Karlsruher Institut für Technologie

EFFICIENT AND MASSIVELY PARALLEL IMPLEMENTATION OF THE EVOLUTIONARY PLACEMENT ALGORITHM

Master thesis of

Pierre Barbera

At the Department of Informatics
Chair of High Performance Computing in the Life Sciences

May 2016

Reviewer:	Prof. Dr. Alexandros Stamatakis
Second Reviewer:	Prof. Dr. Achim Streit
Advisors:	Dr. Tomas Flouri

KIT – University of the State of Baden-Wuerttemberg and National Research Center
of the Helmholtz Association *www.kit.edu*

ABSTRACT

Phylogenetic placement is an important method in the emergent field of metagenetics. It is used to establish abundance- and diversity profiles of microorganisms for a given environment. The Evolutionary Placement Algorithm (EPA) is an algorithm that performs phylogenetic placement of query sequences on a reference tree and reference alignment in linear time and space.

Previous implementations of the EPA achieved high parallel efficiency on shared memory systems. In this work, I present Parallel-EPA (P-EPA): a complete reimplementation of the EPA that performs similarly on shared memory systems. Additionally, I present a design that enables parallelization using distributed memory systems such as supercomputers and clusters. This represents the first parallelization scheme that scales with the size of the tree, enabling phylogenetic placement on extremely large reference trees.

In my evaluation I show that my implementation produces placement results that are highly similar to those of its two main competitors, pplacer and RAxML.

ZUSAMMENFASSUNG

Phylogenetische Platzierung ist eine wichtige Methode im Gebiet der Metagenetik. Sie wird verwendet, um Häufigkeits- und Vielfältigkeitsprofile von Mikroorganismen eines gegebenen Milieus zu erstellen. Der Evolutionary Placement Algorithm (EPA) ist ein Algorithmus, der phylogenetische Platzierungen von Sequenzen auf Referenzbäumen und deren Referenzalinierungen in linearer Zeit-, und mit linearem Speicherbedarf durchführt.

Vorherige Implementierungen des EPA erreichten eine hohe parallele Effizienz auf Rechnersystemen mit gemeinsam genutztem Speicher. In dieser Arbeit stelle ich Parallel-EPA (P-EPA) vor: eine komplette Neuimplementierung des EPA, welche auf Rechnersystemen mit gemeinsam genutztem Speicher eine vergleichbare Effizienz vorweist. Zusätzlich präsentiere ich einen Entwurf, der die Parallelisierung auf verteilten Rechensystemen wie Supercomputern und Rechnerbündeln ermöglicht. Dies stellt das erste Parallelisierungschema dar, das mit der Größe des Baumes skaliert und somit phylogenetische Platzierungen auf extrem großen Referenzbäumen ermöglicht.

In meiner Auswertung zeige ich, dass meine Implementierung Resultate produziert, die vergleichbar mit deren der Konkurrenzprogramme pplacer und RAxML sind.

[...] whilst this planet has gone cycling on
according to the fixed law of gravity,
from so simple a beginning endless forms
most beautiful and most wonderful
have been, and are being, evolved.

— *Charles Darwin, On the Origin of Species [5]*

ACKNOWLEDGMENTS

I want to thank Alexandros Stamatakis for being my advisor for this work, as well as everyone in the SCO group at the Heidelberg Institute of Theoretical Studies. This includes (in no particular order) Lucas Czech, Alexey Kozlov, Tomas Flouri, Diego Darriba, Paschalia Kapli, and Sarah Lutteropp. This work would not have been possible without their support.

I also want to thank Erick Matsen and his group at the Fred Hutchinson Cancer Research Center in Seattle for hosting me for a month, and the Klaus Tschira Foundation for making the trip possible.

CONTENTS

LIST OF FIGURES

ACRONYMS

BLAST Basic Local Alignment Search Tool

BLO Branch Length Optimization

CLV Conditional Likelihood Vector

DNA Deoxyribonucleic Acid

EMD Earth Mover's Distance

EPA Evolutionary Placement Algorithm

FPA Felsenstein Pruning Algorithm

GTR Generalized Time Reversible

INLP Integer Nonlinear Programming

JSON JavaScript Object Notation

LWR Likelihood Weight Ratio

MC Markov Chain

ML Maximum Likelihood

MSA Multiple Sequence Alignment

N-R Newton-Raphson

NGS Next Generation Sequencing

NUMA non-uniform memory access

P-EPA Parallel-EPA

K-R Phylogenetic Kantorovich-Rubinstein

PLL Phylogenetic Likelihood Library

RAM Random Access Memory

1 INTRODUCTION

In recent years, analyses of genetic material have become increasingly important in biological and clinical research. In part, this is due to the advent of cheap Deoxyribonucleic Acid (DNA) sequencing. DNA sequencing is the determination of the order of nucleotides that make up a DNA molecule.

DNA plays perhaps the most central role in all of biology. It is the blueprint that governs the growth and behavior of cells. As such, defects in the DNA of an organism have been identified as the cause of a number of diseases. For example, a genetic mutation in the BRCA2-gene leading to its deactivation significantly increases the chances of developing breast cancer. [13]

While sequencing has been possible since the 1970s [28], only recently have advances, dubbed as Next Generation Sequencing (NGS), driven it to the forefront of research. The sequencing of the first complete human genome [34] in 2001 took over a decade of global effort. After its completion, the cost for sequencing a full human genome was estimated to be around $100M. In contrast, nowadays this cost has decreased to just above $1000 [36].

Because of the richness of genetic data, and its high availability due to NGS, its analysis is the goal of many research projects. For instance, *phylogenetic* studies try to infer evolutionary trees of species, based on data obtained by sequencing specific regions of their genomes [9]. Doing so can reveal the evolutionary history between species and tell us how they are related to one another.

Phylogenetics can also be used to find out how a species relates to an already existing evolutionary tree, or *phylogeny*. This is the goal of the EPA. It tries to find the most likely position of a given anonymous sequence on a phylogenetic tree.

This can be especially useful when encountering a new species, or piece of viral DNA, as illustrated in Figure 1. During outbreaks of new viruses, a race begins to identify effective treatments, cures, and eventually, vaccines. Finding the phylogenetic placement of a new virus can help to quickly identify its characteristics and thus develop possible treatments.

This was the case during a viral outbreak in New York in 1999 [12]. Initially, physicians diagnosed it as a type of *St. Louis encephalitis*. Phylogenetic analysis of the viral sequences revealed it to be a strain of the *West Nile virus* [16], a virus rarely seen outside Africa and the Middle East.

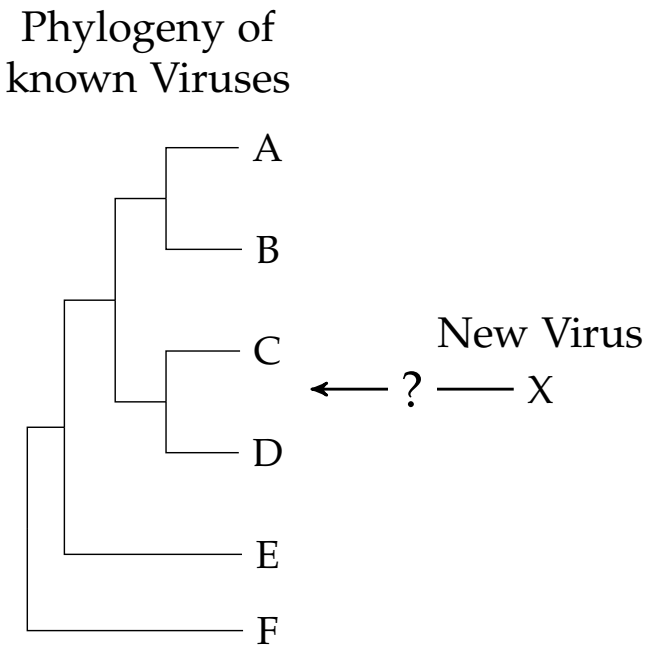


Figure 1: *Example application of phylogenetic placement.*

Another common application of phylogenetic placement lies in the field of *metagenetics*. In metagenetic studies, genetic data is extracted directly from environmental samples. As such, the resulting data usually originates from a high number of different organisms. Often, a central goal is to establish diversity profiles of the samples [19, 30]. In clinical studies, these may be used to infer correlations between bacterial composition and disease status [30].

The EPA facilitates such studies by informing the user how the sequences relate to known species. Downstream analysis can use its output to determine what kind of species are present. This is called *taxonomic classification*.

Non-phylogenetic methods for taxonomic classification typically use the Basic Local Alignment Search Tool (BLAST) [1, 15] to find similar sequences. However, they lack the detailed placement location information that phylogenetic approaches provide.

In this work, I present a new implementation of the EPA. A primary goal in this work was to extend previous parallelization schemes for the algorithm to massively parallel, distributed computer systems. In doing so my design achieves high scalability, allowing placement to be performed on reference trees of almost arbitrary size, given enough computational resources.

In Chapter 2 I introduce the basic principles and terminology used throughout this work. This also includes some background on the type of biological data used as input. I mainly focus on the basics of phylogenetic inference as well as the basics of phylogenetic placement.

In Chapter 3, I outline the algorithmic structure of the EPA and its time-saving extensions, as they were previously described by Berger et al. [2]. In Chapter 4, I briefly outline the differences of my work to other programs implementing phylogenetic placement.

The following Chapter 5 introduces the parallel design used in this work. I present schemes for EPA parallelization in both, shared, and distributed memory systems.

Finally, in Chapter 6, I present the results of the evaluation and verification of my software. I conclude with a summary in Chapter 7.

2 BASIC PRINCIPLES

The following chapter aims to cover the basic principles and methodologies used in this work. I cover fundamental terminology regarding the type of biological data used, as well as the principles of the statistical methods deployed in phylogenetic inference and phylogenetic placement. Additionally, I briefly cover the basics and terminology of cluster computing, the key target environment of the software I designed in this thesis.

2.1 MOLECULAR DATA

Molecular data, primarily sequences of Deoxyribonucleic Acid (DNA), dominate analysis tasks in the field of bioinformatics. DNA is a chain of molecules that makes up the genetic material of an organism. It consists of four different nucleotides, or *bases*: Adenine, Thymine, Guanine, and Cytosine, or A, T, G, and C for short.

While linkage of bases occurs along the chain, they are also mirrored in a duplicate copy of the chain. The two chains are connected, forming the famous DNA double helix. The bases however do not pair exactly: A will only correctly pair with T, and C with G, respectively.

DNA serves as the molecular blueprint for all molecules produced by a living cell. Contiguous regions of DNA that encode for a specific molecule, such as a protein, are called *genes*. The collection of genes of an organism governs its physical characteristics. As such, DNA is the basis on which speciation by natural selection operates. This is because genes, or variations thereof, that are beneficial to the survival

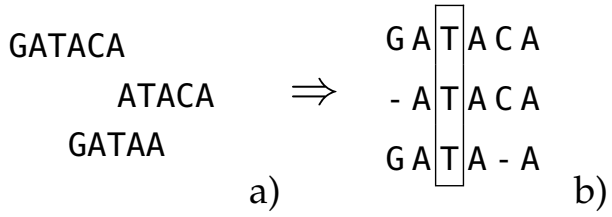


Figure 2: *Basic terminology of DNA sequence data.* a) a collection of DNA *sequences*, comprising characters representing the four nucleotides `A`, `T`, `C` and `G`. b) a Multiple Sequence Alignment (MSA) for these sequences. The alignment process inserts gap characters `-`. A column in the alignment, as highlighted by the rectangle, is called a *site* in the alignment.

and reproduction of an organism, are more likely to propagate to subsequent generations.

Significant advances in sequencing technologies, which determine the order of the DNA bases, have recently generated an abundance of sequence data. Sequencing produces decoded, textual, *sequence* data consisting of short subsequences of genes, up to complete genomes.

Figure 2 outlines some basic DNA terminology. A common way to process a collection of related sequences is to infer a Multiple Sequence Alignment (MSA). As the name suggests, it is the result of an *alignment* procedure. A MSA process arranges sequences in a matrix-style representation. It assigns each sequence to a row, and shifts the characters of the sequences such that similar sub-sequences share the same columns. This is done because after alignment, columns in the MSA, henceforth called *sites*, shall be *homologous*.

