# Phylogeny-Aware Placement and Alignment Methods for Short Reads

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

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

**genehmigte**

**Dissertation**

von

**Simon A. Berger**

aus München

| | |
|---|---|
| Tag der mündlichen Prüfung: | 14. 1. 2013 |
| Erster Gutachter: | Prof. Dr. Alexandros Stamatakis<br>Karlsruher Institut für Technologie |
| Zweiter Gutachter: | Prof. Dr. Christian von Mering<br>University of Zurich |

Hiermit erkläre ich, dass ich diese Arbeit selbständig angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht habe. Ich habe die Satzung der Universität Karlsruhe (TH) zur Sicherung guter wissenschaftlicher Praxis beachtet.

München 21. 01. 2013 _________________________

Simon Berger

# Zusammenfassung

Eine der wichtigsten Entwicklungen der letzten Jahre im Bereich der Molekularbiologie war die zunehmende Verbreitung von sogenannten DNA Sequenzierungstechniken neuer Generation (next generation sequencing; NGS). Diese Methoden können innerhalb kürzester Zeit mehrere Millionen sog. kurzer Sequenzabschnitte (short reads) generieren. Die dadurch entstandene Datenflut stellt die Bioinformatik vor neue Herausforderungen in Bezug auf die Auswertung dieser Daten. Um den aktuellen und künftigen Anforderungen gerecht zu werden, ist es daher notwendig, dass die verwendeten Algorithmen das Potential der zur Verfügung stehenden parallelen Rechnerarchitekturen voll ausnutzen.

Ein wichtiges Anwendungsgebiet der NGS Methoden sind metagenomische Studien, bei denen die DNA aus mikrobiellen Proben extrahiert und sequenziert wird, ohne dass von vornherein die genaue Zusammensetzung der in diesen Proben enthaltenen Arten bekannt ist. Ein wichtiger Analyseschritt besteht darin, die in einer Probe enthaltenen anonymen Arten anhand ihrer Sequenzinformation, beispielsweise durch Suche in einem Referenzdatensatz, zu identifizieren. Als Teil dieser Arbeit wurden Algorithmen für die phylogenetische Identifikation solcher short reads, also deren Analyse unter Berücksichtigung der evolutionären Zusammenhänge, entwickelt. Ein Ziel dieser Methoden besteht darin, die short reads in eine bestehende Referenzphylogenie (Stammbaum) zu platzieren.

Die zugrunde liegende Problemstellung lässt sich wie folgt formulieren: Die Ausgangsdaten bestehen zum einen aus einer bereits existierenden Referenzphylogenie, welche ein multiples Sequenzalignment (reference alignment; RA) und einen dazugehörigen phylogenetischen Baum (reference tree; RT) umfasst. Die Erstellung des RA und des RT ist nicht Teil dieser Arbeit; diese werden als gegeben angenommen. Zum anderen existiert eine, potenziell sehr große, Zahl von 100,000 - 10,000,000 short reads (query sequences; QS) die in den RT platziert werden soll. Die Einfügepositionen/Platzierungen der short reads entsprechen in diesem Fall einer Kante der Referenzphylogenie.

**Evolutionäre Platzierung von Short Reads** Die Berechnung einer derartigen Platzierung kann anhand des hier vorgestellten Evolutionary Placement Algorithm (EPA) vorgenommen werden. Dem EPA liegt die phylogenetische Maximum-Likelihood-Methode (ML) zugrunde, welche unter anderem zur Berechnung phylogenetischer Bäumen häufig verwendet wird. Generell erlaubt das ML-Verfahren den Vergleich verschiedener Phylogenien (Baumtopologien) gemäß ihrer Plausibilität auf Basis eines gegebenen Sequenzalignment. Der EPA basiert auf der Eigenschaft, dass jede mögliche Platzierung einer QS im RT einer Baumtopologie entspricht. Das ML-Kriterium ermöglicht somit, alternative Platzierungen einer QS anhand ihrer Plausibilität zu vergleichen. Die Platzierung der einzelnen QS im RT gibt Aufschluss über die evolutionären Verwandschaftsverhältnisse zwischen den im RT enthalten Organismen und den durch die QS repräsentierten Organismen.

Diese Methode weißt grundlegende Vorteile gegenüber einer identifikation, die ausschließlich auf Sequenzähnlichkeit basiert auf (beispielsweise unter Verwendung von BLAST). Insbesondere wenn der RT keine nahen Verwandten der QS enthält kann es irreführend sein anzunehmen, dass die Referenzsequenz mit der höchsten hnlichkeit auch der nächste Verwandten einer QS ist. Idealerweise kann der EPA in einem solchen Fall erkennen, dass die QS und einige im RT enthaltenen Organismen einen gemeinsamen Vorfahren haben. Die Genauigkeit der Methode wurde experimentell auf Basis von 8 bestehenden Phylogenien und Sequenzalignments von realen biologischen Daten ermittelt. Dazu wurden einzelne Organismen aus der Refernzphylogenie entfernt und anschließend als sogenannte emulierte QS mit dem EPA und einem nicht phylogenie-basierten Vergleichsverfahren wieder in den Baum platziert. Vor der Platzierung wurden die emulierten QS anhand verschiedener Verfahren verkürzt um die von NGS Methoden generierten short reads zu simulieren. Da unter Anwendung dieses Verfahrens a priori bekannt ist, aus welcher Kante im Baum die QS entfernt wurde (Referenzposition), kann nach deren Platzierung die Distanz zwischen der Einfügeposition und der wahren Referenzposition ermittelt werden (beispielsweise als Länge des Pfades im Baum von der Referenzposition zur errechneten Einfügeposition). Im Vergleich zu evolutionär agnostischen Verfahren konnten die Platzierungsgenauigkeiten um den Faktor 1.12 bis 2.06 verbessert werden.

Die algorithmische Komplexität des EPA ist durch das zugrundeliegende statistische Modell höher als bei rein sequenzbasierten Verfahren. Daher stellt die Laufzeitoptimierung einen wichtigen Teil der Arbeit dar. Der EPA wurde durch Multithreading für aktuelle shared-memory Mehrkern- und Mehrprozessor-Systeme optimiert. Der EPA wurde als integrativer Teil der populären phylogenetischen Software RAxML entwickelt und kann beispielsweise auf die bestehenden hochoptimierten Routinen zur Berechnung der

Likelihood zurückgreifen. Diese Routinen nutzen auch SSE3 und AVX Vektorinstruktionen. Des weiteren wurden Heuristiken entwickelt, um die Anzahl der untersuchten Einfügepositionen einzuschränken, und damit einhergehend die zeitaufwändige Platzierung anhand des ML-Kriteriums weiter zu beschleunigen.

Der EPA ist grundsätzlich nicht auf die Verarbeitung von genetischen Sequenzdaten beschränkt, sondern kann beispielsweise auch zur Platzierung von Organismen anhand morphologischer Merkmale (Sichtbare Merkmale z.B. an Knochen) verwendet werden. Damit ist es möglich Fossile, für welche in den meisten Fällen keine genetischen Sequenzdaten vorliegen, in eine haupstächlich anhand genetischer Daten berechnete Phylogenie bestehender Arten zu platzieren. In diesem Zusammenhang wurden neue Kalibrierungsmethoden zur Verbesserung der morphologiebasierten Platzierung von Fossilien in Referenzbäumen entwickelt und getestet.

**Alignierung von Short Reads** Die oben beschriebene Platzierung von QS setzt voraus, dass die QS in einem ersten Schritt zum bestehenden RA hinzualigniert werden. Alignieren bedeutet, dass zueinander homologe (d.h., von einem gemeinsamen Vorfahren abstammende) Sequenzabschnitte einander zugeordnet werden. In diesem Fall werden die Sequenzen des RA als bereits (korrekt) aligniert angenommen, so dass die QS zum bestehenden RA hinzugefügt werden. Es handelt sich dabei also um ein Alignment zwischen einem multiplen Alignment und mehrerer einzelner Sequenzen. Im Rahmen dieser Arbeit wurde zu diesem Zweck ein spezialisierter Algorithmus entwickelt, der neben den Sequenzdaten auch die im RT enthaltene phylogenetische Information nutzt (Phylogeny Aware Parsimony Based Short Read Alignment; PaPaRa). Das Verfahren basiert darauf, dass für mehreren Sequenzen ein evolutionäres Sequenzprofil erstellt wird, gegen welches die QS einzeln aligniert werden.

Bei PaPaRa werden diese Profile unter Zuhilfenahme der Phylogenie erstellt: Innere Knoten im RT entsprechen möglichen Vorfahren der heutigen Spezies, wobei nur Sequenzdaten für die externen Knoten (heutige Spezies) bekannt sind. Die (unbekannten) Sequenzen der Vorfahren werden aus den bekannten Daten rekonstruiert. Es ist anzumerken, dass Teile der anzestralen Sequenz aufgrund des Vorhandenseins mehrerer Mutationen nicht eindeutig bestimmt werden können. Die hierbei entstehenden Mehrdeutigkeiten können im Profil entsprechend dargestellt werden. Das Verfahren erlaubt es also, QS gegen die (nicht eindeutig bekannten) anzstralen Sequenzen der Vorfahren zu alignieren. Wie bereits beim EPA besteht auch hier der Vorteil darin, dass man nicht darauf angewiesen ist, einen, unter Umständen inexistenten, Referenzdatensatz mit nahen Verwandten der QS zur Verfügung zu haben.

Zur Rekonstruktion der Profile benutzt PaPaRa die Maximum-Parsimony Methode (MP). Der eigentliche Alignmentalgorithmus verwendet die, in der Sequenzalignierung weit verbreitete, dynamische Programmierung. Neben der Sequenzinformation beinhalten die Profile auch Informationen über die im RA enthaltene Verteilung von Gaps (Lücken). Das Verfahren rekonstruiert dabei die mögliche Verteilung der Gaps in den Vorfahren. Zur rekonstruktion werden zwei Methoden (PaPaRa Version 1.0 und eine Weiterentwicklung in Version 2.0) zur Modellierung der Gapverteilung eingeführt.

Im Vergleich zu phylogenetisch agnostischen Verfahren (man kann ein Profil auch auf basis des gesamten multiplen Alignments erstellen, wie z.B. bei Profile Hidden-Markov Modellen (profile-HMM) der Fall), erhöht PaPaRa die Genauigkeit der anschließenden Platzierung mittels des EPA wegen der verbesserten Alignmentqualität. Die Genauigkeit wurde hier, in Analogie zur Evaluierung des EPA, anhand emulierter QS bestimmt. Hierbei wurden zusätzlich typische, technisch bedingte, Sequenzierungsfehler in die verkürzten QS eingeführt. Auf den emulierten QS konnte die Platzierungsgenauigkeit des EPA im Vergleich zu profile-HMM Methoden um bis zu Faktor 5.8 verbessert werden.

Wie bereits beim EPA, ist die algorithmische Komplexitaet von PaPaRa höher als bei den existierenden evolutionär agnostischen Verfahren. Auch PaPaRa nutzt Multithreading zur Laufzeitreduzierung. Eine zusätzliche Laufzeitreduzierung um Faktor 12 wurde anhand einer SSE Vektorisierung erreicht.

**Fazit** Die in dieser Arbeit eingeführten Algorithmen EPA und PaPaRa bieten Vorteile bei der phylogenetischen Analyse von short reads. Trotz ihrer vergleichsweise hohen Laufzeitanforderungen konnte gezeigt werden, dass sie in der Praxis für grosse Datensätze besser geeignet sind.

Eine mögliche Entwicklungsperspektive für die Algorithmen geht über deren Anwendung auf short reads hinaus. Eine grundlegend andersartige Anwendung könnte in deren Anwendung auf die Neuberechnung phylogenetischer Bäume liegen. Der Standardansatz ist dabei, zuerst eine Menge von Sequenzen untereinander zu alignieren, und anschließend auf diesen einen Baum zu berechnen (etwa mit RAxML anhand des ML Kriteriums). PaPaRa und EPA könnten in diesem Zusammenhang zur Erweiterung eines bestehenden Baumes um neue Spezies verwendet werden, ohne dass das Alignment und der Baum vollständig neu berechnet werden müssten. Dies könnte einen wesentlichen Schritt in Richtung einer engeren Kopplung von multiplem Sequenzalignment und Baumrekonstruktionsmethoden darstellen.

# Acknowledgements

I would like to thank a couple of people who have contributed to this thesis. First of all, special thanks go to my supervisor Prof. Alexis Stamatakis for giving me the opportunity to work in his group. I am especially grateful for setting up an environment of trust, freedom and openness to pursue my research, while at the same time providing valuable input and direction. I would also like to thank Prof. Christian von Mering, who agreed to be the 2nd reviewer on this thesis, and who also provided valuable input on various parts of my research. Furthermore I would like to thank Prof. Stefan Kramer and Prof. Burkhard Rost for hosting me at Technical University in Munich (TUM) and for providing access to their technical infrastructure.

My thanks also go to all my collaborators and co-workers inside and outside our group. Three of them I would like to mention especially are Nikos Alachiotis, long-time office mate and fellow Ph.D. student from the very beginning, who worked on FPGA and GPU implementations of PaPaRa, Denis Krompass, who implemented the EPA web-server and Zsolt Komornik, who helped implementing the parallel version of the EPA.

Finally, I would like to thank my friends and family for everything outside research.

My position was funded under the auspices of the Emmy-Noether program by the German Science Foundation (DFG), grant STA 860/2.

## 0. Acknowledgements

# Contents

CHAPTER 1

# Introduction

This chapter provides the motivation for conducting research on phylogeny aware methods for analyzing short sequence reads. It summarizes the scientific contribution and describes the overall structure of this thesis.

## 1.1 Motivation

In recent years bioinformatics has entered and exciting new phase: New sequencing methods, generally referred to as 'Next Generation Sequencing' (NGS) have become widely available. They have increased the amount of available sequence data by several orders of magnitude. Due to enormous cost savings, NGS is now commonly available in research labs as well as at the industrial level in the form of sequencing services (e.g., in June 2012, Illumina announced a whole genome sequencing service `http://investor.illumina.com/phoenix.zhtml?c=121127&p=irol-newsArticle&id=1706799`). Moreover, spurned by the economic success of available NGS methods, research on alternative, more powerful, sequencing methods is a high priority goal for the academic field as well as commercial vendors. There exist new sequencing technologies based on direct observation of RNA translation [22] and direct electrostatic characterization of single DNA chains [50]. These developments could lead to broad availability of 'personal sequencing' devices that can be used in the field. In February 2012, Oxford Nanopore announced the first miniaturized, portable sequencing device `http://www.nanoporetech.com/news/press-releases/view/39`. The consequences of these developments are obvious: The amount of sequence data acquired per experiment will increase with the technological advances while, at the same time, the size of the sequence data bases will grow exponentially. Clearly, handling and analyzing these amounts of data is a non-trivial task: In 2010, for example, the Bejing Genomics Institute (BGI) already generated 10 terabytes of raw sequencing data every 24 hours and, for analysis and assembly, relied on a 500 node compute cluster [69].

Another technology, that has experienced exponential growth is the semiconductor industry. 'Moore's Law', introduced in 1965, correctly predicts that the complexity of integrated circuits (e.g., in CPUs) doubles roughly every 18 months. Importantly, over a long time, the increasing complexity used to be directly proportional to increasing performance, due to higher clock frequency and improved efficiency per clock-cycle. This growth has been a convenient solution for many technical problems, requiring a large amount of computing power. Since 2005, the impact of the complexity increase has gradually changed: While Moore's Law still applies, it does no longer translate into a direct, automatic, performance improvement at the same magnitude as before. Now, there is a general trend towards providing parallel computing resources. Instead of a single, faster core, new CPU generations offer more than one core at the same, or slightly increased, speed as preceding generations. The main performance increases are mostly due to parallel computing resources: multi-core CPUs, multi CPU systems, distributed memory compute clusters, SIMD vector units and specialized accelerator architectures like graphics processing units (GPUs). A thorough review of past and current developments in this field is provided in `http://herbsutter.com/welcome-to-the-jungle/`.

These two technological trends, the 'biological data flood' caused by NGS methods and the 'end of the free lunch' caused by diminishing *automatic* performance increases induced by new computer hardware currently have a big influence on the way problems in bioinformatics are being solved. If bioinformatics embraces the 'biological data flood' it needs to cope with the 'end of the free lunch' at the same time. This is especially true for Metagenomic studies of microbial communities. These studies deal with the problem of analyzing genetic material from environmental samples. Therefore, they often yield a large amount of short-read sequences, whose taxonomic provenance is unknown. Here, the first step in metagenomic studies consists in identifying the biological identity of the reads. Such an assignment of short reads to known organisms allows for analyzing and comparing microbial samples and communities (see [91]).

This thesis will introduce algorithms for phylogeny aware analysis of short sequence reads, as generated by NGS methods in the context of metagenomic studies. In particular it deals with two important steps for phylogeny aware short read analysis: (1) aligning the short reads against a known multiple reference alignment, and (2) placing the reads into a known reference tree. Because of the aforementioned challenges induced by the multi-core revolution, a considerable part of this work focuses on the technical (w.r.t. performance) challenges of these new algorithms, beyond the biological modeling. In detail, the algorithms have been developed specifically to exploit parallelism on multi-threaded, SIMD vector, and GPU architectures.

## 1.2 Scientific Contribution

The initial working title of this thesis was “Simultaneous Tree Building and Multiple Sequence Alignment”. This title encompasses two important fundamental principles of computational molecular biology: The creation of multiple sequence alignments (MSA), that is, the process of aligning homologous characters in sequences to each other, and, based on such a multiple alignment, inferring a biologically meaningful phylogeny. During the curse of the project, the research was focused onto a related, but more narrowly defined field: Individual placement of a large number of (new) sequences into an existing phylogeny. As already mentioned, one of the most fundamental recent changes in computational molecular biology is the wide availability of NGS methods. These methods can increase the amount of data generated by a single sequencing run by orders of magnitude (compared to traditional methods) at the same or even lower cost. Thus, an emerging challenge in bioinformatics is to develop tools that can analyze this amount and type of data. The main contribution of this thesis are two novel algorithms that allow for phylogeny-aware analysis of NGS data: The Evolutionary Placement Algorithm (EPA) is a method designed to place a, potentially huge, number of sequence reads into a given reference phylogeny. While not limited to NGS reads, their analysis is a natural application for the EPA. The Phylogeny Aware Parsimony based Short Read Alignment (PaPaRa) Algorithm, was designed for aligning individual sequence reads against an existing MSA and reference phylogeny. More specifically, PaPaRa does not treat the MSA as a monolithic entity, but also uses information from the associated phylogenetic tree to carry out a more informed alignment process. In summary, PaPaRa and the EPA can be used as a complete pipeline for phylogenetic analysis of short sequence reads.

Because both methods target NGS data, performance optimization constitutes an important part of this work: The EPA is based on the phylogenetic likelihood function, which has a considerable computational cost. Thus, another important part of this work focuses on parallelization and vectorization of the underlying computational kernels. The PaPaRa algorithm on the other hand is based on a pair-wise sequence alignment kernel using a dynamic programming algorithm. Dynamic programming alignment is a computationally heavy task per se (the run-time complexity is quadratic w.r.t. to the input data size). Furthermore, PaPaRa targets a huge amount of sequences produced by NGS, hence the algorithm needs to scale on large input data sets. As with the EPA, low-level technical optimization represented an important part of the work.

The scientific work of this thesis has been conducted over the curse of more than three years. The algorithms and results presented here have been published in 4 journal articles [6, 12, 17, 19], 3 peer-reviewed conference papers [14, 15, 82] and one non-reviewed report [13]. Research on other topics related to HPC and bioinformatics not covered by this thesis was published in 7 articles: These articles covered work on low-level thread-synchronization methods [16], offloading computational kernels to field programmable gate arrays (FPGAs) [4, 5, 11], MPI parallelization of a novel graph clustering algorithm [76], use of the EPA by other groups [85] and integration of the vectorized phylogenetic likelihood kernel in RAxML-Light [80].

## 1.3 Structure of this thesis

This thesis is structured around the two novel algorithms described in Chapters 3 and 5: The EPA and PaPaRa. Chapter 4 describes an extension of the EPA to non-molecular (morphologic) data. A general introduction to methods on phylogenetic trees is given in Chapter 2. Following the method descriptions in Chapters 3, 4 and 5, experimental evaluations of the methods are presented in Chapter 6. Finally, Chapter 7 concludes the thesis and discusses relevant future improvements to the introduced methods. It also briefly discusses the possibilities of integrating the concepts of the EPA and PaPaRa as a possible way towards the goal of simultaneous phylogenetic tree inference and multiple sequence alignment.

CHAPTER 2

# Methods on Phylogenetic Trees

The present chapter provides an overview of the basic concepts of phylogenetic trees and their application in biology, as well as a quick introduction to the mathematical models used to infer them. Section 2.1 gives a short introduction to the underlying concepts of phylogenetic trees. Section 2.2 introduces optimality criteria under which phylogenetic trees can be evaluated.

## 2.1 Basic Tree Concepts

A Phylogenetic tree is one possible representation of evolutionary relationships. As such, it is commonly used to represent genealogical relationships among, for example, species or genes. While phylogenetic trees are widely used in practice, they do by no means represent *the only* model for representing evolution. Phylogenetic networks, which can be seen as a generalization of phylogenetic trees, strive to integrate events like hybridization, horizontal gene transfer, recombination, and reassortment, which can normally not be modeled by a simple tree [45]. Still, phylogenetic trees are not yet fully understood, and the extension to a more complex model would further increase the problem complexity. The methods described here are targeted at handling the ever-growing amount of sequence data. At the same time, the amount of easily available computational speed is increasing at a much slower pace than in the past. We will therefore not discuss the relative merits of the competing evolutionary models, and exclusively concentrate on tree based methods instead.

In a mathematical context, a tree is defined as a connected graph without cycles. A graph in turn, is a set of *vertexes* connected by a set of *edges*. In a biological context, vertices are commonly refered to as *nodes* and the term *branches* is often interchangeable with *edges*. In the same context, *external* nodes are often called *tips* or *leaves*. In a phylogenetic tree, these nodes often stand for a present-day species (*taxon*). Normally, some form of sequence data (i.e., DNA or amino acid data) is available at the tips of the tree. The internal nodes, on the other hand, commonly represent extinct hypothetical ancestors, for which no sequence data is available.

Depending on the underlying modeling (see below) and experimental requirements, a phylogenetic tree can be either *rootet* or *unrooted*. In a rooted tree, the ancestor associated with the root-node can be interpreted as the common ancestor of all sequences in the tree. Often, an unrooted tree can be transformed into a rooted tree by turning one of the internal nodes into the root. There exist several methods for selecting the root. Under the *molecular clock* assumption, the root can be selected using distance matrix and maximum likelihood methods. Applicability of molecular clock rooting depends an whether the rate under which the sequences evolved has been constant over time. If this is not the case, *outgroup rooting* can be applied as an alternative. This type of rooting inserts one or more distantly related species (the outgroup) into the phylogenetic tree. A new node — the root of the tree — is then inserted into the edge leading to the outgroup.

Depending on the type of tree, it can contain two distinct pieces of information: the branching pattern of the tree defines the tree topology. The topology is a model for the grouping of species within the tree: a common interpretation is to consider species in a common subtree as descendants of a common ancestor that is located at the root of the subtree. Secondly, the lengths of the edges can carry additional information about the amount of sequence divergence or a time period associated with the edge. Trees containing only the topological information without edge lengths are called *cladograms*. Tree with both, topological information and egde lengths, are called *phylograms*.

**Tree Search** The basic goal of phylogenetic tree inference is to infer a tree that is a plausible representation of evolutionary relationships between the species under study. A common way to achieve this goal algorithmically, is to search for a tree that is optimal according to some optimality criterion (see Section 2.2). For the time being, let us assume that an optimality criterion simply is a function that maps a phylogenetic tree (depending on the method this can either be a phylogram or a cladogram) and the associated sequence data of its taxa to a numeric value. Once an optimality criterion is established, it is possible to compare different phylogenetic trees or evolutionary hypotheses to each other (i.e., tree A is better than tree B). Such a criterion thus allows to search for the best tree (note however that, depending on the optimality criterion it is possible that there is not just a single best tree).

As already mentioned, phylogenetic tree inference can be algorithmically described as searching the space of all possible phylogenetic trees with the goal of finding the best-scoring tree. As with many search problems, the optimal phylogenetic tree can, in theory, be found using an exhaustive search: one simply has to generate all possible tree topologies and score all of them. The best-scoring tree can then be returned. However, in the real world executing such an exhaustive search is not feasible: Even when edge lengths are not part of the optimality criterion, the number of possible (unrooted) tree topologies increases super-exponentially with the number of taxa. Incorporating edge lengths in the scoring function further complicates the problem: In addition to the vast number of tree topologies, each edge in each of the topologies can be assigned an arbitrary length. Generally, phylogenetic tree inference under ML has been shown to be NP-hard [24].

In order to be applicable to real world data set sizes, practical tree search algorithms have to use heuristics to explore only a small, promising part of the search space in reasonable time. Commonly used heuristic methods like *nearest-neighbor interchange* (NNI) and *subtree pruning and regrafting* (SPR) [99] do not enumerate all possible tree topologies, but rather incrementally explore the search space from a given starting configuration. The concept of restricting the search-space will be revisited in Chapter 3, where we introduce an algorithm to rapidly place a (large) number of short sequence reads into a fixed reference tree. The following section will briefly introduce the optimallity criteria that are relevant for the algorihms indroduced in Chapters 3 and 5.

## 2.2 Optimality Criteria

As described above, one underlying principle of phylogenetic inference is to use some optimality criterion to score different candidate trees. This Section provides an overview of commonly used optimality criteria for phylogenetic inference: Maximum parsimony (MP) and maximum likelihood (ML). Theses two techniques are fundamental prerequisites for the novel algorithms introduced in Chapter 3 and Chapter 5: The representation of ancestral states used in MP is the basis for the ancestral sequence profiles used in PaPaRa [17]. The ML criterion is used to place reads into branches of the given reference tree in the evolutionary placement algorithm (EPA) [12].

### 2.2.1 Phylogenetic Likelihood

Maximum likelihood is a statistical technique used in a wide range of scientific fields. It is defined as the probability to observe data under a certain set of model parameters. The likelihood is a function of the model parameters and the input data. In the case of a phylogenetic tree, the data consist of the aligned homologous sequences (i.e., the known DNA or amino acid sequences associated with the living species). It is fundamental to understand that likelihood values can only be compared to each other, when the underlying data (the sequences *and* their multiple alignment) are identical. To compute the likelihood, a statistical model that describes how the data are generated is required.

In case of the phylogenetic likelihood, we first need a model that provides the probability of one sequence $S_1$ evolving into another sequence $S_2$ within time $t$. Also it is assumed that the aligned sites of the sequences evolve independently. Under this restriction the final score of a tree can be obtained by calculating the probabilities individually and independently on a per site basis and then calculating the product over all per-site probabilities. Considering DNA/RNA data (with the character alphabet A, C, G and T), the probability of a character $i$ evolving into another character $j$ in time $t$ is given by the transition probability function $P_{i,j}(t)$, where $i, j \in \{A, C, G, T\}$. For this function, a Markov-process is assumed, such that the value of $P_{i,j}(t)$ is independent of the prior evolutionary history of character $i$. Furthermore, the Markov-process should be *reversible*, which basically means that, the evolutionary process is the same whether the time runs *forward* or *backward*. The reason why reversibility is important is explained further down in this section. The transition probability function can be generated according to a collection of different models such as the Jukes-Cantor (JC69 [48]) or the general time-reversible (GTR [88, 97, 100]) model. A complete and much more detailed review of available sequence evolution models is provided in Chapter 1.1 of [99].

