compared to $n - 2$ CLVs in most standard phylogenetic tree inference programs. Note that, storing the CLVs clearly dominates the memory requirements of all likelihood-based phylogenetic inference tools.

A solution to reduce the number of CLVs that need to be concurrently held in memory is to exploit the recursive structure of the FPA. In particular, once a parent CLV has been successfully computed from its child CLVs, the data held by the children is no longer needed (hence the name 'pruning algorithm') to compute the overall likelihood of the specific, fixed tree. Thus, the memory allocated to the children CLVs can be overwritten by CLV entries required at other internal nodes. Hence, for a given tree topology and (virtual) root, there exists some minimum required number of CLVs that need to be held in memory. Izquierdo-Carrasco. *et al.* [48] have shown that this minimum number of required CLVs is $log_2(n) + 2$ in the worst case for a fully balanced binary tree with n leaves. We will henceforth refer to this method as the *logn* approach.

Further, Izquierdo-Carrasco. *et al.* [48] described a data structure and a CLV management approach, to dynamically determine which CLVs to overwrite. Central to this is the concept of a *slot*, which denotes the allocated memory that stores one CLV. Different CLVs occupy this set of slots at different stages of the tree traversal as induced by the FPA. Note that the number of available slots does not need to be set to the minimum of $log_2(n) + 2$, but can also be set to a larger value. In particular, it can be set such that the CLV storage space corresponding to the number of slots matches the amount of memory available on the system. When CLVs can be reused between applications of the FPA, which is the case for placement, providing more memory than absolutely necessary reduces the number of CLVs that have to be recomputed. In other words, the memory versus runtime tradeoff can be tuned via the number of available slots.

Here, we implement a generalized version of this *Actively Managed CLVs* (AMC) mechanism into our free, open source library for ML phylogenetic likelihood calculations LIBPLL-2 on which EPA-NG relies. By deploying this approach, the user can now set an approximate explicit limit for the memory footprint in EPA-NG. This allows for placing sequences on substantially larger reference trees than before.

Recall that in placement, the goal is to find a set of most likely edge(s) from a given RT that a QS belongs to. When using the maximum likelihood approach to placement, we calculate the likelihood of a QS placement as the likelihood of the RT, extended at a given branch by the QS. To save time, EPA-NG pre-calculates and stores the CLVs for each possible insertion branch in the RT in memory (see Section 3.2). This means that the placement of a QS comprises calculating (i) one CLV summarising the signal from the insertion branch for a specific insertion point along that branch, (ii) setting the CLV at the newly added leaf, and (iii) using these two CLVs to compute the placement likelihood.

While the above implementation exhibits good runtime performance and facilitates parallelization (i.e., offers a high degree of parallelism), it also requires a comparatively large amount of main memory as we need to allocate memory for all possible CLVs in the tree. To address this, we can apply the *logn* approach to substantially reduce the peak memory consumption of EPA-NG.

However, this comes at the cost of increased execution times, as for each iteration over the tree, the per-branch CLVs will have to be re-computed, potentially including the re-computation of CLVs in the respective subtrees defined by this branch. Further, using the *logn* approach to save memory substantially complicates the parallel placement procedure implemented in EPA-NG, as it relies on the assumption that immediate random access to any desired CLV is given at any time. To address this, we now split up the parallel placement of QSs into blocks of RT branches. The CLVs for the next branch block are pre-computed asynchronously, while we work on placing QSs on the current branch block.

Another challenge with such an active CLV memory management approach is to minimize the general computational overhead under memory constraints. EPA-NG utilizes additional memoization techniques, trading additional memory, beyond the CLVs storage space, for increasing speed. More specifically, we use a lookup table that contains constant, precomputed placement results for every branch that allow to rapidly pre-score putative placements (see Section 3.2.1 and Figure 3.2). When executing EPA-NG in default mode with memory saving disabled, this lookup table already provides a substantial (~15 fold) speedup. As we will see in Section 4.5, executing EPA-NG with AMC, using this lookup table improves execution times by up to ~23 times. The reason for this is straight-forward: when using prescoring heuristics, the only time every QS is matched against every branch is during this first phase (see Section 3.2.1). Branch block computation, if done for every branch, has an extremely high computational cost compared to the rest of the program. Thus, we can eliminate the vast majority of the computational effort in the AMC case by using the lookup table, as we will only have to utilize the branch block computation for the second phase of placement, where each QS only gets matched against a small set of promising branches.

An additional parameter affecting runtime performance is the number of QSs processed per iteration, called the *chunk size*. EPA-NG processes QSs in chunks (i) to overlap I/O with computations and (ii) to limit the impact of the sheer QS data volume on the overall memory footprint (see Section 3.2.2.2). Note that a comprehensive placement based analysis involves on the order of 10^7 QSs or greater [70].

When AMC is enabled, using a larger chunk size decreases the number of times the CLVs of the tree need to be recomputed, as the CLVs have to be recomputed at least once per QS block. However, a higher QS block size also means that there is less memory available for other data structures. This becomes a limiting factor for large RTs, as there are internal intermediate datastructures that save results for each combination of RT branch and QS, which can occupy a significant fraction of the available memory (see Section 4.5). Hence, there is an additional trade-off to consider here.

4.3 Related Work

The *logn* approach to performing the FPA was originally implemented as proof-of-concept option in RAxML-Light, where it enabled phylogenetic inference of trees comprising more than 100, 000 leaves [106]. To our knowledge the only other ML phylogenetics software that offers AMC is IQ-TREE 2 [81] which also explicitly uses the *logn* [48] approach.

With respect to placement software there are, to our knowledge, no other programs that employ an active memory management strategy. Of those based on ML methods [5, 8, 75], only pplacer offers a dedicated option to handle input data sets with large memory footprints. For large datasets, the user can specify the location of a memory-mapped file, which will be used for larger memory allocations, thereby reducing the peak main memory consumption. Consequently, the runtime performance of this approach depends on the latency and bandwidth of the underlying file system.

As outlined in Section 3.3, there exist several placement tools that do not rely on ML methods [4, 11]. These programs are characterized by their extremely low memory consumption, that is typically several orders of magnitude lower than for ML methods. For example, APPLES [4] is a distance-based approach that uses least-squares minimization to determine the placement of a QS on a RT. APPLES was used to perform placement on a tree with 200, 000 leaves, by only using ~4GiB of main memory.

4.4 Implementation

Our AMC implementation in libpll-2 comprises two major components, which we illustrate in Figure 4.2. The first component is the mapping of a potentially large number of global CLVs to a substantially smaller set of physical memory locations available to hold them, called *slots*. This can be efficiently implemented by using two arrays that map the global index of a CLV to its slot index, and vice versa. When a slot is currently not associated with a CLV index, or when a given CLV index is not present in memory (not *slotted*), we use dedicated values to denote these special states.

The second major component is the mechanism for choosing which slotted CLVs to overwrite. This slotted CLV overwriting mechanism is in part analogous to cache line replacement policies, but under additional constraints as we can not overwrite

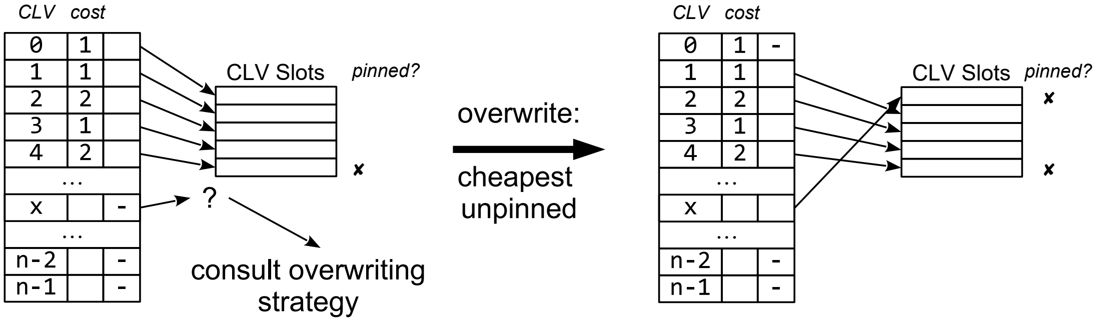


Figure 4.2: Illustration of the CLV management and replacement strategy. We show a snapshot of the CLV management data structures. These include the *cost* of each CLV, which we approximate by the number of descendant nodes that the CLV summarizes (i.e., the nodes in the subtree rooted by the CLV). We also show the mapping of CLV indexes to their locations in physical memory, called CLV *slots*. If a CLV has not been assigned to a slot, it is marked by (-). For each slot, we also record if it is *pinned*, meaning that it can not be overwritten. Note that these illustrations correspond to the tree shown in Figure 4.1. On the left, we show the situation where CLV x has to be computed and stored, while we have not yet assigned a physical memory location for x, and all slots are occupied. Thus, we need to invoke the *overwriting strategy* to select an appropriate slot among the unpinned slots. The strategy is to select the slot occupied by the CLV with the smallest recomputation cost. On the right we show the result of this operation: we assign x to the slot that previously belonged to the CLV with ID 0. Finally, we also mark this slot as pinned, as this example is part of an FPA execution where x is a current *parent*.

any slotted CLV we wish, due to the tree traversal order that defines the data access pattern. Hence, designing an appropriate overwriting strategy is not trivial and can evidently affect runtime performance. It is beneficial to retain slotted CLVs that are likely to be accessed again in the near future during subsequent likelihood calculations. The overwriting strategy heavily depends on the order of CLV operations which might be highly program-/algorithm- specific. Thus, we have implemented a generic replacement strategy interface via a set of callback functions that allow the developer to fully customize how a slot is chosen/overwritten. As default strategy, we have implemented an algorithm that chooses which CLV to replace based on its approximate recomputation cost. We approximate the recomputation cost of a CLV by the number of descendant leaves the CLV summarizes (i.e., the size of the subtree it roots).

4.4.1 Pinning

Finally, an additional mechanism for maintaining consistency is required which is called *pinning*. As mentioned before, some intermediate CLVs *must* remain slotted/pinned to the slots during the FPA that traverses the tree in post-order to be able to compute the likelihood score. We illustrate this mechanism via the tree shown in Figure 4.1. Let us assume that we have just calculated the CLV of node 2. Next, the FPA recurses into the subtree rooted by node 3. However, we can not yet discard the result stored at node 2, and thus need to pin the associated memory slot as it will be required to compute the CLV at node 4. The pinning mechanism can also be exploited beyond a single tree traversal in order to retain and re-use CLVs among successive full tree traversals. However, care has to be taken to not pin too many slots, as this could cause the FPA to fail if an insufficient number of unpinned slots is available.

As EPA-NG operates on a static tree (i.e., the underlying RT is fixed and does not change), we are able to utilize the CLVs pinning mechanism between successive iterations over the tree to minimize recomputation cost. This is of particular use for the aforementioned branch block precomputation. Here, we traverse the RT, and for each branch we visit, we recompute the two CLVs at either end of the branch. To do so, we first determine on which CLVs in the respective subtree defined by that branch these depend, and consequently also have to be computed. For the CLVs in the subtrees, we construct a list of those that are currently slotted, along with a value reflecting their approximate recomputation cost. From this list we retrieve the entries with the highest recomputation cost, and subsequently pin the corresponding CLVs to their slots. We choose the number of slots we pin such that after this high-cost CLV pinning step, the number of unpinned slots is at least $log_2(n) + 2$ (Section 4.1), which ensures that we can successfully execute the FPA.

4.4.2 Parallelization

The AMC strategy also has implications on the degree to which we are able to effectively parallelize the code. Normally, parallelization of placements is straightforward: we merely perform the core placement procedure for each QS and branch pair to be evaluated via random accesses to the corresponding precomputed CLVs. However, when the AMC strategy is enabled, the number of these fine-grained placement tasks is limited by the block of branches (subset of branches in the RT) and corresponding CLVs available to the current iteration.

Further, computing the CLVs of a branch block using the FPA is all but straightforward to parallelize efficiently, as the two main options for doing so have severe limitations. One approach is to parallelize over the width of the alignment, segmenting it into one subregion per thread. However, when subregions become too small, this can actually *increase* the computation time compared to a single-threaded approach [59, 61]. Due to the often low-width nature of alignments used in phylogenetic placement, this situation can arise for even modest number of threads, substantially limiting the potential for parallel efficiency. Another approach is to utilize the recursive nature of the FPA, calculating independent CLVs concurrently whenever possible. This approach is complicated by the limitations of CLV availability when using AMC, as it introduces further dependencies between normally independent CLVs.

As a consequence, this branch block precomputation for the subsequent branch block can constitute a bottleneck. This is especially true when we can not deploy the preplacement lookup table due to memory limitations (Section 4.2). In this case, the computational effort required to compute the CLVs needed for the branch blocks dominates the execution time.

4.5 Experimental Setup and Results

We assessed the performance impact of AMC on EPA-NG using 3 representative empirical datasets with distinct characteristics. We list these datasets in Table 4.1 where we show the number of leaves of the RT, the number of sites in the alignment of the reference sequences with the QS, the type of the underlying data (NT for nucleotide, AA for Amino Acid), as well as the reference to the data source.

The *neotrop* (see also Section 3.4.1.1) dataset covers the QS number/volume dimension. This dataset was used to evaluate the microbiome of neotropical soils, including a reference tree that was tailored to the studied environment [70]. The corresponding reference alignment and QSs are 16S rRNA nucleotide data, and their QSs comprise 95, 417 sequences.

The *serratus* dataset aims to cover an increased alignment width/length and resulting CLV-size dimension by using a reference alignment with 10, 170 amino acid sites. This dataset was the result on our work on the Serratus open science project [30] which uncovered new sequence diversity from the Sequence Read Archive [64]. More specifically, the reference alignment spans 546 sequences from the *Coronaviridae* virus family. Here, the QSs only comprise 136 RdRP sequences from assembled genomes that showed high sequence similarity to the Coronaviridae family.

Lastly, the *pro_ref* dataset covers the RT-size dimension. This dataset comprises the largest default RT from the PICRUST2 software, spanning 20, 000 reference sequences [27]. To this, we added a set of 3, 333 16S QSs, sampled from the wild blueberry rhizosphere [117].

All scripts and datasets used for our experiments are available online at: `https://github.com/pbdas/memsaver-paper`

Table 4.1: Datasets

Name	**leaves**	**sites**	**#QSs**	**type**	**reference**
neotrop	512	4, 686	95, 417	NT	[70]
serratus	546	10, 170	136	AA	[30]
pro_ref	20, 000	1, 582	3, 333	NT	[27, 117]

4.5.1 Execution Time

To assess the memory versus runtime trade-offs of EPA-NG, we used the PEWO testing framework [67]. Using this framework, we have extended an already available workflow to measure the runtime and the memory footprint with our new EPA-NG memory saving mode.