Homology is the property of descent from a common ancestor. Consequently it is of central importance in analyses trying to reconstruct ancestry. Phylogenetic inference is one such method, and it relies heavily on the assumption of homology at MSA sites, which is usually its primary input.

2.2 PHYLOGENETIC TREES

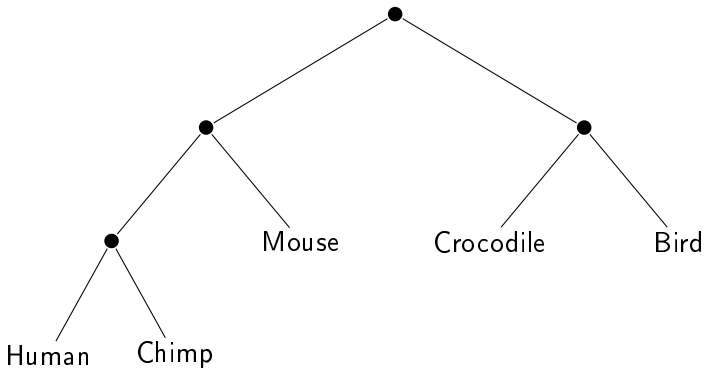


Figure 3: An example for a phylogenetic tree.

Phylogenetic trees, or evolutionary trees, are graphs showing the evolutionary relationships of multiple species that are represented as tips of the tree. Typically these will be *extant*, or surviving, organisms.

Following the edges of the graph, as illustrated in Figure 3, one can trace evolutionary history. Inner nodes of the graph are divergence events and represent the common ancestor of all organisms below them.

The closer two organisms are in the graph, the more closely they are related. For example, Figure 3 shows, that Humans are more closely related to Chimpanzees than to Birds, as Humans and Chimps share a more recent common ancestor.

Phylogenetic trees can be *rooted* or *unrooted*. Figure 3 shows a rooted phylogenetic tree. Starting from the root of the tree at the top of the graph, every inner node splits the rest of the tree in two *subtrees*. This is also called a *bifurcation*.

Any rooted tree can be unrooted by simply removing the root node and connecting its two child nodes by a single branch. Conversely, every unrooted tree can be rooted by placing a new inner node on any chosen branch of the tree. Such a node is referred to as the *virtual root* of an unrooted tree.

Methods to infer such trees can use different input data. While the first methods used vectors describing the presence/absence of morphological traits in an organism, modern methods rely on molecular data, such as DNA or amino acid sequences.

In principle, the majority of methods for inferring phylogenies will do so by successively clustering organisms by similarity of their representative sequences. Simple methods try to minimize the changes required to transform an organism's sequence into its closest neighbor in the tree. One such method is the *parsimony* criterion, which aims to minimize the number of evolutionary steps required to explain the data for a given tree.

More advanced methods use statistical models of molecular evolution to simulate the stochastic changes of nucleotides over time. As a consequence, they assign different possible evolutionary histories to respective probabilities, and through this compute the *likelihood* of a tree.

2.3 PHYLOGENETIC LIKELIHOOD

The phylogenetic likelihood is used as the objective function in *tree search*. Tree search is the traversal of the parameter space that determines the topology and characteristics of a phylogenetic tree. Doing so efficiently is the central issue in software implementing phylogenetic inference (such as RAxML [32]). Section 2.4 will cover tree search and its challenges in greater detail. First, I will elaborate on the phylogenetic likelihood itself.

Like for most optimization problems, finding the *best* tree requires some function to select among two or more distinct trees. In likelihood based methods, this function is derived from the *odds ratio* [9]

$$\frac{P(T_1|D)}{P(T_2|D)} = \frac{P(D|T_1)}{P(D|T_2)} \frac{P(T_2)}{P(T_2)} \tag{1}$$

between trees T_1 and T_2 given the data D. In phylogenetics, D is the MSA described in Section 2.1. Using Equation 1, we can compare two proposed trees by their *posterior probability* $P(T|D)$.

To calculate the ratio, we need to calculate its two terms. The first is $P(T)$, also called the *prior probability*. This is the probability of observ-

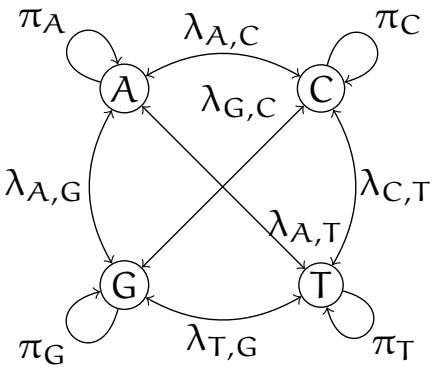


Figure 4: *A general depiction of a Markov chain process for nucleotide evolution.* $\lambda_{x,y}$, with $x, y \in \{A, T, C, G\}$, denotes the rate of transitioning from state x to state y for a given time step, and is called the *transition rate*. π_x is the special case $\lambda_{x,x}$, called the *stationary frequency*. For reasons of legibility, we assume $\lambda_{x,y} = \lambda_{y,x}$.

ing the tree T prior to the evaluation of any data. The second term is $P(D|T)$, or the probability of observing the data D, given a tree T. This is called the *likelihood* and, not surprisingly, it is the focus of the Maximum Likelihood (ML) branch of phylogenetics.

With an increasing volume of independent observations, the likelihood $P(D|T)$ is the product of these individual observations [9]:

$$P(D|T) = \prod_i P(D^{(i)}|T) \tag{2}$$

Consequently, the more data is available, the more the likelihood term will dominate the posterior probability $P(T|D)$ of the tree under the given data. This is a central premise behind ML: given enough independent observations of the data, the prior probability becomes negligible.

Before discussing in detail how ML methods compute the overall likelihood of a tree, it is important to discuss how we model changes in nucleotide sequences over time.

2.3.1 *Models of Nucleotide Evolution*

In nature, genetic material can change over time, through mechanisms such as, for instance, imperfect replication, or repair, of DNA. Over time, and successive generations, two populations of organisms that at one point were practically identical, can start to diverge, given that no, or little, interbreeding occurs. If the organisms have diverged past a certain point, biologists may consider them as being separate species. This is the process of *speciation*, and as discussed in Section 2.2, it is the defining feature of phylogenetic trees. Such events are represented by the inner nodes in the tree.

Thus, to evaluate the likelihood of a tree, based on genetic data, one needs to take into account this change of nucleotides over time. For statistical methods, a first step in developing such a model is to quantify single nucleotide changes. This done by the rate $\lambda_{x,y}$ describing the transition of a base x to base y, where $x, y \in \{A, T, C, G\}$ in time dt. Henceforth we will denote the state alphabet $\{A, T, C, G\}$ as N. The special case $\lambda_{x,x}$, denoted as π_x, is the frequency with which the Markov process remains in state x. Consequently, it is called the *stationary frequency*.

We can depict the four possible DNA states, with respective transition rates between them, as shown in Figure 4. In doing so, and by assuming that the mechanism of nucleotide change is a stochastic process, we can construct a Markov Chain (MC), which forms the basis of most statistical models of evolution.

Also, most models assume that the MC is *time reversible*. For a MC to be time reversible, the transition rate from state x to y must be equal to the transition rate from state y to x:

$$\pi_x \lambda_{x,y} = \pi_y \lambda_{y,x} \tag{3}$$

Some models, such as JC69 [17] or F81 [7], additionally constrain the possible choices of $\lambda_{y,x}$. The model used in this work allows the greatest freedom to choose values for λ and π, while still adhering to time reversibility, and is called the Generalized Time Reversible (GTR) [33] model.

Next, we need to define a quantity of how many state changes occur between nodes in the phylogenetic tree. These are usually represented as the branch lengths, denoted as v here. It is the rate at which a nucleotides change is expected to occur on average along the sequence.

Many biological processes influence v, such as the frequency of DNA replication, environmental pressures, and time.

2.3.2 *The Felsenstein Pruning Algorithm*

Given a Markov model M and a branch v, we can compute $P_{x \mapsto y}(v)$ to be the probability of a nucleotide transitioning from state x into state y after moving along a branch v. As phylogenetic trees have inner nodes, whose state we do not know, a central point of likelihood calculations on trees is to account for these unknown states. The phylogenetic likelihood method does so by summing over the probabilities of every possible state z of an inner node.

Consider the following scenario, depicted in Figure 5. We have an inner node k that is connected by branches v_l and v_m to tip nodes l and m.

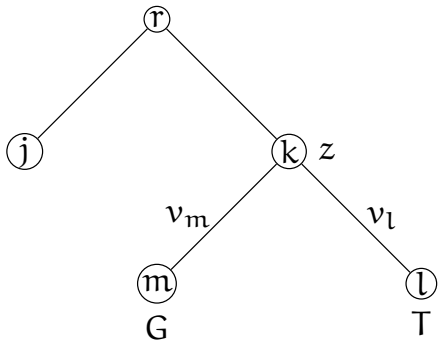


Figure 5: *An example phylogeny for which the conditional likelihood is to be computed.* z denotes the possible states at node k at a given site in the underlying MSA. The tip nodes l and m are depicted with their states T and G, at the same site in the alignment. v_l and v_m denote the branch lengths. As the states of l and m are known, because they are tip nodes, the conditional likelihood at node k to be in state z for a given site in the MSA can be computed according to Equation 4.

We now want to calculate the likelihood of the possible states z at the inner node k at a given site (i) in the MSA. Suppose that we know the states of the tip nodes l and m at site (i) to be T and G, respectively. The likelihood is then calculated as the product of the transition probabilities of the inner state z toward the tip states of l and m:

$$\mathcal{L}_k^{(i)}(z) = P_{z\mapsto G}(v_m)P_{z\mapsto T}(v_l) \tag{4}$$

Equation 4 is called the *conditional likelihood* $\mathcal{L}_k(z)$ at node k.

For any given tip node t in the tree with a corresponding sequence $s = s_1 s_2 \dots s_n$, where n is the number of sites in the MSA, the conditional likelihood is defined as

$$\mathcal{L}_t^{(i)}(z) = \begin{cases} 1, & z = s_i \\ 0, & z \neq s_i \end{cases} \tag{5}$$

Using this, we can extend Equation 4 to work at an arbitrary inner node k by factoring in the likelihoods $\mathcal{L}_l^{(i)}(x_l)$ and $\mathcal{L}_m^{(i)}(x_m)$ of the possibly unknown states x_l and x_m at the child nodes l and m for site (i) in the MSA:

$$\mathcal{L}_k^{(i)}(x_k) = \left(\sum_{x_l \in N} P_{x_k \mapsto x_l}(v_l)\mathcal{L}_l^{(i)}(x_l) \right) \times \left(\sum_{x_m \in N} P_{x_k \mapsto x_m}(v_m)\mathcal{L}_m^{(i)}(x_m) \right) \tag{6}$$

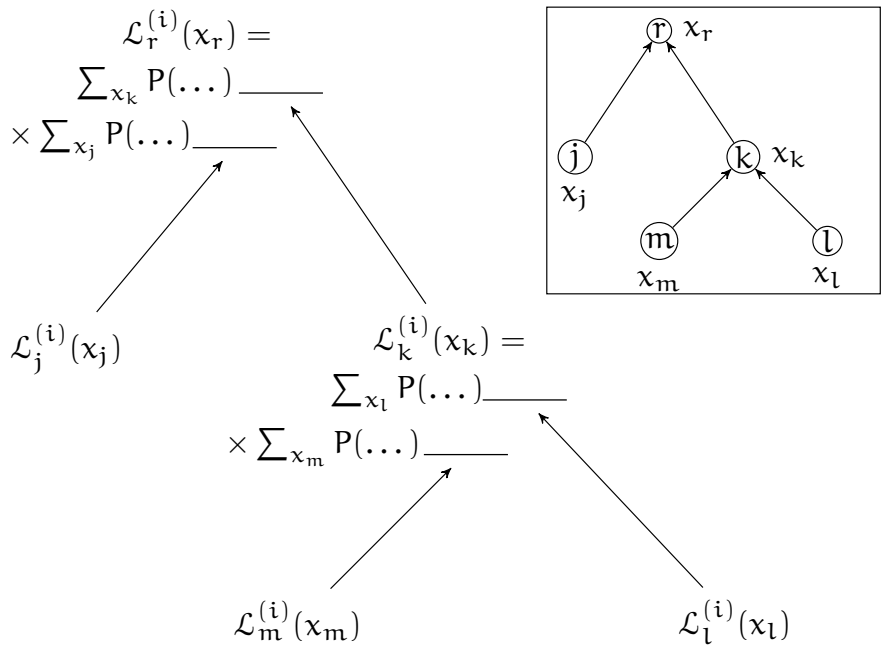


Figure 6: *An illustration of the Felsenstein Pruning Algorithm.* The framed part in the upper right shows a phylogeny with three tips. The application of the Felsenstein Pruning Algorithm (FPA) mirrors the topology of the tree exactly. This is illustrated in the larger picture. Equation 6 is used to compute the conditional likelihoods at the inner nodes r and k. The recursion terminates at the tips of the phylogeny as their state is known. At the tips, the conditional likelihood is set according to Equation 5.