For the example tree in Figure 2.1, the likelihood could be calculated as the product of the individual transition probabilities along the tree branches multiplied with the prior probabilities $\pi_{S9}$ at the root node (the prior probabilities are discussed further below):

$$\begin{aligned} L = \pi_{S9} P_{S9,S8}(b7) P_{S8,S6}(b5) P_{S6,S1}(b1) P_{S6,S2}(b2) P_{S8,S3}(b6) \\ \times P_{S9,S7}(b8) P_{S7,S4}(b3) P_{S7,S5}(b4) \end{aligned} \tag{2.1}$$

As mentioned above, $S6$, $S7$, $S8$ and $S9$ actually correspond to the hypothetical sequences of ancestral taxa. In reality, the actual ancestral sequences

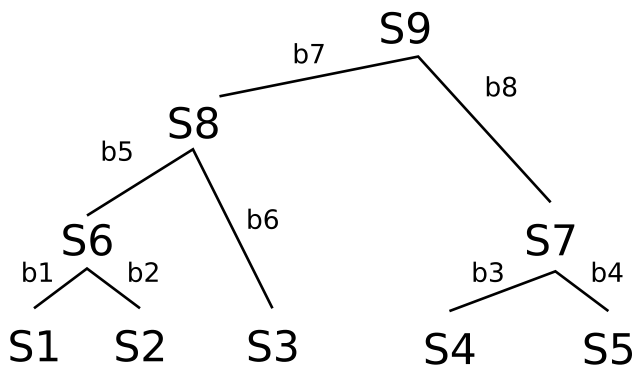


Figure 2.1: Example tree for calculating the phylogenetic likelihood function.

are unknown. Instead of using the ancestral sequences, we instead sum over all possible states at the inner nodes to derive the transition probabilities (Equation 2.2).

$$L = \sum_{S6=A}^{T}\sum_{S7=A}^{T}\sum_{S8=A}^{T}\sum_{S9=A}^{T} [\pi_{S9} P_{S9,S8}(b7) P_{S8,S6}(b5) P_{S6,S1}(b1) P_{S6,S2}(b2) \times P_{S8,S3}(b6) P_{S9,S7}(b8) P_{S7,S4}(b3) P_{S7,S5}(b4)] \tag{2.2}$$

One problem with the summation in Equation 2.2 is that, for the $s-1$ inner nodes we have to sum over all $4^{s-1}$ possible state assignments to the inner nodes. It is clear that the apparent exponential increase in the number of addition operations makes this method infeasible for larger trees. Fortunately, by applying *Horner's* rule (*nesting rule*), the number of required additions can be drastically reduced. In Equation 2.3, the summations are 'pushed' to the right as far as possible, according to this rule.

$$L = \sum_{S9=A}^{T} \pi_{S9} \left[ \sum_{S8=A}^{T} P_{S9,S8}(b7) \left[ \left[ \sum_{S6=A}^{T} P_{S8,S6}(b5) P_{S6,S1}(b1) P_{S6,S2}(b2) \right] P_{S8,S3}(b6) \right] \right] \times \left[ \sum_{S7=A}^{T} P_{S9,S7}(b8) P_{S7,S4}(b3) P_{S7,S5}(b4) \right] \tag{2.3}$$

Note that, the grouping of the tip sequences ($S1$, $S2$, $S3$, $S4$ and $S5$) in the structure of the parentheses follows the structure of the tree: ((($S1$, $S2$), $S3$), ($S4$,$S5$)). This means that the sequence of likelihood calculations for a given tree can be determined automatically in analogy to Equation 2.3. This technique is commonly referred to as Felsenstein's *pruning algorithm* [31]. Essentially it is a dynamic programming algorithm, which recursively calculates and memoizes the probabilities at the root of subtrees. Let $L_i(x_1)$ be the conditional probability of observing data at the tips of the descendants of node $i$, assuming that the nucleotide at node $i$ is $x_i$. If node $i$ is a tip then $L_i(x_i)$ is 1.0, if $x_i$ is the actual nucleotide in the known sequence. Otherwise the probability is 0.0. If $i$ is an inner node with child nodes $j$ and $k$, then $L_i(x_i)$ is derived as follows.

$$L_i(x_i) = \left[\sum_{x_j=A}^{T} P_{x_i x_j}(b_j) L_j(x_j)\right] \times \left[\sum_{x_k=A}^{T} P_{x_i x_k}(b_j) L_k(x_k)\right] \tag{2.4}$$

The overall per-site likelihood $L$ of the tree can then be calculated from the conditional probabilities $L_r$ at the root node $r$ and the prior probabilities $\pi_A$, $\pi_C$, $\pi_G$ and $\pi_T$ according to Equation 2.5. The prior probabilities correspond to the probability of observing the four bases A through T at the root node, which are usually drawn empirically from the input data. A complete description of the phylogenetic likelihood calculation is given in Chapter 4 of [99].

$$L = \sum_{x_r=A}^{T} \pi_{x_r} L_r(x_r) \tag{2.5}$$

Note that the calculation of the likelihood according to the principle described above depends on a rooted representation of the phylogeny. To calculate the likelihood on an unrooted phylogeny, a *virtual root* is temporarily inserted into an edge of the *unrooted* tree, thereby generating a *rooted* representation. Insertion if a virtual root into an edge means, to split an edge and insert a new node at the split point. For example, in the above case, $S9$ could be a *virtual root* inserted between nodes $S7$ and $S8$ of an unrooted tree. Here the reversibility of the Markov-process mentioned above becomes important: According to the *pulley principle* descried by Felsenstein [31], the virtual root can be arbitrarily placed along the edge into which it is inserted, under the condition that the sum of its adjacent edge lengths is the same as the length of the original edge into which it is inserted. In the above example this means that the original, unrooted, phylogeny had an edge of length $b_7 + b_8$ between nodes $S7$ and $S8$. If we add value $x$ to length $b_7$ and subtract the same value from $b_8$, the resulting likelihood is the same. This argument can be applied recursively to show that the virtual root can be placed into any edge of the original, unrooted, tree.

Besides allowing for arbitrary placement of the virtual root, the *pulley principle* is also fundamental for optimizing the edge lengths of a phylogenetic tree. To that end, it is necessary to optimize each individual edge length $b_i$ such that the overall likelihood of the tree is maximized. Here the *pulley principle* is deployed to individually optimize each $b_i$ with respect to the other, fixed, edge lengths. This method is incrementally applied to all edges in the tree until changes no longer increase the likelihood. The *pulley principle* guarantees that the likelihood of the tree constantly increases in this procedure, which allows the overall method to converge. The optimization of the individual edges is commonly carried out using iterative numerical methods like Newton-Raphson.

When applied to all sites of a multiple alignment, the calculations of the $L_i(x_i)$ and the edge length optimization constitute the phylogenetic likelihood kernel (PLK), which usually takes more than 95% of the run time in likelihood-based phylogenetic algorithms. We will revisit the PLK later. Section 3.3 covers the parallelization of the EPA, where the PLK is initially calculated in parallel by distributing the calculations for individual alignment columns to multiple CPUs (this is possible, because, as mentioned above, the likelihood calculations on the alignment sites are independent from each other). Section 3.4 shows how the PLK can be accelerated using single instruction multiple data (SIMD) instructions. This section also refers to a technical detail related to the limited-precision representation of the probability values with floating-point arithmetics: As Equation 2.4 shows, the probability values are constantly multiplied with each other as the tree is traversed, which can result in very small (i.e., close to zero) values on large trees. To prevent numerical underflow, the probability values are checked inside the PLK and multiplied by a sufficiently large value when they get smaller than a certain value $\epsilon$ (the actual value of $\epsilon$ depends on the hardware representation of the floating point numbers). The SIMD acceleration in Section 3.4 demonstrates how these additional operations for preventing numerical underflow can be accelerated. Finally, Section 5.2.1 introduces the probabilistic gap propagation model in PaPaRa 2.0 which uses a variation of the PLK based on a simple two-state model representing the indel pattern of a MSA.

**Likelihood weights** As described in the previous section, the likelihood of a phylogenetic tree can be used to measure how well the tree explains the observed data. Thus, it provides a measure for comparing different tree topologies. The ‘expected likelihood weights’ (ELW [86]) method can be used to normalize likelihood values of different tree topologies such that they can be intuitively compared. Assuming that (for a fixed MSA) there are $n$ trees with likelihood scores $L_1 ... L_n$. The likelihood weight $w_i$ of each tree is defined as

$$w_i = L_i / \sum_{j=1}^{n} L_j \tag{2.6}$$

.

That is, $w_i$ transforms the likelihood of a single tree into a fraction of the sum of all likelihoods. Thereby one can identify, for instance, a single tree with an exceptionally good likelihood compared to the other trees. The EPA (Section 3.2) can calculate such likelihood weights of alternative sequence placements as a method for quantifying placement confidence.

### 2.2.2 Maximum Parsimony

The previous section introduced the phylogenetic likelihood kernel. This section covers maximum parsimony (MP), a simpler tree optimallity criterion, which does not assume an explicit model of sequence evolution. This criterion is based on the idea that a tree is optimal, when it can explain the known sequence data by the smallest possible number of evolutionary changes. W. Fitch [34] introduced a systematic method for efficiently computing the parsimony score on a tree. Similar to the phylogenetic likelihood function, this method uses a representation of the hypothetical and unknown ancestral sequences at the inner nodes of the tree. The ancestral representations are built recursively from the representations of the child nodes. Rather than using a probabilistic representation, parsimony uses sets of sequence characters to represent internal states. For example, parsimony would use the set {A,G} to represent an ancestral site that could either consist of an A or a G. A thorough description of the Maximum Parsimony method is given in Chapter 3.4 of [99].

Figure 2.2 demonstrates, how the ancestral state can be reconstructed for a simple example. When deriving the ancestral character sets, the algorithm first generates the intersection of the two character sets of the child nodes (Figure 2.2a). If the intersection is non-empty, the resulting set is used as the ancestral character set. If the intersection is empty, the union of the two child sets is used as the ancestral character set (Figure 2.2b). The parsimony score is produced alongside the ancestral character sets: For each site, the score corresponds to the number of empty set intersections between the child character sets (i.e., in Figure 2.2 this is the case two times (nodes marked with *), hence the parsimony score is 2). As for likelihood calculations, the character sets and parsimony scores are treated independently for all sites of

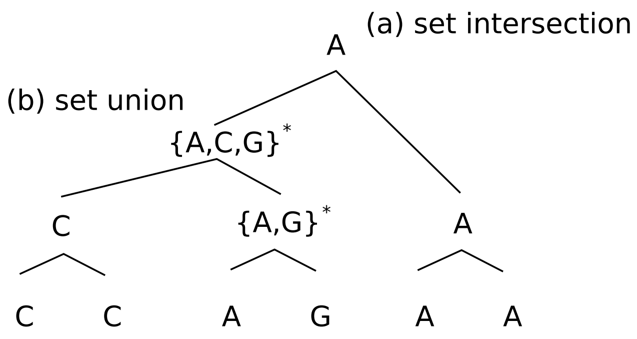


Figure 2.2: Calculation of the ancestral state representations in maximum parsimony.

the sequence alignment, and the overall parsimony score is the sum of the per-site scores. The representation of the ancestral states will be revisited in Chapter 5, where it is used for aligning short reads in the PaPaRa algorithm.

## Summary

This Chapter introduced the phylogenetic methods that are the basis for the novel algorithms presented later. To this end there was a short introduction to phylogenetic trees as a representation of evolutionary relationships in computational methods. In addition, it gave a short description of maximum likelihood (ML) and maximum parsimony (MP) based methods for scoring phylogenetic trees. Both methods are fundamental for the algorithms described in the following Chapters.

CHAPTER 3

# Evolutionary Placement of short Sequence Reads

This chapter introduces the evolutionary placement algorithm (EPA) which has previously been presented in [12]. The multi-grain parallel implementation of the EPA has initially been published in [82].

Identification of organisms from, for example, microbial communities increasingly relies on analysis of DNA that is extracted from soil or water samples containing many, often unknown, organisms, rather than one-by-one from the individual organisms. Recently, the advent of new DNA sequencing techniques (e.g., pyrosequencing [73]) has increased the amount of sequence data available for identification and analysis of microbial communities by several orders of magnitude. This rapid increase in the amount of available sequence data poses new challenges for short-read sequence identification tools. As discussed in Section 1.1, we can not expect that the automatic increase of computer performance according to *Moore's law* will be fast enough to handle this *flood of sequence data*.

In a single run, these new sequencing techniques can generate between hundreds of thousands up to several millions of short DNA reads with a length ranging between 30 to 450 nucleotides [49]. Such sequencing runs can be carried out by individual labs within hours. Besides rapid full-genome assembly, another important application is the sampling of microbial communities from certain environments, for example permafrost-affected soils [37], vertebrate guts [56, 58, 91], hypersaline mats [57], or human hands [33]. As mentioned in Section 1.1, Meta-genomic studies of microbial communities often yield a large amount of short-read sequences, where the provenance of the individual sequences is unknown. In such studies, the first step in analyzing the meta-genomic data consists in identifying the biological origin of the reads. This assignment of short reads to known organisms then allows for examining and comparing microbial samples and communities (see [91]). For instance, 20% of the reads in one sample might be most closely related to a specific taxonomic group of bacteria, while in a different sample only 5% of the reads may be associated to this group.

The Evolutionary Placement Algorithm (EPA) [12], presented here, is a dedicated tool for rapid phylogenetic identification of anonymous query sequences (QS), using a set of full-length reference sequences (RS). The most straight-forward approach to identify the origin of a QS is to use tools that are based on sequence similarity (e.g., BLAST [7]). However, the BLAST-based approach has an important limitation: It can yield misleading assignments of QS to RS if the sample of RS does not contain sequences that are sufficiently closely related to the QS (i.e., if the taxon sampling is too sparse or inappropriate). Any approach based on pair-wise sequence similarity, like BLAST, will not unravel, but silently ignore, potential problems in the taxon sampling of the RS. For instance, given two RS $a$ and $b$, a QS $q$ may be identified by BLAST as being most closely related to $a$. In reality, $q$ might be most closely related to a RS $c$, which is not included in the set of RS. Since this is a known problem [53], recent studies of microbial communities have started employing phylogenetic methods for QS identification [92], despite the significantly higher computational cost. This analysis of short sequence reads is related to phylogenetic tree reconstruction methods that employ stepwise addition of sequences [52], with the difference that each QS is individually placed into the phylogenetic reference tree (RT). If a QS is connected to an internal edge of a RT that comprises the RS (i.e., it is not located near a tip of the tree), this indicates that the sampling of the RS is insufficient to capture the diversity of the sampled QS. This can be used as a means for identifying parts of the tree where taxon sampling is sparse, which can guide sequencing efforts to improve the sampling.

Without a specialized tool like the EPA, phylogeny-based identification of the provenance of anonymous reads can be conducted as follows: firstly the QS are aligned with respect to a reference alignment (RA) for the RS. Then the QS are inserted into the reference tree either via a complete *de novo* tree reconstruction, a constrained tree search, using the RT as a constraint or backbone, or a fast and/or approximate QS addition algorithm, such as implemented in ARB [61], which uses MP. For DNA barcoding, phylogeny-based Bayesian analysis methods have recently been proposed in [67] as well as in [68]; these methods, however, are applied to significantly smaller trees. More recently, Brady and Salzberg have proposed the Phymm and PhymmBL algorithms for metagenomic phylogenetic classification [21], and reported improved classification accuracy for simulated QS compared to BLAST for PhymmBL. Classification by Phymm is based on oligonucleotide composition, whereas PhymmBL uses a weighted combination of scores from Phymm and BLAST. These algorithms classify QS relative to a given database of un-aligned bacterial genomes (the NCBI RefSeq database [70]) and their phylogenetic labels as provided in the RefSeq database (i.e., from Phylum level down to Genus level). This is different to the EPA, which can be used with any set of aligned RS and places QS into a fully resolved bifurcating RT. Because of the different focus of Phymm and the EPA, it is impossible to directly compare their accuracy on the same data sets, because multiple sequence alignments (MSA) and fully resolved phylogenies are not provided by the NCBI RefSeq database. This also hinders direct comparison to other previous phylogenetic classification methods like PhyloPythia [64], which is the only phylogenetic classifier that has been compared to Phymm ([21] shows that Phymm and BLAST substantially outperform PhyloPythia) and MEGAN [44]. However, it is possible to compare the relative performance of the different methods (Phymm, PhymmBL, and EPA) to placements/classifications obtained by using BLAST, on data sets that are appropriate for the respective methods. Therefore, the evaluations presented here mainly compare the accuracy of the EPA to basic BLAST searches and discuss the analogous accuracy evaluation performed for Phymm/PhymmBL.

A very similar algorithm to the EPA, called pplacer [63], has been developed independently by F. Matsen. The execution times of pplacer are comparable to those of the EPA according to a joint performance study conducted by F. Matsen, A. Stamatakis, and S.A. Berger. A comparative study is included in [63].

## 3.1 Motivation

The current standard approach for analysis of environmental reads reconstructs a fully resolved bifurcating tree that often comprises more than 10,000 sequences [33, 91]. The alignments used to reconstruct these trees mostly comprise only a single gene, typically 16S or 18S rRNA. The reconstruction of such large trees with thousands of taxa, based on data from a single gene, is both, time-consuming and hard, because of the weak phylogenetic signal in the alignment, which results in decreased reconstruction accuracy for trees with many, but relatively short sequences (see [20, 66]).

Moreover, in metagenomic data sets a large number of QS will only have a length of approximately 200-450 base pairs if a 454 sequencer is used. Thus, when identifying the provenance of short read QS, the lack of phylogenetic signal becomes even more prevalent and critical if a comprehensive tree is reconstructed. In order to solve the problems associated with the lack of signal and to significantly accelerate the analysis, the EPA uses a different approach that only computes the optimal insertion position for every QS individually in the RT with respect to its ML score.

The following section introduces the EPA, a specialized algorithm for phylogenetic placement of QS. A thorough evaluation of the placement accuracy on eight published data sets is provided in Section 6.3. The impact of QS length on placement accuracy is assessed using tests on emulated short reads derived from original full length sequences of the test data sets. Because phylogenetic placement is inherently more computationally intensive than BLAST-based placement, performance optimization is an important factor in the development of such an algorithm if it is to become a useful and fast alternative to BLAST. Therefore, we have devised several evolutionary placement algorithms and heuristics with varying degrees of computational complexity.

The algorithm, which has been developed and tested in cooperation with microbial biologists, is available as open-source under the GNU General Public License (GPL) as part of RAxML [79] (`https://github.com/stamatak/standard-RAxML`). A web-service that provides an easy to use interface to most of the EPA functionallity is available at `http://sco.h-its.org/raxml`. The approach implemented in the EPA represents a useful, scalable and fast tool for evolutionary identification of the provenance of environmental QS. At the time of their intoduciton, the EPA and pplacer were the only algorithms that could perform the task described here. The parallelization of the EPA [82] and the ability to conduct placements under all time-reversible substitution models and data-types offered by RAxML is a unique feature of the EPA that allows for good scalability on large and diverse data sets. Pplacer can infer placements using either ML (as the EPA) or Bayesian posterior probabilities.

## 3.2 Evolutionary Placement Algorithm

The input for the evolutionary placement algorithm consists of a RT comprising the $r$ RS, and a large comprehensive alignment that contains the $r$ RS *and* the $q$ QS. The task of aligning several QS with respect to a given RS alignment can be accomplished with ARB [61], NAST [25], MUSCLE [28], MAFFT [51] or, as tested here, with HMMER [27]. Chapter 5 introduces PaPaRa [17], a novel phylogeny-aware method for QS alignment based on the RS as well as the RT. Section 6.5 contains a thorough evaluation of the impact of the alignment procedure on EPA placement accuracy.

One key assumption is, that the RT is biologically well-established or that it has been obtained via a preceding thorough phylogenetic analysis. In a typical usage scenario, the EPA could be used for mapping distinct microbial samples, for instance, from time series experiments or different locations/individuals, to the same, microbial reference tree in order to compare communities. Because the same RT can be used in multiple placement runs, the inference of the RT is not part of the EPA.

Initially, the algorithm will read the RT and reference alignment and mark all sequences *not* contained in the RT as QS. Thereafter, the ML model parameters (see Section 2.2.1 for details) and edge lengths on the RT will be optimized using the standard procedures implemented in RAxML.

Once the model parameters and edge lengths have been optimized on the RT, the actual identification algorithm is invoked. It will visit the $2r-3$ edges of the RT via a preorder tree traversal, starting at an arbitrary edge of the tree leading to a tip (i.e., visit the current edge first and then recursively visit the two neighboring edges etc.). At each edge, initially the probability vectors of the RT to the left and the right will be re-computed (if they are not already oriented toward the current edge). Thereafter, the program will successively insert, and subsequently remove again, one QS at a time into the current edge and compute the likelihood (henceforth denoted as the insertion score) of the respective tree containing $r+1$ taxa. The insertion score will then be stored in a $q \times (2r-3)$ table that keeps track of the insertion scores for all $q$ QS into all $2r-3$ edges of the RT. In order to more rapidly compute the per-edge insertions of the QS, the EPA deploys an approximation that is comparable to the Lazy Subtree Rearrangement (LSR) moves in the standard RAxML search algorithm [83]. After inserting a QS into an edge of the RT, it would normally be necessary to re-optimize all edge lengths to obtain the corresponding insertion score according to the ML criterion. Instead, the algorithm only optimizes the three edges adjacent to the insertion node of the QS (see Figure 3.1) to compute the likelihood score of the insertion. This approach rests on the same rationale that was used to justify the LSR moves. The experimental results justify this approximation because it yields high placement accuracy. The EPA uses two methods (similar to those used for the LSR moves) to re-estimate the three edges adjacent to the insertion edge: a *fast* method and a *slow* method that uses the Newton-Raphson method. The *fast* method simply splits the insertion edge, $b_r$, in the RT into two parts, $b_{r1}$ and $b_{r2}$, by setting $b_{r1} = b_{r2} = b_r/2$, and $b_q = 0.9$ (i.e., the edge leading to the QS), where 0.9 is the default RAxML value to initialize edge lengths. These values were chosen empirically for good place-

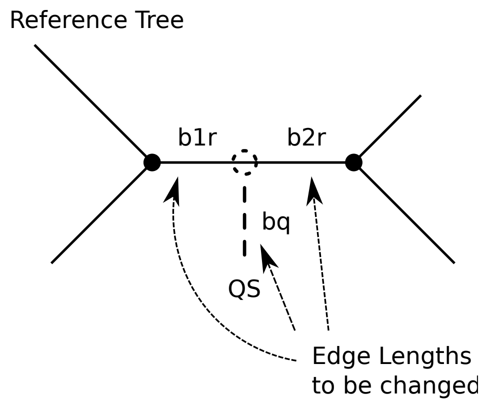


Figure 3.1: Local optimization of edge lengths for the insertion of a query sequences (QS) into the reference tree (RT).

ment accuracy over varying input data. Note that, the *slow* method, that is, the most accurate placement method available in RAxML (not using the aforementioned approximations) is exclusively used for the main accuracy evaluation presented in Section 6.3. Thereafter, a separate assessment of the *slow* and *fast* methods is given for comparison. The *slow* method repeatedly applies the Newton-Raphson method to all three edges ($b_{r1}$, $b_{r2}$ and $b_q$) until no further application of the Newton-Raphson method changes the edge lengths substantially (i.e., when $\epsilon \leq 0.00001$, where $\epsilon$ is the edge length change between two invocations of the Newton-Rhapson method). Alternatively, the algorithm can also use the *fast* method to pre-score and order promising candidate insertion edges in order to reduce the amount of *slow* insertion operations as a heuristic acceleration of the placement process.

The output of this procedure consists of the RT, enhanced by assignments of the QS to edges of the RT. Each QS is attached to the edge that yielded the best insertion score for the specific QS. Hence, the algorithm will return a multi-furcating tree if two or more QS are assigned to the same edge. An example is provided in Figure 3.2.

The EPA algorithm can optionally use the non-parametric bootstrap [32] to account for uncertainty in the placement of the QS. An example for this is shown in Figure 3.3. Thus, a QS might be placed into different edges of the RT (for different bootstrap replicates) with various levels of support. For the bootstrap procedure, we introduce additional heuristics to accelerate the insertion process. During the insertions into the RT using the original alignment we keep track of the insertion scores for *all* QS into *all* edges of the RT. For every QS we can then sort the insertion edges by their scores and for each bootstrap replicate only conduct insertions for a specific QS into 10% of the

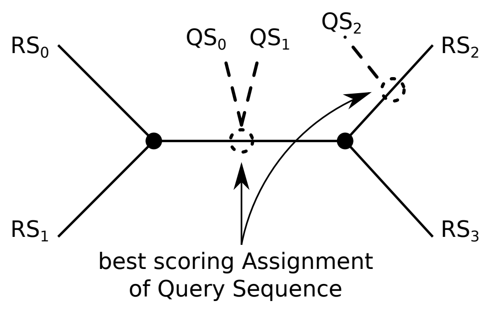


Figure 3.2: Evolutionary identification of 3 query sequences ($QS_0$, $QS_1$, $QS_2$) using a 4-taxon reference tree.

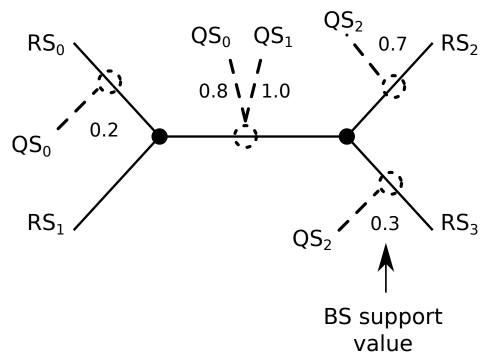


Figure 3.3: Phylogenetic placement of 3 query sequences ($QS_0$, $QS_1$, $QS_2$) into a 4-taxon reference tree with insertion support (IS) score.

best-scoring insertion edges on the RT and original alignment. This reduces the number of insertion scores to be computed per QS and per bootstrap replicate by 90% and therefore approximately yields a ten-fold speedup for the bootstrapping procedure. In a typical application scenario, one may determine the diversity of the environmental sample for every replicate using, for instance, UniFrac [60], and then compute an average diversity over all replicates.

As a faster and as it turns out more accurate alternative to the non-parametric bootstrap, the insertion scores can also be directly used to compute placement uncertainty. von Mering et al.[92] used expected likelihood weights (ELW [86]) to assign QS to an area of a tree with a certain confidence. Methods for calculating a placement uncertainty using ELW are already implemented in the EPA and pplacer.

As described above, the EPA can use optional heuristics that rely on the *fast* scoring approach in order to improve the runtime of the *slow* insertion method. Given the pre-scoring technique, the number of insertion positions considered for the thorough and slow insertion process can be reduced to a fraction of promising candidate edges. The proportion of insertion edges suggested by the rapid pre-scoring heuristics for analysis under the slow insertion method is determined by a user-defined parameter $fh$. As part of our performance evaluation, we tested the heuristics with regard to this parameter setting.

It is a known problem [47] that compositional heterogeneity can bias phylogenetic inference because it implies that the sequences cannot have evolved under the same stationary, reversible and homogeneous conditions (assumed by all the available time-reversible substitution models). In the case of the EPA, this can be problematic for the placement of QS onto potentially very short edges of the RT. Different approaches exist to resolve those problems by using more sophisticated models (e.g., [36, 46]). Nonetheless, time-reversibility is required to accommodate the high computational demands of large scale tree and EPA inferences, in particular with respect to computing the likelihood on trees (the *pulley principle*, see Section 2.2.1 and [31]). Therefore, EPA users should test their RS alignments for evidence that the sequences have not evolved under time-reversible conditions (e.g., using methods published in [3, 42]) before using the EPA. If the data have not evolved under time-reversible conditions, then the findings obtained with the EPA should be discussed with this caveat in mind. These considerations do not affect our accuracy assessment, because trees of full length sequences and placements have been inferred under the same model and the potential errors that may occur because of inappropriate evolutionary models affect both, tree construction and placement.

## 3.3 Exploiting Multi-grain Parallelism in the EPA

The current release of RAxML (available at `https://github.com/stamatak/standard-RAxML.git`) contains a full fine-grain Pthreads-based parallelization of the likelihood function (see Section 2.2.1 and [84]). The initial parallelization of the evolutionary placement algorithm was hence straight-forward because it could take advantage of the existing parallel framework. However, the scalability of this fine-grain parallelization is limited on data sets with few sites such as the real-world single gene bacterial data sets used in the experiments described here.

The main parallelization challenge lies in handling the two different phases of the program that exhibit varying degrees of parallelism: In the *initial phase*, during which ML model parameters are optimized, the edge length optimization procedure is hard to be further parallelized because of intrinsic dependencies between individual edge length optimization iterations. The *insertion phase* consists of computing the $q \times (2r-3)$ insertion scores of the QS on the RT and is easier to parallelize. Hence it is necessary to deploy multi-grain parallelism using the fine-grain parallelization scheme for the *initial phase* and a more coarse-grain parallelization strategy for the *insertion phase*. One can parallelize the insertion phase by essentially conducting *all* $q \times (2r-3)$ insertion score computations simultaneously. Here, the parallelization is done on edges, that is the edges from the RT are distributed to threads in a cyclic manner, where every thread computes insertion scores for inserting *all* $q$ QS into one specific edge. Given the number of edges in real-world RTs (e.g., 9,745 insertion edges, see Section 6.3.8) which is expected to further increase in the future, this represents a sufficiently fine-grain source of parallelism for this algorithm. In order to be able to compute QS insertions simultaneously it is necessary to compute and store all pairs of probability vectors for all edges in the tree. Hence, for every edge there is a contiguous (remember that the fine-grain parallelization of the initial phase uses a cyclic distribution of the probability vector columns) probability vectors attached to the left and right end of each edge. This allows a single thread to independently compute all insertions of the QS into one edge, since only those two vectors are required to compute the insertion score.

In order to store and compute contiguous probability vectors, the EPA initially allocates a data-structure of length $2r-3$ that corresponds to the total number of edges in the RT. For every entry of this edge data structure two contiguous (full-length) probability vectors are allocated that are used to store the probability vectors to the left and the right of that edge in the RT. Calculation of the per-edge probability vectors proceeds as follows: The entire RT is traversed in depth-first order and the virtual root is placed into the edge that is currently being visited. The probability vectors to the left and right of the current edge are recomputed if necessary. Once the strided vectors using the cyclic probability vector column distribution have been computed, a gather operation stores the per-thread columns contiguously in the vectors attached to the edge data structure. This traverse and gather procedure is outlined in Figure 3.4. The overhead of this full tree traversal and the gather operation is negligible (less than 0.5% of overall execution time) compared to the overall execution time as well as the time required for initial model parameter optimization in fine-grain mode. Once the probability vector pairs for all edges have been stored, the algorithm proceeds by simply distributing work to threads in a cyclic way, that is, thread 0 will insert all QS into edge 0, thread 1 will insert all QS into edge 1, etc.