In our tests, we evaluated how constraining the memory available to EPA-NG increases overall execution times. We performed placement on the datasets described in Table 4.1 for different maximum memory settings (set by `--maxmem`) in each run. Every `--maxmem`/dataset configuration was executed five times, and the results we show are calculated as the mean over all five runs, both for the execution time and memory footprint. Note also that PEWO limits individual executions to one worker thread per run (i.e., `--threads 1` for EPA-NG). We denote the fastest run using the default EPA-NG parameters (i.e., whith AMC *disabled*) as *reference run.*

The results of these tests are shown in Figure 4.3. In this graph we show the fraction of memory used, compared to the memory required by the reference run on the x-axis. On the y-axis we show the execution time slowdown of each run, relative to the reference run. For legibility we have scaled the y-axis using the binary logarithm.

Additionally, we show absolute execution time and memory footprint values for these runs in Table 4.2. Here we distinguish between the reference runs which we denote by O, the runs using AMC to the fullest extent (i.e., with the greatest memory limitation possible) denoted by F, and an intermediate setting denoted by I. We chose the intermediate run such that it represents the setting, unique to each dataset, for which we still observed comparatively low execution times (i.e., before the execution time rises sharply).

We observe two distinct characteristics.

Firstly, there is a high variance with respect to the lowest possible memory footprint we can achieve. For the serratus dataset we are able to reduce the memory footprint to 4% of the reference run. In contrast, for the neotrop dataset we are only able to

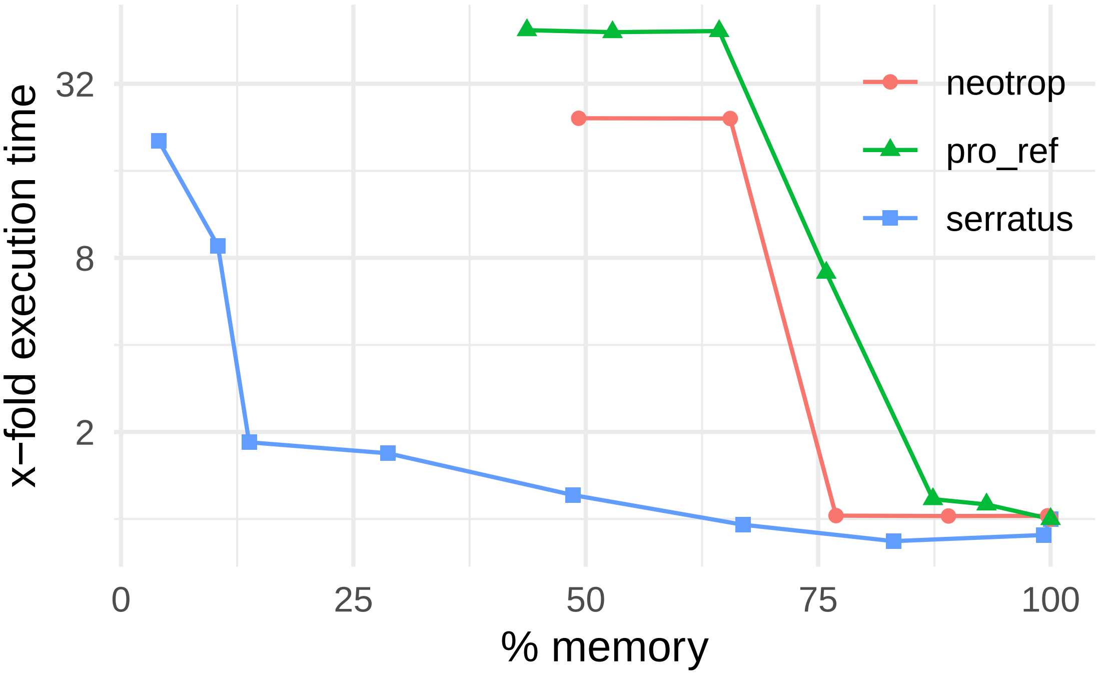


Figure 4.3: Runtime characteristics of varying memory limitation settings (`--maxmem`), relative to the fastest execution *without* active CLV management (reference execution). The x-axis indicates the memory limitation setting as a percentage of memory used by the reference execution. The y-axis indicates the slowdown relative to the reference execution. The sharp sudden decline in execution times occurs when the set memory limit is large enough to allow using the preplacement lookup table.

Table 4.2: Memory footprint and execution time without (O), with full (F), and with intermediate (I) AMC

dataset	**time (s)**			**memory (MiB)**		
	O	I	F	O	I	F
neotrop	160	165	3,908	416	319	205
serratus	18	34	370	6,344	875	258
pro_ref	104	123	5,134	8,701	7,600	3,800

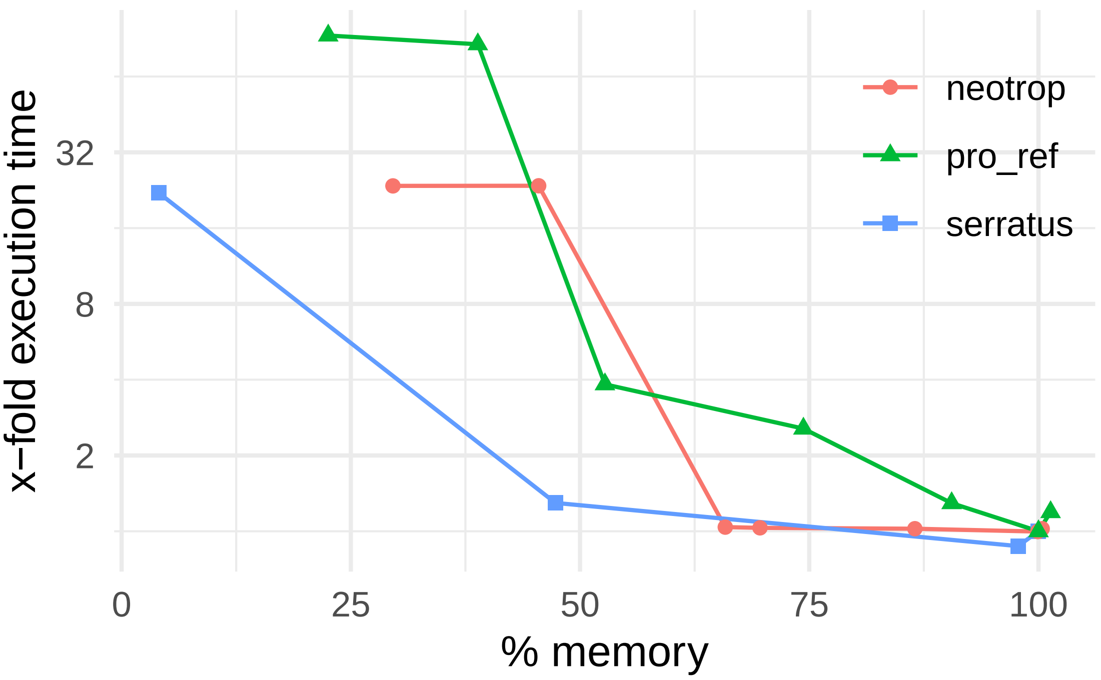


Figure 4.4: Runtime characteristics of varying memory limitation settings, using a lower QS chunk size. The tests are identical to those presented in Figure 4.3, with the exception that the QS `--chunk_size` was lowered from the default of $5,000$, to 500. This illustrates the improved lowest possible memory footprint, and the impact of chunk size on the memory footprint, as well as the execution time.

decrease the memory footprint to 48.7% of the reference run. This limitation of the lowest possible memory setting is in part due to the QS chunk size (see Section 4.2), which, in this set of experiments, was set to the default value of $5,000$ QS per chunk. A larger chunk size increases the size of intermediate result structures, which EPA-NG allocates proportionally to the number of QSs in each chunk. Thus, decreasing the chunk size, that is, reducing the number of QS that are processed in one pass over the tree allows to further decrease the minimum required memory footprint, but at the expense of increased execution time. We show this behavior in Figure 4.4.

Secondly, as we approach this minimum possible memory footprint, there is a sharp, sudden increase in execution time. This effect is caused by not being able to allocate the lookup table memoization (Section 4.2) any more when the available memory does not allow for it.

To further showcase the previously mentioned impact of the chunk size on both, the minimum possible memory footprint, as well as the increase in execution time, we repeated the above experiment using a chunk size of 500. For this additional experiment we also repeated the reference runs using the lower chunk size. We show the results of this test in Figure 4.4. In general, we observe an analogous behavior as for the initial experiment. As expected, we now also observe a lower minimum memory footprint of ~25% for both, the neotrop, and pro_ref data. We also observe that the increase in execution time for the neotrop data remains largely unaltered. Both, in this test, and the initial test, we measured a ~23-fold increase in execution times relative to the respective reference runs. In contrast, for the pro_ref data and a chunk size of 5, 000, we observe a ~49 fold increase in the execution time over the reference run. For the same data and with a chunk size of 500, we observe a ~90 fold increase in the execution time over its reference run. To provide some perspective, this particular run took ~2.4 hours on a single core, which should be acceptable for most users.

This substantial deviation between the behavior of the neotrop and pro_ref analyses is due to the large difference in respective RTs sizes (512 versus 20, 000 taxa), as the latter requires an increased number of CLV (re-)computations. The results for the serratus data remain unchanged, as for both chunk size settings, all QSs fit into a single chunk. Note also that one pro_ref datapoint shows both, a higher memory consumption as well as a slightly increased execution time compared to the reference run. We attribute this behavior to imperfect memory consumption accounting of our implementation. This accounting is the basis of a memory budgeting based upon which we determine the amount of resources we are able to spend before exceeding the limit set by the user. Thus, a flaw in this accounting can lead to a larger than desired memory footprint. We intend to address this issue in the future.

Finally, as expected, execution times improve with increasing memory available. Once the memory limit allows allocating the lookup table memoization, the execution time rapidly approaches the execution time of the reference run. As described in Section 4.2, computing the lookup table represents a one-time computational overhead. Subsequently it is used to accelerate *all* QS pre-placement operations. Thus, if the QSs of a chunk are pre-placed on a small common subset of RT branches, the high computational overhead of successive re-computations of reference CLVs is substantially reduced. Analogously, if there is only one single QS chunk to be processed, the run time impact of re-computing the CLVs in the RT is less pronounced.

4.5.2 Comparison with pplacer

Next, we showcase the capabilities of `EPA-ng` when using AMC, compared to the closest competitor software `pplacer` [75].

To our knowledge, `pplacer` is the only other ML phylogenetic placement software that offers an option to reduce the memory footprint. The `pplacer` tool does so by allocating a significant part of the required memory using a memory-mapped file, effectively extending the available main memory by using disk space. This is an on/off approach, meaning it does not allow the user to more finely control the impact on execution time.

For our showcase test, we chose the two datasets that have the highest memory footprint (serratus and pro_ref). For both datasets, `EPA-ng` requires more than 4GiB of memory, which is a common main memory size on older personal laptops, and which we believe to represent a common current constraint for users with limited

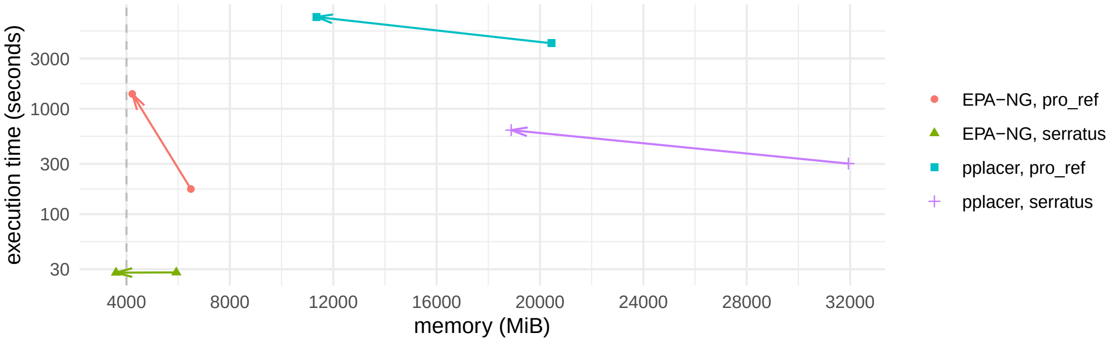


Figure 4.5: Showcase comparison between `pplacer` and `EPA-ng`. We show runs for two different datasets (serratus and pro_ref). We ran each combination of dataset and placement tool both with and without memory saving techniques enabled. Arrows indicate the change in memory footprint and execution time going from memory saving disabled to having memory saving enabled.

access to newer hardware. Thus, the goal of this test was to show how limiting the memory to 4GiB affects the execution times of `EPA-ng` and `pplacer`.

We show the results of this test in Fig. 4.5. We ran both softwares using their default parameters, with the exception of limiting the chunk size of `EPA-ng` to 500. We ran each combination of dataset and placement tool both with and without memory saving techniques enabled. As before, each run was repeated five times, and we report the mean of the runs. Arrows indicate the change in memory footprint and execution time going from memory saving disabled to having memory saving enabled.

We observe that, for the same data, `EPA-ng` performs significantly better than `pplacer` with respect to memory consumption and execution time, both for memory saving disabled and enabled runs. With its memory saving enabled, `pplacer` achieves a significant relative reduction in memory consumption at a moderate cost in execution time. However, when `pplacer` has memory saving enabled, its memory consumption is ~2-3 times higher than `EPA-ng` with its memory saving techniques (AMC) *disabled*.

Regarding `EPA-ng`, as in previous tests, we again observe the significant difference on the execution time increase of AMC between the serratus and pro_ref datasets.

4.5.3 Parallel Efficiency

Due to our adapted parallelization approach for the memory saving option, we also re-evaluated the per-node PE of EPA-NG. For each dataset, we evaluated the PE under three scenarios: limiting the memory as much as possible (*full*), limiting the memory such that the maximum number of slots can be allocated (i.e., three CLVs per inner node), resulting in approximately the same memory footprint as without memory limitation (*maxmem*), and not limiting the memory at all, that is, disabling the memory saving mode (*off*). Furthermore, we again choose the fastest out of five runs as representative datapoint. We compare the run time of each parallel

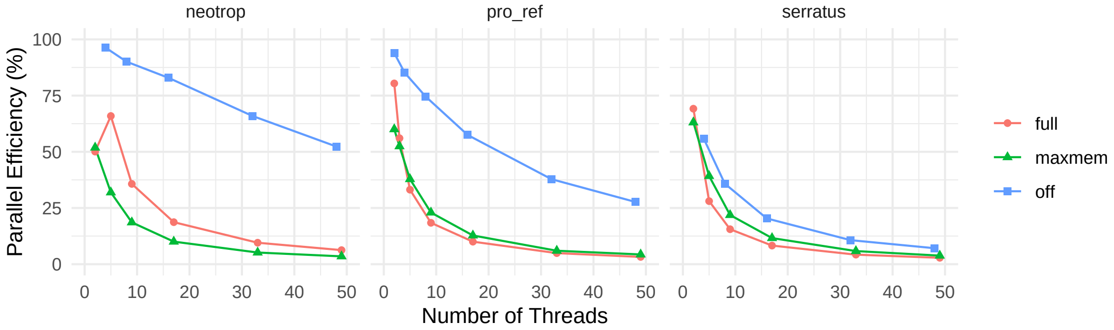


Figure 4.6: Parallel efficiency, measured across the three test datasets, with varying memory limitation settings: no *Actively Managed CLVs* (AMC) (off), minimum memory AMC (full), and maximum memory AMC (maxmem).

configuration to the fastest *serial* run under the same configuration. For the serial runs under each of the above three configuration, we compiled a dedicated version of EPA-NG under a setting that disables any kind of multithreading (`EPA_SERIAL=1 make ...`). We performed all tests on a shared memory system with 48 physical cores (two Intel® Xeon® Platinum 8260 Processors), and 754GB total available RAM.