Equation 6 is referred to as the Felsenstein Pruning Algorithm (FPA) [8]. Note its recursive nature, further illustrated in Figure 6: it follows the structure of the tree exactly, such that starting at the root, it computes different subtrees independently of each other. This allows for efficient computation of the likelihood of a site, and by extension, the tree.

To calculate the total site likelihood for site (i) at node k, we sum over all possible states x_k at node k, multiplied by the base frequency π_{x_k} of a given state:

$$\mathcal{L}_k^{(i)} = \sum_{x_k \in N} \pi_{x_k} \mathcal{L}_k^{(i)}(x_k) \tag{7}$$

Consequently, the full tree likelihood is, once pruning has computed every site (i) at the root node r, simply the product of the site likelihoods, where n is the number of sites in the underlying MSA:

$$P(D|T) = \prod_{i=1}^{n} \mathcal{L}_r^{(i)} \tag{8}$$

For a given node k, we refer to a vector of $\mathcal{L}_k^{(i)}$ over all sites (i) in the alignment as a CLV. The implementation of programs for calculating phylogenetic likelihoods mirrors this definition exactly, and as such, it represents an important notation in this work.

2.4 TREE SEARCH AND PARAMETER OPTIMIZATION

In the previous section, I described how to compute the likelihood of a tree. This did, however, not cover how to find the most likely tree.

The number of possible tree topologies grows super-exponentially with the number of species at the tips. As Felsenstein shows [9], for n species the number of possible unrooted, bifurcating tree topologies is

$$1 \times 3 \times 5 \times 7 \times \cdots \times (2n - 5) \tag{9}$$

meaning, that for only 20 input sequences there will already be about 2.22×10^{20} possible trees. Contrast this with the fact that currently it is not unusual to infer trees with thousands of species [14, 29, 18].

Exploring this search space constitutes the central operation of tree inference programs such as RAxML [32]. Typically, they operate in two phases, which alternate iteratively (given an initial topology and parameter configuration):

1. optimize the branch lengths of a given topology, as well as the model parameters of the model used
2. optimize the topology of the tree, given the model parameters

A common method to perform Branch Length Optimization (BLO) is *Newton's* method, also known as the Newton-Raphson (N-R) procedure, which is an iterative approximation algorithm to determine the root(s) of any given real-valued function. In our case, this function is an extended form of the phylogenetic likelihood $P(D|T, v)$, where v is the current branch length value of the branch being optimized. As we want to find the maximum of the likelihood, we use the N-R method to find the root of the likelihoods first derivative $P'(D|T, v)$.

N-R approximates values for v, using $P'(D|T, v)$ and its derivative $P''(D|T, v)$, iteratively:

$$v_{n+1} = v_n - \frac{P'(D|T, v_n)}{P''(D|T, v_n)} \tag{10}$$

The approximation algorithm iterates until the change in v between two iterations falls below some user specified threshold, that is, until the optimization *converges*. Usually, this procedure is repeated for every branch in the tree.

BLO is central in the *placement algorithm* described in Section 3.1.2. There, the smallest possible sub-tree, containing three tips, is subjected to BLO to obtain more accurate likelihood values.

2.5 PHYLOGENETIC PLACEMENT

While phylogenetic ML methods are useful to infer trees, one can also use them for related problems. One such related problem is *phylogenetic placement*, which is the central topic of this work.

The goal of placement is to gain information about how a sequence relates to an already existing tree and associated MSA. As such, placement requires two inputs: a set of sequences, or *query sequences*, and a *reference tree*, including the MSA from which the tree was built. The reference tree forms the context into which the query sequences are placed. Therefore, the query sequences must also be aligned to the reference MSA. Calculating this *extended MSA*, comprising the reference *and* query sequences, is typically done prior to the phylogenetic placement procedure.

Phylogenetic placement is an emerging tool in *metagenetics* [21, 24], a field which studies genetic material that is obtained directly from environmental samples. One of the opportunities of sampling directly from the environment is the possibility to study inter-microbial interactions. In contrast, classical biological studies of microbes cultivate organisms in the lab, in isolation from their environment. Additionally, many organisms cannot yet be cultivated in the lab to begin with.

Thus, being able to study genetic sequences obtained directly from their environments presents a plethora of opportunities for new insights. For example, one may use placement algorithms to establish bacterial abundance and diversity profiles [19, 30, 26, 20]. A recent clinical study by Srinivasan et al. [30] utilized such analyses to identify correlations between composition of the microbial environment in the human vagina and diagnosis of bacterial vaginosis.

Choosing an appropriate reference tree, and associated MSA, is a daunting task in itself. Phylogenetic placement can only tell us where on a tree a sequence most likely fits to. It cannot tell us if that same sequence would have been much more likely placed at a different location in the *real*, true evolutionary tree as it exists in nature. Thus, the user has to consider the selection of organisms from which to build the reference tree carefully to avoid potential biases.

For example, consider a recent study using placement methods to detect Protist biodiversity in rain forest soils by Mahe et al. [19]. It revealed significant populations of marine organisms among its samples. A more naïve approach may not have included such organisms in the reference tree, potentially missing a significant component in the composition of rain forest environments.

Ideally, placement should always rely on the most comprehensive tree available. As ML tree inference is NP-Hard [9, 3], one needs to make a trade off between the completeness of the tree and the feasibility of inferring it.

The core placement algorithm itself does not choose or infer the reference tree. Its main operation is to take each query sequence and place it onto the tree as a new branch. Then, the overall likelihood of the tree can be re-evaluated. A given edge in the tree, onto which the algorithm has placed a sequence, is called the *insertion branch*. After placing the sequence, the likelihood of the tree is regarded as the likelihood of the sequence to be correctly placed on the specific edge. The algorithm will repeat this for all pairs of query sequences and tree edges.

It is important to note that placement does not extend the tree permanently: after evaluation of the likelihood of a placement, the reference tree is reset to its original configuration. In other words, we only place one query sequence into the tree at a time.

The result of the placement algorithm is a mapping of sequences to their most likely positions on the tree. We can quantify the differences in placement likelihoods for a single query $q = \{p_1, ..., p_n\}$, where p_i is a placement of the sequence associated with q on branch i out of n branches. We can do this by normalizing their likelihoods $P(D|T, p)$:

$$LWR(p) = \frac{P(D|T, p)}{\sum_{i=1}^{n} P(D|T, p_i)} \tag{11}$$

This is called the *likelihood weight ratio* [35], and it represents the confidence that the placement p of a query sequence is correct. Using the Likelihood Weight Ratio (LWR), the result of the placement algorithm is a probability distribution for a query over the reference tree, a concept that is useful in downstream analyses, as explained in Section 2.6.

In this thesis, and in previous works implementing the phylogenetic placement procedure, the placement results are written to a dedicated, standard file format called JPLACE.

2.5.1 JPLACE *File Format*

The JPLACE file format [23] is a portable, human-readable format for storing phylogenetic placements of labeled query sequences on a reference tree. The format itself builds on the widely used JavaScript Object Notation (JSON), allowing for seamless parsing on most platforms.

The first basic element the JPLACE-file contains is the reference tree, encoded in the Newick file format [10]. The JPLACE-format augments this tree representation by a unique indexing of the branches, indicated with brackets after branch lengths or edge labels. By convention, the indexing follows a post-order traversal of the tree. For rooted trees (Section 2.2) the post order traversal starts at the root. For unrooted trees, the traversal depends on the choice of an inner node, that serves as the root point (also called *top level trifurcation*).

This numbering is required for the second major block in the file: the list of query sequences, and their calculated placements on the tree. The sequences themselves are uniquely identified by their sequence labels, as they occurred in the query MSA (Section 2.5). The format specifies that a list of placements follows this identifier, each placement containing a collection of fields specifying the index of the placement branch, its likelihood, and the LWR. The content of a placement field is extensible. In the software presented here, for example, each entry also contains the relative position along the branch of the reference tree. This relative position is called the *distal length*, its associated branch being the *distal* branch. The branch that was produced during placement, connecting the tip containing the query sequence to the tree, is called the *pendant* branch. Its length, called the *pendant length*, is included as well. This pendant/distal branch length terminology is also illustrated in Figure 7.

The pendant length is especially useful as an indication of the quality of the fit of a sequence on the tree. Consider, for example, that the most likely placement is located on an unusually long pendant branch. This could indicate that the organism represented by the query sequence may not be closely related to any of the organisms included in the reference tree. The presence of such placements can

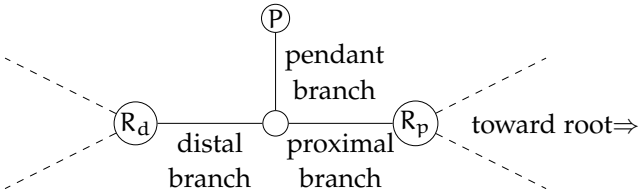


Figure 7: *Basic terminology of tree extension on a given edge of the reference tree.* Shown here is the edge of the reference tree, flanked on each end by reference tree nodes R_d and R_d. R_p is the node that is closer to the root of the reference tree and is called the *proximal* node. Thus, R_d is the node of the two that is more distant from the root of the reference tree, and is called the *distal* node. P is the node that phylogenetic placement has temporarily attached to the reference tree, representing the query sequence. It is called the *pendant* node. The central node was also temporarily added by the phylogenetic placement procedure. The branches connecting the central node to the rest of the tree are called the proximal, distal, and pendant branches, depending on their location. Their values are named by the same scheme: proximal, distal, and pendant length. The dashed lines depict the remaining branches in the reference tree.

be a strong indication that the user may need to consider extending the reference tree.

Lastly, the file also contains some fields for meta data. These may include the format version, and the aforementioned types of data recorded for each placement. Further meta data includes the command line invocation string of the placement program, which is useful for reproducibility.

2.6 METRICS ON COLLECTIONS OF PLACEMENTS

After obtaining the likelihoods and LWRs of each placement for a query sequence on the reference tree, one can project the obtained information back onto the tree to visualize how the sequence relates to the tree. The LWR (Equation 11) is useful for this, since it is already normalized and directly reflects the confidence into the placement. Note that, the LWRs of a single query sum to one, thus yielding a probability distribution over the tree.

When comparing the set of placements of two sequences, representing them as probability distributions is useful, as it allows us to define a distance between them in terms of distances between distributions. One such metric is the Earth Mover's Distance (EMD) [27]. The name derives from its central metaphor: the distribution is a *mass* and moving it requires *work*. The distance between two distributions is the minimum amount of work required to transform one distribution into the other.

For distributions on phylogenetic trees, Evans and Matsen have previously introduced the K-R distance [6]. It operates by the same basic principle as the EMD, moving units of mass (LWRs) across the phylogenetic tree, calculating the total amount of work required to match two distributions.

In this work, I apply the K-R distance to evaluate the correctness of my implementation. I do this by performing phylogenetic placement of a shared set of query sequences on identical trees using different implementations of phylogenetic placement, as well as my own. Then, I compare the distributions of placement locations of query sequences between the different programs I tested (See Section 6.1).

2.7 DISTRIBUTED COMPUTING

Scientific software typically pushes the limits of the computer system it operates on. Taking phylogenetics as an example, common applications of algorithms such as tree inference require large amounts of computing power and memory [31]. To address this need, scientific institutions often have their own computing centers that house a multitude of servers, usually connected via high performance network systems. Such systems are called *computational clusters*, or are also often referred to as a *supercomputer* due to their aggregated power.

Each independent computer in such a networked system is called a *computational node*. A common approach to calculating large problems is to split them up and distribute the computation to as many such nodes as possible.