An additional rationale for deploying a per-edge parallelization strategy is that this can also faciliate a distributed memory parallelization of the

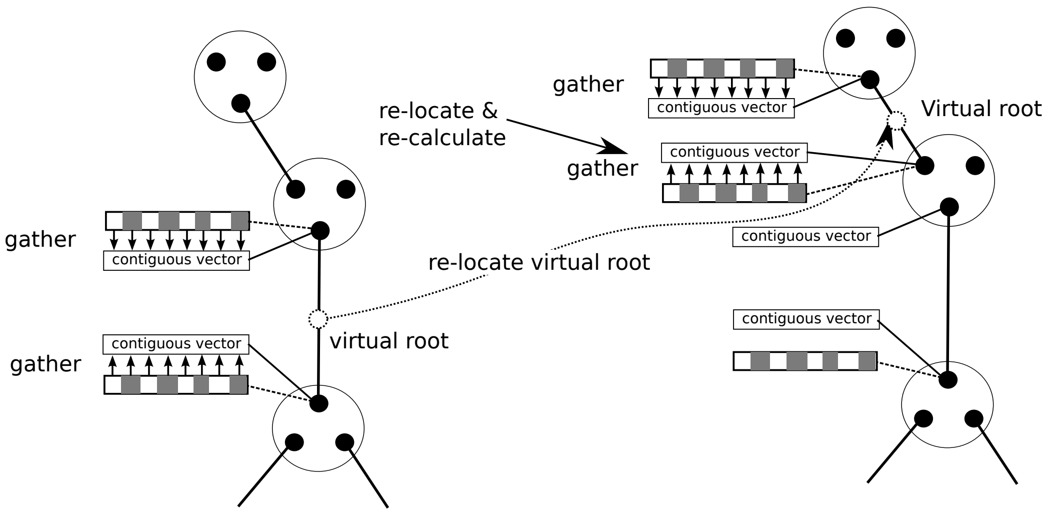


Figure 3.4: Multi-grain gather operation of the per-edge probability vectors on the reference tree.

evolutionary placement algorithm with MPI (Message Passing Interface). If the parallel version was parallelized over insertion sequences, it would need to hold all insertion edge vectors in the memory of each process which might lead to memory shortage, in particular, on systems such as the IBM BlueGene/L or BlueGene/P.

## 3.4 SIMD Vectorization of the PLK

As described in Section 2.2.1, deriving the likelihood score of a phylogenetic tree requires calculating the ancestral probability vectors at the internal nodes of the tree. The total program run time of likelihood based phylogenetic algorithms is often dominated by these calculations, and the EPA is no exception. Usually, ancestral probability vector calculations account for about 65% of the overall run time. It is therefore crucial to optimize the calculation as far as possible. While the shared memory parallelization described in Section 3.3 yields good scalability on multiple cores in a shared memory system, the single instruction multiple data (SIMD) extension on current CPUs allow to exploit additional parallelism by means of vectorization. We specifically target the SSE instruction set (see [1] Chapter 10ff), which is widely available on common Intel and AMD desktop and server CPUs.

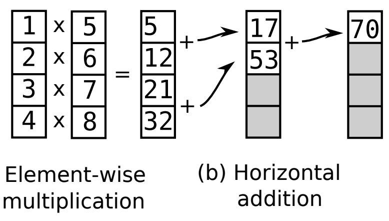


Figure 3.5: Calculating the scalar product of two 4-element vectors [1,2,3,4] and [5,6,7,8] using (a) element-wise multiplication and (b) horizontal addition of the multiplication results.

**SSE3 for Likelihood Computations:** The operations we vectorize in the PLK are are special cases of a general dense matrix multiplication; the computations on $L_j(x_j)$ and $L_k(x_k)$ in Equation 2.4 over all sites c and all nucleotides A, C, G, T are matrix products of the form $P_{x_i x_j}(b_j)L_j(x_j)$ and $P_{x_i x_k}(b_j)L_k(x_k)$. We assessed the usage of highly optimized ATLAS-BLAS routines (`http://math-atlas.sourceforge.net/`), but because of the unfavorable matrix dimensions (multiplication of the $4 \times 4$ matrix P with the $4 \times m$ matrix $L$) we even observed a slowdown. Therefore, the key to optimizing the PLK is to efficiently calculate the scalar product (see Figure 3.5) between vectors of length 4, on which the matrix multiplications rely. The multiplication of the vector elements can be mapped directly to the element-wise arithmetic operations (Figure 3.5a) introduced for Intel CPUs with SSE2 ([1] Chapter 11.4.1). The problem is then to finish the scalar product by summing over the vector elements. Efficient sum calculation requires the horizontal addition instructions (Figure 3.5b) introduced in SSE3 ([1] Chapter 12.3.5). With this scheme, the 7 arithmetic operations necessary for calculating the scalar product of two 4-element vectors are realized with three SSE instructions.

**SSE3 for Likelihood Scaling:** We also vectorized the scaling procedure mentioned in Section 2.2.1 using SSE3 instructions. SSE3 instructions are used to efficiently determine the maximum value of $L_A(c)$, $L_C(c)$, $L_G(c)$, $L_T(c)$ (see Section 2.2.1) and then compare the maximum of these to the $\epsilon$ value, thereby eliminating several conditional statements. Note that, before carrying out explicit vectorization via intrinsics, we conducted numerous failed attempts to re-write the loops such as to facilitate auto-vectorization by the Intel icc compiler (see [14]; version 11.1 at the time of conducting these experiments). SSE3 vectorization was implemented by inserting SSE3 intrinsics into the C code, rather than via inline assembly. This leaves room for further optimization of the instruction schedule and register allocation by the compiler.

## 3.5 EPA Web-Service

A Web-Server that represents the EPA algorithm is available at `http://sco.h-its.org/raxml` free of charge and will be continuously developed and improved. The server runs on a dedicated machine with 48 AMD cores and 256GB of main memory.

Users can upload RTs and RAs and chose to align QS to the RA with HMMER or upload an alignment that already contains the QS. When the QS are aligned by the server, they can also be clustered using UCLUST (`http://www.drive5.com/usearch/`) prior to alignment with HMMER. The UCLUST option can be used to reduce the number of reads that will subsequently be placed and aligned. Finally, the Web-Server also offers a JAVA-based result visualization tool that uses the Archeopteryx framework [40] and provides a simple visualization of the read distribution in the RT.

## Summary

This chapter introduced the EPA, a method for rapid and phylogeny-aware placement of sequence data into an existing phylogeny. The algorithm is based on maximum likelihood (ML) and is specifically designed towards handling the large amount of sequence data produced by next generation sequencing techniques. To enable the algorithm to scale on large data sets, we presented algorithmic as well as technical optimizations. Firstly, an optional heuristic was introduced to reduce the number of insertion positions by the computationally complex scoring procedure. Secondly, the implementation of the algorithm can exploit the inherent parallelism of the placement procedure. Finally, we presented the free web-server and a GUI program to visualize the placement results.

CHAPTER 4

# Application of the EPA to non-molecular data

The following chapter shows how the EPA can be used on non-molecular, in this case, morphological data. The approach has previously been published in [15]. The methods presented here have in the meantime been used successfully in a separate study [19].

The on-going extension of analysis capabilities for partitioned (phylogenomic) data sets in programs for ML based tree inference allows to address novel methodological questions. Since additional ML substitution models for binary morphological data [55] have been integrated into RAxML [79] (available at: `https://github.com/stamatak/standard-RAxML.git`). Current versions of RAxML allow for the analysis of super-matrices (also called total evidence approach or multi-gene/phylogenomic alignments) that contain a mix of data-types, that is, an input alignment may consist of concatenated morphological, DNA, and protein (amino acid) sequence partitions that represent the organisms under study.

An example for such a phylogenomic alignment with 4 present-day organisms (the great apes) and one fossil taxon (e.g., some common extinct ancestor of the human and the chimpanzee) that entails a binary/morphological data partition with 6 sites (columns/morphological characters) and a DNA data partition with 24 sites is provided below. As already mentioned, there will typically not be any molecular data available for the fossil taxa under study, hence the molecular sequence part of the fossil taxon is filled with gaps, which are treated as undetermined characters in all standard ML-based and Bayesian implementations. Thereby, they do not influence the likelihood computations on DNA data.

```
Fossil     001101------------------------
```

```
Human      000111A-GGCATATCCCATACAAAGGTTA
Chimp      000100ATGGCACACCCAACGCAAGGGTGA
Gorilla    001111ATGGCCAACCACTCCCAAAAGTCA
Orangutang 111011CGGGCACATGCAGCGCAA-A-T-A
```

Here we analyze the potential applications of morphological (throughout this we exclusively use binary characters, i.e., we do not consider multi-state morphological characters) data for gaining novel evolutionary insights. The evaluation presented in Section 6.4 uses 5 real-world partitioned input data sets that contain both, morphological, and molecular (DNA) partitions as well as more than 2,500 simulated morphological data sets. It is important to emphasize that, at present it is hard to use a larger number of real-world data sets for the purposes of this study, because they are not readily available in standard tree and alignment repositories such as TreeBase [74].

A general problem with morphological data within the phylogenomic context is that only a few morphological character sites (typically 50–500 alignment columns, see Table 6.5) are available compared to a constantly growing number of molecular character sites (typically 1,000 to tens of thousands in current phylogenomic studies, see, e.g., [26]). Thus, the overall per site log likelihood contribution of the morphological sites will be very small and therefore only have a negligible impact on the shape of the overall tree topology that is inferred based on the concatenated morphological and molecular data set. In addition, there can be a significant incongruence between best-known ML trees (remember that ML for phylogenetic trees is NP-hard [24]) obtained from individual tree searches on *either* the morphological *or* the molecular partitions of the input data set. Given that tree shapes are largely dominated by the molecular part of the input data sets because of the masses of molecular data that have now become available, the question arises what the potential use of those comparatively few (a couple of hundred compared to tens of thousands) morphological columns might be, since they will mostly add some insignificant noise to the signal of broadly sampled phylogenomic data sets. Currently, there exist two application scenarios that make use of a given "true" RT, which will be assumed to be the molecular tree here, though this assumption can evidently be challenged. As a RT one may also consider using a well-established species tree from the literature or the NCBI (National Center for Biotechnology Information, `http://www.ncbi.nlm.nih.gov/`) taxonomy to obtain a "true" RT.

*Scenario I:* Given a RT and a morphological data matrix one may use this matrix for the phylogenetic placement of a fossil taxon for which no molecular data exists. This essentially means that a well-established reference topology is imposed onto the morphological character matrix. The morphological part of the matrix is then used to insert (place) the fossil by computing the best-scoring insertion position under ML in the RT.

*Scenario II:* One may also be interested in inferring the ancestral states on a fixed and dated reference topology to determine at which point of time in the past (on which edge) a transition between, for instance, green eye color and blue eye color appeared.

A problem (as demonstrated in Section 6.4), inherent to both use cases for morphological data is that of incongruence, that is, conflict of phylogenetic signal between the morphological tree and the (molecular) RT. The issues pertaining to *Scenario I* will be addressed here, that is, the accuracy of fossil placement for morphological data with an incongruent tree signal will be assessed. The computational experiments show that placement accuracy under ML is already relatively good (above 85%) and robust against noise, despite conflicting signals in the data. We further refine the method using a new statistical method that improves fossil placement accuracy by approximately 20-25% on average.

The method which is denoted as "morphological weight calibration", can infer weights for morphological alignment sites in such a way that sites which are congruent to the RT obtain a higher weight than incongruent sites. As a result, they contribute more to the overall likelihood during the phylogenetic placement of the fossil. The above methods have been implemented in the current version of RAxML which is freely available as open-source code at `https://github.com/stamatak/standard-RAxML.git`.

## 4.1 Related Work

The assignment of weights to morphological sites, which is henceforth denoted as weight calibration, has previously mainly been addressed within the framework of correct value range treatment for quantitative versus qualitative traits [89, 95], that is, not with the goal to reduce incongruence, but with better biological models in mind. Those methods are primarily used in phylogenetic analyses under MP, where each morphologic trait (character) needs to have the same relative weight. For MP, weight calibration is used to eliminate unequal weightings that may arise from different value ranges on multi-state morphological characters. Because nothing is known about the relative informativeness of transformations on different characters, equal weighting should be assumed a priori [30, 89]. As pointed out by J.J. Wiens [95] this issue has generally received little attention, despite its importance and biological relevance.

In contrast to the above, here we investigate (i) to which extent incon-

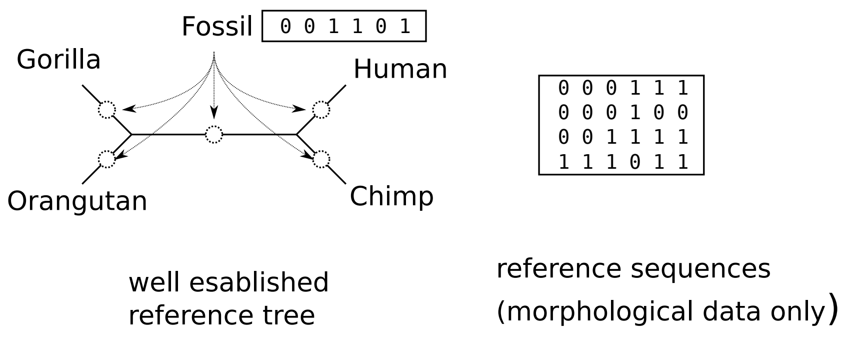


Figure 4.1: Example for a phylogenetic fossil placement problem.

gruent signal in the morphological and molecular data partitions can bias placement accuracy and (ii) if morphological site weight calibration can be used to filter out morphological sites that are highly congruent to the RT.

Previously, J.J. Wiens [93], addressed the question if the addition of molecular data (instead of using morphological data alone) can improve the phylogenetic position/placement of fossils (for which molecular data is not available) in trees, by using simulated data sets and Bayesian as well as MP methods. While he finds that the usage of molecular data in addition to morphological data can increase accuracy, or will at least not affect accuracy, he does not address the effects of incongruent signal in the morphological and molecular partitions on placement accuracy. The approach presented here is different in that it assumes that the molecular tree is the RT, and that there may be a significant amount of incongruence in the trees favored by the molecular and morphological partitions. Section 6.4 will also demonstrate this incongruence on real data sets.

## 4.2 Fossil Placement Algorithm

Initially, we devise a method to place one or more fossils into a given molecular or otherwise well-established RT by exclusively using the morphological part of data for which fossil data is available. An example is provided in Figure 4.1, where we want to place a fossil into a reference tree with 4 current-day organisms, once again using the example of the great apes. For the sake of simplicity we only consider the case of placing a single fossil into the tree; the placement procedure for more than one fossil is analogous.

The input for the fossil placement algorithm in RAxML consists of the RT $t_{ref}$ that comprises the $n$ morphological sequences (4 in our example) of present-day species. The input alignment contains the $n$ present-day sequences as well as the fossil sequence(s) that shall be placed into the tree. As already mentioned, it is assumed that $t_{ref}$ has been obtained via a thorough ML analysis of the corresponding molecular sequence data or by using a well-established species tree, for example, obtained from the literature or the NCBI taxonomy database. Initially, the algorithm will read $t_{ref}$ and the alignment and mark all sequences (in this case only the one fossil sequence) in the alignment that are *not* contained in $t_{ref}$ as query sequences. The remaining placement procedure is equivalent to the EPA which was introduced in Section 3.2. An important feature of the EPA in this scenario is the ability to accept a user-supplied weight vector. The weight vector contains per column (per alignment site) integer weights, which in our case, represent weight calibration algorithm results (see Section 4.3).

## 4.3 Site Weight Calibration

As already mentioned, there may be significant incongruence between the phylogenetic signal, that is, the trees that are favored by the morphological and molecular partitions of the data. As such, one of the key questions is to which extent this incongruency affects the placement accuracy of fossils into a tree derived from molecular data, and if there exist mechanisms to efficiently determine which sites of the morphological data partition are congruent to the molecular RT.

Therefore, it is necessary to devise a criterion to determine *those* sites from a—in this case—morphological data partition that are highly congruent to a given RT. Ideally, one would like to calibrate the site weights, that is, execute the fossil placement algorithm described in Section 4.2 with a weight vector that enhances the signal of morphological sites that *are* congruent to the RT. This weighting scheme should ideally increase fossil placement accuracy and decrease the impact of noise caused by incongruent morphological character sites.

In order to achieve this, a randomized statistical procedure can be used that works as follows: Initially, it reads in the RT $t_{ref}$ and the morphological alignment and optimizes ML model parameters on this tree, without changing the tree topology. Once the model parameters have been optimized it stores the per-site log likelihood values of the RT in an appropriate vector $\vec{L}_{ref}$ of length $m$, where $m$ is the number of sites (columns) in the morphological data.

Thereafter, it generates a set of $n = 100$ random trees, $r_1, ..., r_{100}$. For each random tree $r_i$, where $i = 1...100$, the ML model parameters are optimized again in order to compute the per-site log likelihood scores $\vec{L}_{r_i}$ for random tree $r_i$.

Once the per-site log likelihood scores $\vec{L}_{ref}$ on the RT and $\vec{L}_{r_i}$ on the 100 random trees have been computed, one can determine the degree of congruence between a specific site $j$, where $j = 1...m$, and the RT, by counting in how many random trees $r_i$ the per-site log likelihood of site $j$ is worse than the per-site log likelihood $\vec{L}(j)_{ref}$ in the RT $t_{ref}$. For each site $j$ the procedure computes a weight vector entry $\vec{W}(j) = \sum_{i=1}^{n} \delta_{j,i}$ where $\delta_{ref,i}$ is defined as follows:

$$\delta_{ref,i} = \begin{cases} 1 & if \quad \vec{L}(j)_{ref} > \vec{L}(j)_{r_i} \\ 0 & else \end{cases} \tag{4.1}$$

The above definition means that sites which are highly incongruent with respect to $t_{ref}$ will have low weights close to 0, while sites that have weights close to 100 are highly congruent to $t_{ref}$. The rationale for the above approach is that a site that is highly congruent to the RT will score worse on random trees, while a site that is highly incongruent will score better or at least not worse on most random trees. The above weight vector $\vec{W}$ can be used directly as input for a fossil placement run. The weight vector $\vec{W}$ can also be used to derive a binary weight vector $\vec{W}_{bin}$ in which all elements with $\vec{W}(j) \geq 95$ are set to 1 and all elements $j$ with $\vec{W}(j) < 95$ are set to 0 (using a typical 5% cutoff). This criterion can be used to more radically filter out incongruent sites. The comparison of the per-site log likelihoods does not explicitly use a method for determining, if values are significantly different from each other, but rather compares site-wise log likelihoods directly, since those effects will be averaged out by the randomized re-sampling procedure. Finally, initial tests indicated (results not shown) that the computation of 100 random trees is sufficient to infer stable weight values. The experimental results on simulated data clearly show that the above approach is able to discriminate well between congruent and incongruent sites and thereby justify this approach (see Section 6.4).

## 4.4 Phylogenetic Binning

To facilitate the usage of the placement methods presented here, we have designed a flexible JAVA based binning tool [18, 19]. The tool reads in a RT and the results of the phylogenetic placement as obtained from the EPA algorithm in RAxML. It also allows the user to specify an arbitrary number of phylogenetic bins (clades/subtrees, see Figure 4.2a). This can be done by supplying a plain text input file that specifies those taxa of the RT that shall

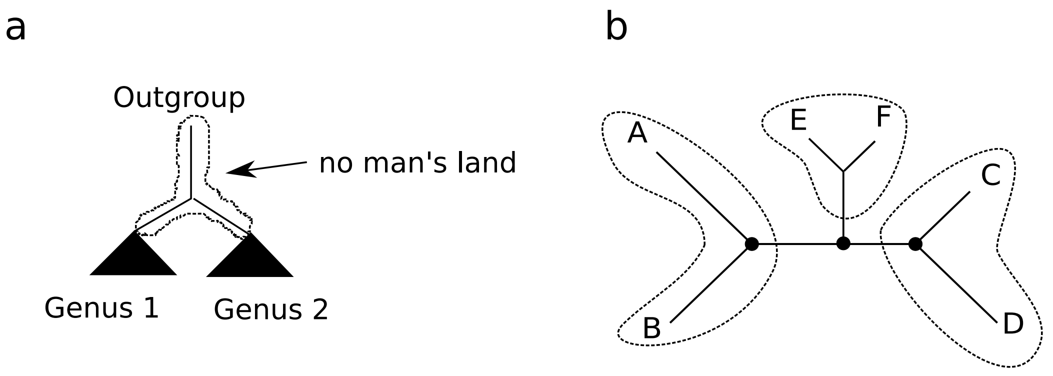


Figure 4.2: (a) Division of a phylogenetic tree in bins. (b) Defining bins by taxon lists: three bins are defined by the three taxon lists [A,B], [C,D] and [E,F].

form a bin. In other words, the user needs to provide a list of taxon lists that form bins (see Figure 4.2b), typically corresponding to monophyletic lineages in the RT. The user can also chose if the edge to which a bin is attached shall form part of the respective bin or not. All placements into edges that do not form part of a bin are assigned to a dedicated 'no man's land' bin. Alternatively, as implemented in a previous study on lichen [19], the user can simply specify an outgroup name. Our JAVA tool will then automatically divide the tree into three bins as shown in Figure 4.2a. The source code of the binning tool is available under the GNU general public license (GPL) at `https://raw.github.com/sim82/java_tools/master/src/ml/EpaBinning.java`.

## Summary

This chapter described an extension of the EPA to non-molecular data. In this context, the algorithm can be used to place taxa for which no molecular sequence data is available (e.g., fossils), into a given molecular reference phylogeny. We discussed the problem of incongruence between molecular and non-molecular data, and introduced an algorithm for automatic selection/ weighting of columns from a non-molecular alignment that are congruent to a molecular reference phylogeny. Additionally, we introduced a flexible binning tool that simplifies analysis of EPA-generated placements.

CHAPTER 5

# Phylogeny-Aware Alignment of Short Reads

This chapter presents the PaPaRa algorithm for short read alignment against a fixed reference MSA. There exist two versions of the algorithm: PaPaRa 1.0, which represents the initial proof-of-concept implementation has been published in [17]. The algorithm has been fundamentally reworked, resulting in PaPaRa 2.0 which has been previously presented in a technical report [13].

The previous chapters introduced the general concept of evolutionary placement of short sequence reads. The accuracy of such a likelihood-based placement of reads depends upon the multiple sequence alignment, that entails the RA *and* the short sequence reads (denoted as query sequences: QS). Therefore, a prerequisite for phylogenetic placement algorithms is, that the QS need to be aligned to the RA (Figure 5.1A), before conducting a placement run. As previously shown, a straight-forward way to achieve this, is to use the methods provided by the HMMER [27] tool-suite for aligning QS with respect to a RA. HMMER initially builds a profile Hidden Markov Model (HMM) of the RA. Thereafter, the QS are aligned against the profile-HMM that represents the RA. HMMER implements a dedicated method, HMMALIGN that allows for aligning multiple QS (one at a time) against the *fixed* profile-HMM of the RA. HMMALIGN will then output an alignment that contains the RA and the QS that have been aligned with respect to the profile-HMM of the RA. Note that, HMMALIGN frequently also modifies the RA by inserting gaps, if needed. When using a profile-HMM, the entire RA is represented by a monolithic —flat— probabilistic profile that does not use the phylogenetic information of the RT. MUSCLE and MAFFT offer similar options to align sequences (in our case QS) against a monolithic profile that is derived from an existing RA. It has already been shown that

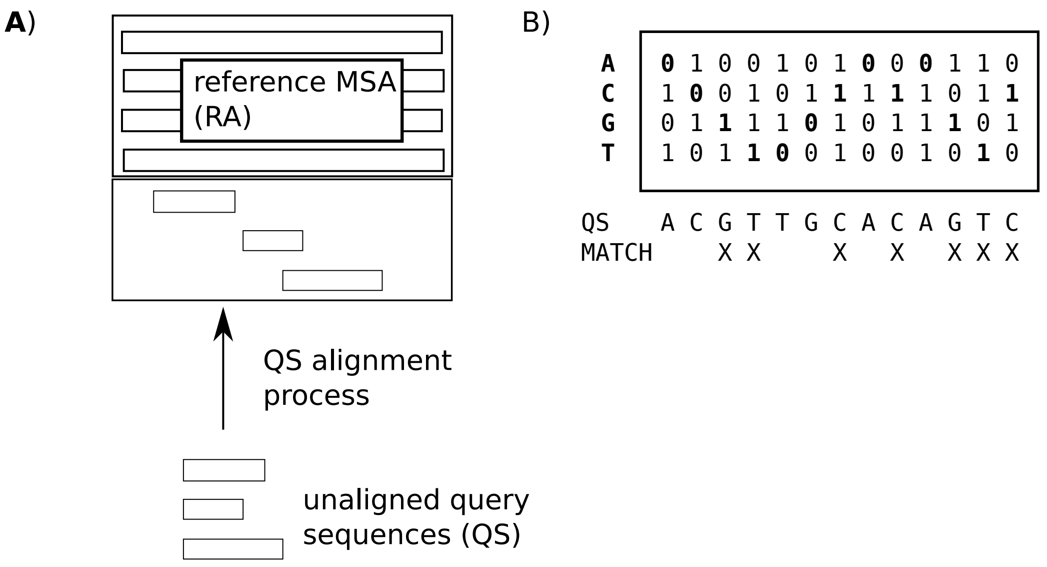

A)
reference MSA (RA)
QS alignment process
unaligned query sequences (QS)
B)
A 0 1 0 0 1 0 1 0 0 0 1 1 0
C 1 0 0 1 0 1 1 1 1 1 0 1 1
G 0 1 1 1 1 0 1 0 1 1 1 0 1
T 1 0 1 1 0 0 1 0 0 1 0 1 0
QS A C G T T G C A C A G T C
MATCH X X X X X X X

Figure 5.1: (A) General scheme of the QS alignment procedure. (B) Matching a QS against an ancestral state vector.

QS alignment using HMMALIGN performs reasonably well with respect to phylogenetic placement accuracy [12, 63].

However, depending on the specific alignment strategy/philosophy deployed to generate the RA, better alignment quality (as quantified by QS placement accuracy), can be achieved by incorporating the phylogenetic signal of the RT into the QS alignment process. Here we will investigate the problem of aligning short reads against a given reference alignment and introduce PaPaRa (PArsimony-based Phylogeny-Aware short Read Alignment), a novel, phylogeny-aware method for QS alignment. The performance evaluation in Section 6.5 systematically evaluates phylogenetic QS placement accuracy of the EPA for different QS alignment methods. The baseline for comparisons consists of corresponding results for EPA-based placement accuracy based on QS alignments using HMMALIGN. In particular, the performance evaluation especially focuses on the adaptability of QS alignment methods to the underlying, implicit RA structure. While MUSCLE and MAFFT also offer modes for sequence-profile alignment (that can be deployed for QS alignment), the focus is exclusively on HMMALIGN as a representative of monolithic profile-based approaches for the following reasons: MUSCLE offers an option to conduct profile-profile alignments which corresponds to aligning two MSAs. Thus, either all QS need to be represented by a single profile (i.e., they have to be 'pre-aligned' with respect to each other) or MUSCLE needs to be invoked separately for each QS and the individual results will have to be combined afterwards. Representing all QS by a single profile does not represent a good option, since it may be impossible to align the QS to each other if the short fragments do not exhibit sufficient overlap. For the second MUSCLE alternative, it is unclear, how the resulting individual per QS MSAs —possibly containing gaps in the RA as well— can be synthesized/merged into a single, comprehensive MSA. In contrast to MUSCLE, MAFFT offers an analogous option for QS alignment as proposed here. However, in preliminary tests, MAFFT returned considerably worse QS alignments than HMMALIGN, with respect to our evaluation criteria (placement accuracy; see Section 6.2). For the above reasons, we focus on comparing phylogeny-agnostic HMMALIGN performance against phylogeny-aware performance of PaPaRa that is described below.

## 5.1 The PaPaRa Algorithm

PaPaRa is a novel method for short read alignment against a fixed reference MSA (RA) and the corresponding phylogenetic reference tree (RT). It is available as open source code at `http://sco.h-its.org/exelixis/software.html`. The following section describes the initial PaPaRa implementation (denoted as PaPaRa 1.0) introduced in [12]. The underlying idea of PaPaRa is to align the QS against the ancestral state vector of each edge in the RT. These ancestral state sequences are conceptually similar to the profiles used in HMMER. However, PaPaRa does not deploy a probabilistic model because of prohibitive run times. A key difference to HMMALIGN is that, PaPaRa derives one profile per edge (branch) in the RT, as opposed to the single, monolithic profile that represents the whole RA in HMMALIGN. Thus, given an RT with $r$ taxa, $n$ sites, and $q$ QS, PaPaRa needs to execute $O(rq)$ alignment steps or $O(rqn^2)$ operations (the individual QS are usually shorter than $n$, so we assume the that the complexity of each alignment step is in $O(n^2)$). Note that $q$ is typically significantly larger than $r$. Because of this high time complexity, the implementation of PaPaRa also contains a proof-of-concept parallelization. The ancestral state vectors, as used here, provide two different types of information: the ancestral sequence profile and a tree-derived gap signal (see following paragraphs).

### 5.1.1 Ancestral Sequence Profile

After reading the input data, the algorithm visits the $2r-3$ edges of the RT via a depth-first tree traversal, starting at an arbitrary terminal edge leading to a tip. At each edge, it computes the parsimony state-vectors [35, 75] of the

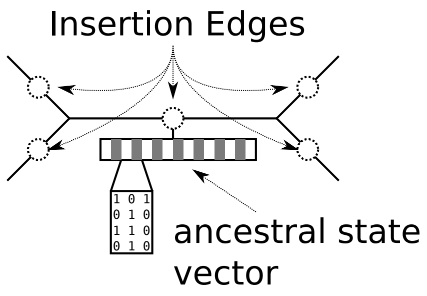


Figure 5.2: Unrooted reference tree (RT) and possible query sequence (QS) insertion positions. The QS are aligned against the ancestral state vectors at the candidate insertion positions.