The results of the PE test are shown in Figure 4.6. In this graph we show the number of threads used in each run on the x-axis. Note that, when AMC is enabled, the asynchronous CLV precomputation means that we are using one additional worker thread. Thus, results using the *full* or *maxmem* setting include one additional thread. On the y-axis, we show the PE. To calculate the PE, we first calculate the parallel speedup $S(r)$ of a run r with execution time $T(r)$ as

$$S(r) = \frac{T(s)}{T(r)} \tag{4.1}$$

where s denotes the serial run, and $T(s)$ the execution time of the serial run, respectively. Subsequently we calculate the *Parallel Efficiency* (PE) $E(r)$ of a run r as

$$E(r) = \frac{S(r)}{P(r)} \tag{4.2}$$

where $P(r)$ denotes the total number of threads/processors utilized in the run.

When AMC is disabled, we observe similar results to our previous evaluation of EPA-NG (see Section 3.4). Further, we observe that PE improves with an increasing number of QS.

In contrast, when AMC is enabled, the PE decreases substantially. This is due to the overhead of CLV recomputations for the branch buffer, which is only parallelized insofar as running on a separate, asynchronous thread. The goal of this

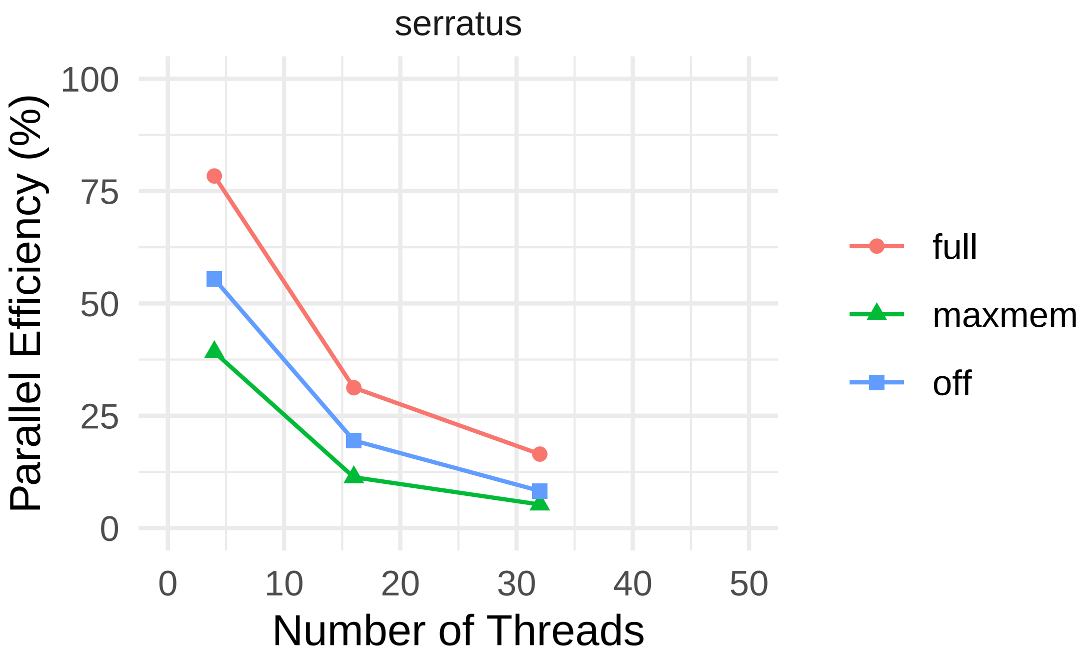


Figure 4.7: Parallel efficiency using an experimental across-site parallelization scheme to accelerate branch buffer precomputation. We measured the serratus dataset, with varying memory limitation settings: no *Actively Managed CLVs* (AMC) (off), minimum memory AMC (full), and maximum memory AMC (maxmem).

parallelization approach was to overlap the CLV computation for one branch block with the QS placement computations of another branch block. The key limiting factor is, that based on our experience with concurrent likelihood calculations, parallelizing the CLV recomputation per se on typical reference datasets, is difficult. Hence, the PE of such an additional parallelization at this level is expected to be sub-optimal. We nonetheless investigated the impact on PE when using a version of LIBPLL-2 that parallelizes CLV calculations over individual sites of the reference alignment. For this, we modified EPA-NG to perform the branch buffer precomputation *synchronously*. In this version, we first use *all* available worker threads for the CLV precomputation of one block and subsequently use *all* worker threads to perform the placement operations on this block. We show the results of this test in Figure 4.7.

Due to time constraints we were only able to obtain data for the serratus dataset, which also presents the most promising candidate for this type of parallelization scheme. For this dataset we observe a substantial improvement in PE. When using 32 threads and the *full* mode, the asynchronous approach showed ~4% PE, whereas our experimental approach yields ~16%. The *maxmem* setting performed nearly identical for 32 threads for both the asynchronous and experimental approaches. However for the same number of threads and when not using AMC, the asynchronous approach had a PE of ~10%, whereas the experimental approach achieved ~8%. Note however, that the serratus data presents the rather atypical case of a very wide alignment. Wide alignments with a large number of alignment sites are known to increase the efficiency of the across-site parallelization approach we deploy.

In contrast, supplying too few sites per thread can be detrimental to the overall execution time [60]. Our preliminary results for combining the experimental parallelization suggest the same behaviour for the neotrop dataset. With the neotrop data and the *full* mode, we observed the run using 32 threads to be ~20% slower than the serial reference. Thus, we can only recommend an across-site parallelization for sufficiently wide alignments.

4.5.4 Verification

Both EPA-ng and libpll-2 include regression testing to ensure the validity of any changes made to the code. Additionally, EPA-ng includes a suite of unit tests. We executed all available tests for the versions of EPA-ng we evaluated here, and detected no fault or deviation from previous results.

4.6 Summary

My previous work on EPA-ng focused on runtime performance and scalability with regards to the number of QS. However, when the goal of the user is to increase the number of references, the use of EPA-ng can be limited by its comparatively high memory consumption.

To alleviate this, I have implemented a memory saving approach for likelihood-based placement which involves active CLV management. By default, this approach does not require any sort of user intervention as the limit for the amount of available memory is determined automatically by our program. However, when the user desires to do so, he/she can explicitly set a specific memory limit via a simple command line option (`--maxmem`).

Thereby, this feature enables maximum likelihood based phylogenetic placement on large scale datasets under resource limitations and/or on extremely large reference trees. The evaluation shows that, while in some cases, the slowdown of EPA-ng induced by aggressive memory saving can be substantial, this only occurs for the most extreme case. In particular, when the allotted memory allows for using the EPA-ng memoization technique, the memory saving to runtime trade-off is acceptable and practical. For example, using the most aggressive memory saving setting, when applied to the pro_ref data, we are able to reduce the memory footprint by ~77% by increasing the execution time from 1.6 minutes to 2.4 hours. With the memoization technique enabled, we are able to reduce the execution time to ~6 minutes, while still reducing the memory footprint by ~43%. This holds in particular for large datasets where otherwise using EPA-ng would merely not have been possible due to memory limitations.

As I have generalized and implemented the CLV memory management in the free, open source phylogenetic maximum likelihood library libpll-2 [40], other likelihood-based tools such as, for instance, RAxML-NG[60] can now also deploy this technique.

5. SCRAPP: A tool to assess the diversity of microbial samples from phylogenetic placements

This chapter is based on the peer-reviewed open-access publication:

P. Barbera, L. Czech, S. Lutteropp, and A. Stamatakis. “SCRAPP: A tool to assess the diversity of microbial samples from phylogenetic placements.” *Mol Ecol Resour*, 2021, Volume 21, Pages 340—349

accessible online at: `https://doi.org/10.1111/1755-0998.13255`

5.1 Introduction

A drawback of phylogenetic placement is its inability to resolve relationships between individual QSs, even when they are placed in close proximity to each other on the RT. This is sensible as it substantially reduces the computational effort while still producing highly accurate results, especially for short read sequences with weak phylogenetic signal. Nonetheless, resolving relationships between QSs constitutes a desired feature by many users. Furthermore, we expect this feature to become more important with the increasing adoption of fourth generation sequencing technologies, which yield substantially longer reads. We have previously demonstrated the value of resolving between-QS relationships with longer read data [50] and hope that the methods presented here constitute a step into this direction.

One key goal of molecular studies is to assess the diversity of a sample. As mentioned in Section 2.5, a number of distinct metrics exist to quantify the diversity within a sample (α-diversity), and between samples (β-diversity) [110]. For a subset of these metrics, phylogenetic information can be used to calculate both α (e.g., PD [34], and Phylogenetic Species Variability [45]) and β (e.g., the UniFrac distance [69]) diversities. A relatively recent approach to quantifying α-diversity using sequence data is phylogeny-aware molecular species delimitation [41, 53, 115, 118]. These methods rely on a given phylogenetic tree to identify species boundaries, essentially resulting in a clustering of the tips into distinct species.

Here, I present a combined approach of phylogenetic placement [5] and species delimitation [53, 118] to devise a measure of phylogeny-aware relative α-diversity. Our SCRAPP (Species Counting on Reference trees viA Phylogenetic Placement) tool, quantifies diversity by initially grouping QSs by the branch on the reference tree to which they most likely belong with respect to their phylogenetic likelihood score. Subsequently, for each such group of QSs placed onto the same reference branch, we infer a separate phylogenetic tree comprising the QS of that group, optionally including an outgroup sequence from the reference tree. We call such a tree a *Branch Query Phylogeny* (BQP). Generating such BQPs constitutes a major part of the analysis (in terms of run time), and is a feature that has, thus far, been missing for post analyzing phylogenetic placements. Therefore, we include the set of inferred BQPs in the SCRAPP output.

Finally, we apply mPTP [53] to the BQP to obtain a species count for the corresponding reference branch. The output of SCRAPP is a branch-annotated reference tree that depicts how species diversity is distributed over the reference tree for a given sample.

SCRAPP is implemented in Python and relies on mpi4py [18–20] for the respective parallel implementation targeting both, shared, and distributed memory systems.

Some of the concepts used in SCRAPP are based on the difficult to use EPA-PTP tool, an early attempt to integrate phylogenetic placement with species delimitation [118]. The goal of SCRAPP is thus to quantify diversity for each branch of the reference tree individually and to improve usability. Further, in contrast to SCRAPP, EPA-PTP used phylogenetic placement to calculate a single, overall species delimitation over the entire reference tree extended by all BQPs simultaneously.

5.2 Description

An overview of the SCRAPP tool is provided in Figure 5.1. SCRAPP takes as input a jplace [76] file (see Section 2.4.2) containing the placements and the associated reference tree, as well as the corresponding MSA of the QSs. From this, we generate per-branch QS MSAs. These include all QSs whose most likely placement was on the given branch. However, we remove those placements from this set, whose best LWR is below a given threshold (`--min-weight`, default 0.5).

If desired, an outgroup from a user specified reference MSA is included in each branch QS MSA such that the corresponding BQP that is produced in the subsequent step can be rooted at this outgroup. We automatically choose the outgroup sequence for a given BQP as the leaf sequence in the reference tree that is most distant from the given branch. Note that, mPTP species delimitation operates on rooted phylogenies. Thus, specifying an outgroup can be beneficial if a more reliable root for the BQP is desired. If a root is not provided, mPTP will automatically root the BQP on its longest branch.

If the number of QSs in a given branch QS MSA exceeds a user-specified maximum (500 by default), we reduce the number of QS to that maximum using a two-stage clustering method (described in Section 5.2.2). This option is necessary to maintain BQP tree inference times within reasonable limits. On empirical datasets, specific reference branches can contain more than $100,000$ QSs, hence yielding the inference of a BQP computationally challenging. We strongly recommend that the QSs are dereplicated or even OTU-clustered (see Section 2.1) prior to executing SCRAPP, or, for that matter, prior to performing placement.

Once the query MSAs have been generated for all branches of the reference tree, we infer a phylogeny for each of them separately using RAxML-NG [60]. As there may be a large number of trees (potentially as many as there are branches in the reference tree) with highly variable sizes to infer, we use ParGenes [82] to orchestrate this tree inference process in a parallel, scalable, and efficient way. The inferred BQPs are then processed using mPTP to obtain a species delimitation, and corresponding species count. The information produced by each mPTP run is tracked for each branch that contains QSs in the reference tree.

We note that the species delimitation itself constitutes a clustering of the QSs, which may represent a desirable output to the user. Particularly, if the original placement input data has *not* already been OTU clustered, the combination of placement with species delimitation can be regarded as phylogeny-aware OTU clustering. However, here we focus on the diversity metric aspects of SCRAPP, and consider further potential applications as future work.

5.2.1 Variance and Output

The set of inferred BQPs can optionally be expanded to calculate species count variance. Two options are available to calculate this variance: *rootings*, generates a tree set on each BQP by enumerating all possible rootings for the unrooted BQP, or *bootstraps*, generates a given number (20 by default) of bootstrapped branch QS MSAs and then re-optimizes the branch lengths on the original BQP for each of the bootstrapped branch QS MSAs. When using these expanded BQP sets, we calculate the final species count as median over all per-branch species delimitation results (i.e., over all rootings or all bootstrap replicates).

The *rootings* and *bootstraps* options constitute two of the three principal operating modes of SCRAPP. The third operating mode, the *outgroup* mode, offers the rooting of the BQPs via inclusion of a reference outgroup (as described above).

Finally, SCRAPP generates two types of output files. Firstly, it outputs an annotated version of the reference tree in the extended NEWICK format, that can easily be visualized by a number of tree viewers (e.g., iTOL [65] or Dendroscope [46]). This is useful for obtaining a high level overview of the diversity, as diversity is represented by just one species count value per reference tree branch.