3 ALGORITHMS

The following chapter describes the operation of the EPA as it was originally conceived by Berger et al. [2]. One of the main goals of this work was to re-implement the EPA.

The core of the EPA is the part that evaluates the likelihood of placements. Additionally, it can be augmented through the use of heuristic approaches to pre-filter promising placement branches in order to limit the computational work while not substantially decreasing accuracy. In the following chapter I will describe both, the core placement algorithm, as well as its heuristic extensions.

3.1 PLACEMENT

The placement algorithm can be divided into three phases.

1. Preprocessing
2. Query Placement
3. Result Aggregation

The following sections describe these phases in detail.

3.1.1 *Preprocessing*

During the first phase, the unrooted reference tree, as well as the corresponding reference MSA, are read into memory, and an internal tree structure is built.

The placement itself requires the CLVs at either end of the insertion branch (Section 2.5) to be computed. These could be recomputed using the FPA (Section 2.3.2), via a full tree traversal, for every placement. However, the CLVs of the edges of the reference tree do not change throughout the placement procedure, across an arbitrary number of queries. Thus, we compute the CLVs of every branch. For each branch we assume that the virtual root of the unrooted tree is located somewhere on that branch, and store the CLVs in memory.

This has the additional advantage of producing the basis for fine-grained parallelization of the query placement phase. To place a query sequence on a branch of the tree, one only needs the two CLVs belonging to that branch. This allows for high memory scalability of the algorithm, subsets of the tree can be distributed independently to an arbitrary number of compute nodes.

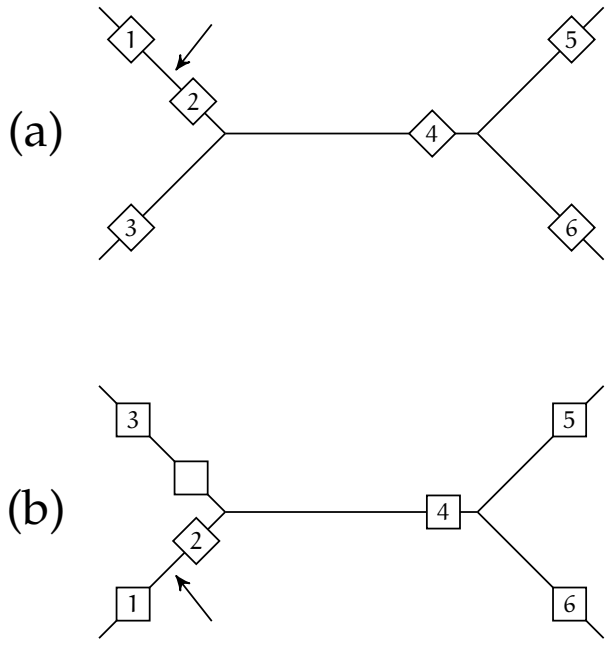


Figure 8: *Illustration of CLV precomputation.* The goal of CLV precomputations is to compute every possible CLV (Section 2.3.2) of the given reference tree. Depicted here are two iterations of said process for the same four taxon phylogeny. First iteration (a)): the branch indicated by the arrow is selected as the first branch of the CLV precomputation procedure. The virtual root of the unrooted tree (Section 2.2) is assumed to be located somewhere on the branch. A post order traversal of the tree, starting at the virtual root, is conducted to identify those CLVs that have not already been computed. Such CLVs are indicated here as numbered diamonds. In this case, we identify the need to compute CLVs 1 and 2, as they are adjacent to the virtual root, 3 and 4 as they are required to compute 2, and 5 and 6 as they, in turn, are required to compute 4. Next, the Felsenstein Pruning Algorithm (FPA) (Section 2.3.2) is applied to compute the indicated CLVs. In the subsequent iteration (b)), the process is repeated. Now, however, only one CLV, indicated as number 2 needs to be directly computed, as the previous iteration has computed all other required CLVs (indicated by squares). The procedure terminates when all branches have been the subject of an iteration.

Figure 8 illustrates these CLV precomputations. For a given edge in the tree, the virtual root is assumed to be located somewhere on that edge. A full traversal of the tree is conducted only once to calculate any CLVs toward that specific virtual root, that have not yet been computed. This is done through the use of the FPA. Any newly calculated CLV is stored in memory.

Note that, each internal node of the tree will be associated with three independent CLVs, one for every adjacent edge and direction of the virtual root. This process is repeated for every edge in the tree. Consequently, every possible CLV in the tree is computed.

3.1.2 *Query Placement*

With precomputed CLVs, the likelihood computation of the placement of a sequence on the tree can be seen as being highly local. All that is needed to compute the likelihood of a placement of a query on a given edge are its two adjacent CLVs.

In the following, I will describe the full operation of the query placement phase. An example of this is illustrated in Figure 9.

Let R_p and R_d denote the nodes of the reference tree directly adjacent to an edge e, on which a placement of a query sequence $q \in Q$ is evaluated using ML. Further, let l_o denote the original length of e in the reference tree.

The algorithm begins by placing a new inner node C between R_p and R_d, splitting edge e, and its branch length l_o in half. It assigns the new branch lengths of the edges connecting C to R_p and C to R_d to be $l_o \times 0.5$. Additionally, it places a new tip node P adjacent to C, representing the sequence q. The length of the branch connecting C to P is initially set to a default value. The result of this is an independent, unrooted tree of minimal dimensions, containing three CLVs and one inner node.

Next, the algorithm performs Branch Length Optimization (BLO) (Section 2.4) on this minimal tree. This represents a first sacrifice of accuracy, as ideally the BLO should be computed on the entire reference tree, extended by the query sequence. However, as this would impose significantly higher computational effort, the algorithm limits itself to the three edges closest to the inserted sequence q.

After the BLO converges, the algorithm computes the likelihood of the placement. It does so by first choosing an arbitrary edge r from the three edges of the minimal tree. It assumes that the virtual root of the tree is located on r and then performs the overall likelihood computation as described in Section 2.3.

It is worth noting that choosing the edge arbitrarily only works under a time reversible model of nucleotide evolution.

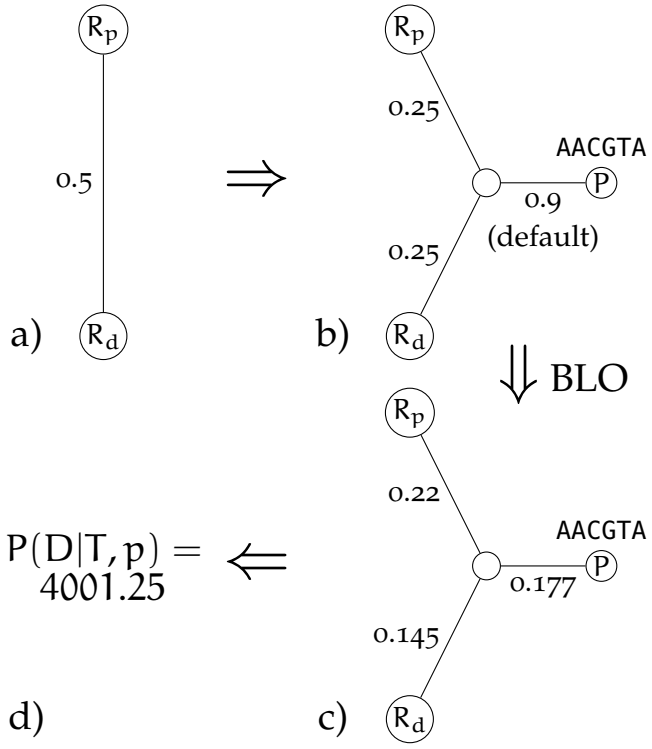


Figure 9: *Basic placement of a query sequence by extension of the tree at a given edge.* a) the reference tree edge for which we want to evaluate the likelihood of a placement for a given query sequence. The end points of the edge correspond to nodes in the reference tree, denoted by R_p for the proximal, and R_d for the distal node. In this example the original branch length is 0.5, as denoted by the edge label. b) the placement algorithm attaches a new node P to the edge representing the query sequence, here exemplary as the sequence AACGTA. It assigns the branch length adjacent to the new node as a default value, here 0.9, and the two new branch lengths adjacent to the original nodes R_p and R_d to be half the original branch length. c) the placement algorithm performs a local branch length optimization, confined to the edges shown. d) the algorithm produces a likelihood based on the new branch lengths, assuming the virtual root to be placed on one of the three edges. This is the placement likelihood result of the sequence on the given branch of the tree.

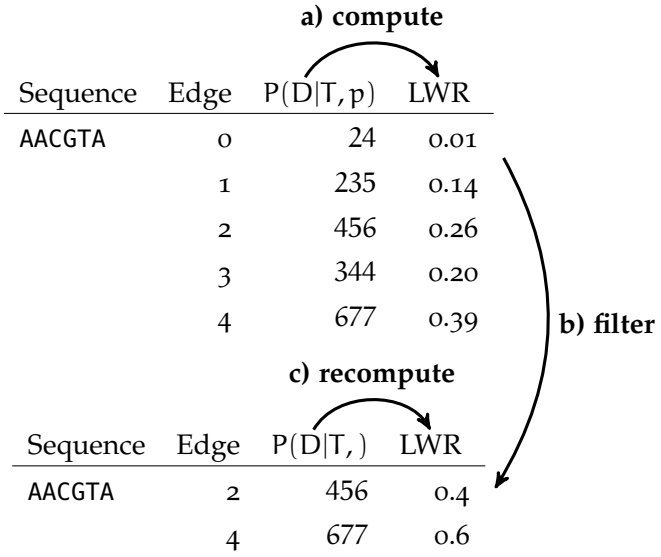


Figure 10: *Post processing of placements during the filtering phase.* The figure shows is the tabulated result of a full *query* of a sequence `AACGTA` on a phylogeny with 5 edges. $P(D|T, p)$ denotes the likelihood of a single placement of the sequence at a given edge. During the aggregation phase, all placements of a query are used to compute their respective LWRs (Section 2.5, Equation 11) (**a)**). The query is then filtered based on the LWR of the placements. In this example, all placements with a LWR value below 0.25 are discarded (**b)**). Lastly, the LWRs are recomputed based on the reduced set of placements.

3.1.3 *Result Filtering*

During the last phase, the algorithm tabulates the likelihood results of a sequence placed on every edge of the tree. It then calculates their respective LWRs, as described in Section 2.5. The process is depicted in Figure 10.

Optionally, the user can specify a threshold and a method to determine what placements are discarded at this point. This is especially useful to minimize the data volume the program produces, as it would otherwise output one placement for every edge in the tree, for every query sequence.

The user can choose between two methods. In the first, all placements that fall below a specified LWR are discarded. In effect, this allows the user to specify what the minimum support value of a placement should be. The second method sorts the placements by LWR, then iteratively adds placements to the list of those that are retained, until the cumulative LWR of that set exceeds a user specified value.

After this selection process, the algorithm outputs the placements to a JPLACE file (Section 2.5.1).

3.2 PRESCORING HEURISTIC

While it may not be immediately obvious, the majority of the computational effort of the EPA (85–90% according to valgrind-callgrind) is exerted during local BLO (Section 3.1.2). It involves a varying number of iterations on branch lengths, and more importantly, recomputations of the phylogenetic likelihood.

Berger et al. originally outlined the *prescoring* heuristic [2], which tries to economize in BLO invocations. It interjects during the normal operation of the algorithm, specifically during query placement. While normal placement performs BLO during placement of *every* sequence of *every* edge, prescoring skips optimization and instead computes the likelihood based on the minimal tree immediately after the new tip P is inserted into the tree based on the appropriate placement likelihoods. It subsequently chooses a drastically reduced set of candidate edges for every query sequence, based on a user specified threshold. For each selected candidate edge, the algorithm repeats the query placement phase of the algorithm, however this time it does perform the localized BLO.

To set the fraction of candidate edges for thorough BLO, the user can choose from two methods analogous to the discarding of placements described in Section 3.1.3. In the first method, the user specifies a percentage of edges in the reference tree. The algorithm sorts the prescored placements by their LWR and then discards all but the most probable placements, as specified by this percentage. The second method is similar, however the user specifies an accumulated LWR threshold. The placements are again sorted by their LWRs as before. However, this time, edges belonging to placements are added to the set of candidates until the accumulated LWR threshold has been reached.