RT at each end of the edge. The signal from those two state-vectors is then combined using parsimony, to obtain the ancestral parsimony state for an imaginary root-node located on the current insertion edge (Figure 5.2). For DNA data, every edge $b$ in the RT will thus be represented by a parsimony state vector $A_b = A_b^1, ..., A_b^n$, where the individual $A_b^i$ are the parsimony states for each alignment site $i$ of the RA. Each entry $A_b^i$ is a bit-vector; each bit corresponds to a character in the sequence alphabet (see Figure 5.1B). For DNA data, a bit vector at a site $i$ can have the following state set: $A_b^i = a_b^i(A), a_b^i(C), a_b^i(G), a_b^i(T) \in \{0,1\}^4$, where the $a_b^i$ are the bits which correspond to the four DNA characters. For practical reasons, the $A_b^i$ are implemented using one 32-bit integer per site (e.g., $S_b^i = a_b^i(A) + 2a_b^i(C) + 4a_b^i(G) + 8a_b^i(T)$ for DNA data). This approach is not limited to DNA data; it can be extended to alphabet sizes with up to 32 states in the current implementation.

### 5.1.2 Gap Signal Propagation

In addition to the parsimony states, the algorithm also uses phylogenetic information on the gap structure as induced by the tree during the alignment process. This gap information is calculated in conjunction with the parsimony state vectors when the RT is traversed. For each alignment site it recursively computes two flags. One flag (denoted as 'consistent gap'; CGAP) is used for indicating that for a specific site in the RA, there consistently appears a gap. The second flag (denoted as 'potential gap'; OPEN) is used to indicate if the gap status of a site $i$ is inconsistent. This tree-derived gap signal is based on similar ideas as used in PRANK$_{+F}$ [59], which has been designed for de-novo MSA construction. The two 'gap flags' are deployed in an analogous way as 'compulsory gaps' (CGAP) and 'potential free gaps'

(OPEN) in PRANK$_{+F}$. Because the signal is calculated from the tips toward the current insertion edge (Figure 5.3), the recursive algorithm needs to consider three cases for combining gap signals during a post-order tree traversal: TIP/TIP, TIP/INNER, and INNER/INNER. Thus, the combination of the the gap signals from the child nodes in each recursive step is governed by a set of empirical rules. In the TIP/TIP case (the children to the left and right of the node at which we intend to compute the 'ancestral' gap signal are tips) the gap signal coming from the two tips can either be gap or non-gap. If both tips have a gap, the result is CGAP, which indicates that in the subtree defined by the current ancestral node the two tips have a gap signal at site $i$. If only one tip has a gap, the outcome is OPEN, indicating a 'potential gap'. For the TIP/INNER case the flags are computed as follows: If either both child nodes have a gap or the tip has a gap and the ancestral child node has a potential gap (indicated by OPEN), the result is a CGAP. In this case, the 'potential' gap signal coming from the INNER node is upgraded (promoted) to a consistent gap. If only one child node signals a consistent or potential gap, the result is OPEN. Finally, for the INNER/INNER case (i.e., two ancestral child nodes), only two consistent CGAP signals will result in a CGAP at the ancestral node. If only one child node has a CGAP, the result at the ancestral node is OPEN. This rule set for combining and propagating the gap signal through the tree has been derived empirically. While the rule set can evidently be further re-fined, it already yields promising results on real biological data sets. Note that, PaPaRa aligns the QS to ancestral states derived from the edges of the RT. Thus, for each edge, the gap signals of the two adjacent nodes is combined. This combination of gap signals is accomplished by using the same rules (TIP/INNER and INNER/INNER cases) as described above. Essentially, this corresponds to placing a temporary root in the middle of the insertion edge.

### 5.1.3 Dynamic Programming Algorithm

Once the gap signal and the ancestral parsimony state at the candidate insertion edge have been computed, they are deployed to calibrate the alignment scoring scheme for the QS at this edge by modifying the match/mismatch and gap open/extend penalties. Only the CGAP flag and *not* the OPEN flag influences the scoring scheme of the alignment algorithm (see Equation 5.1). In general, the CGAP flag will calibrate the scoring scheme such that aligning QS characters against sites with a CGAP flag is strongly penalized. Opening and extending gaps at these CGAP positions will be preferred. Thereby, if a QS is aligned against the ancestral state of a tree region, where gaps are common for certain alignment sites, it is very likely that the QS alignment

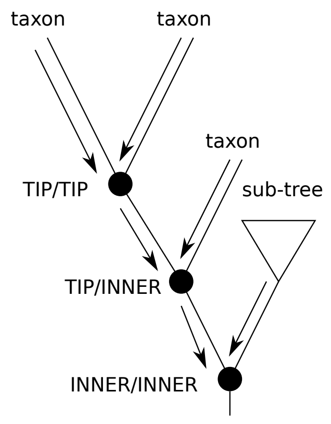


Figure 5.3: Gap signal 'flow' from the tips towards the QS insertion position.

will also contain gaps at these sites.

The actual alignment of the QS against each ancestral state vector is carried out by a standard dynamic programming algorithm for pair-wise alignment using affine gap penalties [38]. Pairwise alignment is conducted with two modifications. Firstly, PaPaRa deploys a 'free shift' or overlapping alignment strategy [43], that is, gaps inserted at the beginning and/or end of the QS are not penalized. Secondly, the affine gap model is only used for inserting gaps into the QS (i.e., deletions in the QS). For inserting gaps into the RA (i.e., insertions in the QS), there is a flat gap penalty. In practice, instead of inserting gaps into the RA, these insertion characters are instead simply deleted in the QS. The rationale for this is that, introducing gaps in the RA does not provide any additional information for QS placement using the EPA. In other words, 'empty' RA columns that entirely contain gaps (modeled as undetermined characters in standard ML implementations) will not affect the EPA placements, since only one QS is aligned at a time. While inserting gaps in the RA may be useful for aligning the QS with respect to each other, the focus here is on evolutionary placement of the QS relative to the RA.

The alignment scoring function is provided in Equation 5.1. The equation recursively defines the score of the dynamic-programming matrix cell $S^{i,j}$ in column $i$ and row $j$ for aligning site $A^i$ of the ancestral state vector against site $B^j$ in the QS.

$$
\begin{aligned}
CG^i &= \begin{cases} 10 & \text{if CGAP is set for site } i \\ 0 & \text{otherwise} \end{cases} \\
(GP^i_{OE}, GP^i_E) &= \begin{cases} (2,1) & \text{if} CG^i = 0 \\ (0,0) & \text{otherwise} \end{cases}
\end{aligned}
$$

$$
\begin{aligned}
M^{i,j} &= \begin{cases} 0 & \text{if } A^i \text{and} B^j \text{match} \\ 3 & \text{otherwise} \end{cases} \\
I^{i,j} &= D^{i,j-1} + 3 \\
D^{i,j} &= min \begin{cases} S^{i-1,j} + GP^i_{OE} \\ D^{i-1,j} + GP^i_E \end{cases}
\end{aligned}
$$

$$
S^{i,j} = min \begin{cases} S^{i-1,j-1} + M^{i,j} + CG^i \\ D^{i,j} \\ I^{i,j} \end{cases} \tag{5.1}
$$

The term $CG^i$ is used to adapt the scoring scheme for sites where the 'constant gap' (CGAP) flag is set. Thereby, matching a QS site against such sites in the RT/RA is substantially penalized while introduction of gaps in the QS at such positions is not. The remaining definitions correspond to a standard dynamic-programming implementation of Gotoh's algorithm [38] for sequence alignment with affine gap penalties. As described above, every state $A^i$ is a bit-vector with one bit per alphabet character. Thus, one may think of the ancestral parsimony state vector as a simple profile, where the bits determine which character of the QS can be aligned for 'free' against an ancestral state character (Figure 5.1B). If, for example $A^i = 1,1,0,0$, this means that, As and Cs in the QS can be matched against alignment site $i$ for this ancestral state vector without incurring a mismatch penalty. Thus, the score $M^{i,j}$ is 0 (i.e., no penalty is induced), if the bit corresponding to character $j$ of the QS is set in $A^i$. Otherwise, the scoring scheme $M^{i,j}$ will return the default mismatch penalty of 3. Note that, the numerical values given in Equation 5.1 represent the default parameters (used in all our experiments), which have been derived empirically. PaPaRa can also deploy user-defined parameters. While there exist more elaborate probabilistic methods (e.g., TKF92 [90]), 'ad-hoc' scoring schemes (e.g., BLAST or Smith-Waterman) are still widely used for bioinformatics analyses. Moreover, because of the high computational complexity of our approach ($O(rqn^2)$), it is currently not computationally feasible to explore more elaborate scoring schemes. In other words, there is a clear trade-off between model accuracy and execution times.

### 5.1.4 Proof of Concept Implementation

PaPaRa 1.0 is implemented in C/C++ as experimental extension of RAxML [79]. It uses the existing routines for parsing alignment files and trees, as well as the existing parsimony implementation. Initially, the algorithm reads and parses the RT, RA, and, the QS. The taxon names in the RT (Newick format) and the RA (relaxed PHYLIP format, see RAxML v7.0.4 Manual; `http://sco.h-its.org/exelixis/oldPage/RAxML-Manual.7.0.4.pdf`) need to be consistent: all taxa in the RT must have a corresponding sequence in the RA. The QS that shall be assigned to the RT can be read from a separate FASTA file or be included in the RA (for details see [17]).

The aligner uses a custom-built sequential dynamic-programming implementation (i.e., the core alignment algorithm is single-threaded and not vectorized). However, as Farrar et al. demonstrated [29, 72] for the smith-waterman algorithm [77], dynamic programming algorithms can be significantly accelerated by means of vectorization. A fundamentally improved version of PaPaRa, which includes a vectorized alignment implementation is presented in Section 5.3.

Alternatively, there is an implementation of a one-sided version of the alignment method, where gaps are only inserted in the QS and not the RA. The respective, simplified dynamic-programming algorithm exhibits fewer dependencies between matrix cell computations. This property can be exploited for further performance improvements. This comes at the cost of alignment quality if insertions (with respect to the sequences in the RA) are common in the QS. For further details please refer to [17].

As already mentioned, PaPaRa relies on a free-shift alignment strategy. Therefore, after the dynamic-programming matrix has been filled, the algorithm searches for the optimal alignment score (minimum) in the last row of the dynamic-programming matrix. This allows for insertion of free gaps at the end of the QS. In standard free-shift alignment procedures, one has to search for the minimum score in the last row *and* the last column of the matrix, because it allows for free gaps at either end of both sequences. As shown in Section 5.2, it is possible to deploy the standard free shift alignment method, which allows for free gaps also at both ends of the RA. This allows PaPaRa to handle cases where the procedure presented here is not applicable (e.g., QS not fully contained in the RA).

## 5.2 Improvements in PaPaRa 2.0

While the proof-of-concept implementation of PaPaRa 1.0 was enough to demonstrate that the basic approach worked, it was fundamentally flawed from a software engineering perspective: because it was originally created as C++ extension to RAxML (ANSI-C codebase), integration into the official release of RAxML was not an option, as this would require to either introduce C++ code to RAxML or porting back PaPaRa to ANSI-C. While PaPaRa 1.0 depends on significant parts of RAxML (i.e., tree-IO, alignment-IO and parsimony state vector generation), the largest part of the RAxML codebase was unused (i.e., ML, tree-search algorithm, MPI infrastructure etc.). Also integration of the new features planned for further extension of PaPaRa would have required more fundamental changes to the RAxML codebase. Therefore PaPaRa 2.0 was created as a complete rewrite, where most of the code not specific to PaPaRa (e.g., tree-/sequence-IO) is kept in a separate library (open source code available under the GPL at https://github.com/sim82/ivy_mike). The open source code of PaPaRa 2.0 is also available under the GPL at https://github.com/sim82/papara_nt.

At the algorithmic level, PaPaRa 2.0 introduces a new, less ad hoc, method to derive the gap signal from the RT and RA, based on a probabilistic model, rather than a set of empirical rules (see Section 5.1.2). Furthermore, the alignment scoring scheme is fundamentally improved by introducing versatile handling of freeshift gaps. Furthermore, PaPaRa 2.0 also allows for optionally inserting gaps into the RA instead of deleting insertions from the QS. At the technical level, it uses a new SSE-based SIMD vectorization, which can speed up the PaPaRa kernel by a factor of 15.

### 5.2.1 Probabilistic Gap Model

Methodologically, the new gap model, represents the most substantial change in PaPaRa 2.0. PaPaRa 1.0, as described in Section 5.1, uses a set of gap-flags (OPEN/CGAP) for each site in the ancestral state vectors to keep track of those sites in the ancestral state vectors that are likely to contain a gap. For positions that contained a gap according to these rules, the scoring scheme of the dynamic programming algorithm in PaPaRa is modified, such that, matches were penalized and gap insertions in the QS are favored.

PaPaRa 2.0 introduces a probabilistic scheme to model gaps (indels). Here, the ancestral state (AS) vectors of each node $x$ are augmented by a set of two probability vectors $L_0^{(x)}$ and $L_1^{(x)}$, which contain the respective probabilities of observing a non-gap or a gap character at each site of the

AS. The number of entries in the two gap probability vectors corresponds to the number of columns (sites) in the RA. For the terminal (tip) nodes of the RT, the values in $L_0^{(x)}$ and $L_1^{(x)}$ are initialized according to the sequence of the corresponding taxon in the RA. For non-gap characters, the elements in $L_0^{(x)}$ and $L_1^{(x)}$ are set to 1.0 and 0.0 respectively. For gap characters, they are set to 0.0 and 1.0. For an inner node $p$ in the RT, the probability vectors are derived from the two child nodes $c_1$ and $c_2$ according to a simple time-reversible substitution model that is computed recursively with the Felsenstein pruning algorithm [31]. The model uses a set of transition probabilities $P_{ij}(t)$, where $i$ and $j$ take values 0 and 1 for the non-gap and gap cases. Each $P_{ij}$ is the probability of a state $i$ transforming into $j$ within time $t$ (e.g., $P_{01}(0.1)$ is the probability of a non-gap character turning into a gap within time 0.1). The edge values $t$ are fixed and correspond to the edge lengths in the RT that are estimated using the nucleotide data and standard likelihood model. The transition probability matrix for time $t$ is obtained by $P(t) = e^{Qt}$, where $Q$ is the instantaneous transition rate matrix, that is initialized by the prior probabilities $\pi_{\text{non-gap}}, \pi_{\text{gap}}$ of observing non-gap/gap characters in the RA (for more details about this binary model, see [55]):

$$Q = \begin{vmatrix} -\pi_{\text{gap}} & \pi_{\text{gap}} \\ \pi_{\text{non-gap}} & -\pi_{\text{non-gap}} \end{vmatrix} \tag{5.2}$$

As described in [31], the probability vectors of a parent node $p$ are calculated from the probability vectors of child nodes $c_1$ and $c_2$ and the respective edge lengths $t_1$ and $t_2$ according to

$$L_k^{(p)} = \left( \sum_{i \in \{0,1\}} P_{ki}(t_1) L_i \right) \left( \sum_{j \in \{0,1\}} P_{kj}(t_2) L_j \right) \tag{5.3}$$

As mentioned above, PaPaRa 2.0 works by aligning QS against AS vectors that represent the edges in the RT. The corresponding gap probability vectors of an AS are generated by inserting a temporary virtual root node $x$ in the center of the edge under consideration. The corresponding $L_0^{(x)}$ and $L_1^{(x)}$ are then calculated according to Equation 5.3.

The gap probability vectors are not directly used in the dynamic programming algorithm. Instead of directly modulating the scoring scheme with the gap/non-gap probabilities, PaPaRa 2.0 derives a single binary gap-flag per site. The flag at a site is set, if the probability of observing a non-gap character ($L_0^{(x)}$) at that site is smaller than the probability of observing a gap character ($L_1^{(x)}$). Thereby, the integration of the gap signal into the scoring scheme is analogous to PaPaRa 1.0 (see Equation 5.1). The main difference is that, the gap-flag is no longer derived according to some empirical rules, but using a probabilistic model. There are two main arguments in favor of 'compressing' the gap probabilities into a binary flag rather than using them directly in the alignment scoring scheme. Firstly, the overall scoring scheme of the alignment algorithm is not based on probabilities, but on empirical values. Thus, there is no obvious and intuitive approach for directly integrating the gap-probabilities into this ad hoc empirical scoring scheme. Nonetheless, we carried out tests that directly use the gap probabilities to modulate the scoring scheme. However, this approach performed worse with respect to accuracy (results not shown) than the simple flag-based method described here. Secondly, with respect to program run-times, using the gap probabilities directly, would require using floating point arithmetic in the score calculations, which are generally slower than integer arithmetic (e.g., on current Intel CPUs, throughput of most integer instructions is twice as high as for their corresponding floating point instructions; see `http://www.agner.org/optimize/instruction_tables.pdf`). Using 16-bit integers for calculating and storing alignment scores also allows for a more efficient SIMD vectorization than, for instance using 32-bit single precision floating point values (see Section 5.3).

### 5.2.2 Unified Use of Freeshift Gaps

Previously, PaPaRa 1.0 used distinct approaches for penalizing gaps in the QS and in the RA (i.e., deletions and insertions). Gaps in the QS (deletions) were penalized using affine gap penalties, that is, there are distinct penalties for opening and extending gaps. Gaps on either end of the QS were not penalized (freeshift gaps). The rationale for using the freeshift penalty is, that the QS are short compared to the RA and are expected to be fully contained within the genomic region (e.g., a single 16S gene) covered by the RA. Thus, gaps on the RA side (insertions) were treated with a constant gap extension penalty, and gaps on either side of the RA were not free. When gaps are added to either ends of the RA, this effectively means that the QS emitting the gap is not fully contained within the genomic region of the RA. While the initial assumption was that this would not occur, it turned out to be overly restrictive in some real use cases. Therefore, PaPaRa 2.0 was extended to also allow for freeshift gaps on either side of the RA. Thereby, QS that do not entirely fit within the RA are allowed to have an unpenalized 'hangover' at the boundaries of the RA. The scoring scheme was further modified, such that gaps inserted into the RA are also treated with affine gap penalties. The respective recursive formula for the dynamic programming alignment of an AS $A$ and a QS $B$ is given in Equation 5.4:

$$
\begin{aligned}
CG^i &= \begin{cases} -3 & \text{if GAP-flag is set for site } i \\ 0 & \text{otherwise} \end{cases} \\
(GP_O, GP_E) &= (-3, -1) \\
(GP_O^i, GP_E^i) &= \begin{cases} (GP_O, GP_E) & \text{if} CG^i = 0 \\ (0,0) & \text{otherwise} \end{cases} \\
M^{i,j} &= \begin{cases} 2 & \text{if } A^i \text{and} B^j \text{match} \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

$$
\begin{aligned}
I^{i,j} &= max \begin{cases} S^{i,j-1} + GP_O + GP_E \\ I^{i,j-1} + GP_E \end{cases} \\
D^{i,j} &= max \begin{cases} S^{i-1,j} + GP_O^i + GP_E^i \\ D^{i-1,j} + GP_E^i \end{cases}
\end{aligned}
$$

$$
S^{i,j} = max \begin{cases} S^{i-1,j-1} + M^{i,j} + CG^i \\ D^{i,j} \\ I^{i,j} \end{cases} \tag{5.4}
$$

The basic structure of the scoring method is similar to the original one described in Section 5.1. There we also give the definition of a match between $A^i$ and $B^j$. The main difference is the term $I$ that accommodates affine gap penalties for insertions (i.e., gaps in the RA), which were previously only allowed for deletions (i.e., gaps in the QS). A more subtle difference is that, for the insertion score calculation ($I$) we now use the fixed gap-open/gap-extension penalties $GP_O$ and $GP_E$, whereas for deletions ($D$) we used the scores as modulated by the site-specific gap-flags ($GP_O^i$ and $GP_E^i$). Finally, the score minimization used in PaPaRa 1.0 was transformed into a maximization, such that match scores are positive values, while gap penalties correspond to negative values (i.e., we maximize the score). Using positive values for matches and negative values for mismatches simply yields the scoring scheme more intuitive and easier to interpret.

### 5.2.3 Inserting Gaps into Reference Alignments

To extend the applicability beyond downstream analyses using phylogenetic placement methods (EPA, pplacer), PaPaRa 2.0 also offers an option to insert

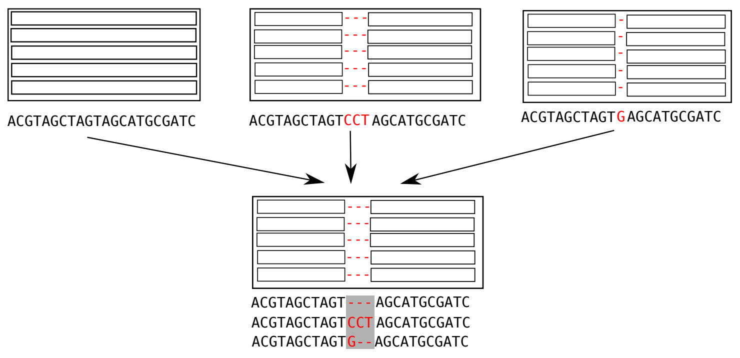

ACGTAGCTAGTAGCATGCGATC
ACGTAGCTAGTCCTAGCATGCGATC
ACGTAGCTAGTGAGCATGCGATC
ACGTAGCTAGT---AGCATGCGATC
ACGTAGCTAGTCCTAGCATGCGATC
ACGTAGCTAGTG--AGCATGCGATC

Figure 5.4: Merging overlapping gaps introduced to the RA by different QS

gaps into the RA instead of deleting insertion characters from the QS. This allows for preserving the complete sequence information of a QS. However, based on the underlying PaPaRa principle, it still aligns only one QS at a time against the RA and RT. In other words, if two or more QS insert gaps into the RA at the same position, these gaps have to be merged. The merging procedure currently does not aim to correctly align QS characters with respect to each other within the RA gaps (see Figure 5.4). The result of this merging procedure resembles a full MSA consisting of the previous RA and the newly inserted QS. Due to the simplistic nature of the merging procedure, QS characters which are aligned against a gap in the RA (e.g., the QS part highlighted by the gray box in Figure 5.4) should not be interpreted as being correctly aligned, even if they appear as aligned characters in the output file. When the resulting MSA is used with the EPA, the parts of the QS which are aligned against gaps in the RA will not contribute to the likelihood signal, since gaps are treated as missing data in all standard ML implementations. Because of this mathematical property, the QS characters inside these RA gaps are simply 'ignored' by the EPA. When a PaPaRa alignment is used for downstream analysis with the EPA or pplacer, it is therefore suggested to switch off the RA gap insertion feature.

## 5.3 Parallelism in PaPaRa

As described in Section 5.1.4 the PaPaRa algorithm exhibits a high degree of parallelism, because a —typically— large number of independent pair-wise QS and AS alignment calculations need to be performed (for $q$ QS and a RT with $r$ taxa, the number of independent pair-wise alignments is $qr$).

**Multithreading** PaPaRa 1.0 already deployed multithreading to exploit this coarse-grain parallelism, obtaining good speedups on multi-core architectures. As mentioned in Section 3.4, most modern processors also offer SIMD instructions (most commonly SSE [1] on Intel and AMD CPUs) which can be used to exploit data-parallelism at a more fine-grained level.

**SIMD Vectorization** Using SIMD instructions to accelerate dynamic programming algorithms (e.g., the smith-waterman algorithm), has been extensively studied. See [29, 72] for reviews of previous work. Generally, the techniques to exploit SIMD parallelism in pair-wise sequence alignment can be categorized into intra-sequence and inter-sequence approaches. Intra-sequence approaches accelerate the alignment of a single sequence pair (e.g., the SWPS3 program [29]). The main drawback of this approach is that, when vectorizing the calculation of a single dynamic-programming matrix one has to rely on exploiting wave-front parallelism which limits the degree of parallelism that can be achieved. In contrast to this, inter-sequence vectorization approaches conduct several simultaneous pair-wise alignments (compute multiple matrices) with a single vector instruction. This is typically achieved by aligning a single sequence against multiple (usually 4, 8, or 16) other sequences at a time. In other words, instead of calculating a single matrix cell $S^{i,j} \in \mathbb{Z}$ per step/instruction, SIMD operations are used to calculate multiple matrix cells $\bar{S}^{i,j} \in \mathbb{Z}^w$, where $w$ is the vector width of the SIMD instructions. The inter-sequence method is particularly efficient if a large number of pair-wise all-against-all alignments need to be conducted, as is the case with PaPaRa.

The actual value $w$ depends on the data type used for score calculations. Depending on the type of input data, PaPaRa 2.0 internally uses either 16 bit (DNA data) or 32 bit (protein data) integers with corresponding vector widths of 8 and 4 (i.e., 8x16bit and 4x32bit vectors) respectively. The reason for using different integer types and thus vector widths is that, for greater efficiency, the alignment kernel in PaPaRa 2.0 uses the same data types to store scores as well as the elements of the ancestral state vectors. For protein data, the ancestral state vectors require 20 bits per site, and therefore need to be stored in 32 bit integer values, which correspond to a vector width of 4. For DNA data there are only 4 bits per ancestral state vector site, which can be stored in a 16 bit integer. In this case we also use 16 bit values in the alignment scoring algorithm. For the typically short QS, numeric overflow of these 16-bit integers is currently not an issue.

The vectorized implementation is analogous to the corresponding sequential dynamic programming implementation (Equation 5.4). The main difference is that, for vector data one can not directly use the standard C++ data types (PaPaRa 2.0 is implemented in C++) and arithmetic operators. Instead, one has to use vector intrinsics. For Intel SSE3 intrinsics, for instance, the `__m128i` data type can either be interpreted as a vector of 8x16bit or 4x32bit integers. It replaces the `short` and `int` data-types and the intrinsic vector operation `_mm_add_epi16` is used instead of the `+` operator for 8x16bit vectors. Corresponding intrinsic functions exist for most of the built-in C/C++ data types and arithmetic operations, which makes inter-sequence vectorization relatively straight-forward.

To further improve performance, the vectorized implementation does not keep the whole dynamic-programming matrix in memory. This is because, storing the entire matrix only becomes necessary if the actual alignments shall be generated via a trace-back step. In PaPaRa 2.0 the vectorized algorithm is only used for initial all-against-all score calculations because this step largely dominates execution times. It is therefore sufficient to simply store one row of the matrix at a time (see Equation 5.4: the values in matrix row $j$ only depend on values from rows $j$ and $j-1$. Therefore, the alignment kernel can overwrite the matrix values from row $j-1$ with the new values calculated for row $j$). To further increase the cache efficiency of the aligner, the matrix score calculations are carried out in a block-based way: Instead of calculating the matrix values for an entire row at a time (Figure 5.5a), only a subset (block) of $B$ columns/entries are calculated (Figure 5.5b). The block size $B$ defines how many values of the previous matrix row need to be accessed, and is set (e.g., empirically depending on the cache size of the target CPU) such as to keep the referenced values that are read and written in the L1 cache at all times. The rightmost column of each block is stored and used as starting value for the next block (Figure 5.5b; highlighted columns). An earlier SIMD implementation without this block-based optimization yielded a substantial slowdown when the RA length exceeded 10000 columns [6]. The blocking optimization alleviates this performance degradation.

## 5.4 PaPaRa for Graphics Processing Units (GPUs)

To explore further possibilies of accelerating PaPaRa, N. Alachiotis ported the alignment kernel of PaPaRa 1.0 to NVIDIA GPUs using OpenCL (Open Computing Language [2]). This work was partly conducted as an experiment and applied a ‘competing programmer’ approach, that is, roughly the same amount of work was spent on porting and tuning the GPU kernel as was spent on the SIMD CPU implementation. Note that the GPU kernel was compared against an intermediate vectorized version of PaPaRa 1.0, which

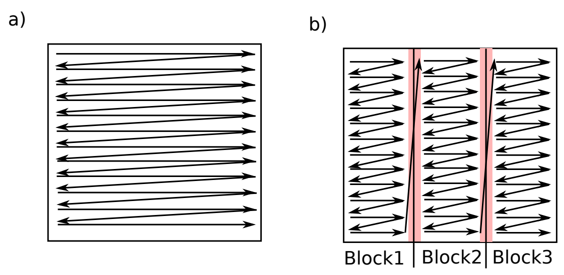


Figure 5.5: The order of matrix cell calculations in a standard (a) and block-based (b) inter-sequence dynamic programming implementation. The block-based approach increases data access locality and thereby cache efficiency.

was not published separately from [6]. The basic architecture of current GPUs is comparable to other SIMD (vector-)architectures. Because of their greater flexibility (i.e., there are no fixed vector operations, but rather multiple threads that can execute the same instructions in lockstep), we will refer to GPUs as SIMT (single instruction multiple threads) architectures.

Both, the SIMD and the SIMT implementation of the PaPaRa kernel use an inter-sequence memory organization (see Section 5.3 and [6, 72]). On the SIMD (Intel x86) architecture, this memory organization facilitates the vectorization, while on the SIMT architecture it allows for coalesced global memory (RAM) accesses and thereby enables high data throughput (i.e., in theory a SIMT architecture could read/write data to/from widely scattered memory locations, but for good performance, data should be retrieved from contiguous addresses). For a detailed description of the GPU implementation please refer to [6].