To allow users to explore the results more thoroughly, for example, by inspecting the variance of the median species count, we also produce a comprehensive output file in a `json`-based file format that is analogous to the JPLACE format [76]. This format, called Tree Edge Annotations (TEA), contains the reference tree with enumerated branches, as specified in JPLACE, followed by annotation information. The annotation comprises a list of per-branch values. In SCRAPP this annotation includes the median species count, and the species count variance, among others. A full specification and an example of the TEA format is provided in Appendix A.1, as well as online at `https://github.com/pbdas/scrapp/wiki/TEA-format`.

5.2.2 Placement Space Clustering

In general, phylogenetic diversity metrics face a fundamental scalability issue, as they rely on a phylogeny inferred on the QSs. With increasing sequencing volumes, inferring such phylogenies under maximum likelihood becomes prohibitively expensive. Moreover, as metabarcoding/metagenomic samples typically comprise short sequences, the available signal for reliable tree inference on thousands or tens of thousands of taxa is mostly insufficient [10]. This was the key motivation for the development of phylogenetic placement methods as a scalable and more reliable alternative.

Nonetheless, SCRAPP faces this same computational issue again at a different level as a reference branch may contain tens of thousands of QSs. To alleviate this, we have implemented a two-stage clustering method called *Placement Space Clustering* (PSC) in SCRAPP. PSC leverages the fact that the pendant and distal/proximal lengths of a placement can be interpreted as the coordinates in a two-dimensional space (hereafter called *placement space*). When using PSC, we map the set of placements on a branch into placement space and then perform a standard k-means clustering on the respective datapoints. Subsequently, we select a small number x of placements from each cluster as representatives of that cluster, such that $k * x$ equals the maximum desired number (as specified by the user) of sequences per branch QS MSA. More specifically, we select the top $x := 10$ sequences by number of informative (non-gap or non-undetermined) sites, thereby maximizing the potential phylogenetic signal for the subsequent tree inference.

5.3 Evaluation

We assessed the accuracy of SCRAPP using both, simulated, and empirical data.

5.3.1 Simulated Data

We generated *true* species trees using the MSPRIME ([55], v0.7.3) coalescent simulator. We then used SEQ-GEN ([95], v1.3.4) to generate MSAs on those trees. We

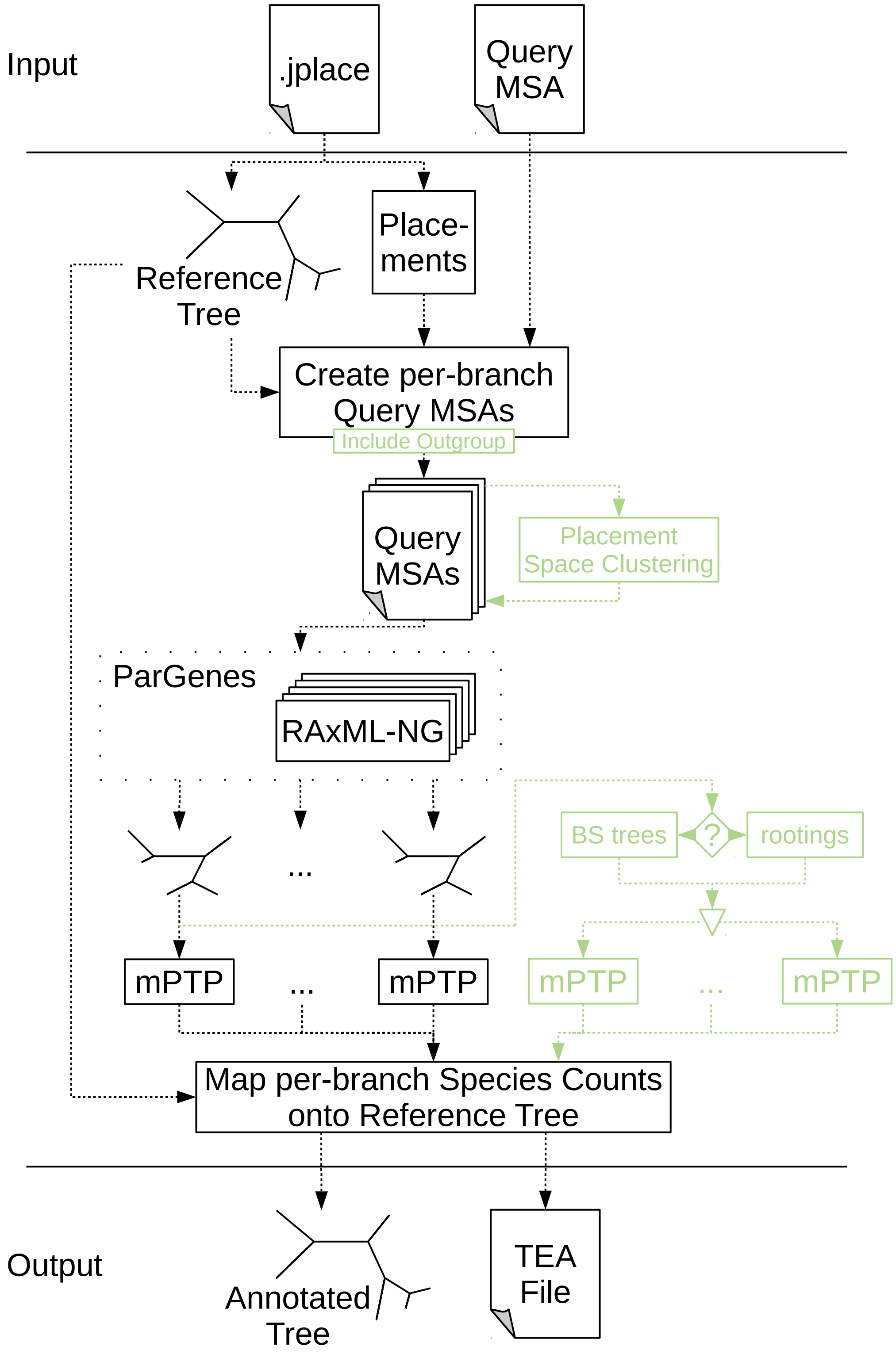


Figure 5.1: Overview of the major components of the SCRAPP pipeline. In green, we highlight optional components (inclusion of reference sequences for BQPs outgroup rooting, *placement space clustering* for limiting computational effort, bootstrapping or re-rooting for delimitation variance assessment).

Parameter	Value
seq_length	4000
prune_fract	0.25
pop_size	1e6
species	400
mut_rate	1e-8
sample_size	50

Table 5.1: Default simulation parameters as used in msprime and Seq-Gen.

generated the trees and MSAs such as to evaluate SCRAPP under a broad range and combination of simulation parameters. The parameters include: the number of starting populations (which we call *species*) (range: $200 - 600$), the sequence length (range: $1000-4000$), the number of individuals per population (called *sample size* by MSPRIME) (range: $20 - 80$), the overall MSPRIME population size (range: $10^5 - 10^7$), and the mutation rate (range: $10^{-7} - 10^{-8}$). In particular, we investigated the influence on each parameter individually while keeping the remaining parameters fixed to a set of default values, specified in Table 5.1.

From each simulated *true* tree and MSA, we first pruned a set of QSs by removing all but one individual from each starting population. To account for incomplete reference data with lower taxon sampling density, we subsequently further pruned a given fraction (denoted as `prune_fract`, $[0.1, 0.4]$) of leaves uniformly at random from the trees. We then labeled the branches of the remaining reference tree by the number of query species (here assumed to be equal to the number of populations) whose *true* location is on that given branch.

We then used EPA-NG to place the query data back onto the tree. Next, we evaluated these phylogenetic placement results using SCRAPP, yielding an annotated NEWICK tree. Finally, we compare the reference tree with the inferred species count annotations (hereafter SCRAPP-tree) to the reference tree with the *true* species count annotations.

All scripts used for generating the simulated data can be found in the SCRAPP repository: `https://github.com/Pbdas/scrapp/tree/master/simtest`

5.3.2 Empirical Data

In addition to the tests on simulated data, we replicated part of the evaluation of [78]. McCoy and Matsen [78] evaluated different diversity metrics by the quality of their fit with clinical metadata, which are known from literature to correlate with α-diversity.

We chose to replicate and extend the evaluation of the Bacterial Vaginosis dataset [105] (hereafter called **BV**), as we already had access to the data (see Section 3.4.1.2) and the specific dataset has been particularly well studied [15]. The clinical metadata included in the BV dataset are based on two methodologies indicating the presence or absence of bacterial vaginosis for a patient: Amsel's criteria [2], and the Nugent score [87]. Amsel's criteria comprise four distinct criteria, three of which need to be fulfilled to positively diagnose a patient with bacterial vaginosis. In the BV dataset, ”Amsel” is provided as a binary value indicating whether a patient was diagnosed as being positive or negative. The Nugent score is a composite score based on gram-stained vaginal swabs. The score ranges from negative $(0-3)$, through intermediate $(4-6)$, to positive $(7-10)$.

Unfortunately, due to patient data protection issues, we cannot make the BV dataset publicly available. Please refer to [105] and [15] for an exhaustive exploration of the BV dataset, and a detailed description of the phylogenetic placement of the per sample data, respectively.

Firstly, to obtain the OTU-derived diversity measures used in the evaluation of [78], we performed OTU clustering using SWARM ([71, 72], v3.0, `-d 1 -f`), and utilizing VSEARCH ([97], v2.6.2) for dereplication and filtering. We further analyzed the resulting OTU table using the R package PHYLOSEQ ([79], v1.22.3, function `estimate_richness`) to obtain the Shannon [100], Simpson [101], ACE [12], and Chao1 [14] indices.

Secondly, to assess the placement based methods, we computed a phylogenetic placement of the sample data. Note that, we did not use the reference tree given in the original publication [105], as we found that the inclusion of multiple strains of the same bacterial species can produce a very flat likelihood distribution for potential placements of a single QS across individual branches of the tree [15]. Therefore, we used an appropriately modified version of the reference tree, as shown in Figure S1 in [15]. This modified reference tree only retains consensus sequences of all reference strains, such that only one taxon per species remains. The modified reference tree comprises 198 taxa.

Based on this placement data, we obtained the measures outlined in [78], on a per-sample basis, using the GUPPY command `fpd` ([75, 78], v1.1.alpha19-0-g807f6f3). Note that, we chose to omit the `guppy fpd --include-pendant` option to avoid overestimating diversity. The placement process does not resolve relationships between individual QS. Thus, the distance of each individual QS to the RT is denoted by a so-called pendant length. Consequently, if two or more QS are phylogenetically close to each other, but relatively distant to the RT, the common distance to the RT may be counted once per QS in the PD calculation. This can lead to potential overestimation.

Lastly, we applied SCRAPP to the placement data, running the analysis in the `bootstrap` operating mode, and limiting the maximum number of taxa per BQP to $1,000$. This again yields a SCRAPP-tree (see Section 5.3.1).

In the interest of comparability, we chose to re-implement the BWPD function using the GENESIS library [17], in a way such that it can be applied to SCRAPP-trees. The BWPD relies on a one-parameter function family interpolating between classical PD and an abundance weighted version of the PD. McCoy and Matsen [78] chose to implement and evaluate the BWPD on placement results, which consist of precise locations and branch lengths of queries on the reference tree. In contrast to this, SCRAPP-trees comprise assignments of absolute numbers (species counts) to branches of the tree, without any more specific branch length information. To remedy this discrepancy, when calculating the BWPD on a SCRAPP-tree, we treat the species count of a branch as if it were a single placement, located at the middle of said branch, without a pendant length.

All data handling and analysis scripts used in the empirical data evaluation can be accessed online at `https://github.com/Pbdas/diversity-compare`.

5.3.3 Clustering and Showcase

Finally, we include a showcase test and analysis for two additional empirical datasets.

In one set of experiments, we use the neotrop dataset (see Section 3.4.1.1 and Section 4.5) to evaluate our PSC methodology (Section 5.2.2). This data is particularly challenging for phylogenetic placement, as the available reference data is too sparse to cover the diversity that was sampled. For our purposes, we randomly selected small subsets of $1,000$ QSs from this dataset and placed them on the reference tree described in [70] (512 reference taxa). We then executed SCRAPP for distinct settings of `--cluster-above`, thereby limiting the maximum number of sequences per branch used in the subsequent BQP tree searches. As the randomly selected $1,000$ QSs subsets produced a maximum of 298 QSs placements per branch, a threshold value of 300 was selected as the benchmark against which all other runs are compared to, since this constitutes the "no clustering" case. For each clustering threshold setting and each operating mode we performed 5 independent runs of the same data in order to quantify the variability introduced by the randomization component in the clustering algorithm. Scripts and data used in this experiment can be found in the repository at `https://github.com/Pbdas/scrapp/tree/master/test`.

In a second set of experiments, we used a large dataset from the UniEuk project [9] as a showcase for deploying SCRAPP on a standard parallel compute cluster. For this test, we used a phylogenetic placement of $585,050$ QSs, which resulted from an OTU clustering of roughly 300 million sequences, on a reference tree comprising 800 taxa. From this, SCRAPP identified $254,103$ QSs as being placed with a LWR above the default 0.5 threshold (see Section 5.2). We limited the maximum number of sequences per branch to 800, and used the *bootstrap* operating mode, generating 100 bootstrap trees per BQP. This resulted in the inference of $1,070$ BQPs, the largest tree containing 797 taxa. SCRAPP further evaluated each of them via 100 distinct bootstrap MSAs.

5.3.4 Error Metrics

For the simulated data, we calculate two distinct accuracy values. The first is the absolute difference between the inferred and the true species count on a branch in the reference tree. This absolute difference is then averaged over all branches of the reference tree that have non-zero values in either tree. We denote this accuracy metric as Mean Absolute per-branch Error (hereafter **MAE**).

More formally, let S and T be two trees with identical topologies and branch-associated values s_i and t_i for a given branch index i, respectively. T denotes the *true* tree, while S denotes the *SCRAPP-tree* (Section 5.3.1). Let B be the set of branch indices for which either S or T have non-zero values. We can now write the MAE as

$$MAE = \frac{\sum_{i \in B} |t_i - s_i|}{|B|} \tag{5.1}$$

Our second accuracy metric is based on normalized per-branch species counts. For a given branch with index k, we calculate this normalized count based on a absolute species count x_k as

$$x_k^{norm} = \frac{x_k}{\sum_{i \in B} x_i} \tag{5.2}$$

where k denotes the index of a given branch, and B is as defined above.

Further, instead of calculating the absolute difference, we calculate the relative difference:

$$rel(t_k, s_k) = \frac{|t_k - s_k|}{(|t_k| + |s_k|)/2} \tag{5.3}$$

Again, s_k and t_k are the values for a given branch with index k, of two given trees S and T as defined above. Note that here we compute the relative difference by normalizing via the arithmetic mean of s_k and t_k. This ensures that the metric produces well-defined values in cases where $t_k = 0$. The term $rel(t_k, s_k)$ is also known as the *Relative Percent Difference.* Note that $rel(t_k, s_k)$ is bounded between 0 and 2.