The second method, called *adaptive prescoring* is, to my knowledge, a novel method. Consequently, it was not available in the previous implementations of the EPA. It represents a more adaptive heuristic, as it adjust itself to the likelihood weight distribution produced by prescoring. Should prescoring already produce a very likely candidate edge for placement, only this edge and perhaps a very small number of additional edges are in fact evaluated more thoroughly with BLO. When prescoring produces a very flat distribution, prescoring will not limit itself to a maximum number of candidate edges and instead reevaluate most edges by their relative importance.

The correctness of the adaptive prescoring method is evaluated in Section 6.1.

4

RELATED WORK

In this chapter I outline the differences of my work to other programs implementing phylogenetic placement. Currently, not many implementations exist. The two most widely used by far are RAxMLs implementation of the EPA, and PPLACER. My implementation will henceforth be referred to as PARALLEL-EPA (P-EPA) to avoid confusion.

Software performing taxonomic classification, one of the main uses of phylogenetic placement, is also often built using BLAST [1] or similar methods. However, as BLAST does not find the most likely phylogenetic placement, I will not cover BLAST-based approaches here.

4.1 RAXML-EPA

Prior to this work, the EPA had only been implemented as a part of RAxML [32], an efficient and highly parallel ML-based program for phylogenetic tree inference. As stated before, the primary goal of this work is to re-implement the EPA, in analogy to the RAxML implementation. Consequently, the basic operations of my algorithm differ only in few, key points from the original EPA [2].

As P-EPA, RAxML-EPA calculates a set of placements of a query sequence on a given reference tree (Section 2.5). It does so by evaluating individual placements ML (Section 2.3). RAxML-EPA also utilizes prescoring to identify a set of candidate edges for more thorough evaluation, as described in Section 3.2. However, while in RAxML-EPA the user needs to supply a fixed percentage of candidate edges to be selected during prescoring, P-EPA also offers the option to determine the candidates based on a cumulative LWR (Equation 11) threshold.

RAxML-EPA also utilizes precomputation of the CLVs at either end of the edges to be evaluated. This is the key factor allowing the placement algorithm to scale linearly in space and time with the size of the reference tree.

The parallelization scheme employed in RAxML-EPA is, again, similar to the one used in this work. Both approaches split up the reference tree by edges, then evaluate the placement of each query sequence on the given edge subset. However, in contrast to P-EPA, RAxML-EPA does not support parallelization using distributed memory systems, such as supercomputing clusters. For running the EPA on such systems, the users of RAxML have to first split up the query sequences into appropriate chunks and then input them into instances of the program running on separate computing nodes.

While such a method scales decently with the number of query sequences, it does not with the size of the tree, since every node will always have to compute the placements for every edge. This is in contrast to P-EPA, which automatically scales with both tree size and volume of query sequences (Chapter 5).

Improving the scalability was in fact one of the main goals of this thesis, along with achieving higher maintainability and code quality than RAxML-EPA.

P-EPA also generates the JPLACE file format as its output (Section 2.5.1).

4.2 PPLACER

The PPLACER software [22] is a phylogenetic placement tool with similar functionality to RAxML-EPA (Section 4.1). It also scores placement locations by extending a reference tree by an additional tip. As a score, PPLACER offers ML scores, but also Bayesian posterior probabilities.

In PPLACER, Matsen et al. employ what they call *baseball heursitics*. Baseball heuristics are conceptually similar to the prescoring heuristic presented in Section 3.2, as they also try to select a set of candidate edges. On these, a more thorough placement is then conducted, utilizing BLO.

Candidate edge selection in the baseball heuristic works as follows. For a given sequence, placement likelihoods are computed for the middle of every edge. The resulting list of per edge likelihoods is then sorted from most likely, denoted as L, to least likely. The authors call this the *batting order*. The procedure then goes through the list in order, performing thorough placement using BLO. This continues until a thoroughly evaluated placement yields a likelihood lower than the initially highest likelihood L minus some user specified value D, called the *strike box*. Once this point is reached, a *strike* is counted. The evaluation terminates once a number of strikes, specified by the user, has been reached.

How PPLACER performs BLO is different than how P-EPA and RAxML-EPA do it. PPLACER does not use the N-R method. Instead, the attachment location on the insertion branch is changed iteratively, without changing the sum of the distal and proximal branch lengths. Each iteration also changes the length of the pendant branch. For optimizing the pendant branch length, however, Brent's method is used. This is repeated until some likelihood threshold is reached.

Like P-EPA and RAxML-EPA, PPLACER outputs the placements in JPLACE format (Section 2.5.1).

Unlike other placement tools, PPLACER is a comprehensive software package, and also includes tools for further analysis and visualization of the placement results.

Among others, this includes PLACEVIZ, which converts the output of the placement algorithm into tree formats that can be viewed by

common tree visualization tools. It visualizes the number of placements per branch by proportionally altering the branch thickness. The tool also visualizes placement uncertainty through a color gradient.

Another associated tool is PLACEUTIL, which can be used to manipulate JPLACE-files. This includes merging separate placement files, checking for inconsistencies such as different reference trees, or splitting them apart based on regular expressions.

5 DESIGN

One of the main goals of this work was to produce a more scalable implementation of the EPA, both regarding the size of the input reference tree, as well as the number of query sequences.

In practice, scaling with the number of input query sequences using programs such as RAxML-EPA[2] or PPLACER [22] is done by first splitting up the sequences into disjoint chunks of equal size. Then, the user runs many instances of the program, each placing one such chunk of sequences on the common reference tree, across a number of distributed compute nodes.

This approach fails to scale with the size of the tree, as every compute node retains the full tree in memory. As the size of the tree, or the size of the reference alignment for that matter, grows, memory requirements can quickly become the limiting factor. Large trees under the old parallelization approach are prone become inefficient because of swapping data between Random Access Memory (RAM) and the hard disk.

Additionally, RAxML-EPA performs model parameter and branch length optimization of the reference tree for every instance of the program, when in theory this only needs to be done once.

In contrast, the distributed design presented here allows to evenly distribute the reference tree among an arbitrary number of compute nodes. This includes a separation of the preparation phase, that performs reference tree optimization and precomputes the CLVs. This allows P-EPA to perform this task only once.

Section 5.2 will introduce the design in detail. First, however, in Section 5.1 I will outline the basic parallelization scheme by which the distributed, as well as the local parallelization operate.

5.1 PARALLELIZATION OVER EDGES

The basic parallelization approach employed for P-EPA is to split the reference tree into equally sized subsets of edges. The main idea that allows for this is the fact that, once the preprocessing step of the EPA (Section 3.1.1) has computed the CLVs for every edge in the tree, the edges themselves can be regarded as independent with respect to the placement procedure. The finest granularity that can be achieved is therefore to assign one edge of the reference tree to one thread in the multi-threaded version of the implementation.

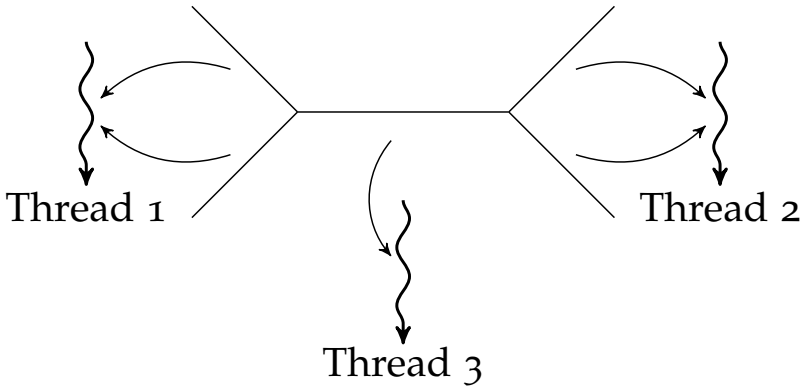


Figure 11: *Example of a possible mapping of reference tree edges to compute threads.*

RAxML-EPA [32] (Section 4.1) uses the same parallelization scheme in its implementation. The authors showed that the scheme is a good fit, and even achieves super-linear speedups.

While parallelization of the placement phase of the EPA (Section 3.1.2) is straight forward, the pre- and post processing phases are more challenging.

The approach described so far is implemented using the OpenMP [25] programming interface. OpenMP allows the user to write shared memory multiprocessing code for C/C++ or Fortran programs. It allowed the shared memory parallelization of the software presented here to be achieved with minimal programming effort, while also achieving speedups comparable to RAxML (see results in Section 6.3).

To cope with large numbers of query sequences, an automatic chunking approach is used. Query sequences are read into memory in fixed-size chunks. Each chunk is then fully processed by the program before their result is written to the jplace output file.

This decoupling of input, processing, and output forms a simple pipeline, allowing query sequences to be streamed into the placement core. Treating the input as a stream has the added benefit of allowing P-EPA to read partial outputs from the program producing the query MSA, allowing for further time savings.

For distributed memory systems, P-EPA employs a parallelization scheme that expands on the pipeline paradigm.

5.2 DISTRIBUTED PIPELINE

The basic implementation presented so far already parallelizes the placement phase of the EPA efficiently on shared memory systems. The use of automatic chunking enables it to scale almost arbitrarily with the number of input query sequences and RAM requirements. Parallelizing the placement algorithm over edges in the reference tree allows for scalability proportional to the size of the reference tree.

Memory scaling, however, is quickly limited as the size of the structures needed to represent the reference tree exceed the size of the memory in the shared memory system.

P-EPA introduces a parallelization scheme for distributed memory systems. In principle, the scheme mirrors that presented in Section 5.1: The tree is split into independent subsets of edges and distributed to computational nodes. In the distributed memory system, these nodes are compute nodes in a supercomputer or cluster (Section 2.7). Each of these nodes performs placement of all query sequences for the current chunk of the query MSA on its subset of the tree. I denote this collection of nodes that compute placements of sequences on edge subsets a *placement stage*.

To facilitate the distribution of edge subsets, the *preprocessing* phase of the EPA (Section 3.1.1) is performed ahead of the actual placement procedure. The result of this separate run, two CLVs per edge in the tree, is written to a file. This file can then be accessed by all assigned nodes in the cluster in parallel.

As discussed in Section 3.1.3, the placement results for a given sequence are subject to LWR computations and filtering. Herein also lies the biggest issue this scheme has to overcome, as at the end of the placement computation the results are fragmented by edge, while we want to *aggregate* them by sequence. Thus, the scheme assigns the task of collecting the results of a given sequence to a distinct set of nodes in the cluster called the *aggregation stage*.

After placement of a chunk of sequences on its subset of the tree, each compute stage node forwards the placement results of a given sequence to the aggregation stage node that is responsible for collecting the results of that sequence. This communication scheme is illustrated in Figure 12. It is important to note that while the aggregation stage computes the LWRs of a query, and filters the result, the placement stage can begin computing placements for the next chunk of sequences.

This forms the basis of the distributed pipeline. Figure 13 illustrates how the interplay of the stages forms the pipeline. Sequences enter the pipeline in chunks from the left as they are read by the nodes in the compute stage. The resulting placements are forwarded to the correct nodes in the aggregation stage, which filter the results and write them to the JPLACE output file.

When the prescoring heuristic (Section 3.2) is used, the design of the pipeline becomes more involved, as shown in Figure 14. The pipeline is extended to four stages: computing the placements on the whole tree without BLO, aggregating the results and selecting the candidate edges, thorough placement on the candidate edges, and finally aggregating, filtering, and outputting the placement results.