To leverage the maximum computing power available in current desktop systems we created a hybrid version of PaPaRa, consisting of the vectorized, multi-threaded CPU implementation and the GPU implementation. Internally, the CPU and GPU alignment cores compete for work during the PaPaRa alignment phase (see [6] for the technical details of the work distribution). We demonstrate that on a typical desktop system equipped with a low-cost gaming GPU the overall system performance of this CPU-GPU system can be more than doubled, compared to a stand-alone CPU implementation (see Section 6.5.6).

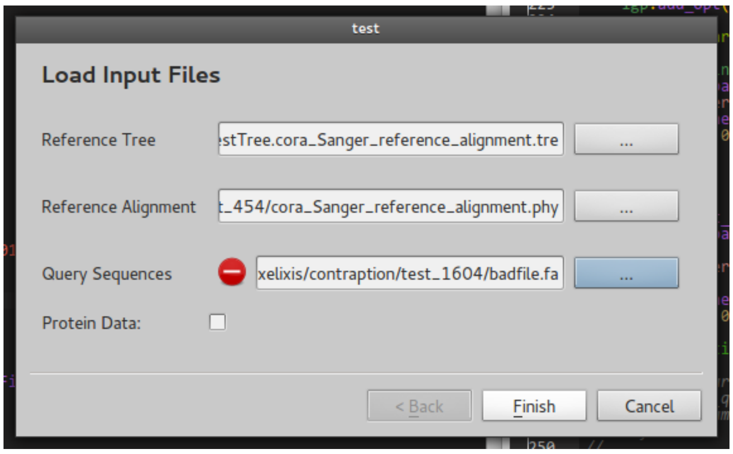
## 5.5 Interactive graphical frontend: Visual PaPaRa

One of the design goals of PaPaRa 2.0 was to make the code modular and reusable. It was therefore relatively straight-forward to transform the PaPaRa 2.0 core into a library that can be used by other software. To test the limitations of this approach and to make the algorithm more attractive for users (not familiar with the UNIX command-line), the PaPaRa 2.0 core was directly integrated into Visual PaPaRa, an interactive graphical frontend. Visual PaPaRa was developed in a platform independent manner based on Qt (`http://qt.nokia.com/products`) and is available for Linux, Windows and Mac OSX (GPL source code at `https://github.com/sim82/contraption`).

Visual PaPaRa includes an interactive alignment viewer, which can be used to assess the result of the QS alignment. The direct integration of the PaPaRa 2.0 core in the frontend made it possible to rapidly assess changes caused by different alignment parameter settings almost interactively (depending on the computional requirements of the data set). Figure 5.8 shows the main window of the Visual PaPaRa for an example data set.

**Input Data** When the program is started, the user is prompted to specify the input files via a dialog (Figure 5.6). The user has to specify valid files for the RT, RA and QS. The initial dialog performs some basic sanity checks of the input files and warns the user when they obviously contain the wrong data (e.g., the user accidentally selected a phylip file in the RT field). The sanity checks are only intended as a quick (i.e., constant time) check and only examine the first few characters of the files. Once the files have been specified, the user can close the dialog by pressing the 'Finish' button. The program will then read in the input files and open the main window. If the input files can not be processed properly, despite the basic sanity check (e.g., the internal structure of a newick tree file is incorrect), an error message is shown to the user.

**Main Window** Initially the main window is almost empty, except for the scoring parameter definition fields on the lower right hand corner of the window (Figure 5.7). The user can specify the numerical scoring parameters used by the alignment scoring function (Equation 5.4: open, extend, match and cgap are used for $GP_O$, $GP_E$, $M^{i,j}$ and $CG^i$). With the RefGaps checkbox, the user can activate/deactivate gap insertion into the RA (see Section 5.2.3). The alignment procedure with the specified parameters is then started by pressing the *run* button in the lower right hand corner of the main



stTree.cora_Sanger_reference_alignment.tre

t_454/cora_Sanger_reference_alignment.phy

xelixis/contraption/test_1604/badfile.fa

Figure 5.6: Initial dialog of Visual PaPaRa. The user is prompted to select the three input files: Reference Tree (RT), Reference Alignment (RA) and Query Sequences (QS). The warning symbol next to the QS selection field indicates that the user has selected either a non-existing file or a file of the wrong type.

window. When the parameter values are changed by the user, the alignment procedure must be re-run to update the resulting alignment. The *run* button is highlighted in red to signal that the parameters have been changed and the alignment is therefore no longer up-to-date. Once the QS alignment has been calculated, the RA is shown in the alignment viewer in the upper half of the main window, while the aligned QS are shown in a corresponding viewer in the lower half. The zoom control slidebar on the right border of the window allows to zoom in and out in the RA and QS alignment viewers. A high zoom setting (Figure 5.8a) allows to view details of the QS alignment, while a low setting (Figure 5.8b) provides a big picture of the QS alignment. Changing the parameters and seeing the resulting changes in the alignment result in quasi-realtime allows the user to iteratively tune the alignment parameters.

The current alignment (i.e., the RA and the aligned QS) can be saved in a phylip file using the *Save As...* button in the lower left hand corner of the main window. The alignment kernel used in Visual PaPaRa corresponds to the parallelized and vectorized PaPaRa 2.0 alignment kernel. By default, Visual PaPaRa will use all available CPU cores (in a future version, the number of cores will be specified by the user). Visual PaPaRa requires a SSE3-capable CPU.

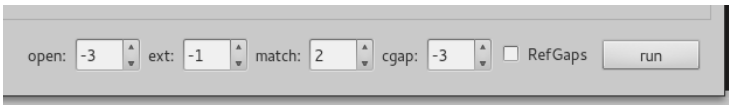
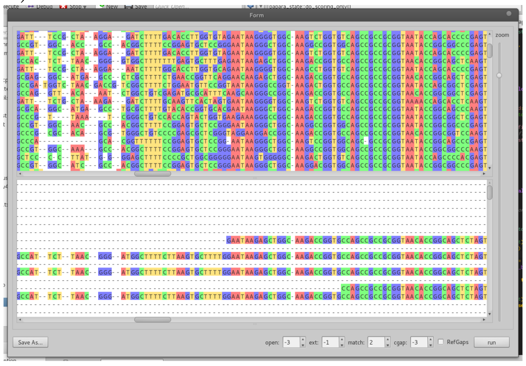

open: -3 ext: -1 match: 2 cgap: -3 RefGaps run

Figure 5.7: Control fields for the PaPaRa algorithm in the main window.

a)



b)

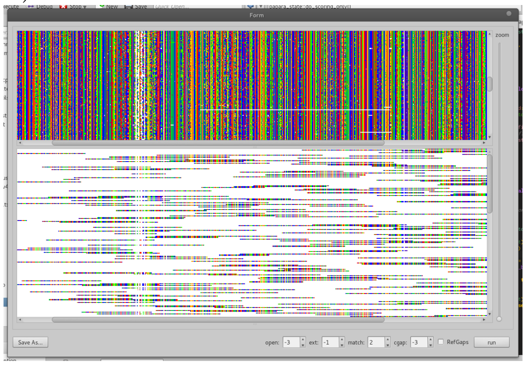


Figure 5.8: The main window of the PaPaRa GUI, using a high (a) and low (b) zoom setting on the build-in alignment viewer.

## Summary

The present chapter covered the problem of aligning short sequence reads against a given reference alignment and the corresponding reference phylogeny. It introduced the PaPaRa algorithm, which at time of publication was the only algorithm specifically designed for this purpose. Because the field is developing rapidly, two version of the algorithm have already been released incrementally: The initial proof-of-concept implementation (PaPaRa 1.0), as well as a fundamentally improved re-implementation (PaPaRa 2.0). Program run-time is an important aspect, especially because the algorithm has to be able to cope with the ever increasing amount of sequence data produced by NGS methods. Therefore, PaPaRa 2.0 is able to exploit parallelism using SIMD vectorization *and* multi-threading.

The next chapter will present the experimental evaluation of the EPA and PaPaRa.

CHAPTER 6

# Results and Discussion

This chapter contains the evaluation of the EPA and PaPaRa, the two novel algorithms introduced in Chapters 3, 4 and 5. For both algrithms, this chapter presents accuracy as well as run-time evaluations. Section 6.3 covers the EPA, Section 6.4 covers the application of the EPA on non-molecular data and Section 6.5 covers PaPaRa. The evaluation results have previously been presented in the corresponding publications: The evaluation of the EPA and its parallel version in [12, 82], the EPA on non-molecular data in [15], and PaPaRa and its GPU version in [6, 13, 17].

## 6.1 Data Sets

To test the accuracy of the EPA and competing approaches, we used eight reference alignments (RA) of nucleotides or amino acids containing 140 up to 1,604 sequences. The experimental data span a broad range of organisms and include rbcL genes (D500), small subunit rRNA (D150, D218, D714, D855, D1604), fungal DNA (D628) and amino acid sequences from *Papillomaviridae* (D140). For each set we computed the most likely tree and obtained bootstrap support (BS) [81] values for the internal edges; this ML tree is denoted as RT. The data sets are available at `http://www.exelixis-lab.org/epaData.tar.bz2`. The selection of the data sets and data types per se is not important, as long as QS with well-supported positions can be extracted. It should be noted that the number of data sets that could be assessed was limited by the excessive computational requirements of the leave-one-out experiments described in Section 6.3. The question we intend to answer by these experiments is this: when a full-length sequence is pruned from the

| Data | type | length | # taxa | # QS | # inner QS |
|---|---|---|---|---|---|
| D140 | AA | 1104 | 140 | 95 | 9 |
| D150 | N | 1269 | 150 | 66 | 10 |
| D218 | N | 2294 | 218 | 80 | 14 |
| D500 | N | 1398 | 500 | 205 | 29 |
| D628 | N | 1228 | 628 | 210 | 20 |
| D714 | N | 1241 | 714 | 293 | 61 |
| D855 | N | 1436 | 855 | 344 | 48 |
| D1604 | N | 1276 | 1604 | 541 | 83 |

Table 6.1: Data sets used for evaluation of the Evolutionary Placement Algorithm (EPA). The columns contain (from left to right): the name of the data; the type of the data (N: nucleotides; AA: amino acids); the length of the data (i.e., number of sites in the alignment); the number of taxa (# taxa); the number of query sequences (# QS); and the number of inner query sequences (# inner OS) (for definitions of QS and inner QS, see the main text).

RA and RT, can the EPA place a reduced length sub sequence into approximately the same edge of the reduced RT from which the full-length sequence was pruned? We specifically did not include real metagenomic data sets in our study because the phylogenetic positions of the QS are unknown. Moreover, real metagenomic data sets do not allow for comparing the placement accuracy, or inferred placement position, between full-length and short-read sequences, whereas using a full-length sequence alignment and emulating short-reads allows for such a comparison. Therefore, we emulated metagenomic data sets using real-world alignments with ML trees and BS support, which is as close as one can get to reality for assessing placement accuracy using real sequence data. The nucleotide data sets are also used for evaluating the PaPaRa algorithm in Section 6.5. To analyze real metagenomic data we used the parallel version of the EPA, which can be applied to very large, real metagenomic data sets (4,874 RS and 100,627 QS [82]).

## 6.2 Methods to measure Distances within Trees

The accuracy evaluations in this chapter depend on measuring distances between different edges of a phylogenetic tree. The background is that the accuracy evaluations are based on comparing actual placement positions, generated with the EPA, to a known reference position. To quantify the accuracy, we use two distance measures that are based on the topology and edge

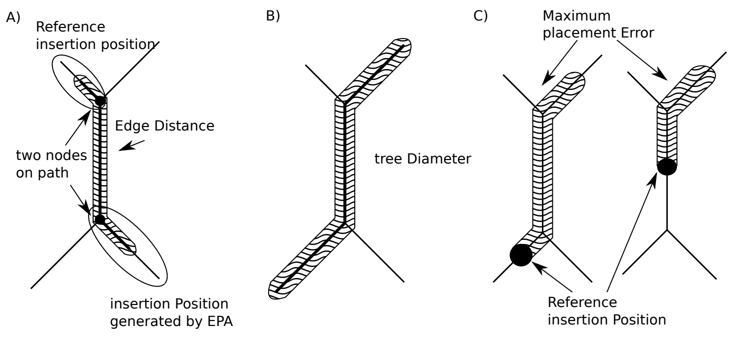

A)
Reference insertion position
Edge Distance
two nodes on path
insertion Position generated by EPA
B)
tree Diameter
C)
Maximum placement Error
Reference insertion Position

Figure 6.1: Illustration of the tree-based distance measures. (A) Example tree with two edges (original and insertion edge) highlighted. There are two nodes on the path, so the node distance is 2. The edge distance corresponds to the length of the connecting path, where of the two end edges only half of the edge length is used. (B) Tree diameter which is used to normalize the edge distance. (C) The maximum placement error for two exemplary reference insertion position.

lengths of the original ML tree: The Node Distance (ND), is the unweighted path length in the original RT between the two edges. This corresponds to the number of nodes located on the path that connects the two placement edges (Figure 6.1a) and represents an absolute distance measure. The second measure is the sum of edge lengths on the path connecting the two edges. This measure also includes 50% of the length of the insertion edge and 50% of the length of the original edge (Figure 6.1a).

For comparability between different trees and in order to obtain a relative measure, we normalize the edge path length by dividing it by the maximum tree diameter (Figure 6.1b). The maximum diameter is the edge path of maximum length between two taxa in the RT. This distance measure is henceforth denoted as normalized edge distance (NED%). For later experiments (i.e., the evaluation of PaPaRa in Section 6.5) we use a revised scheme for normalizing the edge distance: Rather than normalizing it by the tree-diameter (longest path in the tree), we now deploy a position-specific worst-case placement error (Figure 6.1c). This position-specific placement-error corresponds to the QS-specific worst-case scenario, that is, we normalize by the longest path from the 'true' insertion position to a terminal edge.

## 6.3 Placement Accuracy of the EPA

In this section we present the evaluation of the EPA algorithm. It is analyzed in terms of its placement accuracy as well as run-time performance and parallel scalability.

### 6.3.1 Generation of emulated QS

To evaluate the accuracy of our algorithm, we pruned one candidate QS at a time from the existing ML trees before reinserting the QS into the tree. We only pruned and reinserted those QS that were associated with high BS scores in the RT in order to assess placement accuracy for taxa whose position in the original tree is reliable. A candidate QS is considered to have high BS, when the BS of either one of the two edges to which the taxon is attached is $\geq 75\%$ and the other one leads to a tip (Figure 6.2a), or if both of these edges have a BS $\geq 75\%$ (Figure 6.2b). The 75% threshold reflects the typical empirical cutoff that is widely used in phylogenetics [41]. For each QS, a reduced RT is derived by pruning the respective tip from the original tree. The QS associated to that taxon is then placed onto the reduced tree (Figure 6.2c) with our EPA algorithm.

In our test data sets, the QS were always derived from the full-length sequences in the RA. In a typical application scenario, however, the placement algorithm will have to cope with QS that are significantly shorter than the full-length sequences in the RA. Hence, we carry out a systematic assessment of the placement accuracy as a function of the QS length, by artificially shortening the full-length sequences via gap insertions. We deployed three distinct methods to produce a QS that are listed by order of increasing biological realism. In the following, the term candidate QS (CQS) refers to the full length sequences pruned from the RT, while QS refers to the shortened sub sequences:

A first method to emulate short reads involved randomly replacing existing characters by gaps (see Figure 6.2c). While this method arguably does not reflect a real usage scenario, it provides a means to systematically assess the placement of QS over a wide variation of "virtual read lengths", while minimizing the influence of the position specific placement accuracy variation. Position specific effects on placement accuracy have previously been identified as a problem by Chakravorty et al. [23]. For instance, in bacterial 16s rRNA there are nine "hypervariable regions" (V1–V9). Sequence data from individual hypervariable regions (or from combinations of them), was shown to allow for varying levels of differentiation between species [23]. Mul-

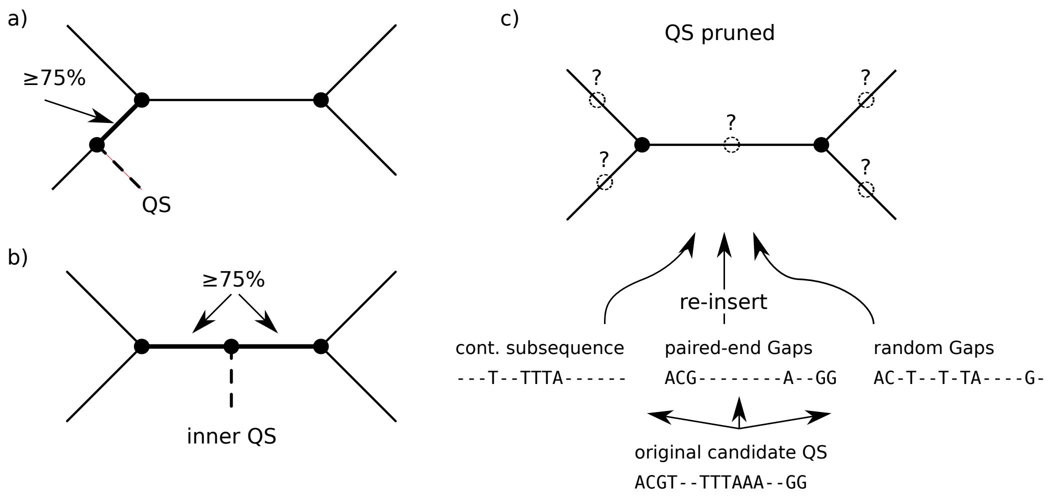


Figure 6.2: Illustration of the criterion for the query sequence (QS) selection and experimental setup. (a) Candidate QS belongs to sub tree of size 2 that is connected to the tree by a well supported edge. It has one other tip as direct neighbor. QS with this property will be referred to as outer QS. (b) Candidate QS is connected to the tree by two well supported edges. QS with this property will be referred to as inner QS. (c) Experimental setting: re-insert shortened candidate QS in to pruned RT. We use three different ways of generating QS with desirable features: contiguous subsequences, paired-end reads and random gaps.

tiple placement runs were conducted for QS with the relative proportion of non-gap characters set to 10%, 20%,..., 90%, up to the full sequence length. Because the sequences have been extracted from the original MSA, the remaining non-gap characters are still correctly aligned to the RA. Because the proportion of gaps is calculated relative to the length of the RA, the maximum proportion of available non-gap characters is alignment dependent. We emphasize that, mathematically, the introduction of random gaps does not influence the calculation of the likelihood function, because the model assumes independence between sites. The results from these experiments show the qualitative relationship between QS length and placement accuracy, in a way that is not feasible with the more realistic QS generation method based on contiguous sub sequences described below. In the evaluations based on contiguous sub sequences, we randomly sample multiple sub sequences per CQS for stability reasons. This means that we have to limit the range of sub sequence lengths to keep the computational requirements of the evaluations manageable.

The second method to artificially shorten CQS involved randomly sampling contiguous sub sequences from each full length CQS (see Figure 6.2C). This method to produce QS closely reflects the main EPA application scenario. Typically, a large number of short sequence reads generated by next generation sequencing methods from unknown positions, will need to be placed onto a RT. For each CQS we sampled 20 QS with uniformly distributed positions and normally distributed lengths (mean length: $200 \pm 60$ bp for nucleotides; $70 \pm 20$ for amino acids). This roughly corresponds to the read lengths generated by current high throughput sequencing technologies. We sampled 20 QS per CQS to minimize the position-specific QS placement bias. Because of the high computational requirements of placing 20 QS per full-length sequence, we did not repeat this evaluation for different mean read-lengths. To assess the relationship between read length and placement accuracy we conducted the random-gap evaluation. Because a method is needed to align the short reads to the RA in a typical analysis using EPA, we also assessed placement accuracy using re-aligned QS and compared it to the QS placement accuracy obtained for the original alignment. We did only conduct the re-alignment experiment for the QS creation methods that produce contiguous sub sequences. We used HMMER to re-align the QS to a profile Hidden Markov Model (HMM) of the RA (sequence-to-profile alignments with MUSCLE and MAFFT resulted in slightly inferior placement accuracy; data not shown). Because the re-alignment procedure is not an integral part of the EPA, and future developments could potentially improve the re-alignment quality, we present results for the EPA with and without HMMER re-alignment.

A third method to produce QS involved generating these such that they correspond to paired-end reads (see Figure 6.2c; in this experiment we excluded data set D140 because it is a multi-gene alignment of amino acid sequences). Thus, in contrast to the previous methods, the position of the extracted sub sequences within the gene remains fixed. This modification reflects another real-world scenario of the EPA, since paired-end sequencing is a widely used technique. Once again, we conducted our placement accuracy assessment on paired-end reads that were artificially generated from the CQS by replacing all characters in the middle of the sequence by gaps. The artificial paired-end reads were 2x50 bp and 2x100 bp in length.

### 6.3.2 Comparison to Placements based on pair-wise Sequence Similarity

We conducted our EPA accuracy evaluation by comparison to a typical application scenario, in which similarity-based tools such as BLAST are used to assign a QS to the most similar RS. In this setting, a QS will always be assigned to one of the terminal edges of the RT. As mentioned above, for the EPA tests we can choose to re-use the alignment information from the original MSA from which the QS were generated. With BLAST we do not have this option, so all QS will effectively be re-aligned against the RS. For this reason we compare BLAST against the EPA with and without previous QS re-alignment using HMMER.

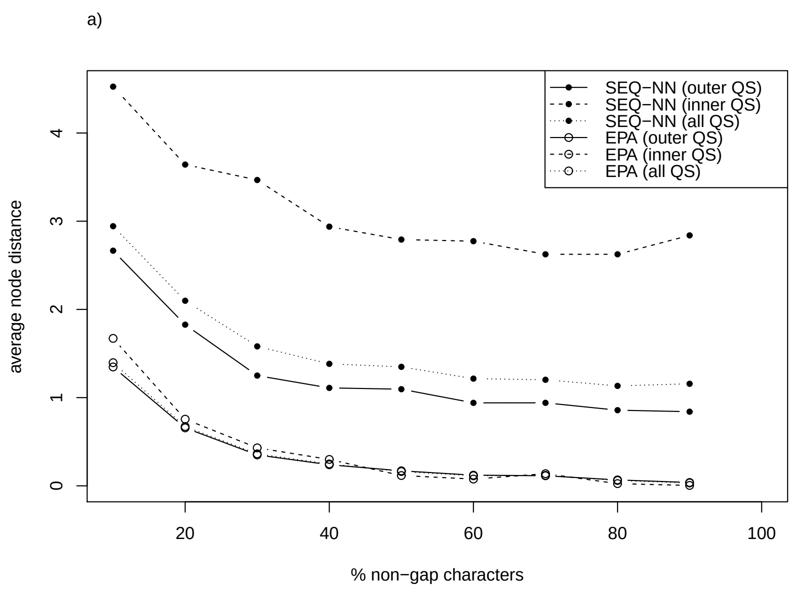
For all tests involving BLAST, we removed all gaps from the MSA and computed a BLAST database for each data set. We also removed all gaps from the candidate QS and concatenated the two ends of the artificial paired-end reads into one sequence. Searches with those sequences were conducted against the corresponding BLAST database. The default parameters of the BLAST program from the NCBI C Toolkit were used for character matches/mismatches (scores 1 and -3) and gaps (non-affine gap penalty of -1). The default values from the NCBI BLAST web site with affine gap penalties were also tested, but produced slightly worse placement results and higher run times than the default settings. Using BLAST has the disadvantage that the information stemming from the RA that is present in the QS cannot be used, so for fairness the BLAST placements should be mainly compared to EPA placements including previous QS re-alignment.

For the tests on random gap QS, we did not use BLAST for comparison, because it is not well suited for aligning such QS. The gap model as well as the local alignment algorithm rely on contiguous sequence stretches of certain lengths, which were not present in the random gap QS. For the random gap evaluation we used a custom distance measure to calculate the pair-wise sequence similarities as an alternative to BLAST. It is defined as follows: For the two aligned sequences in question, we count the number of positions, where two different, non-gap characters are aligned with each other. This measure corresponds to the Hamming distance [39], where a gap is interpreted as a place-holder for any character. Placements derived from this distance measure will be referred to as sequence-based nearest-neighbor (SEQ-NN) placements.

### 6.3.3 Placement Accuracy for Random Gap QS

To assess the influence of QS length and, at the same time, reduce the impact of positional variability on placement accuracy, we tested the accuracy of the EPA on random-gap sequences of various lengths. Placements were carried out on the 8 data sets for varying emulated read lengths. Figure 6.3 provides a detailed plot of the accuracy as a function of the proportion of gaps, averaged over all candidate QS and over all data sets (respective plots for the individual data sets can be found in the supplementary material of [12]). As a measure of accuracy we use the distance between the insertion edge and the original edge. Therefore, a lower distance indicates higher accuracy. For each placement method, we show the distances for the set of all QS, as well as for two disjunct QS subsets (Figure 6.2): *Outer QS*, are QS that have been pruned from an edge leading to a tip in the QS (i.e., they have a direct neighbor), and *inner QS*, which do not have a direct neighbor in the RT.

As expected, the general trend is that placement accuracy increases with QS length. For all three QS subsets, the EPA achieves higher placement accuracy than SEQ-NN. Generally, the distances of the EPA placements are at least two times lower than for SEQ-NN. For SEQ-NN, the placement accuracy is considerably lower for the inner QS subset compared to the full QS set and the outer QS set. Placement based on pair-wise sequence similarity is harder for inner QS than for the outer QS because their CQS do not have direct neighbors in the RT. This decrease of placement accuracy is independent of the QS lengths. For the EPA, the accuracy is distributed in a more uniform way across the three QS subsets. Only for short QS, there is a slight accuracy decrease for inner QS (this effect is more pronounced for the NED% measure). With increasing QS length, the EPA placements become almost equally accurate for the three QS subsets. It is worth noting that, on average, the EPA correctly places almost all QS from all three subsets, when they contain less than 50% gaps. The results suggest that there is a steady increase in accuracy for increased QS lengths up to the 'perfect' placement in our tests. This is particularly promising because read lengths will most



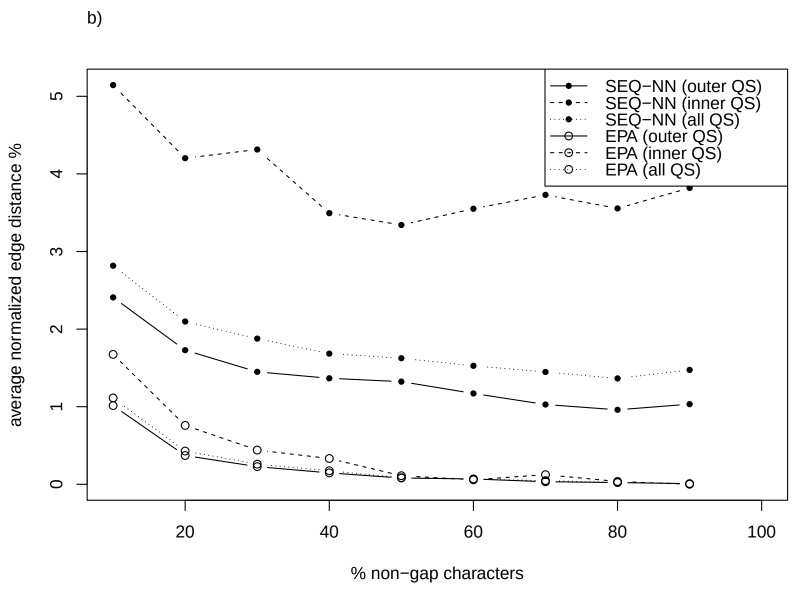


Figure 6.3: Placement accuracy for QS with artificially introduced random gaps. (a) Average node distance (ND) and (b) normalized edge distance (NED%) between insertion positions and real positions.

probably further increase.

The EPA placements on the inner QS compared to the outer QS are especially encouraging because this subset represents a worst-case scenario with respect to taxon sampling in the RT. In contrast to SEQ-NN (i.e., inner-/outer-QS), the original QS position has negligible impact on the placement accuracy. The results on the inner-QS subset are indicative for the performance on data sets with a sparse or inadequate taxon sampling. Since it is hard to determine an adequate taxon sampling *a priori* for an unknown microbial community, our approach can therefore also be used to appropriately adapt the taxon sampling.

### 6.3.4 Placement Accuracy for randomly selected Sub Sequences

Table 6.2a provides the placement accuracy (according to the ND measure) for the uniformly sampled contiguous subsequences of normally distributed lengths (mean: 200±60bp and 70±20 amino acid residues). The table values represent distances between the placement positions and the true positions from which the QS have been pruned. A ND of 0 represents a perfect placement, larger values indicate larger placement errors. As in Section 6.3.3, we provide separate results for outer QS, inner QS and all QS. In the second column, we show the placement accuracy of the EPA in terms of ND. For data set D140, this means that, averaged over all QS, the placements calculated by the EPA are within 0.51 nodes of the original placements. In the next column, we provide the average ND of the EPA for the case when the QS have been re-aligned to the RA using HMMER, prior to the placement run. For D140, the average ND is 0.59, so the additional re-alignment step results in EPA placements that are on average 1.16 times further away from their original position than EPA placements without QS re-alignment. The third column gives the average ND for a BLAST-based approach. For D140, the value of 0.91 corresponds to placements that are on average 1.78 and 1.53 times further away from the true position than the EPA-based and EPA-RA-based placements. Table 6.2b contains the corresponding results under the NED% measure. In general, the results are comparable to the ND results.