Finally, we again calculate the average over all relative normalized species count differences across all branches that have non-zero value, resulting in the Normalized Mean Relative per-branch Error (**NMRE**).

$$NMRE = \frac{\sum_{i \in B} rel(t_i^{norm}, d_i^{norm})}{|B|} \tag{5.4}$$

The MAE captures the deviation of the SCRAPP-based species count from the true species count. The NMRE quantifies the difference between the true and the inferred diversity distribution over the tree.

The accuracy of the methods used in the empirical evaluation is calculated as in [78]. Here, the primary approach is to assess the correlation of the diversity measures with the clinical metadata (see Section 5.3.2). To quantify the correlation with the diagnosis based on Amsel's criteria, we first use the `glm` function in R [93] to fit a generalized linear model to the data. We then calculate the *Amsel accuracy* as the proportion of correctly identified datapoints via a leave-one-out cross-validation.

Just as McCoy and Matsen we perform independent 2-group t-tests between the Amsel diagnosis and the investigated metrics, using the `t.test` R function. The resulting p-value is presented here as the *Amsel p-value*. For comparing against the Nugent score, we fit the diversity measures using a linear regression model, via the `lm` function in R. The function also returns the R^2 of the fit, which is the proportion of the variation that is explained by the model.

5.4 Results

5.4.1 Simulated Data

We performed a total of 270 independent simulation runs, covering all simulation space dimensions, all of their combinations with the SCRAPP operating modes, and repeating runs for each individual configuration 5 times. We show high level results across all runs, and stratified by operating mode, in Table 5.2. We observe a mean NMRE of 0.508 over all experiments. When stratified by the different operating modes, we observe the lowest overall NMRE for the rootings mode (0.471 mean NMRE).

To summarize our exploration of the impact of different simulation parameters, we find that result accuracy in terms of mean NMRE increases with increasing overall population size, sample size (number of individuals per population), and sequence length, as well as decreasing `prune_fract` (Section 5.3.1). While less pronounced, there is a trend for the NMRE to improve with increasing total tree size which may be attributed to improved taxon sampling density [44]. This can be observed in Figure 5.2, which shows data for those simulation runs where we only varied the total number of starting populations (here called species).

Further, we observe a general trend for overestimating the species count across all simulation parameters, as indicated by the high MAE values (Table 5.2). Specifically, the *rootings* mode appears to overestimate the species count the most, while the *bootstrap* mode performs best in this regard. We therefore recommend that users deploy the *bootstrap* mode when the goal is to obtain as accurate as possible estimates of the absolute species counts. However if, one desires to obtain the most accurate *relative* distribution of species counts over the tree, we recommend the *rootings* mode, as it consistently showed the lowest NMRE.

Further, we observe a divergent relationship between the MAE and NMRE scores for the population size, sample size, sequence length, and species parameters. For the first three parameters this is due to a decrease in the fraction of mPTP delimitation results that yield the null model. Note that, when mPTP yields the null model, it cannot distinguish between a delimitation into one or n species, where n is the number of leaves in the tree (the BQP in our case). As the fraction of null model results decreases, the *relative* mPTP accuracy increases, yielding more accurate results with respect to the NMRE metric. At the same time, this also increases the average *absolute* species count, and thereby generates higher MAE values.

	NMRE	σ^2	CV	MAE	σ^2	CV
bootstrap	0.518	0.013	0.221	5.69	2.99	0.304
outgroup	0.535	0.017	0.241	7.71	5.48	0.304
rootings	0.471	0.019	0.289	8.15	5.76	0.294
across-all	0.508	0.016	0.254	7.18	5.86	0.337

Table 5.2: We report the mean NMRE and mean MAE, across all runs (last row) and across all runs of the specific operating modes (middle rows). σ^2 denotes the variance of the given means, and CV denotes the coefficient of variation. As a reference, the mean variance among simulation replicates (identical parameter configurations but different random number seeds) was 1×10^{-3} and 3×10^{-2} for the NMRE and the MAE, respectively.

For the species parameter (the number of populations in the coalescent simulation), the negative relationship between MAE and NMRE is less pronounced. It can be explained by the fact that an increase in the species parameter yields a larger simulated tree, but, at the same time, unlike the other three parameters does not increase the phylogenetic signal for reconstructing the BQPs. As a consequence, the fraction of mPTP null model results remains constant over the species parameter range. Further, as phylogenetic placement is not exact, a larger reference tree with an increased number of branches also implies a larger potential for misplacing QSs. This increases the chance of reference tree branches for which the true number of placed QSs should be 0, to contain misplaced QSs, and thereby yield a minimum species count of 1. As the delimited species to which a misplaced QSs belongs may already be accounted for on another branch, the total species count increases. As a result, the MAE will increase as well.

Finally, Figure 5.3 shows the full exploration of varying individual simulation parameters, using the MAE and NMRE metrics, as well as the fraction of null model results.

5.4.2 Empirical Data

The most important results of our evaluation based on the BV dataset are shown in Table 5.3. We were able to closely replicate the results of [78] (their Table 2), although we generally observe higher values for the Amsel accuracy and Nugent R^2. The exception to this are the R^2 values obtained from the ACE and Chao1 measures, that substantially underperform compared to the results of [78]. As ACE and Chao1 are the only tested OTU-based metrics that specifically assign a higher weight to rare observations (i.e., OTUs observed only once or twice), we speculate that our data handling approach has reduced the number of rare OTUs. However, our results confirm the general trend that phylogenetic methods outperform OTU methods with respect to the aforementioned metrics.

Further, we observe a high level of agreement between metrics directly calculated from phylogenetic placement results, and metrics derived from SCRAPP results.

Measure	Amsel accuracy	Nugent R^2	Amsel p-value	Mean rank
bwpd_0.25.guppy	0.877	0.777	2.01e-35	2.33
bwpd_0.25.scrapp	0.874	0.785	4.02e-34	2.33
phylo_entropy.scrapp	0.873	0.782	4.70e-34	4.00
bwpd_0.5.guppy	0.873	0.757	1.03e-34	4.33
bwpd_0.5.scrapp	0.872	0.786	1.37e-33	4.67
bwpd_0.scrapp	0.873	0.767	1.49e-33	5.67
quadratic.scrapp	0.869	0.779	1.60e-32	8.33
bwpd_0.75.guppy	0.870	0.725	2.46e-33	9.00
bwpd_0.75.scrapp	0.868	0.772	5.10e-32	10.33
quadratic.guppy	0.869	0.718	7.97e-33	10.33
bwpd_0.guppy	0.872	0.713	2.00e-31	11.17
unrooted_pd.guppy	0.872	0.713	2.00e-31	11.17
phylo_entropy.guppy	0.869	0.716	1.43e-32	11.33
rooted_pd.guppy	0.871	0.701	5.73e-31	13.00
bwpd_1.scrapp	0.861	0.741	1.30e-29	13.67
bwpd_1.guppy	0.867	0.691	8.36e-32	14.33
Shannon	0.826	0.387	5.03e-18	17.00
ACE	0.822	0.242	1.41e-10	18.00
Chao1	0.810	0.213	6.35e-09	19.00
Simpson	0.788	0.168	3.61e-08	20.00

Table 5.3: Correlation and predictive power of SCRAPP in comparison with analogous approaches on the Bacterial Vaginosis data. Amsel accuracy, Nugent R^2, Amsel p-value, and mean rank are calculated exactly as in [78]. Rows are sorted by mean rank. Measures suffixed by `.guppy` are calculated using guppy fpd [75], whereas measures suffixed by `.scrapp` were calculated based on results produced by SCRAPP. Shannon, ACE, Chao1, and Simpson values were calculated based on an OTU clustering of the same data (see Section 5.3.2).

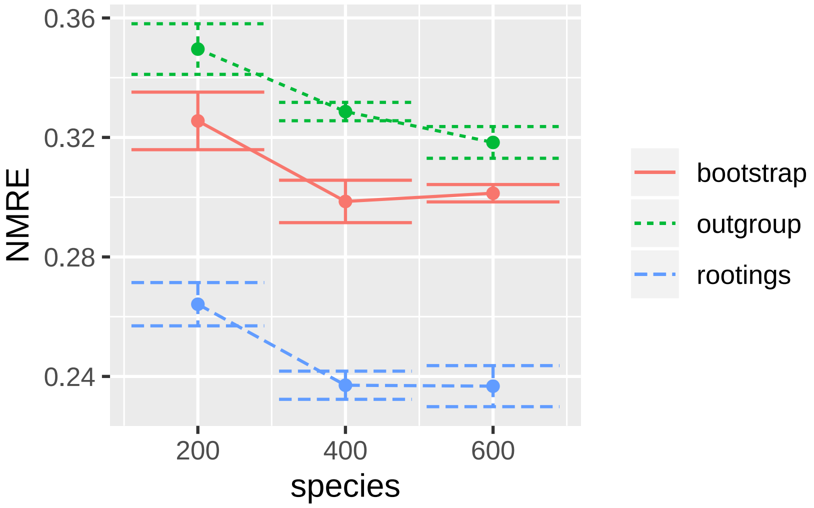


Figure 5.2: NMRE (Equation 5.4) for several runs on simulated datasets where we only varied the total species count of the ”true” tree (the number of individual populations). Error bars denote the first standard deviation from the mean. Data was stratified by the three different operating modes of SCRAPP (see Section 5.2).

5.4.3 Clustering and Showcase

The results of evaluating PSC with varying clustering thresholds are shown in Figure 5.4. Both, the *bootstrap*, and *rootings* operating modes produced stable results, that are qualitatively similar to the tests on simulated data. However, the *outgroup* operating mode proved to be highly sensitive to the PSC, yielding high species count deviations starting at a clustering threshold of 200 (a data reduction of ~33%). Due to the known issues with the eukaryotic soil reference dataset at hand we hypothesize that the cause for this behavior is the sparse taxon sampling in the reference MSA. This incomplete taxon sampling induces a high branch length value between the ingroup QSs and the outgroup, as SCRAPP selects the phylogenetically most distant taxon in the reference tree as outgroup.

As a final showcase for the scalability of SCRAPP on distributed computing clusters, we analyzed a large dataset of $585,050$ QSs placed on a 800 taxon reference tree, utilizing 50 compute nodes comprising a total of 800 cores. Running this analysis involved handling about 1 million files, of which approximately $8,500$ had to be retained as intermediate results for further downstream analysis. The total runtime under this setting was 26.5 hours. We regard this as being fast, since the overall computational task includes hundreds of tree inferences with up to 797 taxa, and handling approximately 1 million intermediate files.

5.5 Summary

In this chapter I presented SCRAPP, a highly scalable and fully automated pipeline for diversity quantification of phylogenetic placement data. The primary goal of

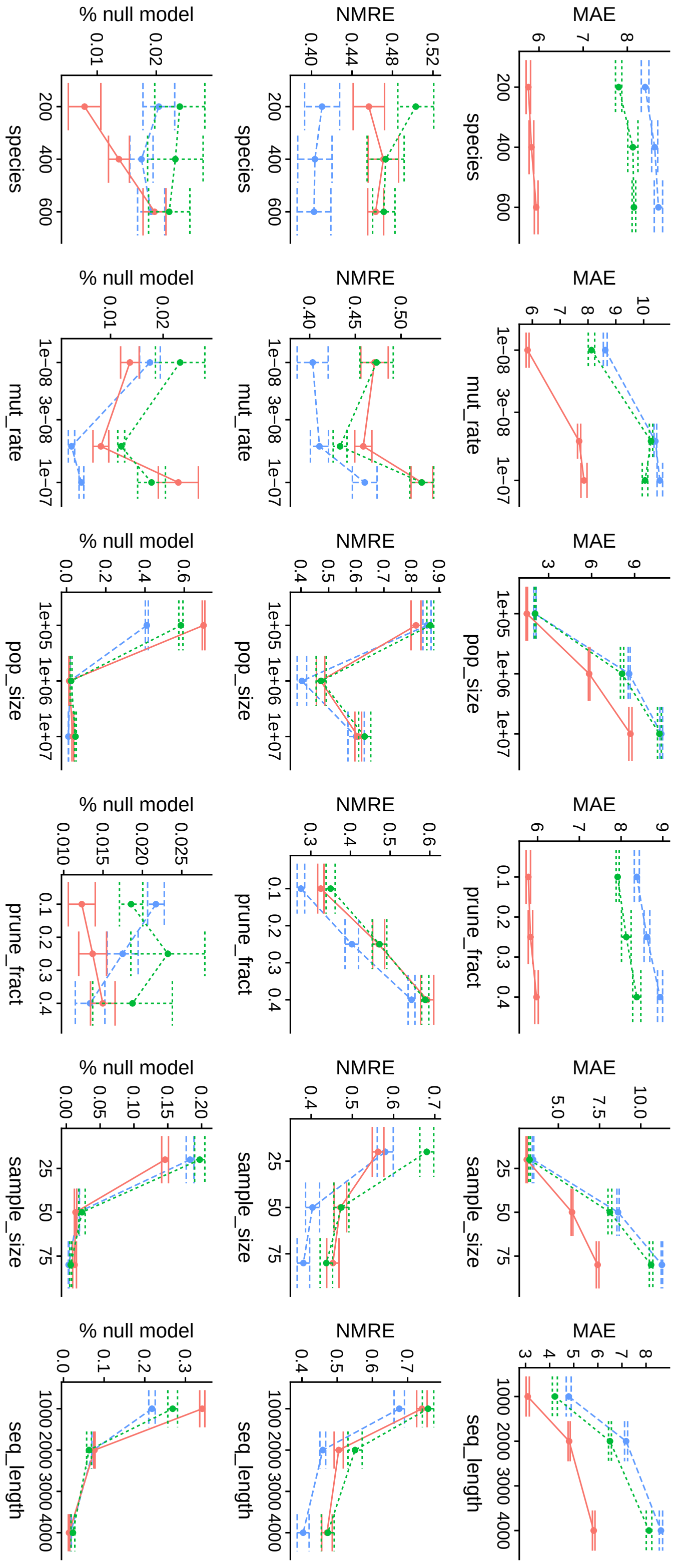


Figure 5.3: Full exploration of varying individual simulation parameters, using the MAE and NMRE metrics, as well as the fraction of null model results.