A key difference is that in the second round of placement computations, the amount of computation is not guaranteed to be dis-

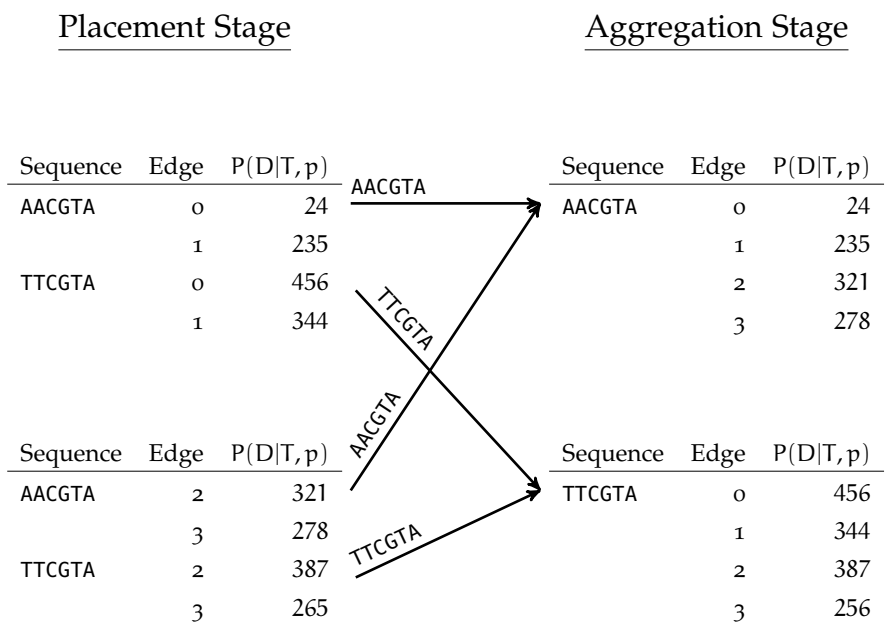


Figure 12: *Communication between the placement-, and aggregation stages in the pipeline.* In this example, two nodes in the placement stage are tasked with placing the sequences `AACGTA` and `TTCGTA` on their respective subsets of edges. Subsequent to placement, they send their results to the aggregation phase, here made up of two nodes. Each aggregation node in this example is responsible for computing the LWRs of one of the two sequences. Thus, communication between the stages constitutes a reduce operation.

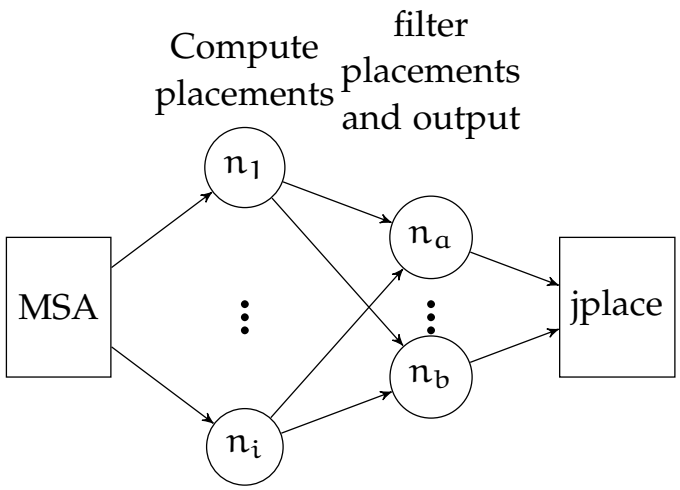


Figure 13: *Pipepline without prescoring.* n denote the nodes per stage, i being the number of nodes in the placement stage. a and b denote different sequence subsets, for which aggregation nodes n_a and n_b collect placement results. MSA and jplace denote input and output files.

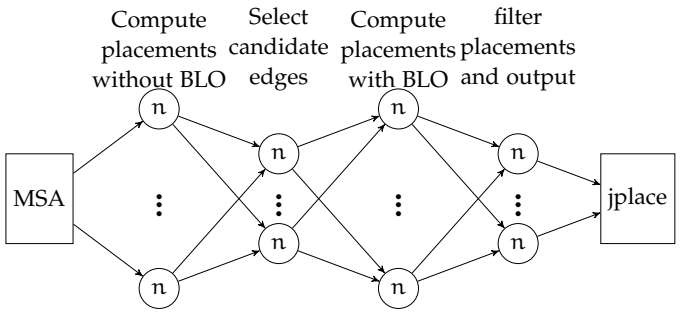


Figure 14: *Pipepline with prescoring.* Although in this depiction nodes are denoted simply by n and not explicitly numbered, typically each node will occupy a distinct compute node in the cluster. Further, the mapping of sequences to intermediate stages, as shown in Figure 13, persists here as well.

tributed equally across the tree, as only a small subset of edges per sequence will be evaluated. This complicates the otherwise straightforward mapping of edge subsets to placement stage nodes. Instead, nodes are now assigned specific combinations of edges and sequences for thorough placement. The assignment is done such that the workload is distributed equally among the nodes computing the thorough placement. At the same time, the assignment tries to "pin" edge subsets to compute nodes, such that frequent loading of the stored CLVs can be avoided.

The inherent challenge in using pipeline parallelism is to find a mapping of compute nodes to stages in the pipeline, such that the overall throughput of the pipeline is maximized. We call a given mapping of (compute) nodes to stages a *schedule*, and a schedule that maximizes the pipeline throughput as *optimal*. As we can expect the time it takes to compute a chunk of data in a stage of the pipeline to vary for different input data, a static schedule is not optimal. In the following, I describe a dynamic algorithm that approximates an optimal schedule by continuously measuring the throughput of each stage and periodically adjusting the schedule accordingly.

5.2.1 *A Scheduling Algorithm*

Let N be the set of compute nodes, linked by an interconnect network, available to the program. Let S be the set of stages in the pipeline. A schedule is a mapping $N \to S$. Let $s \in S$ be a stage in the pipeline and x_s be the number of nodes assigned to stage s in a given schedule.

There are three constraints that a schedule must fulfill.

1. compute nodes cannot be subdivided:

$$x_s \in \mathbb{N}^+ \tag{12}$$

2. every stage must be assigned at least one node:

$$1 \leqslant x_s \leqslant |N| \tag{13}$$

3. every node must be assigned to a stage:

$$\sum_{s \in S} x_s = |N| \tag{14}$$

In the beginning, the pipeline starts with some initial schedule. To dynamically measure the efficiency of the schedule, during operation, each node records the average time t it takes to process a chunk of input sequences. t is called the *turnaround time*. Periodically, these values are aggregated, and a stage-wide average of t can be computed, here denoted as t_s, with $s \in S$. From this, the *difficulty* of a stage can be computed as the time of a stage s relative to the fastest stage e:

$$d_s = \frac{t_s}{\min_{e \in S}(t_e)} \tag{15}$$

Based on this empirically determined d_s, we want to find a new schedule such that each stage s receives a number x_s of nodes proportional to its computational difficulty. This can be expressed as

$$x_s = d_s x_e \tag{16}$$

Having our choices of x_s constrained to integers, approximating an optimal schedule constitutes solving the following Integer Nonlinear Programming (INLP) optimization problem:

$$\begin{aligned} &\text{minimize} && \sum_{s \in S} (d_s x_e - x_s)^2 \\ &\text{subject to} && \sum_{s \in S} x_s = |N| \\ & && x_s \in \mathbb{N}^+, \forall s \in S \end{aligned} \tag{17}$$

The objective function of the optimization problem is the sum of the squared differences between the nodes in a stage compared to their difficulty. By minimizing it, we assure a schedule that, assuming past throughput of the pipeline stages predicts future throughput, schedules the nodes in such a way that the future throughput is maximized.

The periodicity of this reschedule operation is flexible. A high frequency of reevaluation may allow the pipeline to quickly adapt to changing load. However, as rescheduling requires flushing the pipeline, a high re-scheduling frequency negatively impacts the overall efficiency of the pipeline.

One possible strategy is to perform the re-scheduling in an exponential fashion: much more frequently in the beginning, in the hope that the pipeline quickly converges on an optimal configuration. This initial frequent re-scheduling is followed by longer and longer periods of uninterrupted operation during which the bulk of the work is processed.

In future work I hope to evaluate these, and other, scheduling approaches. I do not believe allowing the user to specify a default schedule is appropriate, as it would further complicate the use of the program.

6 EVALUATION

The implementation of the algorithms and parallel design presented in Chapter 3 and Section 5.1 was evaluated with respect to three key properties: validity of the results, serial runtime, and parallel efficiency.

Unfortunately, due to time constraints, I was unable to complete the implementation of the distributed pipeline, and its scheduling algorithm, presented in Sections 5.2 and 5.2.1. As a premature performance evaluation of the distributed approach would be meaningless, I hope to conduct a thorough evaluation in future work.

All tests are based on a subset of the query sequences used in [19]. As this is a recent real-world dataset, based on modern sequencing technologies, I assume it to be representative of the type of data that will be the typical input for P-EPA. Further, as I compare P-EPA's output to RAxML-EPA and PPLACER, both of which have undergone validation through simulation study, I assume empirical data to be sufficient.

The reference tree and alignment were used in full. Only a small subset of the query sequences was used to accelerate the evaluation. The sequences themselves remained unchanged.

A modified version of RAxML (v 8.2.4) was used in the tests. It was modified to not use empirical base frequencies. RAxML estimates empirical base frequencies using the entire MSA, including the reference and query sequences. This was disabled as P-EPA is built using a low-level interface of the Phylogenetic Likelihood Library (PLL) [11], which so far supports computing empirical frequencies from the reference alignment only.

PPLACER (v 1.1.alpha18) was built from source prior to conducting the tests. It did not have to be modified regarding empirical base frequencies, as model parameters can be supplied directly to the program. The supplied model parameters for the reference tree and MSA were the ones produced by the RAxML-EPA runs. The base frequencies were set to be equally distributed (set to 0.25 each) and the remaining model parameters were either the result of RAxML's optimization routine or its defaults, depending on the test.

6.1 VERIFICATION

To verify the correctness of P-EPA, I compared its output to the outputs of PPLACER [22] and RAxML-EPA [2]. Each program was run

on the same reference tree, reference alignment, and 1000 query sequences, taken from [19].

Three different tests were conducted on the same dataset with 1000 query sequences. In the *thorough test*, all programs were run without their respective heuristics, that is, they evaluated every edge using BLO. As noted in Section 4.2, the BLO used in PPLACER differs from the type used in RAxML-EPA and P-EPA.

A description of the heuristics can be found in Section 3.2 and Chapter 4. For PPLACER, disabling the baseball heuristics was achieved by setting the number of maximum strikes to 0 (`--max-strikes 0`).

In the *heuristic test*, both RAxML-EPA and P-EPA were invoked with the `-G` option. This option allows the user to specify a percentage fraction of the total number of edges that should be selected as candidates during the initial placement run. The fraction was set to 0.02 to ensure consistency across the tested programs. For PPLACER, the baseball heuristic was used and configured such that it behaved identical to RAxML-EPA's `-G 0.02` option. This was done by setting the strike box to 0 and the maximum number of pitches to 20 (0.02×1021, 1021 being the total number of branches in the reference tree).

Lastly, in the *adaptive prescoring* test, the adaptive prescoring method, introduced in Section 3.2, was compared to the results of RAxML-EPA and PPLACER from the previous, heuristic, test. As mentioned, adaptive prescoring stops selecting edges from the sorted list of candidates after an accumulated LWR threshold is reached. The threshold used here was 0.95, meaning at least 95% of the sum total likelihood weight of all placements is considered in the thorough re-evaluation. This is also the default value when using adaptive prescoring in P-EPA.

Subsequently, the K-R metric (Section 2.6) was used to calculate a pairwise distance between the placement results of two programs at a time. To achieve this for a given sequence, the K-R distance was calculated between the placements produced by program A and the placements produced by program B. This was done for each of the 1000 query sequences. The overall distance between the output of A and B was then calculated to be the mean between the distances for individual query sequences. The GENESIS toolkit [4] was instrumental in this procedure.

Figure 15 visualizes the result of the three tests. The three tested programs are shown in the corners of the triangle. The distance between the corners is the pairwise K-R distance.

As the K-R metric is a distance between placement distributions of a sequence, we can use it to establish a baseline between two programs that have already been determined to produce highly similar results [2, 22]. This is done by calculating the distance between the two programs, RAxML-EPA and PPLACER. We can then calculate the distances of the results of those programs to the results of P-EPA.