The values for D140 reflect the general trends which can be observed on most data sets. The EPA placements are on average more accurate than the BLAST placements, by factors ranging between 1.12 and 2.06. Except for the two smallest (in terms of number of taxa) nucleotide data sets, the advantage of the EPA over alternative methods clearly improves for the inner QS. For the smallest data sets the number of inner QS (see Table 6.1, remember that

Table 6.2: Accuracy on randomly sampled short sub sequences in terms of ND (Table a) and NED% (Table b) from the original position. The second column (EPA) shows the average ND of the EPA placements (using *slow* insertions under the $GTR + \Gamma$ model) for the data sets in question. The third column (EPA-RA) shows the average ND for the EPA with previous re-alignment using HMMER. The last column (BLAST) shows the average ND for a BLAST-based approach.

(a)

| | Data | EPA | EPA-RA | BLAST |
|---|---|---|---|---|
| outer QS | D140 | 0.49 | 0.58 | 0.82 |
| | D150 | 1.14 | 1.2 | 1.96 |
| | D218 | 1.7 | 2.01 | 3.66 |
| | D500 | 1.31 | 1.37 | 2.36 |
| | D628 | 2.44 | 2.95 | 2.69 |
| | D714 | 1.71 | 1.82 | 2.36 |
| | D855 | 2.87 | 2.97 | 3.53 |
| | D1604 | 2.26 | 2.45 | 2.87 |
| inner QS | D140 | 0.74 | 0.71 | 1.81 |
| | D150 | 3.09 | 3.1 | 4.62 |
| | D218 | 2.24 | 2.66 | 3.81 |
| | D500 | 2.05 | 2.37 | 4.16 |
| | D628 | 3.28 | 3.65 | 4.04 |
| | D714 | 1.78 | 1.93 | 3.51 |
| | D855 | 3.68 | 3.74 | 4.91 |
| | D1604 | 2.23 | 2.38 | 3.86 |

(b)

| | Data | EPA | EPA-RA | BLAST |
|---|---|---|---|---|
| outer QS | D140 | 1.88 | 2.22 | 2.97 |
| | D150 | 0.61 | 0.67 | 1.05 |
| | D218 | 3.82 | 4.42 | 7.2 |
| | D500 | 2.33 | 2.63 | 4.12 |
| | D628 | 2.2 | 3.06 | 2.4 |
| | D714 | 2.28 | 2.43 | 3.04 |
| | D855 | 1.71 | 1.8 | 2.09 |
| | D1604 | 0.93 | 1.06 | 1.25 |
| inner QS | D140 | 3.29 | 2.96 | 8.47 |
| | D150 | 2.66 | 2.71 | 6.83 |
| | D218 | 4.85 | 5.53 | 7.69 |
| | D500 | 3.12 | 4.14 | 7.06 |
| | D628 | 2.08 | 2.65 | 2.52 |
| | D714 | 2.82 | 3.1 | 5.75 |
| | D855 | 2.47 | 2.52 | 3.31 |
| | D1604 | 2.06 | 2.24 | 3.73 |

we only select well-supported tips as CQS) is very small; thus, this variation may be attributed to random effects. For the outer QS, the advantage over BLAST is less pronounced. For two of the larger data sets (D1604 and D755), the absolute accuracy of the EPA is approximately equal for outer QS and inner QS, while there is a larger accuracy decrease for BLAST. The re-alignment using HMMER (see Table 6.2 columns EPA and EPA-RA) before placement by the EPA has only a small negative impact on placement accuracy. The re-alignment step decreases the accuracy of the EPA with respect to the two distance measures by factor 1.03–1.2. For one data set only (D628), the combination of EPA and HMMER was found to be slightly less accurate than placement with BLAST on outer QS. We conclude that profile-HMMs as implemented in HMMER offer a useful method for adding short reads to a RA in this scenario, while there is still room to improve the QS alignment on certain data sets. In Table 6.3 we show the placement precision of the EPA and BLAST, in terms of percentage of QS placed within a certain ND of their true position. Up to a ND threshold of 10 nodes, the EPA with HMMER re-alignment (column EPA-RA) outperforms BLAST by 2.2–8%. As before, the EPA without re-alignment has slightly higher accuracy compared to EPA-RA. For the inner QS subset, the difference between BLAST and EPA-RA is larger, reaching 19.7% for a ND threshold of 1 (by definition, BLAST cannot correctly place inner QS, thus the ND will be at least 1 for inner QS). Brady et al. [21] evaluated Phymm/PhymmBL in a similar experimental setup, by measuring the classification accuracy of Phymm, PhymmBL, and BLAST at different taxonomic levels. PhymmBL achieved an accuracy improvement of approximately 6% over BLAST on simulated QS.

As previously noted here, there can be position-specific effects that can influence the placement accuracy depending on which part of the gene is used to generate QS. To minimize the impact of this position-specific bias, we sampled multiple QS (at different gene regions) from every input sequence and report averages in this study. This broad sample of QS along the gene can also be used to plot the site-specific, mean placement accuracy over the length of the RA. Accuracy plots for data sets D1604 and D855 are shown in Figure 6.4 . For every alignment column the graph shows the mean placement accuracy (ND) over all QS fragments that covered the column. Note that the columns at both ends of the alignment were covered by less fragments than the columns in the middle (due to the uniform sampling of sub sequences from the original QS). Therefore the graphs contain a high amount of noise near the extreme ends of the alignment. Similarly there is low coverage and high noise in regions that contain a high amount of gaps (see columns 300 to 400 in D1604). Both graphs show that the EPA has higher placement

Table 6.3: Percentage of QS placed correctly within a certain node distance (max(ND)) of the original position over all data sets.

| | max(ND) | EPA | EPA-RA | BLAST |
|---|---|---|---|---|
| outer QS | 0 | 58.4% | 56.4% | 56.3% |
| | 1 | 71.1% | 69.5% | 64.1% |
| | 2 | 77.6% | 76.0% | 69.5% |
| | 5 | 87.9% | 86.8% | 82.6% |
| | 10 | 94.6% | 93.9% | 91.9% |
| inner QS | 0 | 37.0% | 35.1% | 0.0% |
| | 1 | 64.3% | 63.4% | 43.7% |
| | 2 | 74.7% | 73.1% | 56.9% |
| | 5 | 86.5% | 85.8% | 77.9% |
| | 10 | 94.1% | 93.6% | 90.5% |

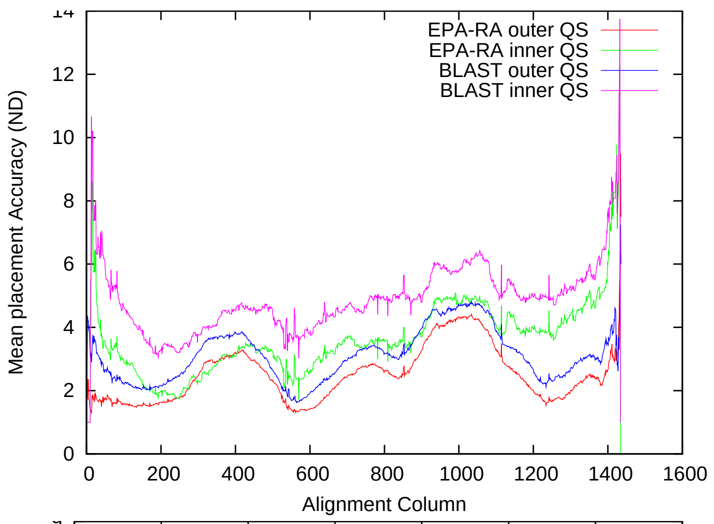
accuracy than SEQ-NN over the whole length of the alignments. Also the EPA shows less decrease in accuracy on the harder sub-set of inner QS. This approach can also be used, a priori, on a full length sequence alignment to determine appropriate gene regions for short-read generation.

### 6.3.5 Placement Accuracy for paired-end Reads

Table 6.4a provides the results of the experiments with virtual paired-end reads of length 2x100bp (the results for 2x50bp reads are provided in Table 6.4b). Similar to Section 6.3.4 the placement accuracy is given in terms of ND and NED%, averaged over all QS. For D150, the EPA places the QS within 1.26 nodes of their true position while for BLAST, the average ND of 3.67 corresponds to placements that are 2.91 times further away from the original position, relative to the EPA. The Table also contains corresponding values for the NED% measure, which yields similar results as the ND measure. As in Section 6.3.4, the general trend is similar for all data sets. The accuracy of EPA placements are on average 1.58–5.87 times better than for BLAST.

Figure 6.5 provides histograms for the ND values of individual placements computed by the EPA and BLAST for 2x100bp paired-end reads on data set D855. Respective histograms for all data sets on 2x100bp and 2x50bp reads are available in the supplementary material of [12]. The plots suggest that the placement error for both methods approximately follows a power law distribution. The placements obtained via the EPA are, on average, closer to the true position and yield smaller maximum placement errors than BLAST.

Table 6.4 highlights that, the EPA consistently outperforms BLAST-



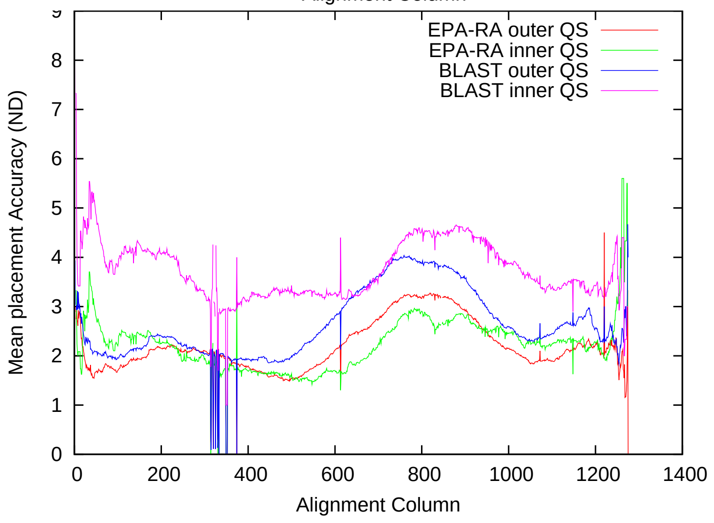


Figure 6.4: Accuracy profiles for D855 (upper) and D1604 (lower). The plots are derived from the evaluation results of the contiguous sub sequence placement experiments.

Table 6.4: Accuracy of the placement of 2x100 bp (Table a) and 2x50bp (Table b) paired-end reads. The values given are the node-distance (ND) and the normalized edge distance (NED %). The methods used are the Evolutionary Placement Algorithm (EPA) (*slow* insertions under the $GTR+\Gamma$ model) and BLAST-based nearest neighbor.

(a)

| | | ND | | NED % | |
|---|---|---|---|---|---|
| | Data | EPA | BLAST | EPA | BLAST |
| outer QS | D150 | 1.14 | 3.38 | 0.63 | 1.76 |
| | D218 | 1.44 | 4.77 | 3.6 | 8.45 |
| | D500 | 0.84 | 4.88 | 1.75 | 7.35 |
| | D628 | 0.56 | 1.83 | 0.83 | 1.83 |
| | D714 | 1.31 | 3.59 | 1.77 | 4.78 |
| | D855 | 2.0 | 5.98 | 1.11 | 3.35 |
| | D1604 | 1.58 | 3.67 | 0.79 | 1.65 |
| inner QS | D150 | 1.9 | 5.3 | 2.23 | 5.64 |
| | D218 | 4.43 | 5.21 | 8.16 | 9.38 |
| | D500 | 1.59 | 9.41 | 3.15 | 12.24 |
| | D628 | 0.85 | 2.95 | 0.55 | 1.13 |
| | D714 | 1.66 | 4.9 | 2.9 | 7.76 |
| | D855 | 2.9 | 7.54 | 2.04 | 4.61 |
| | D1604 | 1.57 | 5.3 | 1.69 | 4.84 |

(b)

| | | ND | | NED % | |
|---|---|---|---|---|---|
| | Data | EPA | BLAST | EPA | BLAST |
| outer QS | D150 | 3.84 | 6.21 | 1.9 | 4.65 |
| | D218 | 2.7 | 6.35 | 5.44 | 13.37 |
| | D500 | 3.72 | 12.23 | 6.09 | 19.22 |
| | D628 | 1.26 | 3.46 | 1.43 | 3.63 |
| | D714 | 2.01 | 4.1 | 2.91 | 5.7 |
| | D855 | 3.82 | 9.33 | 2.55 | 7.16 |
| | D1604 | 2.53 | 5.73 | 1.21 | 3.62 |
| inner QS | D150 | 2.1 | 9.2 | 5.37 | 11.08 |
| | D218 | 4.0 | 9.29 | 8.4 | 17.72 |
| | D500 | 3.86 | 11.03 | 7.33 | 15.66 |
| | D628 | 1.8 | 4.55 | 1.5 | 1.9 |
| | D714 | 2.61 | 4.48 | 4.32 | 6.93 |
| | D855 | 3.79 | 10.69 | 2.79 | 9.36 |
| | D1604 | 2.27 | 6.13 | 1.9 | 5.76 |

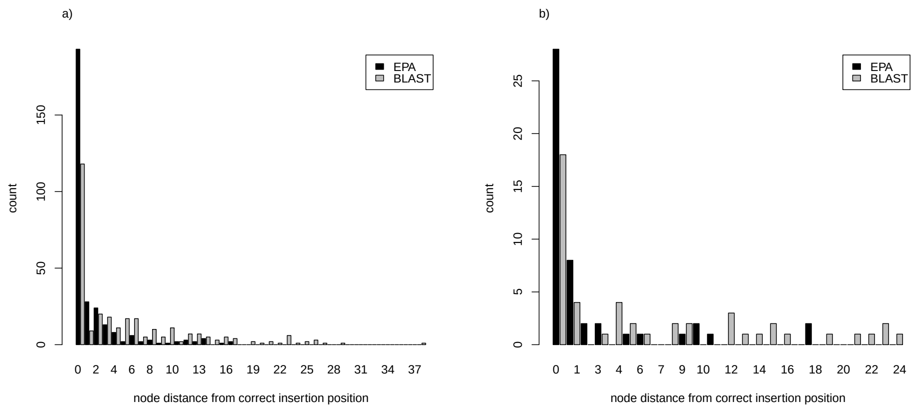


Figure 6.5: Histogram showing the placement accuracy, based on node distances, for the placement of 2x100 bp paired-end reads from D855, using outer (a) and inner (b) QS.

based placements for paired-end reads and that placements are approximately twice as accurate on average. Generally, the placement accuracy for paired-end reads of length 2x100bp is better than was expected from the test with randomly selected contiguous sub sequences of mean length 200bp Section 6.3.4. One contributing factor is that, many of the sub sequences in the previous experiment were significantly shorter than 200bp, because we use normally distributed lengths. There also appears to exist a positive effect, generated by selecting sub sequences from two distinct regions of the gene (the start and the end) from the original QS; in combination, phylogenetic information from both ends, may contain a stronger phylogenetic signal than a single, randomly selected sub sequence.

### 6.3.6 Impact of Placement Algorithms and Substitution Models on Accuracy

All preceding computational experiments were carried out using the most thorough (*slow*) version of the EPA under the GTR+$\Gamma$ and WAG+$\Gamma$ (AA) models. In this mode, the EPA optimizes edge lengths via the Newton-Rhapson method for every possible insertion edge on the RT. As mentioned in Section 3.2, we also devised a *fast* version of the EPA where this optimization is deactivated for QS insertions. These heuristics can speed up the EPA by one order of magnitude, when a large amount of QS is being placed onto a RT. An additional speedup by a factor of 3 to 4 can be achieved by using the CAT approximation of rate heterogeneity [78].

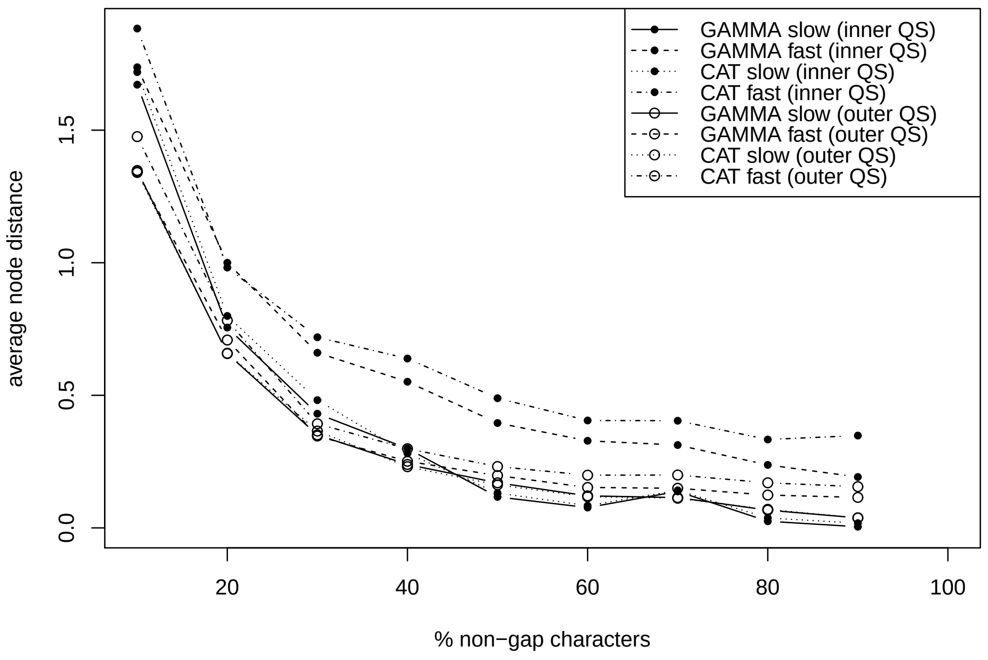


Figure 6.6: Average node distance for different versions of the EPA (*fast*/*slow* insertions) algorithm and model types (GTR+Γ, GTR+CAT) on inner QS and outer QS from all data sets.

Figure 6.6 shows the impact of EPA heuristics and rate heterogeneity models on placement accuracy for all QS over all data sets (analogous plots for the individual data sets are available in the supplementary material of [12]). For the *slow* insertion method, there is practically no difference in placement accuracy between the Γ model and the CAT approximation. For the *fast* insertion method, there is a notable decrease in placement accuracy for the CAT as well as the Γ models. The decrease is less pronounced for the outer QS while for the inner QS the effect of using the *fast* insertion method is more pronounced. As already mentioned, correct placement of the inner QS is harder than placement of the outer QS, because outer QS have direct neighbors in the RT. The results of this experiment show that, the *slow* version of the EPA which includes edge length optimization can produce better placement results than the *fast* version, especially when QS are placed on inner edges of the RT.

### 6.3.7 Heuristics for Slow Insertions

As shown in Section 6.3.6, the loss of accuracy induced by the *fast* insertion method is minimal. Nonetheless, a slight accuracy improvement can be attained by using the *slow* insertion method, especially regarding the more precise edge length estimate at the insertion position that can be used for post-analysis purposes. Using the rapid insertion edge pre-scoring heuristics already described, it is possible to accelerate the *slow* insertion algorithm with little to no impact on placement accuracy. Here, we evaluate the accuracy trade-offs associated with these heuristics. We also provide run-time measurements for the EPA and BLAST.

In contrast to the previous accuracy assessments, we do not test the placement of one QS at a time onto an existing RT from which the QS has been previously pruned. Instead, we randomly split the alignments into two subsets that each comprise 50% of the taxa. The first subset is used to infer a best-known ML tree with RAxML onto which the remaining taxa (of the second subset) are placed via the EPA. Here, we assume the *slow* EPA placements to be the true placements. In this experiment, we reduce the length of the QS to 50% non-gap characters. The non-gap characters are a contiguous sequence fragment that starts at the beginning of the respective sequence; that is, the QS represent roughly the first half of the gene.

Figure 6.7 shows the accuracy on the largest data set D1604 (placement of 802 QS onto a RT with 802 RS). The fraction of insertion edges considered for the slow insertion phase is controlled by the parameter $fh$. In the plot the accuracy of the heuristics for values of $fh = \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}, \frac{1}{64}, \frac{1}{128}, \frac{1}{256}$ is shown (i.e., on this data set approx. 400, 200, 100, 50, 25, 12, 6 out of 1601 possible insertion edges are considered). For the lowest $fh$ values, there is a notable decrease in placement accuracy (sharp rise of the curves on the left side of the plot). For 50 or more tested insertion edges, the accuracy remains virtually constant. The results suggest that, on this data set, it is sufficient to more thoroughly analyze only 50 out of 1601 ($fh = \frac{1}{32}$) candidate insertion edges proposed by the heuristics to obtain the best possible accuracy (even for $fh = \frac{1}{64}$ there is only a very small deviation from the reference positions). Another important result is that the MP heuristics produce equally accurate placements as the ML heuristics, for all, except the smallest values of $fh$. We conclude that, the MP heuristics with a parameter setting of $fh = 1/32$ (using the $\Gamma$ model for *thorough* insertions) are sufficient for achieving placement accuracy comparable to the reference placement, but with computational requirements (290 seconds) that are in the same order of magnitude as a simple BLAST search (216 seconds) of the QS against the sequences in the RS. Re-alignment of the QS to the RA takes 224 seconds

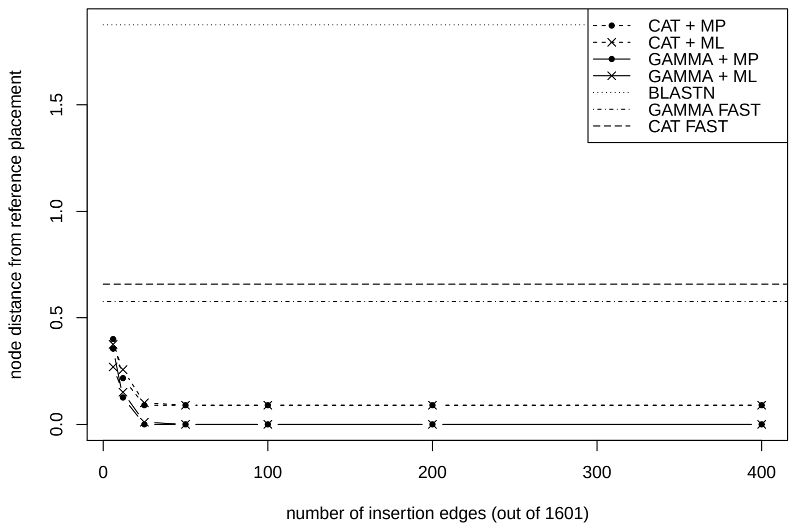


Figure 6.7: Accuracy of the EPA as a function of the number of edges considered for *slow* insertion after heuristic filtering.

using HMMER on this data set. Thus, the combined run time of HMMER and the EPA is approximately 2.5 times higher than a simple BLAST search, but still within the same order of magnitude.

The lowest run time (113 seconds) was achieved by using the CAT model for *slow* insertions, at the expense of a slight loss in accuracy (i.e., on average the ND increased by approximately 0.1). Based on the results in Section 6.3.6, we expect the accuracy difference between the CAT approximation and the $\Gamma$ model of rate heterogeneity to be negligible in a real-world scenario.

The differences in accuracy between the *fast* and *slow* insertion methods as well as between the $\Gamma$ and CAT models are generally larger than in Section 6.3.6. This is not surprising, given that this experiment was not designed to measure the insertion accuracy relative to an assumed correct position, but the deviation between our best, yet slowest, method and less accurate, accelerated methods. Here, we do not constrain the experiment to QS with high support values in the RT, but chose QS at random, which may introduce a certain loss of precision. In addition, the RT (comprising 50% of the taxa in the original RA) is smaller than in the previous evaluations and thus more sparsely sampled. Nonetheless, the deviation between the *fast* and *slow* EPA versions amounts to less than half a node on average and the general finding that *slow* insertions under CAT are more accurate than *fast* insertions under

$\Gamma$ is consistent with previous experiments.

### 6.3.8 Experimental Setup and Benchmark for the parallel EPA version

We used three real-world 16S rRNA data sets with 4,412 [33] (HAND data set), 16,307 [91] (GUT data set), and 100,627 (unpublished) QS that are classified into a single-gene bacterial RT with 4,874 species and a reference alignment length of 1,287 base pairs. Those data sets represent typical current use cases for our algorithm and the two smaller ones are freely available for download together with the multi-grain parallelization of the source-code as part of standard RAxML. As test systems for assessing scalability we used a 4-way quad-core AMD Barcelona system with a total of 16 cores running at 2.2 GHz equipped with 128 GB of main memory, and a Sun x4600 8-way quad-core AMD Shanghai system with a total of 32 cores running at 2.7 GHz, equipped with 64 GB of main memory. The evaluation compares the Parallel Multi-Grain (PMG) described in Section 3.3 to the Parallel Fine-Grain (PFG) version of the EPA. The PFG version uses the fine-grain parallelization scheme in both, the *initial phase*, as well as the *insertion phase*.

Computational experiments were conducted as follows: On the AMD Barcelona system we executed one sequential run and PFG as well as PMG runs on 2, 4, 8, and 16 cores for the *fast* insertion method. On the x4600 we executed the same runs, but up to 32 cores. We also executed two runs using the *slow* insertion method with 16 threads for the PFG and PMG parallelization on the 16-core system. The execution time for the slow method using the PFG approach on the HAND data set was 35 hours compared to *only* 22 hours using the PMG implementation. Thus, for the slower (and more accurate) insertion algorithm on the HAND data set we achieve a parallel efficiency improvement exceeding 35%. The execution times for the slow method on the larger GUT data set where 139 hours (PFG) and 76 hours (PMG) respectively, which corresponds to a parallel run time improvement of 45%.

In Figure 6.8 we depict the speedups for the fast insertion algorithm using the PFG and PMG parallelizations on the Barcelona system for data sets HAND and GUT. Figure 6.8 shows that the PFG version scales reasonably well on the 16-core system and even achieves super-linear speedups due to improved cache efficiency up to 8 threads. However, as outlined in Section 3.3, the scalability of the fine-grain approach is limited for single-gene alignments. The impact of an increasing amount of synchronization events per computation in the PFG parallelization is underlined by the sub-linear

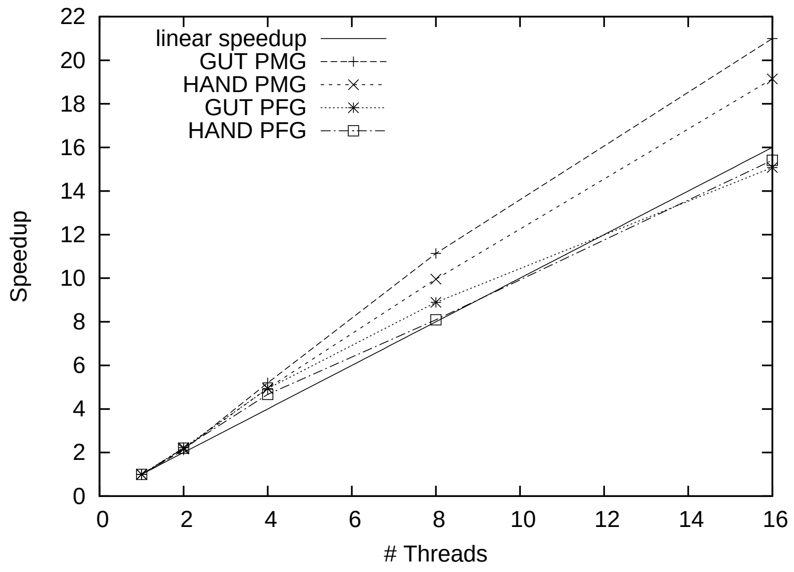


Figure 6.8: Speedups for overall execution times on a 16-core AMD Barcelona system using the fast insertion algorithm.

speedups on 16 cores. The increase in parallel efficiency of the PMG over the PFG approach on 16 cores for the fast insertion method is slightly lower than for the slow insertion method, but we still achieve a 25% improvement on 16 cores. This is because the contribution of the less scalable fine-grain initial model optimization phase to overall execution time is larger for the fast than for the slow insertion method. Nonetheless, speedups are significantly super-linear up to 16 cores for the PMG version. The overall speedups are slightly lower on the 32-core system (Figure 6.9), because of the less scalable fine-grain initial model optimization phase; stand-alone placement without model optimization scales linearly. Nonetheless, the increase in parallel efficiency of the PMG over the PFG approach with 32 cores exceeds 50%.

Figures 6.8 and 6.9 show that the parallel efficiency of the EPA is generally higher on the AMD Barcelona system. The reason for this is that the improved cache efficiency of the parallel algorithm has a larger impact on the Barcelona cores that have 2MB L3 caches compared to 6MB L3 caches on the Shanghai cores. In Figure 6.10 we plot the speedup of the Shanghai over the Barcelona CPUs as a function of the number of threads. When only one Shanghai core is used, the program is almost twice as fast as on a Barcelona core which can not be explained by the higher clock rate and improved micro-architecture alone. The main cause is the three times larger L3 cache size. If the number of threads is increased and cache misses are

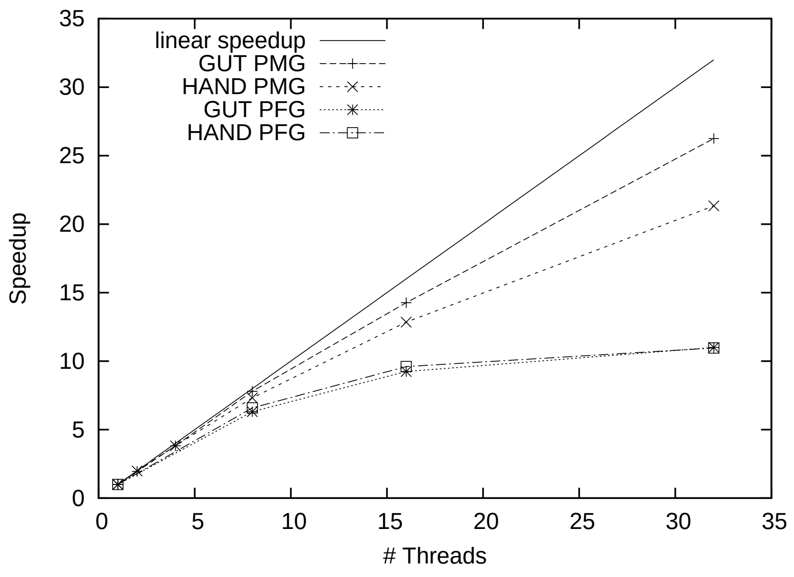


Figure 6.9: Speedups for overall execution times on a 32-core x4600 system using the fast insertion algorithm.

thereby decreased, the speedup converges to 1.23 which corresponds to the clock rate ratio $(2.7GHz/2.2GHz)$.