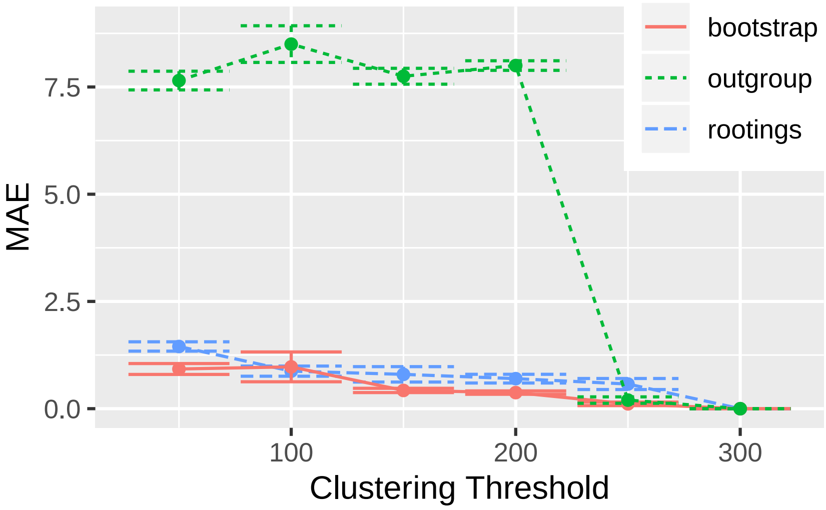


Figure 5.4: MAE (Equation 5.1) of multiple runs of SCRAPP, using different thresholds down to which *Placement Space Clustering* (PSC) reduces the maximum per-branch data volume. The MAE is calculated with reference to the case of the threshold being 300, as 300 was the maximum number of QSs that were placed per-branch. The underlying query and reference data are from the neotrop dataset (Section 5.3.3, [70]).

SCRAPP is to quantify the diversity distribution of a given sample on the reference tree. It does so by grouping QSs by their most probable reference tree branches performing a novel type of dimensionality reduction called *Placement Space Clustering* (PSC) if the size of the group exceeds a limit set by the user. Subsequently, we use the parallel multiple tree search tool ParGenes to infer a ML phylogeny on this group of sequences. The result of each of these tree inferences is then passed to the molecular species delimitation tool mPTP, which outputs a clustering of the leaves of the tree into putative species. Finally, SCRAPP synthesizes the information generated by mPTP and annotates the original reference tree, generating a species count distribution on the tree.

The evaluation of SCRAPP shows that, on simulated datasets, the approach yields phylogenetic diversity distributions with a comparatively low per-branch error rate. On empirical data, we show that α-diversity metrics calculated on the results obtained from SCRAPP rank among the most accurate of those tested in terms of predictive power, and correlation with clinical metadata.

Finally, our experiments show that using PSC, our novel clustering method for placements, SCRAPP is able to to efficiently perform dimensionality reduction of the branch QSs MSA input data. This enables SCRAPP to tackle the scalability challenge induced by the overabundance of environmental sequence data.

Like the other tools I have developed in the course of this thesis, SCRAPP is available as a free open source software at `https://github.com/Pbdas/scrapp`.

6. Conclusion and Outlook

6.1 Summary

In this thesis I have described my work on high performance computational software for *Maximum Likelihood* (ML) phylogenetic placement, and its application to a novel method for quantifying the species diversity of a given sample. ML phylogenetic placement is a state-of-the-art method for determining the evolutionary context of a genetic sequence with unknown provenance. ML-based phylogenetic methods use statistical models of evolution to compute the phylogenetic likelihood which constitutes the most accurate method for evolutionary tree inference.

In Chapter 3 I described EPA-NG, which combines the features of two previously published ML phylogenetic placement tools and improves single core execution times by up to 30 times. Furthermore, EPA-NG is the first ML phylogenetic placement software that can be executed and scales well on distributed memory systems. These performance improvements enable EPA-NG to perform placement for datasets with an extremely high number of query sequences, as shown in a test involving the placement of 1 billion sequences, using $2,048$ cores.

In Chapter 4 I presented the most recentimprovement of EPA-NG, which allows the user to specify an upper bound for the main memory usage. When the memory requirements of an analysis necessitate it, EPA-NG deploys an active memory management strategy that involves an asynchronous recomputation of vital data structures. Using this mode enables the user to perform ML placement analyses on larger reference trees than ever before, at the cost of low to moderate increases in execution time. This is especially relevant for users that only have access to hardware with limited memory resources. This new mode also has implications on the parallelization scheme employed in EPA-NG, which we explore in the evaluation of the program.

A common use-case of phylogenetic placement is to characterize the composition of a microbial community, using genetic sequencing data obtained from an environmental sample. Such studies typically also aim to quantify the diversity of a sample, and by extension the diversity of the environment the sample was taken from. Moreover, current methods for environmental sample diversity quantification often rely on phylogenetic information, such as the popular *Phylogenetic Diversity* (PD) metric. Thus, it is beneficial to use results from phylogenetic placement to quantify the diversity of a sample.

In Chapter 5 I described SCRAPP, a software that takes the results of phylogenetic placement as input and calculates a distribution of species counts on the reference tree, which can then be interpreted as a diversity measure. SCRAPP achieves this by combining state-of-the-art tools for phylogenetic inference and molecular species delimitation. Furthermore, it employs a novel approach for clustering placement results, called *Placement Space Clustering* (PSC) to allow the user to constrain computational effort.

Outside of my main stream of work, I have collaborated on additional projects, which I have outlined in Section 1.2. These projects involved the application of phylogenetic placement methods to empirical data [30, 50, 83], the further development of methods and tools for post-analyzing phylogenetic placement data (specifically involving the GENESIS library and the GAPPA command line tool) [16, 17, 50], as well as the application of phylogenetic inference methods to empirical data [30, 83].

6.2 Future Work

This final section covers possible future directions of research in the areas of phylogenetic placement, and placement post-analysis methods.

6.2.1 Streamlining the creation of Reference Trees

Unfortunately, phylogenetic placement has a comparatively high entry barrier for new users, as often the first step is to assemble a reference dataset and infer a reference tree. This step is far from trivial, as it requires highly domain-specific knowledge. The user has to assemble a set of reference sequences that are suitable for their studied environment, ensure the validity and quality of the sequences, apply phylogenetic inference software, and be able to assess the quality of the resulting tree. Moreover, the choices made in the construction of a reference tree have direct impact on the process and outcome of ML phylogenetic placement. For example, a tree containing too many taxa can substantially delay the response time of an analysis due to increased computational effort. Similarly, if a reference tree contains highly similar sequences (perhaps multiple strains of a species), placement of a short sequence may result in multiple placement locations with a low LWR, complicating the interpretation of the results [105]. The consequence is that users often go through multiple iterations of building a reference tree, placing their query sequences, evaluating the results, and refining the reference tree based on the outcome.

There are several possible directions of work that could help to alleviate this situation. Firstly, there currently is a lack of tools to assist the tasks of creating a reference tree. For example, the development of specific objective quality criteria for such trees could allow the user to infer a suitable tree more quickly and consistently.

Secondly, objective quality criteria for reference trees could also form the basis of an automated pipeline that performs a more specialized tree inference procedure according to the requirements of the user, perhaps including constraints regarding taxonomic coverage, or desired taxonomic specificity. Some work in automating the procedure for generating a reference tree has been conducted, specifically for generating a reduced set of consensus sequences from a large reference database [16]. Additional work is required to fully integrate such an approach with existing tree inference and phylogenetic placement tools, which would enable an automated iterative procedure that improves the reference tree based on preliminary placement results.

Thirdly, making available public, curated reference trees that target specific environments studied by many users should be encouraged. There already exist sequence databases for specific research communities to allow for analysis of data acquired from independent research groups [25, 73, 91]. Typically, these databases are the source of reference sequences and thus would be the ideal platform for sharing and curating reference trees as well. While some databases projects have made some effort in this direction [9], and some users have publicly shared their trees in alternative ways [98], this is not a standard procedure yet. Additionally, the use of established reference trees would allow for substantially improved comparability between individual studies.

6.2.2 Algorithmic improvements of existing methods

Despite recent advances in the field, ML methods have remained the "gold standard" in terms of accuracy for phylogenetic placement [4, 11, 66, 67]. However, novel approaches have improved on execution times and memory footprints, sometimes by orders of magnitude [4, 11]. While it is likely that ML methods, such as EPA-NG, have reached a point of diminishing returns for performance improvement on most types of processing environments, further work is required to enable the use of co-processors such as GPGPUs, for instance. The work presented in Chapter 4 is an important advancement in this direction, as the memory constraints of current GPGPUs present a primary obstacle for their use for ML phylogenetic placement.

6.2.3 Expanding the applicability of ML Phylogenetic Placement

One limitation of ML placement arises from the possible and rather typical incongruence between a species tree and a set of constituent gene trees. A species tree is typically constructed from multi-locus data (i.e., multiple genes) rather than simply using a tree inferred on the data from a single gene, as the evolutionary history of a gene may differ from the species tree due to gene loss, gene duplication, and horizontal gene transfer. As sequencing reads in microbial studies are often extremely short, and in the case of metabarcoding are targeting a (subregion of) a single gene, phylogenetic placement is typically applied to a specific gene tree only. Thus, accounting for possible gene tree/species tree incongruence during the placement process would be beneficial and could increase the accuracy/reliability of the method. In a recent preprint, a possible solution to this problem has been proposed, however for distance-based placement methods instead of ML phylogenetic placement [51]. It is currently unclear whether a similar approach can be applied to single-gene ML placement methods. In addition, the possibility of integrating ML placement methods with ML species tree inference methods, such as SpeciesRax [84], warrants further investigation.

Recent advances in sequencing technology that allow for long-read sequencing may aid in this effort as such technologies are able to produce contiguous sequences that span multiple genes [50]. Such sequences could be used to map single reads across multiple loci, possibly enabling multi-gene placement. A related recent advance is the emergence of *Metagenome-Assembled Genome*s (MAGs) [89], which is an attempt to use sequence assembly methods to reconstruct the genome of a single species based on metagenomic data. For such MAG data, multi-gene placement also seems applicable and promising. Such novel types of data can also be expected to alter the execution time performance of existing programs, possibly warranting further algorithmic work on existing placement methods.

Finally, there is a need to devise a robust method for identifying placement results that represent novel groups of organisms. It is likely that the evaluation of novel candidate taxa will continue to be more involved, for example by applying more rigorous phylogenetic methods such as comprehensive phylogenetic inference. For phylogenetic placement results the ability to discriminate between outliers which result from upstream errors such as misalignment or chimeric sequences, and sequences that represent valid signal would be highly beneficial.

A. Supporting Information

Parts of this chapter are based on the peer-reviewed open-access publication:

> **P. Barbera**, L. Czech, S. Lutteropp, and A. Stamatakis. “SCRAPP: A tool to assess the diversity of microbial samples from phylogenetic placements.” *Mol Ecol Resour*, 2021, Volume 21, Pages 340—349

accessible online at: `https://doi.org/10.1111/1755-0998.13255`

A.1 The Tree Edge Annotations Format

A.1.1 Intention

The TEA file format aims to provide the possibility to simply and succinctly assign an arbitrary number of human and/or machine readable values to edges/branches in a given phylogenetic tree.

A.1.2 Description

TEA uses the JSON syntax.

A TEA file has two mandatory key-value pairs: the tree, and the list of per-edge annotations. The edge annotations are also nested within a list of samples. This allows the user to specify multiple, yet semantically distinct annotations for one phylogenetic tree in a single file.

A.1.2.1 "tree"

This key is followed by a single string specifying a single tree in NEWICK format. The tree must include edge IDs in curly brackets, for each edge/branch. The edge ID numbering starts from zero and enumerates the edges in post-order starting at the root of the tree. If the tree is unrooted, the root is set at the top level trifurcation of the given NEWICK string. We define the order of the post-order traversal as leftmost-subtree first, from the perspective of the NEWICK string.

A.1.2.2 "views"

This is a key-value set of individual views of the tree, where each key specifies the name, and its corresponding value is a `view` on the tree. A `view` is a container holding a collection of per-edge annotation values: the `"annotation":{...}` key-value pair.

As an example (see example below), a `view` may be the SCRAPP-based species count data from the phylogenetic placement post-analysis of a metagenomic sample.

Each `view` **must** contain an `"annotation":{...}` key-value pair.

Additionally, a `view` may contain the following optional fields: * `type`: a string describing the type of the given `view`. This may be useful for grouping similar views into one.

"annotation"

This object contains the per-edge annotation information objects. An annotation information object is not required to be complete. That is, not all edges of the tree must have an annotation and the order within this collection does not matter.

Each per-edge object is indexed by the edge number (edge ID) the annotation refers to.

The fields and structure within an annotation object are completely free and can be specified by the user.

A.1.2.3 "meta"

This is an optional (but highly recommended) field intended to contain information about the origin of the file. As such it contains an "invocation" field specifying the command line invocation and arguments that produced the file.

A.1.2.4 "version"

This is a mandatory field specifying which version of the TEA format the file complies with.

A.1.3 Example

This example shows (in redacted form) how a TEA file as produced by SCRAPP might look like. The tree is specified, along with two separate metagenomic views from which the underlying phylogentic placement information was generated. Each sample contains a list of per-edge annotations comprising the results of the species delimitation(s) with mptp. Here, “count” refers to the inferred species count.

```
{
  ”tree”: ”((A:0.2{0},B:0.09{1}):0.7{2},C:0.5{3}){4};”,

  ”views”: {
    ”leaf_emerging”: {
      ”type”: ””,
      ”annotation”: {
        0:{
          ”count_average”: 32.1,
          ”count_median”: 30.0,
          ”count_stdev”: 14.9,
          ”null_score_average”: 748.93,
          ”null_score_stdev”: 2.27e−13,
        },
        1:{”count_average”: 1, ...},
        4:{”count_average”: 2, ...},
      }
    },
    ”leaf_growth”: {
      ”type”: ””,
      ”annotation”: {
        ...
      }
    }
    ...
  },

  ”meta”: {
    ”invocation”: ”./scrapp.py ...”
  },
  ”version”: ”0.1.0”,
}
```

Bibliography

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403 – 410, 1990.

[2] R. Amsel, P. A. Totten, C. A. Spiegel, K. C. Chen, D. Eschenbach, and K. K. Holmes. Nonspecific vaginitis: Diagnostic criteria and microbial and epidemiologic associations. *The American Journal of Medicine*, 74(1):14 – 22, 1983.

[3] S. Ardui, A. Ameur, J. R. Vermeesch, and M. S. Hestand. Single molecule real-time (SMRT) sequencing comes of age: applications and utilities for medical diagnostics. *Nucleic Acids Research*, 46(5):2159–2168, 2018.

[4] M. Balaban, S. Sarmashghi, and S. Mirarab. APPLES: Scalable distance-based phylogenetic placement with or without alignments. *Systematic Biology*, 69 (3):566–578, 2020.

[5] P. Barbera, A. M. Kozlov, L. Czech, B. Morel, D. Darriba, T. Flouri, and A. Stamatakis. EPA-ng: Massively Parallel Evolutionary Placement of Genetic Sequences. *Systematic Biology*, 68(2):365–369, 2018.

[6] J. M. Barea. Future challenges and perspectives for applying microbial biotechnology in sustainable agriculture based on a better understanding of plant-microbiome interactions. *Journal of soil science and plant nutrition*, 15:261 – 282, 2015.

[7] G. M. Barker. Phylogenetic diversity: a quantitative framework for measurement of priority and achievement in biodiversity conservation. *Biological Journal of the Linnean Society*, 76(2):165–194, 2008.