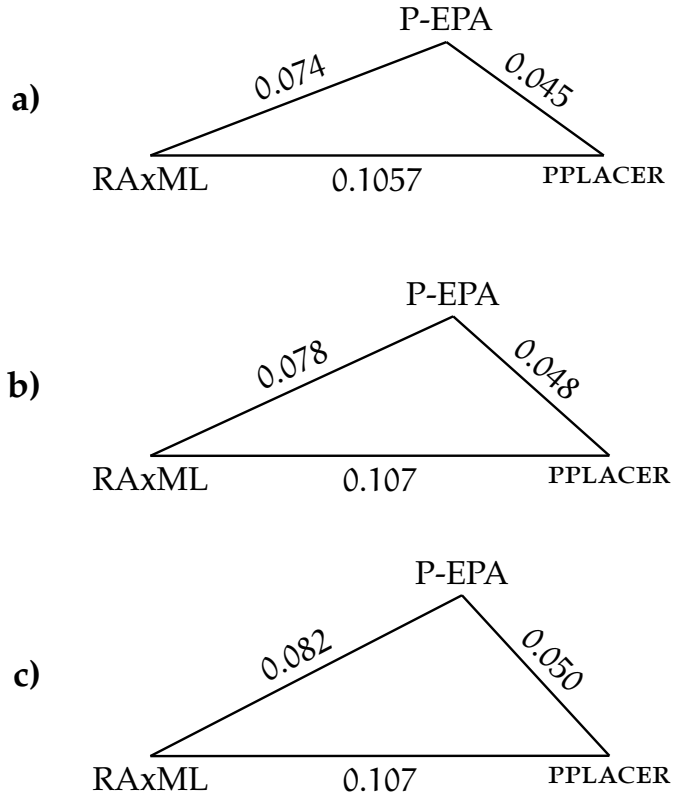


Figure 15: *Average query K-R distances (Section 2.6) between outputs of tested implementations.* The corners of the triangles are labeled by the programs that produced the output. a) shows the mean query K-R distances using the thorough, non-heuristic approach. b) shows the mean query K-R distances using the heuristic approach. c) is analogous to b), however the results of P-EPA used to compute the mean K-R distances were obtained using the adaptive prescoring heuristic (Section 3.2) with the default threshold value of 0.95.

Placing these distances in conjunction, as is done in Figure 15, we can visualize how similar or dissimilar our results are to the baseline.

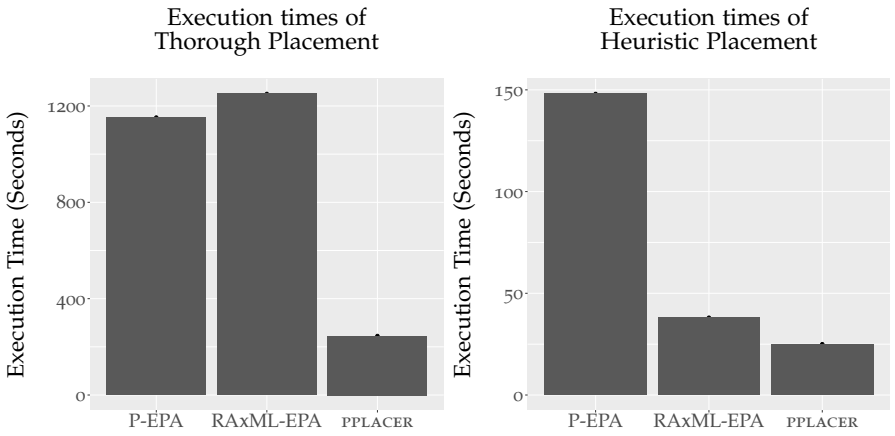
The results of these tests show, that my implementation produces placements that are highly similar to those of PPLACER and RAxML-EPA.

6.2 SERIAL RUNTIME

The serial execution times of RAxML-EPA, P-EPA, and PPLACER were evaluated, using the same reference tree, reference sequences and 1000 query sequences as used in the correctness tests (Section 6.1 and Figure 15). Figure 16 shows the results.

The configuration of the programs was identical to the correctness tests for both the thorough and heuristic placement calculations.

The tests show that, while for thorough placement P-EPA achieves an equivalent execution time as RAxML-EPA, further work is required



(a) (b)

Figure 16: *Comparision of execution times of different phylogenetic placement software.* Shown are the lowest execution times out of 10 repetitions for each program. The data used was the same as in the correctness tests (Section 6.1 and Figure 15). (a) compares the execution times for thorough placement of 1000 sequences. For pplacer, this was achieved by setting the number of strikes to 0. (b) shows the execution times of heuristic placement of the 1000 sequences. The programs were configured such that the 20 most likely edges for every sequence were evaluated in detail. For RAxML-EPA and P-EPA, this was achieved by supplying the `-G 0.02` option. This fraction derives from the fact that the reference tree used has 1021 edges. For pplacer, the options `-strike-box 0 -max-strikes 20` were used.

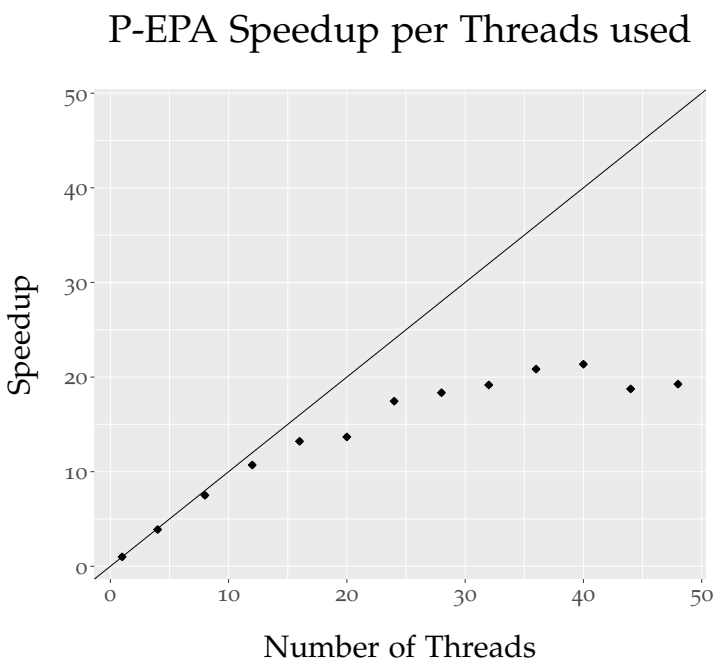


Figure 17: *Speedup of the multi-thread implementation.* Tests were conducted on a machine with 4 AMD Opteron™ 6174 processors, each having 12 physical cores. For each number of threads, 100 query sequences were placed on the reference tree from [19]. This was repeated 10 times each. The result shows the lowest execution time of those 10 runs.

to achieve the same for the heuristic placement mode. It also shows that PPLACER currently outperforms both implementations. After conferring with the authors of PPLACER, we believe this discrepancy to be due to the different way it performs BLO. Closing this execution time gap is the subject of future work.

6.3 EFFICIENCY

The efficiency of the multi-thread implementation (Section 5.1) was tested on a machine with 4 AMD Opteron™ 6174 processors, with 12 physical cores per processor. The machine had 256GB of RAM.

The dataset used in this test was again extracted from the data in [19]. For this test, however, only 100 query sequences were used at a time to limit the overall duration. The 100 query sequences were placed against the full reference tree consisting of 512 species, using the standard (thorough) placement approach (Section 3.1). This was repeated 10 times for every tested number of threads. Of those 10 runs, the shortest run time was taken as the data point, as slower runtimes on identical data are assumed to be due to outside influences.

From a performance standpoint, we are primarily interested in the best possible result, ignoring outside factors.

The results are shown in Figure 17, where the speedup compared to the single-thread implementation is plotted for every run.

The tests show, that P-EPA performs efficiently for a low to intermediate number of threads. A drop off in efficiency can be observed for a high number of threads. Most likely this is due to inefficiencies in distributing the work load using OpenMP, as the tested machine, having four processors, uses non-uniform memory access (NUMA).

7 SUMMARY

In this work, I presented P-EPA, a complete reimplementation of the EPA. I showed that P-EPA has an overall efficiency that is similar to that of previous implementations on shared memory systems. Additionally, I presented a design that will enable P-EPA to utilize computational resources in a distributed memory system. I also presented a method for load balancing the stages of the pipeline by empirical measurements and dynamic adaptations during execution. I presented the first parallelization scheme that can potentially with the size of the tree, potentially enabling phylogenetic placement on extremely large reference trees with billions of query sequences.

Further, I showed the validity of P-EPA by comparing its output to its main competitors: RAxML-EPA and PPLACER. My tests showed that my implementation produces placements that are highly similar to those computed by RAxML-EPA and PPLACER.

7.1 IMPLEMENTATION NOTES

Unfortunately, due to time constraints, I was unable to complete the implementation and testing of the distributed pipeline, as well as the load balancing algorithm, presented in Sections 5.2 and 5.2.1.

P-EPA was developed using C++11. It uses a low-level interface of the Phylogenetic Likelihood Library [11] for most of its phylogenetic likelihood computations, and file input/output operations.

A short command line interface manual for P-EPA can be found in Appendix A.1. The source code is available upon request, at `https://github.com/Pbdas/epa`.

7.2 FUTURE WORK

Section 6.2 showed several shortcomings of P-EPA in terms of execution time. One possible improvement would be the inclusion of PPLACER-inspired BLO. In general, more work in optimizing the core computations of the placement algorithm is needed.

Further, the distributed design I introduced in Section 5.2 still needs to be fully tested and refined. A part of further development in this regard should be the evaluation and further refinement of the scheduling procedure I presented alongside the distributed design in Section 5.2.1.

Other future work may include extending the program to work with a wider variety of input data, such as Amino Acid sequences.

More supported input file formats would be beneficial, too. These features may depend on their availability in the a low-level interface of the PLL, which I use extensively in P-EPA.

APPENDIX

A.1 P-EPA COMMAND LINE INTERFACE MANUAL

-h Display list of options

-t Path to the reference tree file

-s Path to reference alignment file. May include query sequences.

-q Path to query alignment file

-w Path to working directory.

Default: current working directory

-g Use prescoring heuristic with specified value acting as an accumulative LWR threshold beyond which no candidates are selected

Default: off. When used but no value specified the default is 0.95

-G Use prescoring heuristic with specified value acting as the fraction of edges in the reference tree which should be selected as candidate edges

Default: off. When used but no value specified: 0.1

-O Optimize branch lengths and model parameters of the reference tree

-l Filter final result by minimum LWR value below which placements are discarded

Default: 0.01

-L Filter final result by accumulative LWR value. When specified threshold is reached all other placements are discarded

Default: off

-m Specify nucleotide substitution model

Options:

GTR	General Time Reversible (Default)
JC69	Jukes-Cantor
K80	Kimura 80

-b Path to binary file produced by first running the program with the -B option. Performs placement on tree and alignment specified in said file

-B Binary dump mode: only builds the relevant internal structures, performs optimization (if -O was specified) and calculates all possible CLVs. Then dumps it all in a compact file format that can be read by the program using the -b option. Does not perform placement!

BIBLIOGRAPHY

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

[2] Simon A. Berger, Denis Krompass, and Alexandros Stamatakis. Performance, accuracy, and web server for evolutionary placement of short sequence reads under maximum likelihood. *Systematic Biology*, 60(3):291–302, 2011. doi: 10.1093/sysbio/syr010.

[3] Benny Chor and Tamir Tuller. Finding a Maximum Likelihood Tree is Hard. *Journal of the Association for Computing Machinery*, 53(5):722–744, September 2006. doi: 10.1145/1183907.1183909.

[4] Lucas Czech. genesis – A toolkit for working with phylogenetic data. Web page, accessed May 2016. URL `https://github.com/lczech/genesis`.

[5] Charles Darwin. *On the Origin of Species by Means of Natural Selection*. Murray, London, 1859. or the Preservation of Favored Races in the Struggle for Life.

[6] Steven N. Evans and Frederick 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.

[7] Joseph Felsenstein. Evolutionary trees from DNA sequences: A maximum likelihood approach. *Journal of Molecular Evolution*, 17 (6):368–376, 1981. doi: 10.1007/BF01734359.

[8] Joseph Felsenstein. Statistical Inference of Phylogenies. *Journal of the Royal Statistical Society. Series A (General)*, 146(3):246–272, 1983.

[9] Joseph Felsenstein. *Inferring Phylogenies*, volume 2. Sinauer Associates Inc., 2004.

[10] Joseph Felsenstein. The Newick file format. Web page, accessed May 2016. URL `http://evolution.genetics.washington.edu/phylip/newicktree.html`.

[11] T. Flouri, F. Izquierdo-Carrasco, D. Darriba, A.J. Aberer, L.-T. Nguyen, B.Q. Minh, A. von Haeseler, and A. Stamatakis. The Phylogenetic Likelihood Library. *Systematic Biology*, 2014. doi: 10.1093/sysbio/syu084.

[12] Centers for Disease Control. Outbreak of West Nile-Like Viral Encephalitis. Web page, accessed May 2016. URL `https://www.cdc.gov/mmwr/preview/mmwrhtml/mm4838a1.htm`.