Finally, we also conducted experiments on a challenging data set with 100,627 QS to explore the limits of our approach on the x4600 system. We obtained overall speedups of 14.5 and 27.3 for fast insertion runs with 16 and 32 cores respectively. On 32 cores the overall runtime for the placement of 100,627 QS is less than 1.5 hours which makes our approach a viable and more accurate alternative to BLAST and other sequence comparison-based approaches that do not take into account the evolutionary history of the sequences under study.

## 6.4 Fossil placement evaluation

This section presents the experimental evaluation of the EPA for non-molecular sequence data. This includes testing the morphological weight calibration algorithm on real and simulated data. It also includes two case studies on real fossil data.

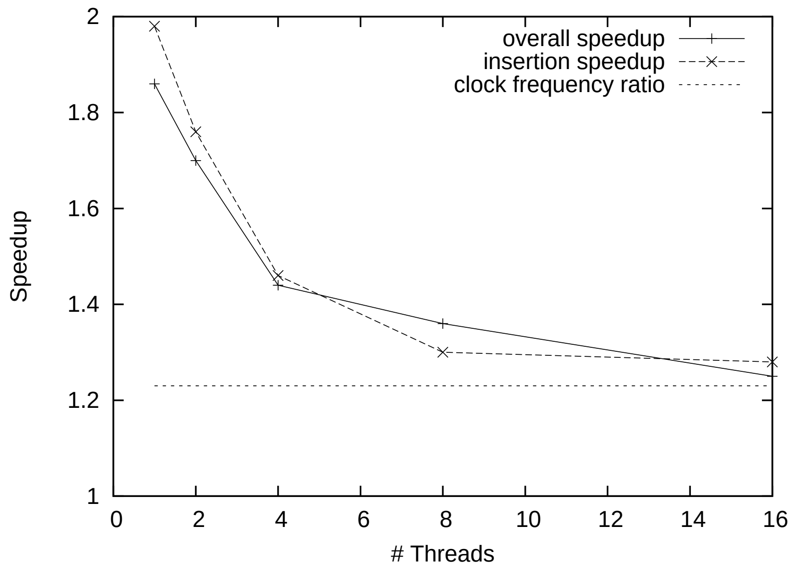


Figure 6.10: Speedup of the 32-core (Shanghai) over the 16-core (Barcelona) architecture.

| name | # taxa | # mol sites | # morph sites |
|---|---|---|---|
| D1 | 35 | 2,006 | 117 |
| D2 | 23 | 16,662 | 414 |
| D3 | 32 | 1,713 | 381 |
| D4 | 81 | 3,675 | 213 |
| D5 | 18 | 266 | 35 |

Table 6.5: Overview of test data sets.

### 6.4.1 Real-World Test Datasets

We used 5 real-world test data sets that contain morphological as well as molecular DNA data. The data sets are labelled as D1 through D5 for ease of reference. Table 6.5 provides the number of taxa and number of molecular as well as morphological sites for all input data sets we used. The real-world data sets can be downloaded at `http://www.exelixis-lab.org/morphologyDatasets.tar.bz2`.

Dataset D1 [62] contains 35 taxa of walnut trees (*Juglandaceae*). The original alignment also contained an additional 5 fossils. D2 [10] comprises 23 Marsupial sequences (the original data set also contained 10 fossils). D3 [96] contains 32 taxa of Amphibians (*Caudates*). D4 [94] contains 81 taxa of tree-frogs (*Hylidae*). Finally, D5 [87] contains 18 taxa that span a wider variety of species than the other data sets, ranging from the chicken to the homo sapiens.

### 6.4.2 Datasets and Methods

An initial literature search revealed that at the time the experiments were conducted, no freely available programs for generating simulated morphological data sets were available. Therefore, we contacted J.J. Wiens, who kindly made available to us the C code for generating simulated data sets that was used in [93]. We completely re-implemented and extended the original C program in Java (GPL source code available at `https://github.com/sim82/java_tools/blob/master/src/ml/EvoSimFossil.java`). The program can now read in two distinct trees, for instance, one that is congruent to a reference topology and a random tree that is incongruent to the reference topology. This allows for generating simulated morphological data sets that entail two partitions with conflicting phylogenetic signal.

In addition, the simulation program can generate morphological partitions of variable length, for example, a partition of 300 sites that are incongruent to the RT and a partition with 100 sites that are congruent to the RT. Moreover, the simulation program allows for generation of an artificial fossil sequence, that is located at the innermost edge (the most distant position from current-day species) of the tree on which the data is being generated. While a fossil in general must not necessarily be located at the innermost edge of a tree (see [62]), this setup ensures that the placement problem as such is more difficult, since the closest current-day relatives of the fossil are located as far away as possible. In our simulated data generation tool, the artificial fossil is thus automatically placed onto the edge that has the longest edge-based path length to the nearest tips.

In order to assess incongruence between trees obtained from morphological and molecular data partitions in Section 6.4.3 we need to compute the topological distances between trees. The standard Robinson-Foulds (RF) [71] distance between two trees is defined as the number of non-trivial bipartitions (splits into taxon label sets induced by the inner edges of a tree) that are contained in one of the two trees but not in both. The RF distance is typically reported as relative distance, that is, the count of distinct bipartitions divided by $2(n-3)$ where $n$ is the number of organisms and $n-3$ the number of inner edges (edges not leading to tips). The number $2(n-3)$ hence represents the worst case for RF, that is, the two trees under comparison do not share any bipartitions.

In addition to the RF distance, one can also define the Weighted RF (WRF) distance that takes into account the bootstrap support values on the edges. If there are incongruent bipartitions in the tree that have low support, for example, 10%, they will contribute a total of 0.2 to the WRF distance, while they would contribute 2 to the RF score. Therefore, the WRF distance provides a better notion of whether trees disagree in strongly (important) or weakly (unimportant) supported bipartitions. The WRF distance also better resembles the way in which Biologists usually assess there results. In our experiments we used the respective RF and WRF options as implemented in RAxML.

### 6.4.3 Incongruence of Morphological and Molecular Data

Initially, we assessed the (in)congruence between the morphological and the molecular data partitions in our real world data sets to substantiate our claim that morphological and molecular partitions typically exhibit incongruent signal.

For this, we split up each real-world data set into the morphological and molecular partitions and conducted thorough ML analyses as follows: For the morphological and the molecular data sets we seperately conducted 100 bootstrap analyses and 50 ML searches for the best-scoring ML tree under the $\Gamma$ model of rate heterogeneity [98] using RAxML.

We then used the corresponding RAxML option to draw bootstrap support values on the respective best-scoring tree out of the 50 ML trees we computed. The RF and WRF distances between the respective best-scoring morphological and molecular trees with bootstrap support values were then computed in order to determine incongruence between the data partitions (see Table 6.6).

The values provided in Table 6.6 clearly show that the molecular and morphological trees are highly incongruent based on the RF and WRF distances. RF distances exceed 50% and WRF distances oscillate around 40% which means that several highly supported bipartitions of the molecular tree are not recovered by the morphological tree.

In order to assess the stand-alone topological stability of the morphological data partitions we conducted an additional 100 ML searches per data set (on the morphological partitions only). We then computed the maximum RF distance and the mean RF distances within those ML tree sets based on all pairwise RF distances between the resulting 100 ML trees (in this case we do not include WRF distances, as the 100 ML trees have been calculated without bootstrap support). The average and maximum distances in those tree sets as shown in Table 6.7 provide a good notion for the general topological instability of the morphological partitions. In most cases the mean RF distance largely exceeds 10% and the maximum RF is larger than 50% in most

| data set | RF(morph,mol) | WRF(morph,mol) |
|---|---|---|
| D1 | 59% | 39% |
| D2 | 60% | 37% |
| D3 | 62% | 47% |
| D4 | 82% | 45% |
| D5 | 80% | 42% |

Table 6.6: Incongruence between morphological and molecular trees & average BS support induced by morphological and molecular partitions.

| data set | max RF | mean RF | data set | max RF | mean RF |
|---|---|---|---|---|---|
| D1 | 68.75% | 21.70% | D2 | 25.00% | 6.27% |
| D3 | 58.62% | 25.56% | D4 | 64.10% | 32.59% |
| D5 | 73.33% | 33.05% | | | |

Table 6.7: Pairwise mean and maximum RF values for sets of 100 ML trees.

cases (i.e., ML trees for the same data set only share 25% of their non-trivial bipartitions). Given that the data sets are relatively small with respect to the number of taxa and that the RAxML search algorithm has been shown to be very efficient in recovering the best-known tree [79] we conclude that there is a *significant* lack of signal with respect to tree reconstruction in the real-world morphological data sets under study.

This initial set of experiments underlines two major claims: *Firstly*, that morphological data partitions can yield significantly different trees compared to molecular data partitions and *secondly*, that morphological data partitions can suffer from a significant lack of or a weak phylogenetic signal and it is therefore difficult to use them for de novo phylogenetic inference. Based on this observation we focus on assessing the usage of morphological data partitions for the placement of fossils in the following computational experiments.

### 6.4.4 Morphological Weight Calibration

Initially we assessed if our statistical method for determining congruent and incongruent sites works on simulated data sets. For this purpose we generated a simulated data set based upon the real molecular tree for data set D3 and generated one incongruent partition with 200 morphological sites and one partition congruent to the molecular tree that also comprised 200 morphological sites. We then executed our algorithm on this data set and plotted the inferred weights over the number of sites in the simulated alignment. As

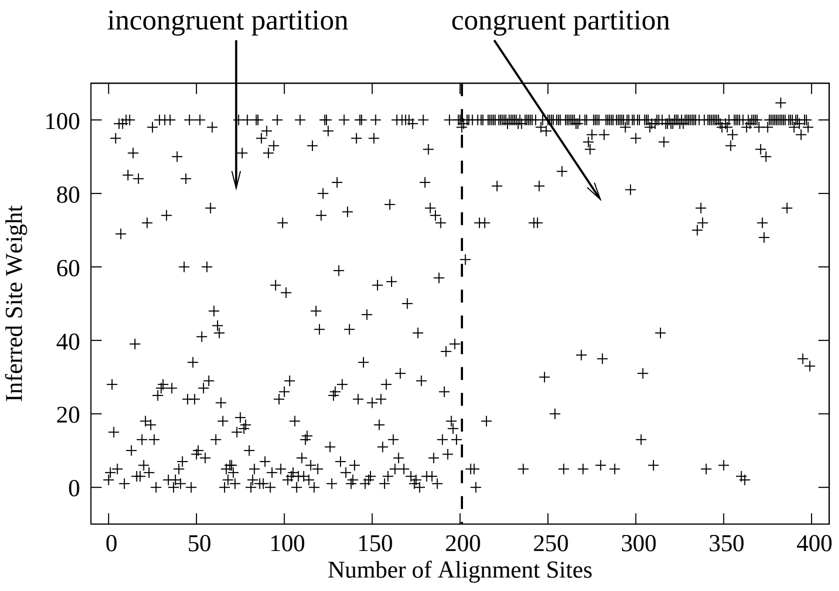


Figure 6.11: Weight assignments for congruent and incongruent (with respect to a RT) data partitions of a simulated morphological data set.

Figure 6.11 clearly shows, we are able to distinguish between incongruent (first half of the alignment, low values) and congruent (second part of the alignment, high weight values) sites.

Results for larger data sets in terms of the number of organisms included and distinct input trees are analogous (results not shown).

### Fossil Placement Accuracy on Simulated Data Sets

To test the weight calibration algorithm on simulated data sets, we generated simulated data sets as follows: Based upon each of the 5 real molecular trees we generated 100 simulated data sets per real tree, by using different random seeds for every simulated alignment and a different random tree for sets of 10 simulated alignments in order to generate morphological sites that are incongruent to the molecular RTs. For each set of those 100 simulation runs per RT we also generated morphological alignments of variable length, that is, data sets containing 100 congruent as well as 100, 200, 300, 400, and 500 incongruent sites derived from the random tree. Thus, for each real input tree we generated a total of 500 simulated alignments. The rationale for this setup is to test, to which extent the degree of random noise in the alignment affects placement accuracy. For ease of reference we denote these simulated data sets as SX_YYY_ZZZ where SX denotes the molecular tree from data sets D1-D5 that was used to generate the congruent data partition, YYY the number of congruent sites, and ZZZ the number of incongruent sites.

For each simulated data set size, we placed the fossil (generated as described in Section 6.4.2) into the RT using our evolutionary placement algorithm (see Section 4.2) with Bootstrapping in order to avoid any random effects that may be caused by placement runs without Bootstrapping. We executed those placement runs for the unweighted case (all morphological sites included, without weight calibration; denoted as **UNW**), the case with integer weights (using the calibrated weights from $\vec{W}$; denoted as **INT**), using only the incongruent data partition (SX_ZZZ; denoted as **BAD**), and only the congruent data partition (SX_YYY; denoted as **GOOD**). The accuracy was then measured using Equation 6.1 to compute the node-based absolute placement accuracy. Relative edge-based accuracy results were analogous (data not shown). Fossil placement accuracy was averaged over the 100 simulated data sets for every data set size.

When the EPA is used with bootstrapping, more than one potential insertion edge can be proposed for a fossil (see Figure 6.12), which means that we need to appropriately adapt our distance measures to incorporate Bootstrap support values. For a bootstrap run with $N_{bs}$ bootstrap replicates, the output of the algorithm contains a set of $i = 1...N$, where $N \leq N_{bs}$, insertion positions for the fossil with bootstrap values $S_i$. Using this information we derive a set of ND or NED% distances $D_i$ to the correct edge for each alternative Bootstrap placement $i$. We use the $D_i$ to represent the bootstrap placement information as a single quantity for the fossil placement accuracy by defining the Weighted Root Mean Squared Distance (WRMSD), $D_{wrms}$ as follows:

$$D_{wrms} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\frac{S_i}{N_{bs}}D_i)^2} \tag{6.1}$$

The results of the simulated data set experiments are provided in Table 6.8. Except for data sets S4, the approach with calibrated site weights using integer values, clearly improves placement accuracy by 25% to over 50%. In some cases (data sets S1 and S3) it even outperforms the fossil placement accuracy achieved by exclusively using the congruent data partition. Overall the weight calibration approach improves the average placement accuracy on all data sets (including data set S4) from 4.72 to 3.84 (i.e., it achieves an accuracy improvement of 20%). Overall, there is a clear tendency for placement accuracy to decrease with an increased amount of incongruent sites. The bad overall performance on data set S4—note that the congruent morphological partition does not achieve a significantly better placement

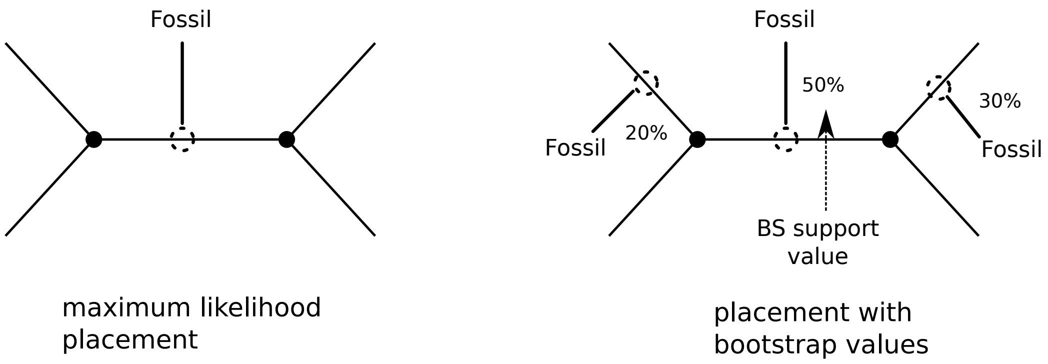


Figure 6.12: Example output of the phylogenetic fossil placement procedure without and with Bootstrap support values.

accuracy than the combined partitions—may be attributed to artefacts introduced by the simulated data generation process. In addition, the molecular tree shape of S4 is particularly difficult, because it has a large number of relatively short inner edges and long edges leading to the tips. As such, the simulated fossil that is placed at the innermost edge of the tree will be hard to place accurately, because of the short internal edges. This also explains the better performance of D4 in the experiments on real data presented in the next section, because in this case we use morphological data from extant species that are mostly attached to long edges to assess placement accuracy. In addition to this, a congruent data partition length of 100 may not be sufficient to compute an accurate placement because the data set has significantly more organisms than all other tested data sets. For 100 simulated data sets S4_400_400, that is, with 400 congruent sites, the placement accuracy of the congruent partition increased to 5.30 and that of the integer weighted placement to 8.55. However, a further increase of the congruent sites to a length of 800 did not yield further significant improvements in placement accuracy.

## Placement Accuracy on Real Datasets

The overall placement accuracy on real data sets was assessed in a different way than on simulated data. While some data sets include fossil data, unlike as for the simulated data sets, we do not know the true phylogenetic position of these fossils. To this end, we decided to base our analysis only on the current-day species for which molecular data is available. We assume that the true position of these taxa is the position in the respective molecular reference topology. In order to thoroughly test placement accuracy, we removed one organism at a time from every real world molecular tree, which we then re-inserted using only the morphological data via the placement algo-

| name | UNW | INT | BAD | GOOD |
|---|---|---|---|---|
| S1_100_100 | 1.37 | 0.00 | 5.02 | 0.66 |
| S1_100_200 | 1.82 | 0.40 | 5.04 | 0.66 |
| S1_100_300 | 2.01 | 0.44 | 5.16 | 0.66 |
| S1_100_400 | 2.82 | 1.13 | 4.82 | 0.66 |
| S1_100_500 | 2.93 | 1.11 | 5.05 | 0.66 |
| S2_100_100 | 2.50 | 1.38 | 5.25 | 0.83 |
| S2_100_200 | 3.29 | 1.64 | 5.71 | 0.83 |
| S2_100_300 | 3.95 | 2.47 | 5.34 | 0.83 |
| S2_100_400 | 3.87 | 2.57 | 5.14 | 0.83 |
| S2_100_500 | 4.24 | 3.07 | 5.62 | 0.83 |
| S3_100_100 | 1.36 | 0.44 | 7.00 | 0.95 |
| S3_100_200 | 2.08 | 0.82 | 6.48 | 0.95 |
| S3_100_300 | 2.44 | 0.85 | 6.66 | 0.95 |
| S3_100_400 | 3.01 | 1.28 | 6.26 | 0.95 |
| S3_100_500 | 3.85 | 1.46 | 6.76 | 0.95 |
| S4_100_100 | 12.02 | 14.95 | 12.26 | 12.40 |
| S4_100_200 | 11.25 | 13.93 | 11.59 | 12.40 |
| S4_100_300 | 11.47 | 12.58 | 11.80 | 12.40 |
| S4_100_400 | 12.08 | 12.36 | 11.49 | 12.40 |
| S4_100_500 | 11.22 | 11.29 | 12.10 | 12.40 |
| S5_100_100 | 2.04 | 1.32 | 5.55 | 0.92 |
| S5_100_200 | 3.74 | 2.35 | 5.37 | 0.92 |
| S5_100_300 | 3.92 | 2.74 | 5.66 | 0.92 |
| S5_100_400 | 4.63 | 3.61 | 5.65 | 0.92 |
| S5_100_500 | 4.14 | 3.28 | 5.54 | 0.92 |

Table 6.8: Absolute average node-distance based accuracy for fossil placements on simulated data sets for unweighted, integer-weighted sites as well as incongruent and congruent data partitions.

rithm. On data set D3 for instance, we conducted 32 placement runs for each of the 32 species. Once again, we used phylogenetic placement with bootstrapping and extracted the average placement accuracy using the Weighted Root Mean Squared Distance (see Equation 6.1). We conducted placement runs for 4 different weighting schemes: unweighted (denoted as **UNW**), binary weights (denoted as **BIN**), integer weights (denoted as **INT**), and all weights set to 100 (denoted as **100**). In Table 6.9 we provide the absolute accuracy in terms of average placement node distance (ND) for all analyzed weighting schemes and all real-world data sets. In Table 6.10 we provide the relative accuracy in terms of average normalized edge distance (NED%). The data presented clearly show that the approach using our weight calibration mechanism with integer weights yields the best results in terms of accuracy. An interesting observation is that the approach where a weight of 100 is assigned to every site performs better than the binary weighting scheme. This can be attributed to the application of the Bootstrap procedure and a too strict cutoff of 5% used for generating the binary weight vector. While our morphological calibration mechanism works well, assigning weights of 100 to each site assures that sites that contain congruent signal will, with high probability, be included in the bootstrap replicates, while this probability is low for binary weights or the unweighted placement that comprise all sites at most once as opposed to 100 times. As the relatively good accuracies obtained for the unweighted case indicate, ML is able to filter out noise, that is, incongruent signal, for placing fossils. However, the standard Bootstrap procedure may occasionally not include some congruent sites in the bootstrap replicates, which can bias the stability of the placement results. The probability for not sampling congruent sites is relatively large, because the morphological data partitions have comparably few sites.

Our placement algorithm using calibrated integer weights yields placements that are approximately 25% better in terms of node distance than the unweighted standard approach and they also achieve a relative average distance improvement (over all data sets) of 25%. Thus, despite the partially highly incongruent phylogenetic signal between morphological and molecular data partitions, we are able to accurately place fossils in well-established RTs. Even using the unweighted approach one can achieve better than 85% accuracy in the worst case.

### Placement Accuracy of Real Fossils: Two Case Studies

We also used morphological data for placing the real fossil taxa that were included in the original biological analyses of data set D1 [62] and D2 [10]. Those fossil taxa had previously been placed and analyzed using different

| name | UNW | BIN | INT | 100 |
|---|---|---|---|---|
| D1 | 1.26 | 0.93 | 1.05 | 1.05 |
| D2 | 0.99 | 0.86 | 0.75 | 0.78 |
| D3 | 1.32 | 1.14 | 0.75 | 0.99 |
| D4 | 3.51 | 3.02 | 2.06 | 2.34 |
| D5 | 3.13 | 3.34 | 2.20 | 2.43 |

Table 6.9: Absolute node-distance based accuracy for fossil placements on real-world data sets for alternative site weighting schemes.

| name | UNW | BIN | INT | 100 |
|---|---|---|---|---|
| D1 | 3.9% | 3.4% | 3.2% | 3.1% |
| D2 | 4.6% | 4.5% | 3.6% | 3.5% |
| D3 | 9.6% | 7.8% | 5.4% | 8.3% |
| D4 | 11.0% | 9.9% | 7.6% | 8.6% |
| D5 | 14.2% | 14.2% | 12.7% | 13.0% |

Table 6.10: Relative edge-distance based accuracy for fossil placements on real-world data sets for alternative site weighting schemes.

placement approaches in the aforementioned studies [10, 62]. While a detailed biological analysis of the placements obtained via the approach we present here is outside the scope of this work, we briefly address placements results using morphological weight calibration and discuss potential interpretations.

Figure 6.13 depicts the placements of the Juglandaceae fossils. The name labels of the fossil taxa in Figure 6.13 are preceded by the word QUERY and we have appended the bootstrap support for the insertion edge at the end of the name label. The placements of the individual fossils are partly comparable to the findings in [62].

The Polyptera, Palaeoplatycarya, and Platycarya fossils are in a clade (subtree) with Carya and Juglans which corresponds to empirical biological expectations, but are not located at the root of the subtree containing all Juglans and Carya. Also the Cruciptera fossil is placed with the Juglans, rather than being located at the root of the subtree containing all Juglans. The largest difference to the study presented by Manos et al. [62] is that the Paleooremunnea fossil is placed at the root of the subtree containing the Oreomunnea, rather than being located at the root of the subtree comprising all Juglans and Carya. However, the placement of the Paleooremunnea fossil is know to be problematic (see [62], p. 425). While in [62] its phylogenetic placement varies significantly, depending on the method used, we obtain an assignment with 100% Bootstrap support for this fossil. Overall, the fossil placements are biologically reasonable and could give rise to new biological hypotheses (D. Soltis, personal communication).

Figure 6.14 shows the placement of the Marsupial fossils from [10]. The placement of the Djarthia fossil is particularly interesting, as it seems to confirm the original placement as a member of Australidelphia, but outside the subtree comprising extant Australasian marsupials (see Figure 3a-b in [10] ). Note that, in contrast to the original studies [10, 62] which include the fossil sequences into a de novo tree inference, we placed them individually into a previously generated molecular tree using morphological data alone. This one-by-one placement of the fossil sequences generates the multi-furcating (non-binary) trees, in which more than one fossil taxon can be placed into the same edge of the molecular RT.

## 6.5 PaPaRa: Experimental Evaluation

The following section contains the evaluation of the PaPaRa algorithm. The main application scenario for PaPaRa is for metagenomic analyses using phylogenetic placement methods such as the Evolutionary Placement Algorithm (EPA) [12] or pplacer [63]. As mentioned in Section 5, for these algorithms the QS need to be in alignment with the RA. To this end, our performance evaluation is specifically designed to assess the accuracy of alignment methods (PaPaRa, HMMALIGN) with respect to analyzing (identifying) short reads by means of phylogenetic placement algorithms. In other words, we do not directly evaluate alignment quality. Instead, we analyze the impact of the QS alignment method on the phylogenetic placement quality/accuracy using the EPA. Therefore, we assess alignment quality by means of the calculated/inferred evolutionary position of the QS. Measures and methods for assessing the placement accuracy of short reads using the EPA are described in Section 6.2. We also carried out a basic assessment of QS placement accuracy when QS are re-aligned with HMMALIGN (v3.0), albeit in a different experimental setup and context. Here, we use the same distance/accuracy measures as described in Section 6.2 (Figure 6.1A). The node distance (ND), which is defined as the number of nodes along the path between the 'true' placement position and the inferred placement position (see below) represents an absolute accuracy measure. The normalized edge distance (NED%), is a relative measure between 'true' and inferred placement positions that is based on the actual edge-lengths in the RT. The NED% reflects the relative evolutionary distance between the two positions. As mentioned in Section 6.2, the

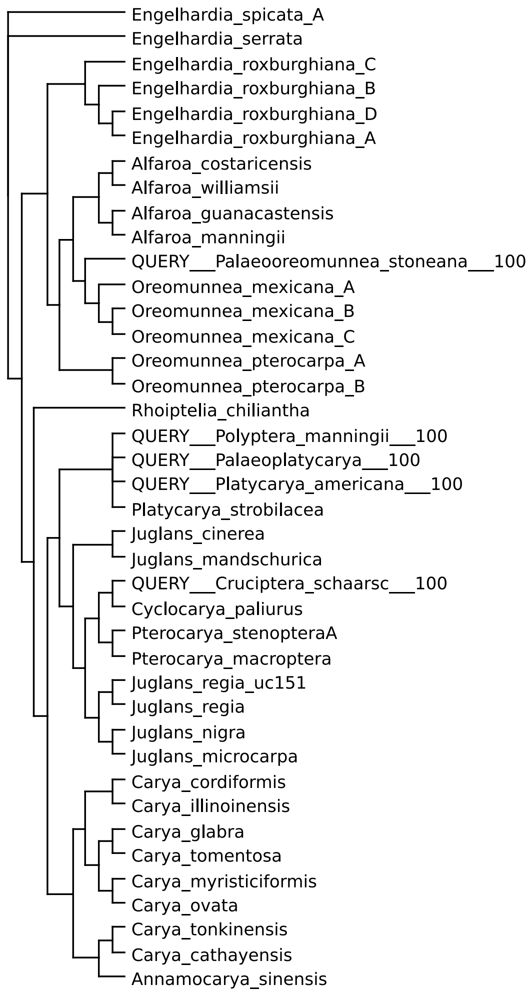


Figure 6.13: Fossil placement in data set D1. The number behind the query sequence names denotes the bootstrap support of the placement.

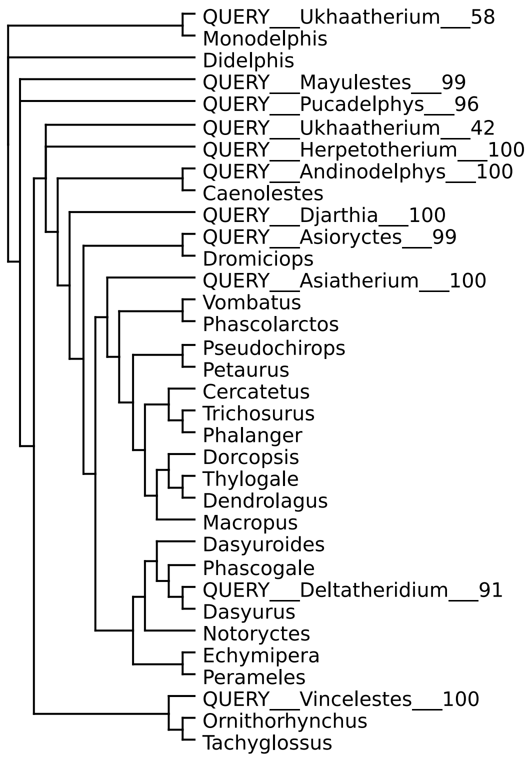


Figure 6.14: Fossil placement in data set D2. Placements with BS support $< 10\%$ have been removed.