[8] S. A. Berger, D. Krompass, and A. Stamatakis. Performance, accuracy, and web server for evolutionary placement of short sequence reads under maximum likelihood. *Systematic Biology*, 60(3):291–302, 2011.

[9] C. Berney, A. Ciuprina, S. Bender, J. Brodie, V. Edgcomb, E. Kim, J. Rajan, L. W. Parfrey, S. Adl, S. Audic, D. Bass, D. A. Caron, G. Cochrane, L. Czech, M. Dunthorn, S. Geisen, F. O. Glöckner, F. Mahé, C. Quast, J. Z. Kaye, A. G. B. Simpson, A. Stamatakis, J. del Campo, P. Yilmaz, and C. de Vargas. UniEuk: Time to Speak a Common Language in Protistology! *Journal of Eukaryotic Microbiology*, 64(3):407–411, 2017.

[10] O. R. Bininda-Emonds, S. Brady, J. Kim, and M. J. Sanderson. Scaling of accuracy in extremely large phylogenetic trees. In *Biocomputing 2001*, pages 547–558. World Scientific, 2000.

[11] M. Blanke and B. Morgenstern. Phylogenetic placement of short reads without sequence alignment. 2020.

[12] A. Chao and S.-M. Lee. Estimating the number of classes via sample coverage. *Journal of the American statistical Association*, 87(417):210–217, 1992.

[13] I. Cho and M. J. Blaser. The human microbiome: at the interface of health and disease. *Nature Reviews Genetics*, 13(4):260–270, 2012.

[14] R. K. Colwell and J. A. Coddington. Estimating terrestrial biodiversity through extrapolation. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 345(1311):101–118, 1994.

[15] L. Czech and A. Stamatakis. Scalable methods for analyzing and visualizing phylogenetic placement of metagenomic samples. *PLOS ONE*, 14(5):1–50, 2019.

[16] L. Czech, P. Barbera, and A. Stamatakis. Methods for automatic reference trees and multilevel phylogenetic placement. *Bioinformatics*, 35(7):1151–1158, 2018.

[17] L. Czech, P. Barbera, and A. Stamatakis. Genesis and Gappa: processing, analyzing and visualizing phylogenetic (placement) data. *Bioinformatics*, 2020. btaa070.

[18] L. D. Dalcin, R. R. Paz, P. A. Kler, and A. Cosimo. Parallel distributed computing using Python. *Advances in Water Resources*, 34(9):1124 – 1139, 2011. New Computational Methods and Software Tools.

[19] L. Dalcín, R. Paz, and M. Storti. MPI for Python. *Journal of Parallel and Distributed Computing*, 65(9):1108 – 1115, 2005.

[20] L. Dalcín, R. Paz, M. Storti, and J. D'Elía. MPI for Python: Performance improvements and MPI-2 extensions. *Journal of Parallel and Distributed Computing*, 68(5):655 – 662, 2008.

[21] D. Darriba, D. Posada, A. M. Kozlov, A. Stamatakis, B. Morel, and T. Flouri. ModelTest-NG: A new and scalable tool for the selection of DNA and protein evolutionary models. *Molecular Biology and Evolution*, 37(1):291–294, 2019.

[22] C. Darwin. Letter no. 2022, 1856. Online: `https://www.darwinproject.ac.uk/letter/DCP-LETT-2022.xml`. accessed on 23 March 2021.

[23] C. Darwin. *On the Origin of Species by Means of Natural Selection.* Murray, London, 1859.

[24] C. de Vargas, T. Pollina, S. Romac, N. Le Bescot, N. Henry, C. Berger, S. Colin, N. Haentjens, M. Carmichael, D. Le Guen, *et al.* Plankton planet:'seatizen'oceanography to assess open ocean life at the planetary scale. *bioRxiv*, 2020.

[25] T. Z. DeSantis, P. Hugenholtz, N. Larsen, M. Rojas, E. L. Brodie, K. Keller, T. Huber, D. Dalevi, P. Hu, and G. L. Andersen. Greengenes, a chimera-checked 16S rRNA gene database and workbench compatible with ARB. *Applied and environmental microbiology*, 72(7):5069–5072, 2006.

[26] S. Dodsworth. Genome skimming for next-generation biodiversity analysis. *Trends in Plant Science*, 20(9):525–527, 2015.

[27] G. M. Douglas, V. J. Maffei, J. R. Zaneveld, S. N. Yurgel, J. R. Brown, C. M. Taylor, C. Huttenhower, and M. G. I. Langille. PICRUSt2 for prediction of metagenome functions. *Nature Biotechnology*, 38(6):685–688, 2020.

[28] S. R. Eddy. Accelerated profile HMM searches. *PLoS Comput Biol*, 7(10): e1002195, 2011.

[29] R. C. Edgar. Search and clustering orders of magnitude faster than BLAST. *Bioinformatics*, 26(19):2460–2461, 2010.

[30] R. C. Edgar, J. Taylor, T. Altman, P. Barbera, D. Meleshko, V. Lin, D. Lohr, G. Novakovsky, B. Al-Shayeb, J. F. Banfield, A. Korobeynikov, R. Chikhi, and A. Babaian. Petabase-scale sequence alignment catalyses viral discovery. *bioRxiv*, 2020.

[31] A. El Kaoutari, F. Armougom, J. I. Gordon, D. Raoult, and B. Henrissat. The abundance and variety of carbohydrate-active enzymes in the human gut microbiota. *Nature Reviews Microbiology*, 11(7):497–504, 2013.

[32] S. N. Evans and F. A. Matsen. The phylogenetic Kantorovich-Rubinstein metric for environmental sequence samples. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 74(3):569–592, 2012.

[33] M. Exposito-Alonso, , H. A. Burbano, O. Bossdorf, R. Nielsen, and D. Weigel. Natural selection on the Arabidopsis thaliana genome in present and future climates. *Nature*, 573(7772):126–129, 2019.

[34] D. P. Faith. Conservation evaluation and phylogenetic diversity. *Biological Conservation*, 61(1):1 – 10, 1992.

[35] A. Feder, E. J. Nestler, and D. S. Charney. Psychobiology and molecular genetics of resilience. *Nature Reviews Neuroscience*, 10(6):446–457, 2009.

[36] J. Felsenstein. Maximum Likelihood and Minimum-Steps Methods for Estimating Evolutionary Trees from Data on Discrete Characters. *Systematic Biology*, 22(3):240–249, 1973.

[37] J. Felsenstein. The Number of Evolutionary Trees. *Systematic Biology*, 27(1): 27–33, 1978.

[38] J. Felsenstein. Evolutionary trees from DNA sequences: A maximum likelihood approach. *Journal of Molecular Evolution*, 17(6):368–376, 1981.

[39] J. Felsenstein. The Newick file format. Web page, accessed March 2021.

[40] T. Flouri, D. Darriba, A. Kozlov, M. T. Holder, B. Morel, and A. Stamatakis. libpll - The Phylogenetic Likelihood Library. Web page, accessed March 2021.

[41] T. Fujisawa and T. G. Barraclough. Delimiting Species Using Single-Locus Data and the Generalized Mixed Yule Coalescent Approach: A Revised Method and Evaluation on Simulated Data Sets. *Systematic Biology*, 62(5): 707–724, 2013.

[42] R. J. Geider, E. H. Delucia, P. G. Falkowski, A. C. Finzi, J. P. Grime, J. Grace, T. M. Kana, J. La Roche, S. P. Long, B. A. Osborne, T. Platt, I. C. Prentice, J. A. Raven, W. H. Schlesinger, V. Smetacek, V. Stuart, S. Sathyendranath, R. B. Thomas, T. C. Vogelmann, P. Williams, and F. I. Woodward. Primary productivity of planet earth: biological determinants and physical constraints in terrestrial and aquatic habitats. *Global Change Biology*, 7(8):849–882, 2001.

[43] M. Hasegawa, H. Kishino, and T.-a. Yano. Dating of the human-ape splitting by a molecular clock of mitochondrial dna. *Journal of molecular evolution*, 22 (2):160–174, 1985.

[44] T. A. Heath, S. M. Hedtke, and D. M. Hillis. Taxon sampling and the accuracy of phylogenetic analyses. *Journal of Systematics and Evolution*, 46(3):239–257, 2008.

[45] M. Helmus, T. Bland, C. Williams, and A. Ives. Phylogenetic Measures of Biodiversity. *The American Naturalist*, 169(3):E68–E83, 2007. PMID: 17230400.

[46] D. H. Huson, D. C. Richter, C. Rausch, T. Dezulian, M. Franz, and R. Rupp. Dendroscope: An interactive viewer for large phylogenetic trees. *BMC bioinformatics*, 8(1):460, 2007.

[47] H. Integrative, L. M. Proctor, H. H. Creasy, J. M. Fettweis, J. Lloyd-Price, A. Mahurkar, W. Zhou, G. A. Buck, M. P. Snyder, J. F. Strauss III, *et al.* The integrative human microbiome project. *Nature*, 569(7758):641–648, 2019.

[48] F. Izquierdo-Carrasco., J. Gagneur., and A. Stamatakis. Trading running time for memory in phylogenetic likelihood computations. In *Proceedings of the International Conference on Bioinformatics Models, Methods and Algorithms - Volume 1: BIOINFORMATICS, (BIOSTEC 2012)*, pages 86–95. INSTICC, SciTePress, 2012.

[49] M. Jain, H. E. Olsen, B. Paten, and M. Akeson. The oxford nanopore MinION: delivery of nanopore sequencing to the genomics community. *Genome Biology*, 17(1), 2016.

[50] M. Jamy, R. Foster, P. Barbera, L. Czech, A. Kozlov, A. Stamatakis, G. Bending, S. Hilton, D. Bass, and F. Burki. Long-read metabarcoding of the eukaryotic rDNA operon to phylogenetically and taxonomically resolve environmental diversity. *Molecular Ecology Resources*, 20(2):429–443, 2020.

[51] Y. Jiang, M. Balaban, Q. Zhu, and S. Mirarab. DEPP: Deep Learning Enables Extending Species Trees using Single Genes. *bioRxiv*, 2021.

[52] T. H. Jukes and C. R. Cantor. Evolution of Protein Molecules. *Mammalian Protein Metabolism*, 3(21):132, 1969.

[53] P. Kapli, S. Lutteropp, J. Zhang, K. Kobert, P. Pavlidis, A. Stamatakis, and T. Flouri. Multi-rate Poisson tree processes for single-locus species delimitation under maximum likelihood and Markov chain Monte Carlo. *Bioinformatics*, 33(11):1630–1638, 2017.

[54] E. Karsenti, S. G. Acinas, P. Bork, C. Bowler, C. De Vargas, J. Raes, M. Sullivan, D. Arendt, F. Benzoni, J.-M. Claverie, M. Follows, G. Gorsky, P. Hingamp, D. Iudicone, O. Jaillon, S. Kandels-Lewis, U. Krzic, F. Not, H. Ogata, S. Pesant, E. G. Reynaud, C. Sardet, M. E. Sieracki, S. Speich, D. Velayoudon, J. Weissenbach, P. Wincker, and the Tara Oceans Consortium. A Holistic Approach to Marine Eco-Systems Biology. *PLOS Biology*, 9 (10):1–5, 2011.

[55] J. Kelleher, A. M. Etheridge, and G. McVean. Efficient Coalescent Simulation and Genealogical Analysis for Large Sample Sizes. *PLoS Comput Biol*, 12(5): 1–22, 2016.

[56] M. Kimura. A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *Journal of molecular evolution*, 16(2):111–120, 1980.

[57] E. V. Koonin. Orthologs, paralogs, and evolutionary genomics. *Annu. Rev. Genet.*, 39:309–338, 2005.

[58] L. B. Koski and G. B. Golding. The Closest BLAST Hit Is Often Not the Nearest Neighbor. *Journal of Molecular Evolution*, 52(6):540–542, 2001.

[59] A. M. Kozlov and A. Stamatakis. Using RAxML-NG in Practice. In C. Scornavacca, F. Delsuc, and N. Galtier, editors, *Phylogenetics in the Genomic Era*, pages 1.3:1–1.3:25. No commercial publisher | Authors open access book, 2020.

[60] A. M. Kozlov, D. Darriba, T. Flouri, B. Morel, and A. Stamatakis. RAxML-NG: A fast, scalable, and user-friendly tool for maximum likelihood phylogenetic inference. *Bioinformatics*, 2019.

[61] O. Kozlov. *Models, optimizations, and tools for large-scale phylogenetic inference, handling sequence uncertainty, and taxonomic validation.* PhD thesis, KIT-Bibliothek, 2018.

[62] L. Legendre and F. Rassoulzadegan. Plankton and nutrient dynamics in marine waters. *Ophelia*, 41(1):153–172, 1995.

[63] C.-A. Leimeister, S. Sohrabi-Jahromi, and B. Morgenstern. Fast and accurate phylogeny reconstruction using filtered spaced-word matches. *Bioinformatics*, page btw776, 2017.

[64] R. Leinonen, H. Sugawara, and M. S. and. The sequence read archive. *Nucleic Acids Research*, 39(Database):D19–D21, 2010.

[65] I. Letunic and P. Bork. Interactive Tree Of Life (iTOL): an online tool for phylogenetic tree display and annotation. *Bioinformatics*, 23(1):127–128, 2006.

[66] B. Linard, K. Swenson, and F. Pardi. Rapid alignment-free phylogenetic identification of metagenomic sequences. *Bioinformatics*, 35(18):3303–3312, 2019.

[67] B. Linard, N. Romashchenko, F. Pardi, and E. Rivals. PEWO: a collection of workflows to benchmark phylogenetic placement. *Bioinformatics*, 2020. btaa657.

[68] R. Logares, T. H. Haverkamp, S. Kumar, A. Lanzén, A. J. Nederbragt, C. Quince, and H. Kauserud. Environmental microbiology through the lens of high-throughput DNA sequencing: Synopsis of current platforms and bioinformatics approaches. *Journal of Microbiological Methods*, 91(1):106–113, 2012.

[69] C. Lozupone and R. Knight. UniFrac: a New Phylogenetic Method for Comparing Microbial Communities. *Applied and Environmental Microbiology*, 71 (12):8228–8235, 2005.

[70] F. Mahé, C. de Vargas, D. Bass, L. Czech, A. Stamatakis, E. Lara, D. Singer, J. Mayor, J. Bunge, S. Sernaker, T. Siemensmeyer, I. Trautmann, S. Romac, C. Berney, A. Kozlov, E. A. D. Mitchell, C. V. W. Seppey, E. Egge, G. Lentendu, R. Wirth, G. Trueba, and M. Dunthorn. Parasites dominate hyperdiverse soil protist communities in Neotropical rainforests. *Nature Ecology & Evolution*, 1(4):0091, 2017.