[13] Bernard Friedenson. The BRCA1/2 pathway prevents hematologic cancers in addition to breast and ovarian cancers. *BioMed Central Cancer*, 7(1):1, 2007.

[14] Pablo A. Goloboff, Santiago A. Catalano, J. Marcos Mirande, Claudia A. Szumik, J. Salvador Arias, Mari Källersjö, and James S. Farris. Phylogenetic analysis of 73 060 taxa corroborates major eukaryotic groups. *Cladistics*, 25(3):211–230, 2009. doi: 10.1111/j.1096-0031.2009.00255.x.

[15] Daniel H. Huson, Alexander F. Auch, Ji Qi, and Stephan C. Schuster. MEGAN analysis of metagenomic data. *Genome Research*, 17(3):377–386, 2007. doi: 10.1101/gr.5969107.

[16] Xi-Yu Jia, Thomas Briese, Ingo Jordan, Andrew Rambaut, Han Chang Chi, John S Mackenzie, Roy A Hall, Jacqui Scherret, and W Ian Lipkin. Genetic analysis of West Nile New York 1999 encephalitis virus. *The Lancet*, 354(9194):1971 – 1972, 1999.

[17] Thomas H Jukes and Charles R Cantor. Evolution of Protein Molecules. *Mammalian Protein Metabolism*, 3(21):132, 1969.

[18] Samuli Lehtonen. Towards Resolving the Complete Fern Tree of Life. *PLoS ONE*, 6(10):1–6, 10 2011. doi: 10.1371/journal.pone.0024851.

[19] Frederic Mahe, Colomban de Vargas, David Bass, Lucas Czech, Alexandros Stamatakis, Enrique Lara, Jordan Mayor, John Bunge, Sarah Sernaker, Tobias Siemensmeyer, Isabelle Trautmann, Sarah Romac, Cedric Berney, Alexey Kozlov, Edward Mitchell, Christophe Seppey, David Singer, Elianne Egge, Rainer Wirth, Gabriel Trueba, and Micah Dunthorn. Soil Protists in Three Neotropical Rainforests are Hyperdiverse and Dominated by Parasites. *bioRxiv*, 2016. doi: 10.1101/050997.

[20] C Manichanh, L Rigottier-Gois, E Bonnaud, K Gloux, E Pelletier, L Frangeul, R Nalin, C Jarrin, P Chardon, P Marteau, J Roca, and J Dore. Reduced diversity of faecal microbiota in Crohn's disease revealed by a metagenomic approach. *Gut*, 55(2):205–211, 2006. doi: 10.1136/gut.2005.073817.

[21] Frederick A. Matsen. Phylogenetics and the Human Microbiome. *Systematic Biology*, 64(1):e26–e41, 2015. doi: 10.1093/sysbio/syu053.

[22] Frederick A. Matsen, Robin B. Kodner, and Virginia 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.

[23] Frederick a. Matsen, Noah G. Hoffman, Aaron Gallagher, and Alexandros Stamatakis. A format for Phylogenetic Placements. *PLoS ONE*, 7(2):1–5, 2012. doi: 10.1371/journal.pone.0031009.

[24] Jenna L. Morgan, Aaron E. Darling, and Jonathan A. Eisen. Metagenomic Sequencing of an In Vitro-Simulated Microbial Community. *PLoS ONE*, 5(4):1–10, 04 2010. doi: 10.1371/journal.pone.0010209.

[25] OpenMP Architecture Review Board. OpenMP Application Program Interface Version 3.0, May 2008. URL `http://www.openmp.org/mp-documents/spec30.pdf`.

[26] Junjie Qin, Ruiqiang Li, Jeroen Raes, Manimozhiyan Arumugam, Kristoffer Solvsten Burgdorf, Chaysavanh Manichanh, Trine Nielsen, Nicolas Pons, Florence Levenez, Takuji Yamada, et al. A human gut microbial gene catalogue established by metagenomic sequencing. *Nature*, 464(7285):59–65, 2010.

[27] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. The Earth Mover's Distance as a Metric for Image Retrieval. *International Journal of Computer Vision*, 40:99–121, 2000.

[28] F. Sanger and A.R. Coulson. A rapid method for determining sequences in DNA by primed synthesis with DNA polymerase. *Journal of Molecular Biology*, 94(3):441 – 448, 1975.

[29] Stephen A. Smith, Jeremy M. Beaulieu, Alexandros Stamatakis, and Michael J. Donoghue. Understanding angiosperm diversification using small and large phylogenetic trees. *American Journal of Botany*, 98(3):404–414, 2011. doi: 10.3732/ajb.1000481.

[30] Sujatha Srinivasan, Noah G. Hoffman, Martin T. Morgan, Frederick A. Matsen, Tina L. Fiedler, Robert W. Hall, Frederick J. Ross, Connor O. McCoy, Roger Bumgarner, Jeanne M. Marrazzo, and David 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, 06 2012. doi: 10.1371/journal.pone.0037818.

[31] Alexandros Stamatakis. *Parallel and Distributed Computation of Large Phylogenetic Trees*, pages 325–346. John Wiley & Sons, Inc., 2005.

[32] Alexandros Stamatakis. RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics*, 30:1312–1313, 2014.

[33] S. Tavaré. *American Mathematical Society: Lectures on Mathematics in the Life Sciences*, volume 17, chapter Some Probabilistic and Statistical Problems in the Analysis of DNA Sequences, pages 57–86. Amer Mathematical Society, 1986.

[34] J. Craig Venter, Mark D. Adams, Eugene W. Myers, Peter W. Li, Richard J. Mural, Granger G. Sutton, Hamilton O. Smith, Mark Yandell, Cheryl A. Evans, Robert A. Holt, Jeannine D. Gocayne, Peter Amanatides, Richard M. Ballew, Daniel H. Huson, Jennifer Russo Wortman, Qing Zhang, Chinnappa D. Kodira, Xiangqun H. Zheng, Lin Chen, Marian Skupski, Gangadharan Subramanian, Paul D. Thomas, Jinghui Zhang, George L. Gabor Miklos, Catherine Nelson, Samuel Broder, Andrew G. Clark, Joe Nadeau, Victor A. McKusick, Norton Zinder, Arnold J. Levine, Richard J. Roberts, Mel Simon, Carolyn Slayman, Michael Hunkapiller, Randall Bolanos, Arthur Delcher, Ian Dew, Daniel Fasulo, Michael Flanigan, Liliana Florea, Aaron Halpern, Sridhar Hannenhalli, Saul Kravitz, Samuel Levy, Clark Mobarry, Knut Reinert, Karin Remington, Jane Abu-Threideh, Ellen Beasley, Kendra Biddick, Vivien Bonazzi, Rhonda Brandon, Michele Cargill, Ishwar Chandramouliswaran, Rosane Charlab, Kabir Chaturvedi, Zuoming Deng, Valentina Di Francesco, Patrick Dunn, Karen Eilbeck, Carlos Evangelista, Andrei E. Gabrielian, Weiniu Gan, Wangmao Ge, Fangcheng Gong, Zhiping Gu, Ping Guan, Thomas J. Heiman, Maureen E. Higgins, Rui-Ru Ji, Zhaoxi Ke, Karen A. Ketchum, Zhongwu Lai, Yiding Lei, Zhenya Li, Jiayin Li, Yong Liang, Xiaoying Lin, Fu Lu, Gennady V. Merkulov, Natalia Milshina, Helen M. Moore, Ashwinikumar K Naik, Vaibhav A. Narayan, Beena Neelam, Deborah Nusskern, Douglas B. Rusch, Steven Salzberg, Wei Shao, Bixiong Shue, Jingtao Sun, Zhen Yuan Wang, Aihui Wang, Xin Wang, Jian Wang, Ming-Hui Wei, Ron Wides, Chunlin Xiao, Chunhua Yan, Alison Yao, Jane Ye, Ming Zhan, Weiqing Zhang, Hongyu Zhang, Qi Zhao, Liansheng Zheng, Fei Zhong, Wenyan Zhong, Shiaoping C. Zhu, Shaying Zhao, Dennis Gilbert, Suzanna Baumhueter, Gene Spier, Christine Carter, Anibal Cravchik, Trevor Woodage, Feroze Ali, Huijin An, Aderonke Awe, Danita Baldwin, Holly Baden, Mary Barnstead, Ian Barrow, Karen Beeson, Dana Busam, Amy Carver, Angela Center, Ming Lai Cheng, Liz Curry, Steve Danaher, Lionel Davenport, Raymond Desilets, Susanne Dietz, Kristina Dodson, Lisa Doup, Steven Ferriera, Neha Garg, Andres Gluecksmann, Brit Hart, Jason Haynes, Charles Haynes, Cheryl Heiner, Suzanne Hladun, Damon Hostin, Jarrett Houck,

Timothy Howland, Chinyere Ibegwam, Jeffery Johnson, Francis Kalush, Lesley Kline, Shashi Koduru, Amy Love, Felecia Mann, David May, Steven McCawley, Tina McIntosh, Ivy McMullen, Mee Moy, Linda Moy, Brian Murphy, Keith Nelson, Cynthia Pfannkoch, Eric Pratts, Vinita Puri, Hina Qureshi, Matthew Reardon, Robert Rodriguez, Yu-Hui Rogers, Deanna Romblad, Bob Ruhfel, Richard Scott, Cynthia Sitter, Michelle Smallwood, Erin Stewart, Renee Strong, Ellen Suh, Reginald Thomas, Ni Ni Tint, Sukyee Tse, Claire Vech, Gary Wang, Jeremy Wetter, Sherita Williams, Monica Williams, Sandra Windsor, Emily Winn-Deen, Keriellen Wolfe, Jayshree Zaveri, Karena Zaveri, Josep F. Abril, Roderic Guigó, Michael J. Campbell, Kimmen V. Sjolander, Brian Karlak, Anish Kejariwal, Huaiyu Mi, Betty Lazareva, Thomas Hatton, Apurva Narechania, Karen Diemer, Anushya Muruganujan, Nan Guo, Shinji Sato, Vineet Bafna, Sorin Istrail, Ross Lippert, Russell Schwartz, Brian Walenz, Shibu Yooseph, David Allen, Anand Basu, James Baxendale, Louis Blick, Marcelo Caminha, John Carnes-Stine, Parris Caulk, Yen-Hui Chiang, My Coyne, Carl Dahlke, Anne Deslattes Mays, Maria Dombroski, Michael Donnelly, Dale Ely, Shiva Esparham, Carl Fosler, Harold Gire, Stephen Glanowski, Kenneth Glasser, Anna Glodek, Mark Gorokhov, Ken Graham, Barry Gropman, Michael Harris, Jeremy Heil, Scott Henderson, Jeffrey Hoover, Donald Jennings, Catherine Jordan, James Jordan, John Kasha, Leonid Kagan, Cheryl Kraft, Alexander Levitsky, Mark Lewis, Xiangjun Liu, John Lopez, Daniel Ma, William Majoros, Joe McDaniel, Sean Murphy, Matthew Newman, Trung Nguyen, Ngoc Nguyen, Marc Nodell, Sue Pan, Jim Peck, Marshall Peterson, William Rowe, Robert Sanders, John Scott, Michael Simpson, Thomas Smith, Arlan Sprague, Timothy Stockwell, Russell Turner, Eli Venter, Mei Wang, Meiyuan Wen, David Wu, Mitchell Wu, Ashley Xia, Ali Zandieh, and Xiaohong Zhu. The sequence of the human genome. *Science*, 291(5507):1304–1351, 2001. doi: 10.1126/science.1058040.

[35] C. von Mering, P. Hugenholtz, J. Raes, S. G. Tringe, T. Doerks, L. J. Jensen, N. Ward, and P. Bork. Quantitative Phylogenetic Assessment of Microbial Communities in Diverse Environments. *Science*, 315(5815):1126–1130, 2007. doi: 10.1126/science.1133420.

[36] KA Wetterstrand. DNA Sequencing Costs: Data from the NHGRI Genome Sequencing Program (GSP). Web page, accessed May 2016. URL `https://www.genome.gov/sequencingcostsdata`.

DECLARATION

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht zu haben und die Satzung der Universität Karlsruhe (TH) zur Sicherung guter wissenschaftlicher Praxis beachtet zu haben.

Karlsruhe, May 2016

Pierre Barbera