[71] F. Mahé, T. Rognes, C. Quince, C. de Vargas, and M. Dunthorn. Swarm: robust and fast clustering method for amplicon-based studies. *PeerJ*, 2:e593, 2014.

[72] F. Mahé, T. Rognes, C. Quince, C. de Vargas, and M. Dunthorn. Swarm v2: highly-scalable and high-resolution amplicon clustering. *PeerJ*, 3:e1420, 2015.

[73] B. L. Maidak, J. R. Cole, T. G. Lilburn, C. T. Parker Jr, P. R. Saxman, J. M. Stredwick, G. M. Garrity, B. Li, G. J. Olsen, S. Pramanik, *et al.* The RDP

(ribosomal database project) continues. *Nucleic acids research*, 28(1):173–174, 2000.

[74] E. R. Mardis. Next-generation sequencing platforms. *Annual Review of Analytical Chemistry*, 6(1):287–303, 2013. PMID: 23560931.

[75] F. A. Matsen, R. B. Kodner, and V. E. Armbrust. pplacer: linear time maximum-likelihood and Bayesian phylogenetic placement of sequences onto a fixed reference tree. *BioMed Central Bioinformatics*, 11(1):1–16, 2010.

[76] F. A. Matsen, N. G. Hoffman, A. Gallagher, and A. Stamatakis. A Format for Phylogenetic Placements. *PLOS ONE*, 7(2):1–4, 2012.

[77] R. L. Mayden. *Species: the units of biodiversity.* Chapman and Hall Ltd, 1997.

[78] C. O. McCoy and F. A. Matsen. Abundance-weighted phylogenetic diversity measures distinguish microbial community states and are robust to sampling depth. *PeerJ*, 1:e157, 2013.

[79] P. J. McMurdie and S. Holmes. phyloseq: An R Package for Reproducible Interactive Analysis and Graphics of Microbiome Census Data. *PLOS ONE*, 8(4):1–11, 2013.

[80] H. W. Meuer, E. Strohmaier, J. Dongarra, H. Simon, and M. Meuer. Performance Development, TOP500 List. Web page, accessed March 2021.

[81] B. Q. Minh, H. A. Schmidt, O. Chernomor, D. Schrempf, M. D. Woodhams, A. von Haeseler, and R. Lanfear. IQ-TREE 2: New models and efficient methods for phylogenetic inference in the genomic era. *Molecular Biology and Evolution*, 37(5):1530–1534, 2020.

[82] B. Morel, A. M. Kozlov, and A. Stamatakis. ParGenes: a tool for massively parallel model selection and phylogenetic tree inference on thousands of genes. *Bioinformatics*, 35(10):1771–1773, 2018.

[83] B. Morel, P. Barbera, L. Czech, B. Bettisworth, L. Hübner, S. Lutteropp, D. Serdari, E.-G. Kostaki, I. Mamais, A. M. Kozlov, P. Pavlidis, D. Paraskevis, and A. Stamatakis. Phylogenetic Analysis of SARS-CoV-2 Data Is Difficult. *Molecular Biology and Evolution*, 2020. msaa314.

[84] B. Morel, P. Schade, S. Lutteropp, T. A. Williams, G. J. Szöllősi, and A. Stamatakis. SpeciesRax: A tool for maximum likelihood species tree inference from gene family trees under duplication, transfer, and loss. *bioRxiv*, 2021.

[85] F. Mus, M. B. Crook, K. Garcia, A. G. Costas, B. A. Geddes, E. D. Kouri, P. Paramasivan, M.-H. Ryu, G. E. Oldroyd, P. S. Poole, *et al.* Symbiotic nitrogen fixation and the challenges to its extension to nonlegumes. *Applied and environmental microbiology*, 82(13):3698–3710, 2016.

[86] P. Mylona, K. Pawlowski, and T. Bisseling. Symbiotic nitrogen fixation. *The Plant Cell*, 7(7):869, 1995.

[87] R. P. Nugent, M. A. Krohn, and S. L. Hillier. Reliability of diagnosing bacterial vaginosis is improved by a standardized method of gram stain interpretation. *Journal of Clinical Microbiology*, 29(2):297–301, 1991.

[88] M. Nute, E. Saleh, and T. Warnow. Evaluating Statistical Multiple Sequence Alignment in Comparison to Other Alignment Methods on Protein Data Sets. *Systematic Biology*, 68(3):396–411, 2018.

[89] D. H. Parks, C. Rinke, M. Chuvochina, P.-A. Chaumeil, B. J. Woodcroft, P. N. Evans, P. Hugenholtz, and G. W. Tyson. Recovery of nearly 8,000 metagenome-assembled genomes substantially expands the tree of life. *Nature Microbiology*, 2(11):1533–1542, 2017.

[90] J. Qin, R. Li, J. Raes, M. Arumugam, K. S. Burgdorf, C. Manichanh, T. Nielsen, N. Pons, F. Levenez, T. Yamada, *et al.* A human gut microbial gene catalogue established by metagenomic sequencing. *Nature*, 464(7285): 59–65, 2010.

[91] C. Quast, E. Pruesse, P. Yilmaz, J. Gerken, T. Schweer, P. Yarza, J. Peplies, and F. O. Glöckner. The SILVA ribosomal RNA gene database project: improved data processing and web-based tools. *Nucleic Acids Research*, 41(D1): D590–D596, 2012.

[92] K. D. Queiroz. Species concepts and species delimitation. *Systematic Biology*, 56(6):879–886, 2007.

[93] R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2017.

[94] R. Rabenseifner, G. Hager, and G. Jost. Hybrid MPI/OpenMP Parallel Programming on Clusters of Multi-Core SMP Nodes. In *2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, pages 427–436, 2009.

[95] A. Rambaut and N. C. Grass. Seq-Gen: an application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees. *Bioinformatics*, 13(3):235–238, 1997.

[96] J. Raphson. *Analysis aequationum universalis: seu ad aequationes algebraicas resolvendas methodus generalis, & expedita, ex nova infinitarum serierum methodo, deducta ac demonstrata*, volume 1. Typis TB prostant venales apud A. & I. Churchill, 1702.

[97] T. Rognes, T. Flouri, B. Nichols, C. Quince, and F. Mahé. VSEARCH: a versatile open source tool for metagenomics. *PeerJ*, 4:e2584, 2016.

[98] L. Rubinat-Ripoll. lrubinat/PhotoRefT: a 16S rDNA reference tree representing the main groups of picophototrophic eukaryotes and prokaryotes, 2019. Online: `https://doi.org/10.5281/zenodo.3476953`.

[99] S. Sarmashghi, K. Bohmann, M. T. P. Gilbert, V. Bafna, and S. Mirarab. Skmer: assembly-free and alignment-free sample identification using genome skims. *Genome Biology*, 20(1), 2019.

[100] C. E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[101] E. H. Simpson. Measurement of diversity. *Nature*, 163(4148):688, 1949.

[102] R. Sinha, , G. Abu-Ali, E. Vogtmann, A. A. Fodor, B. Ren, A. Amir, E. Schwager, J. Crabtree, S. Ma, C. C. Abnet, R. Knight, O. White, and C. Huttenhower. Assessment of variation in microbial community amplicon sequencing by the microbiome quality control (MBQC) project consortium. *Nature Biotechnology*, 35(11):1077–1086, 2017.

[103] R. Sinha, , G. Abu-Ali, E. Vogtmann, A. A. Fodor, B. Ren, A. Amir, E. Schwager, J. Crabtree, S. Ma, C. C. Abnet, R. Knight, O. White, and C. Huttenhower. Assessment of variation in microbial community amplicon sequencing by the microbiome quality control (MBQC) project consortium. *Nature Biotechnology*, 35(11):1077–1086, 2017.

[104] T. F. Smith, M. S. Waterman, *et al.* Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

[105] S. Srinivasan, N. G. Hoffman, M. T. Morgan, F. A. Matsen, T. L. Fiedler, R. W. Hall, F. J. Ross, C. O. McCoy, R. Bumgarner, J. M. Marrazzo, and D. N. Fredricks. Bacterial Communities in Women with Bacterial Vaginosis: High Resolution Phylogenetic Analyses Reveal Relationships of Microbiota to Clinical Criteria. *PLoS ONE*, 7(6):1–15, 2012.

[106] A. Stamatakis, A. J. Aberer, C. Goll, S. A. Smith, S. A. Berger, and F. Izquierdo-Carrasco. RAxML-light: a tool for computing terabyte phylogenies. *Bioinformatics*, 28(15):2064–2066, 2012.

[107] S. Sunagawa, L. P. Coelho, S. Chaffron, J. R. Kultima, K. Labadie, G. Salazar, B. Djahanschiri, G. Zeller, D. R. Mende, A. Alberti, F. M. Cornejo-castillo, P. I. Costea, C. Cruaud, F. Ovidio, S. Engelen, I. Ferrera, J. M. Gasol, L. Guidi, F. Hildebrand, F. Kokoszka, C. Lepoivre, F. D'Ovidio, S. Engelen, I. Ferrera, J. M. Gasol, L. Guidi, F. Hildebrand, F. Kokoszka, C. Lepoivre, G. Lima-Mendez, J. Poulain, B. T. Poulos, M. Royo-Llonch, H. Sarmento, S. Vieira-Silva, C. Dimier, M. Picheral, S. Searson, S. Kandels-Lewis, C. Bowler, C. de Vargas, G. Gorsky, N. Grimsley, P. Hingamp, D. Iudicone, O. Jaillon, F. Not, H. Ogata, S. Pesant, S. Speich, L. Stemmann, M. B. Sullivan, J. Weissenbach, P. Wincker, E. Karsenti, J. Raes, S. G. Acinas, and

P. Bork. Structure and function of the global ocean microbiome. *Science*, 348 (6237):1–10, 2015.

[108] S. Tavaré *et al.* Some probabilistic and statistical problems in the analysis of dna sequences. *Lectures on mathematics in the life sciences*, 17(2):57–86, 1986.

[109] J. D. Thompson, B. Linard, O. Lecompte, and O. Poch. A comprehensive benchmark study of multiple sequence alignment methods: Current challenges and future perspectives. *PLoS ONE*, 6(3):e18093, 2011.

[110] C. M. Tucker, M. W. Cadotte, S. B. Carvalho, T. J. Davies, S. Ferrier, S. A. Fritz, R. Grenyer, M. R. Helmus, L. S. Jin, A. O. Mooers, S. Pavoine, O. Purschke, D. W. Redding, D. F. Rosauer, M. Winter, and F. Mazel. A guide to phylogenetic metrics for conservation, community ecology and macroecology. *Biological Reviews*, 92(2):698–715, 2017.

[111] P. J. Turnbaugh, R. E. Ley, M. Hamady, C. M. Fraser-Liggett, R. Knight, and J. I. Gordon. The human microbiome project. *Nature*, 449(7164):804–810, 2007.

[112] L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *Journal of computational biology*, 1(4):337–348, 1994.

[113] K. Wetterstrand. DNA Sequencing Costs: Data from the NHGRI Genome Sequencing Program (GSP). Web page, accessed March 2021. https://www.genome.gov/sequencingcostsdata.

[114] Z. Yang. Among-site rate variation and its impact on phylogenetic analyses. *Trends in Ecology & Evolution*, 11(9):367–372, 1996.

[115] Z. Yang. The BPP program for species tree estimation and species delimitation. *Current Zoology*, 61(5):854–865, 2015.

[116] T. J. Ypma. Historical development of the Newton–Raphson method. *SIAM review*, 37(4):531–551, 1995.

[117] S. N. Yurgel, J. T. Nearing, G. M. Douglas, and M. G. I. Langille. Metagenomic functional shifts to plant induced environmental changes. *Frontiers in Microbiology*, 10, 2019.

[118] J. Zhang, P. Kapli, P. Pavlidis, and A. Stamatakis. A general species delimitation method with applications to phylogenetic placements. *Bioinformatics*, 29(22):2869–2876, 2013.

[119] A. Zhernakova, A. Kurilshikov, M. J. Bonder, E. F. Tigchelaar, M. Schirmer, T. Vatanen, Z. Mujagic, A. V. Vila, G. Falony, S. Vieira-Silva, *et al.* Population-based metagenomics analysis reveals markers for gut microbiome composition and diversity. *Science*, 352(6285):565–569, 2016.

List of Publications

This list contains all of my publications that are relevant to, and were created during my work on this thesis. For an up to date list, see `https://orcid.org/0000-0002-3437-150X`.

1. **P. Barbera**, A. Kozlov, T. Flouri, D. Darriba, L. Czech, and A. Stamatakis. Massively Parallel Evolutionary Placement of Genetic Sequences. Poster at *ISC 2017 PhD Symposium*, Frankfurt am Main, Germany, June 2017.

2. **P. Barbera**, A. Kozlov, L. Czech, B. Morel, D. Darriba, T. Flouri, and A. Stamatakis. EPA-ng: Massively Parallel Evolutionary Placement of Genetic Sequences. *Systematic Biology*, 68(2):365–369, 2019.

3. L. Czech, **P. Barbera**, and A. Stamatakis. Methods for Automatic Reference Trees and Multilevel Phylogenetic Placement. *Bioinformatics*, 35(7):1151–1158, 2019.

4. M. Jamy, R. Foster, **P. Barbera**, L. Czech, A. M. Kozlov, A. M. Stamatakis, D. Bass, and F. Burki. Long-read metabarcoding of the eukaryotic rDNA operon to phylogenetically and taxonomically resolve environmental diversity. *Mol Ecol Resour*, 20:429—443, 2020

5. L. Czech, **P. Barbera**, and A. Stamatakis. Genesis and Gappa: Processing, Analyzing and Visualizing Phylogenetic (Placement) Data. *Bioinformatics*, 36(10):3263–3265, 2020

6. R. C. Edgar, J. Taylor, T. Altman, **P. Barbera**, D. Meleshko, V. Lin, D. Lohr, G. Novakovsky, B. Al-Shayeb, J. F. Banfield, A. Korobeynikov, R. Chikhi, and A. Babaian. Petabase-scale sequence alignment catalyses viral discovery. *bioRxiv*, 241729, 2020. Submitted to *Nature*

7. **P. Barbera**, L. Czech, S. Lutteropp, and A. Stamatakis. SCRAPP: A tool to assess the diversity of microbial samples from phylogenetic placements. *Mol Ecol Resour.* 21:340—349, 2021

8. B. Morel, **P. Barbera**, L. Czech, B. Bettisworth, L. Hübner, S. Lutteropp, D. Serdari, E. G. Kostaki, I. Mamais, A. Kozlov, P. M. Pavlidis, D. Paraskevis, and A. Stamatakis. Phylogenetic Analysis of SARS-CoV-2 Data Is Difficult. *Molecular Biology and Evolution*, msaa314, 2020

9. **P. Barbera**, and A. Stamatakis. Efficient Memory Management in Likelihood-based Phylogenetic Placement. *IPDPS21 Conference*, 2021