NED% values in this evaluation have been normalized by the position-specific worst-case placement error, rather than the tree-diameter.

### 6.5.1 Realignment of emulated QS

The main part of our performance evaluation compares the placement accuracy of EPA-computed QS placement with respect to the placement position of the optimally aligned QS ('true' placement). The EPA placements obtained without QS re-alignment are regarded as the optimal ('true') reference placements, against which the phylogenetic placements *after* QS re-alignment with PaPaRa/HMMALIGN are compared. For such an evaluation, we require QS that are already in alignment with the RA in order to compute an optimal reference placement with the EPA that represents the 'true' placement. The QS, which are assumed to be correctly aligned in the reference QS alignment, are initially dis-aligned (we simply remove all gaps), and passed to the two QS alignment procedures (PaPaRa and HMMALIGN) for re-alignment. The thereby re-aligned QS are then used as input for the EPA.

#### Data Sets

The correctly aligned QS were extracted from 7 real-world full-length biological MSAs (termed original MSAs). The taxon set of each reference MSA was randomly split into two sub-alignments of equal size (each containing 50% of taxa from the original MSA). One half of the original MSA is then used as RA, on which we compute the best-known ML tree with RAxML [79]. This tree is then used as RT for the RA. The other half of the original RA sequences is used to generate a candidate QS (CQS) set. Because both sub-alignments originally formed part of the same MSA, all sequences in the CQS set (and all sub-sequences of these sequences) are in alignment with the RA. Analogous to Section 6.3, the sequences in the CQS set that are derived from a MSA of full length sequences are then reduced in length (see below for details) to emulate QS that resemble short sequence reads.

For each data set, we carried out our performance analysis using three common MSA methods to generate three original MSA versions respectively. We computed de-novo MSAs using MUSCLE (v3.70), MAFFT (v6.626), and PRANK$_{+F}$. We selected these three programs, because they are widely used state-of-the-art codes for MSA *and* because they are based on fundamentally different alignment philosophies. Since we adopt an agnostic view on what the best MSA strategy may be, we thereby intend to assess the flexibility and adaptability of PaPaRa to diverse MSA philosophies that are implicitly encoded in the underlying RAs. Finally, we also used the (partially manu-

| Data MSA | # sites ORIG | # sites MUSCLE | # sites MAFFT | # sites PRANK | # taxa | # QS |
|---|---|---|---|---|---|---|
| D150 | 1269 | 1272 | 1336 | 1939 | 75 | 1500 |
| D218 | 2101 | 2044 | 1993 | 6425 | 109 | 2180 |
| D500 | 1398 | 1402 | 1402 | 1479 | 250 | 5000 |
| D628 | 1199 | 1761 | 1348 | 2437 | 314 | 6280 |
| D714 | 1241 | 1341 | 1273 | 2205 | 357 | 7140 |
| D855 | 1436 | 1469 | 1443 | 2208 | 427 | 8560 |
| D1604 | 1271 | 1325 | 1278 | 2475 | 802 | 16040 |

Table 6.11: Data sets used for evaluation of the QS alignment algorithms. The the values in columns 2–5 correspond to the four RA per data set, which have been generated with the different MSA approaches (ORIG, MUSCLE, MAFFT and PRANK$_{+F}$).

ally curated) MSAs as provided by the authors of our test data sets. While manual curation is debatable, in particular in the light of reproducibility of results, we nonetheless used the given MSAs because hand-curation is still common practice and may encode empirical biological knowledge about the underlying data. We also conducted some experiments on simulated sequence data, for which the performance of both, PaPaRa and HMMALIGN, is considerably better than on a real data set of comparable size (see [17]; supplementary material). We therefore assume that the QS alignment problem is harder on our test data sets that are derived from real sequences and therefore only used real sequence data for the evaluation. For each of the 7 data sets, we thus have 4 original MSA versions: manually curated (called ORIG in the following), MUSCLE, MAFFT and PRANK$_{+F}$. Table 6.11 contains information about the length (number of RA columns) in the data sets as well as the number of taxa contained in the RAs and the respective number of QS.

In our experiments, we assume that the full length CQS (and the derived short QS) obtained from the 4 MSA versions for each data set, represent ideally aligned QS, with respect to the corresponding RA. To the best of our knowledge, HMMALIGN and PaPaRa currently represent the most suitable methods for aligning short-reads to a RA. Therefore, we specifically did not use real short read data, for which the correct alignment to the RA is not known. Our experiments are designed to systematically test the impact of QS alignment quality on the evolutionary placement accuracy of the EPA. Thus, we did not consider alignment quality criteria, other than the relative QS placement error with respect to the reference QS placement obtained from the original MSA.

Similar to the earlier evaluation of the EPA algorithm, from every full-length QS in the QS set, we randomly sub-sampled 20 contiguous QS with uniformly distributed position and normally distributed lengths (mean length: $100 \pm 10$ bp and $200 \pm 60$ bp). We have already used this method for QS generation in Section 6.3 to create simulated short read sequences that emulate reads obtained from a high throughput sequencer. For each of the 20 sub-sampled QS, we computed an individual reference placement, because the EPA placement of the sub-sampled QS can differ from the placement of the full-length CQS. Thereby, we can more accurately assess the QS alignment impact on placement accuracy, without the potential bias that is induced by QS length variation (see [12]). To yield the evaluation more realistic, we then also modified the subsampled QS by introducing typical next-generation sequencer errors. Based on the methods implemented in Grinder [8] and the empirical data in [9], we re-implemented an appropriate model for simulating representative 454 homopolymer sequencing errors. Each homopolymer (this also includes single characters) that is detected in the raw QS is randomly shortened or elongated, according to the empirical probabilities provided in [9]. The JAVA source code of the simulation program is available at `https://raw.github.com/sim82/java_tools/master/src/ml/SampleDistSubseq.java`.

Because we derive the new RA by splitting the original MSA into two parts (i.e., the RA and the QS set), it is likely that the RA will contain sites that entirely consist of gaps. This is especially true for MSAs generated with PRANK$_{+F}$, that frequently comprise sites with only one or two non-gap characters. Since entirely ‘empty’ columns that only contain gaps are not present in real MSAs, such columns are completely removed from the RA *prior* to QS alignment (using HMMALIGN and PaPaRa) and placement (using EPA) in our experiments.

### 6.5.2 PaPaRa 1.0: Placement Accuracy

For each of the 7 data sets, we determined PaPaRa- and HMMALIGN-based QS placement accuracy for all 4 original MSA versions. Tables 6.12 and 6.13 contain the results for all data sets. The values in the Tables indicate RT-based average ND and NED% distances between the ‘true’ reference EPA placements, based on the QS alignment extracted from the original MSA and the respective EPA placements with QS re-alignment. Values for the two QS re-alignment methods (PaPaRa, HMMALIGN) are provided separately.

In Table 6.12 we provide results for all data sets in terms of number of taxa using QS with a mean length of $100 \pm 10$ bp. When HMMALIGN is used for re-alignment on the manually curated MSA on this data set, EPA placements on the largest data set (D1604) are on average 1.35 nodes (column HMM, row ORIG) away from the reference placement position. For PaPaRa the corresponding node distance (ND) is 0.28 (column PA, row ORIG). When the relative distance (NED%) is used, the corresponding values are 1.35 (HMMALIGN) and 0.71 (PaPaRa). Therefore, PaPaRa reduces the error in QS node placement distance by a factor of 4.87 and factor 1.90 for the relative distance respectively compared to HMMALIGN. For the automated MSA methods (MUSCLE, MAFFT, PRANK$_{+F}$), HMMALIGN and PaPaRa show analogous accuracy differences. The EPA placements of re-aligned QS are on average 3.11–5.88 times closer (1.7–2.52 times for the NED% distance) to the reference placements in terms of node distance (ND) for PaPaRa than for HMMALIGN. On some of the smaller data sets (D150, D218 and D714) the PaPaRa-aligned QS can produce worse placements than the HMMALIGN-aligned QS. However, in most cases, PaPaRa only produces worse results with respect to the NED% measure.

For the longer QS of mean length 200$\pm$60 shown in Table 6.13, placement accuracy increases for both alignment methods in most cases. The improvements are more pronounced for HMMALIGN, where the ND is improved by up to a factor of 2. Generally, accuracy differences between PaPaRa and HMMALIGN decrease. For 5 out of the 7 data sets PaPaRa produces worse results than HMMALIGN at least for some of the tests (i.e., for certain RA and distance measure combinations). As with the shorter QS, this is especially pronounced on the smaller data sets with less taxa. Thus, the advantage of using a phylogeny-aware QS alignment strategy on data sets with few taxa is smaller. In contrast to PaPaRa, on small data sets HMMALIGN can take advantage of its more powerful probabilistic RA model and the stronger signal contained in the 200 bp long QS. However, the typical RT will be considerably larger than the smallest data sets in this study, because of the very dense taxon sampling of the 16S rRNA. Thus, while the accuracy improvement induced by PaPaRa is minor on small data sets, it substantially improves on the larger reference data sets in our experiments.

The rather pronounced difference between the two distance measures (i.e., when the ND is considered, the advantage of PaPaRa over HMMALIGN is larger than for the NED%), can be attributed to the RT shape of this data set (D1604): Visual inspection revealed that, it contains a large number of closely related taxa which gives rise to a large number of relatively short edges (branches) near the tips of the tree. Thus, if a QS is misplaced within such a region of the tree, this can result in a relatively large ND (because there is a large number of nodes in the region), but a small NED% since edges between the nodes are short. The HMMALIGN re-aligned QS tend to be misplaced in such ‘dense’ areas of the tree, which results in a relatively

| | | ND | | NED % | |
|---|---|---|---|---|---|
| | Data Set | PaPaRa | HMM | PaPaRa | HMM |
| D150 | ORIG | **0.52** | 1.31 (2.52) | 1.74 | **1.68 (0.96)** |
| | MUSCLE | **0.57** | 1.18 (2.07) | 2.00 | **1.51 (0.75)** |
| | MAFFT | **0.58** | 1.29 (2.24) | 1.69 | **1.49 (0.88)** |
| | PRANK$_{+F}$ | **0.70** | 2.10 (3.02) | **1.88** | 2.89 (1.54) |
| D218 | ORIG | 2.02 | **1.78 (0.88)** | 6.57 | **5.80 (0.88)** |
| | MUSCLE | 1.95 | **1.73 (0.88)** | 7.63 | **6.25 (0.82)** |
| | MAFFT | **1.86** | 1.92 (1.03) | 6.21 | **6.14 (0.99)** |
| | PRANK$_{+F}$ | 2.04 | **2.03 (0.99)** | 7.18 | **6.86 (0.96)** |
| D500 | ORIG | **0.57** | 0.59 (1.03) | **1.64** | 1.65 (1.01) |
| | MUSCLE | **0.60** | 0.68 (1.13) | **1.71** | 1.91 (1.12) |
| | MAFFT | **0.62** | 0.69 (1.12) | **1.74** | 1.91 (1.10) |
| | PRANK$_{+F}$ | **0.68** | 0.81 (1.20) | **1.88** | 2.22 (1.18) |
| D628 | ORIG | **0.80** | 1.90 (2.39) | **2.07** | 3.89 (1.88) |
| | MUSCLE | **1.09** | 3.75 (3.44) | **2.81** | 9.38 (3.34) |
| | MAFFT | **0.47** | 2.78 (5.90) | **1.11** | 5.00 (4.51) |
| | PRANK$_{+F}$ | **0.50** | 3.32 (6.68) | **1.14** | 5.61 (4.92) |
| D714 | ORIG | 0.55 | **0.54 (0.99)** | 1.71 | **1.28 (0.75)** |
| | MUSCLE | **0.50** | 0.86 (1.70) | **1.40** | **1.40 (1.00)** |
| | MAFFT | **0.45** | 0.75 (1.65) | 1.58 | **1.55 (0.98)** |
| | PRANK$_{+F}$ | **0.51** | 1.28 (2.50) | **1.47** | 2.48 (1.68) |
| D855 | ORIG | **0.59** | 1.32 (2.24) | **1.03** | 1.67 (1.62) |
| | MUSCLE | **0.67** | 1.54 (2.29) | **1.22** | 2.32 (1.90) |
| | MAFFT | **0.66** | 1.03 (1.56) | **1.11** | 1.50 (1.35) |
| | PRANK$_{+F}$ | **0.80** | 2.28 (2.85) | **1.47** | 3.57 (2.43) |
| D1604 | ORIG | **0.28** | 1.35 (4.87) | **0.71** | 1.35 (1.90) |
| | MUSCLE | **0.43** | 1.35 (3.11) | **0.87** | 1.48 (1.70) |
| | MAFFT | **0.29** | 1.21 (4.12) | **0.72** | 1.29 (1.80) |
| | PRANK$_{+F}$ | **0.41** | 2.43 (5.88) | **0.95** | 2.41 (2.52) |

Table 6.12: Placement accuracy for the two QS alignment methods on QS of lengths $100 \pm 10$ bp. The relative accuracy of HMMALIGN compared to PaPaRa is given in parentheses.

| | | ND | | NED % | |
|---|---|---|---|---|---|
| | Data Set | PaPaRa | HMM | PaPaRa | HMM |
| D150 | ORIG | **0.46** | 0.62 (1.34) | 1.26 | **1.00 (0.79)** |
| | MUSCLE | **0.46** | 0.52 (1.12) | 1.44 | **0.64 (0.44)** |
| | MAFFT | **0.46** | 0.60 (1.29) | 1.22 | **0.79 (0.65)** |
| | PRANK$_{+F}$ | **0.52** | 1.08 (2.08) | 1.53 | **1.49 (0.97)** |
| D218 | ORIG | 1.57 | **1.06 (0.68)** | 5.53 | **3.61 (0.65)** |
| | MUSCLE | 1.69 | **0.98 (0.58)** | 6.94 | **4.08 (0.59)** |
| | MAFFT | 1.59 | **1.24 (0.78)** | 5.63 | **4.37 (0.78)** |
| | PRANK$_{+F}$ | 1.62 | **1.26 (0.78)** | 5.88 | **4.57 (0.78)** |
| D500 | ORIG | 0.46 | **0.27 (0.58)** | 1.44 | **0.76 (0.53)** |
| | MUSCLE | 0.44 | **0.28 (0.63)** | 1.38 | **0.84 (0.61)** |
| | MAFFT | 0.45 | **0.30 (0.67)** | 1.37 | **0.89 (0.65)** |
| | PRANK$_{+F}$ | 0.47 | **0.37 (0.79)** | 1.41 | **1.07 (0.76)** |
| D628 | ORIG | **0.71** | 1.81 (2.53) | **1.66** | 3.32 (2.00) |
| | MUSCLE | **0.86** | 2.00 (2.33) | **2.03** | 4.11 (2.02) |
| | MAFFT | **0.36** | 2.05 (5.64) | **0.79** | 3.27 (4.13) |
| | PRANK$_{+F}$ | **0.34** | 1.61 (4.72) | **0.77** | 2.47 (3.21) |
| D714 | ORIG | 0.45 | **0.36 (0.81)** | 1.37 | **0.81 (0.59)** |
| | MUSCLE | **0.45** | 0.48 (1.06) | 1.25 | **0.75 (0.60)** |
| | MAFFT | **0.34** | 0.42 (1.26) | 1.22 | **0.94 (0.77)** |
| | PRANK$_{+F}$ | **0.42** | 0.74 (1.79) | **1.30** | 1.31 (1.01) |
| D855 | ORIG | **0.59** | 0.70 (1.19) | 0.99 | **0.87 (0.88)** |
| | MUSCLE | **0.63** | 0.90 (1.43) | **1.15** | 1.20 (1.04) |
| | MAFFT | 0.61 | **0.51 (0.83)** | 1.09 | **0.73 (0.67)** |
| | PRANK$_{+F}$ | **0.74** | 1.40 (1.90) | **1.33** | 2.05 (1.54) |
| D1604 | ORIG | **0.25** | 0.90 (3.53) | **0.63** | 0.80 (1.28) |
| | MUSCLE | **0.40** | 1.03 (2.61) | **0.76** | 0.92 (1.21) |
| | MAFFT | **0.26** | 0.82 (3.18) | **0.65** | 0.79 (1.21) |
| | PRANK$_{+F}$ | **0.34** | 1.65 (4.80) | **0.84** | 1.39 (1.64) |

Table 6.13: Placement accuracy for the two QS alignment methods on QS of lengths $200 \pm 60$ bp. The relative accuracy of HMMALIGN compared to PaPaRa is given in parentheses.

large average ND compared to PaPaRa re-aligned QS. To this end, by using a phylogeny-aware approach, PaPaRa can better use such densely sampled areas in the RT, while such a fine-grained resolution can not be achieved by using a 'flat' probabilistic profile (as implemented in HMMALIGN). On smaller data sets the differences between the two distance measures are less pronounced.

In most cases, the largest difference in placement accuracy between PaPaRa and HMMALIGN is observed for PRANK$_{+F}$-based MSAs. Because of the specific MSA approach in PRANK$_{+F}$, a strong and consistent gap signal is embedded into the original MSA. In contrast to HMMALIGN, PaPaRa is able to use this embedded gap-signal in combination with the respective RT. In Figure 6.15 we provide histograms of the average ND distribution for QS (with mean length 100bp) over all data sets and for all reference MSAs. PaPaRa-based QS alignments generate placements that are, on average, closer to the 'true' reference position. The histograms also show that, for PRANK$_{+F}$-generated MSAs, the placement accuracy decrease induced by using HMMALIGN is more pronounced compared to other MSA methods. In general, PaPaRa is thus more robust with respect to different MSA philosophies and hence more adaptable.

For the above experiments, we knew a priori, that the QS had sufficiently closely related sequences in the RA. If this is not the case (e.g., if reads from a distant clade that is not contained in the RT are sampled), according to some preliminary experiments, neither the QS alignment method nor the EPA can be expected to produce a reliable result. This observation also holds when the QS stem from a different (e.g., non-orthologous) genomic region than the sequences in the RA. Therefore, we suggest that the QS should be checked beforehand, for example by doing a quick BLAST search against the sequences in the RA to exclude completely unrelated sequences.

### 6.5.3 PaPaRa 1.0 Run-time Performance

We also carried out a runtime assessment of HMMALIGN and PaPaRa 1.0. A serial execution of PaPaRa requires between 385s and 44,270s on the smallest (D150) and largest (D1604) data set respectively (using ORIG MSAs and QS of lengths $200 \pm 60$ bp on an 3.2 GHz Intel Core i5; compiled with `gcc 4.5.1` for Linux). The corresponding HMMALIGN times range between 61s and 1031s. Thus, HMMALIGN is 6.3 – 43 times faster than PaPaRa. This performance difference is not surprising, because PaPaRa runtimes depend on the number of QS *and* the number of taxa in the RT. In other words, PaPaRa exhibits a significantly higher theoretical runtime complexity than HMMALIGN. Therefore, performance optimization of the core alignment procedure

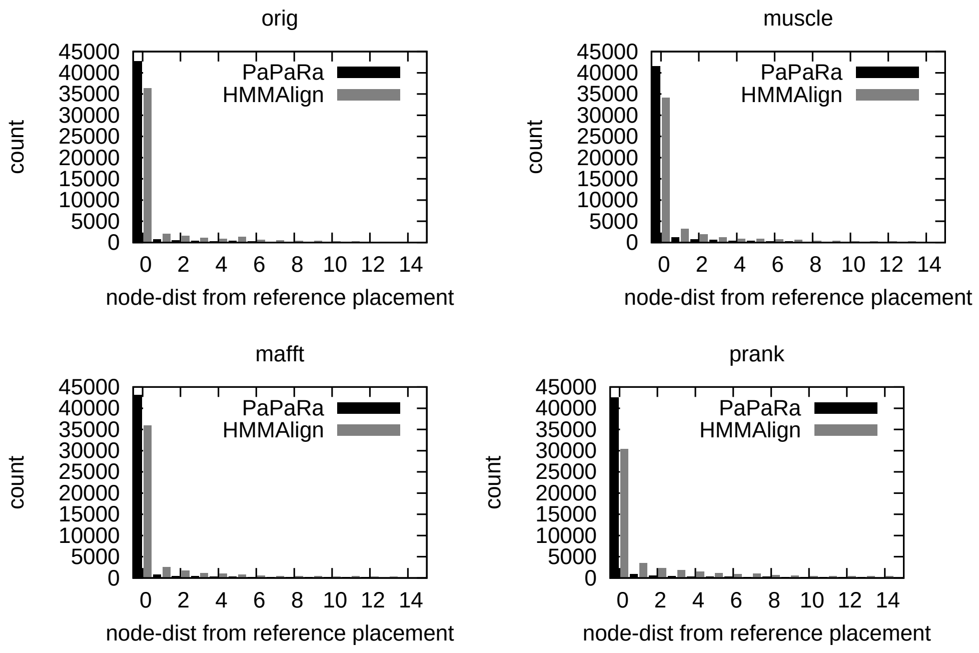


Figure 6.15: Histograms showing the distribution of the placement error (ND) for PaPaRa and HMMALIGN aligned QS, over all data sets.

is essential for overall PaPaRa performance. The inherent —significantly higher— time complexity of PaPaRa is also one main reason for aligning against ancestral parsimony state vectors (i.e., bit-vectors), instead of using a probabilistic approach that would require costly floating point arithmetics.

Currently, PaPaRa creates the QS alignments in two phases: Initially, all QS are aligned, and thereby scored, against all ancestral state vectors (insertion positions/edges of the RT). For performance reasons the actual alignments (i.e., the dynamic programming traceback) are not computed in this phase. Only after the best scoring insertion position has been determined for each QS, the actual alignments are generated by aligning them again to the best positions in a second step. The initial step normally accounts for more than 99% of overall runtime. As mentioned in Section 5.3, the core alignment procedure can be further optimized by deploying 128-bit wide SSE vector instructions. These optimizations were integrated into PaPaRa 2.0, for which we will present performance results in Section 6.5.5. One could also think of a more compact bit-level representation of the input data to reduce memory requirements and cache misses. The sequential dynamic programming implementation used in PaPaRa can perform about 0.12 GCUPS (giga cell updates per second) on the Intel Core i5. For comparison, Farrar et al. report more than 3 GCUPS for an SSE optimized smith-waterman implementation on an older 2.0 GHz Intel Xeon Core 2 Duo processor [29]. Note that HMMALIGN, as used here, already includes SSE vectorization in the alignment algorithm.

### 6.5.4 PaPaRa 2.0 Placement Accuracy

Table 6.14 provides an accuracy evaluation of PaPaRa 2.0 compared to PaPaRa 1.0 and HMMALIGN (for easier comparison, the accuracy values for PaPaRa and HMMALIGN are re-produced from Table 6.12). The number in the respective data set name denotes the number of taxa in the respective data set. We restrict our evaluation to the harder case of short QS of length $100 \pm 10$ bp. As in the preceding evaluation of PaPaRa 1.0, we provide results for 4 different multiple sequence reference alignments per data set (ORIG, MUSCLE, MAFFT and PRANK$_{+F}$). The measure for quantifying alignment accuracy is exactly the same as for PaPaRa 1.0. The values in Table 6.14 correspond to the RT-based mean distances between the 'true' reference EPA placement and the respective EPA placement after re-alignments with PaPaRa 1.0, PaPaRa 2.0, and HMMALIGN.

On most data sets PaPaRa 2.0 outperforms PaPaRa 1.0 as well as HMMALIGN. On the largest data set (D1604) and the original alignment (ORIG), the EPA places the PaPaRa 2.0 aligned QS within 0.23 nodes of the reference placement, while for PaPaRa 1.0 and HMMALIGN the distances are 0.28 and 1.31 respectively. On the MUSCLE and MAFFT generated RAs, the performance improvements are similar, while on the PRANK$_{+F}$ aligned RA, PaPaRa 2.0 performs slightly worse than PaPaRa 1.0 (ND 0.44 compared to ND 0.41). However, with respect to the NED% measure, PaPaRa 2.0 consistently outperforms PaPaRa 1.0 as well as HMMALIGN. More importantly, PaPaRa 2.0 no longer shows the comparably large discrepancy between ND and NED% measures on this data set (see Section 6.5.2). On the remaining data sets (except for D628) PaPaRa 2.0 outperforms version 1.0. In contrast to version 1.0 it also consistently outperforms HMMALIGN under both accuracy measures.

### 6.5.5 PaPaRa 2.0 Run-time Performance

To assess the run-time performance of PaPaRa 2.0, we repeated the performance evaluation from Section 6.5.3 where we reported the overall program run times of PaPaRa 1.0 and HMMALIGN on the smallest (D150) and largest (D1604) data set respectively. In contrast to the accuracy evaluation, the run-time is based on 1500 (D150) and 16040 (D1604) QS with lengths

Table 6.14: Placement accuracy for the three QS alignment methods under consideration. Results are given using the node-distance (ND) and normalized edge distance (NED%) measures.

| | | ND | | | NED% | | |
|---|---|---|---|---|---|---|---|
| | Data Set | PaPaRa | PaPaRa 2.0 | HMM | PaPaRa | PaPaRa 2.0 | HMM |
| *D*150 | ORIG | 0.52 | **0.25** | 1.31 | 1.74% | **1.02%** | 1.68% |
| | MUSCLE | 0.57 | **0.27** | 1.18 | 2% | **0.99%** | 1.51% |
| | MAFFT | 0.58 | **0.30** | 1.29 | 1.69% | **1.02%** | 1.49% |
| | PRANK$_{+F}$ | 0.7 | **0.43** | 2.1 | 1.88% | **0.91%** | 2.89% |
| *D*218 | ORIG | 2.02 | **1.12** | 1.78 | 6.57% | **3.62%** | 5.8% |
| | MUSCLE | 1.95 | **1.12** | 1.73 | 7.63% | **4.34%** | 6.25% |
| | MAFFT | 1.86 | **1.13** | 1.92 | 6.21% | **3.73%** | 6.14% |
| | PRANK$_{+F}$ | 2.04 | **1.19** | 2.03 | 7.18% | **4.02%** | 6.86% |
| *D*500 | ORIG | 0.57 | **0.32** | 0.59 | 1.64% | **0.88%** | 1.65% |
| | MUSCLE | 0.6 | **0.28** | 0.68 | 1.71% | **0.80%** | 1.91% |
| | MAFFT | 0.62 | **0.29** | 0.69 | 1.74% | **0.83%** | 1.91% |
| | PRANK$_{+F}$ | 0.68 | **0.35** | 0.81 | 1.88% | **0.96%** | 2.22% |
| *D*628 | ORIG | 0.8 | **0.77** | 1.9 | 2.07% | **1.85%** | 3.89% |
| | MUSCLE | **1.09** | 1.92 | 3.75 | **2.81%** | 5.15% | 9.38% |
| | MAFFT | **0.47** | 0.55 | 2.78 | **1.11%** | 1.13% | 5% |
| | PRANK$_{+F}$ | **0.5** | 0.58 | 3.32 | 1.14% | **1.05%** | 5.61% |
| *D*714 | ORIG | 0.55 | **0.37** | 0.54 | 1.71% | **0.94%** | 1.28% |
| | MUSCLE | 0.5 | **0.33** | 0.86 | 1.4% | **0.74%** | 1.4% |
| | MAFFT | 0.45 | **0.26** | 0.75 | 1.58% | **0.69%** | 1.55% |
| | PRANK$_{+F}$ | 0.51 | **0.30** | 1.28 | 1.47% | **0.59%** | 2.48% |
| *D*855 | ORIG | 0.59 | **0.53** | 1.32 | 1.03% | **0.63%** | 1.67% |
| | MUSCLE | 0.67 | **0.44** | 1.54 | 1.22% | **0.71%** | 2.32% |
| | MAFFT | 0.66 | **0.58** | 1.03 | 1.11% | **0.81%** | 1.5% |
| | PRANK$_{+F}$ | 0.8 | **0.61** | 2.28 | 1.47% | **0.97%** | 3.57% |
| *D*1604 | ORIG | 0.28 | **0.23** | 1.35 | 0.71% | **0.37%** | 1.35% |
| | MUSCLE | 0.43 | **0.38** | 1.35 | 0.87% | **0.50%** | 1.48% |
| | MAFFT | 0.29 | **0.21** | 1.21 | 0.72% | **0.37%** | 1.29% |
| | PRANK$_{+F}$ | **0.41** | 0.44 | 2.43 | 0.95% | **0.58%** | 2.41% |

$200 \pm 60$ bp. We used these QS lengths to obtain comparable run times to the evaluation in Section 6.5.3. For our experiments we used the same Intel core i5-750 CPU as in the original evaluation, with a more recent gcc version (4.6.2). All measurements were carried out using a single CPU core.

For PaPaRa 2.0 (which implements the vectorized alignment kernel described in Section 5.3, the overall program run times range between 27s (D150) and 2590s (D1604). The corresponding run times with the other methods are 345s-39746s (PaPaRa 1.0) and 61s-1030s (HMMALIGN). Note that, the values reported for PaPaRa 1.0 are approximately 10% lower than in Section 6.5.3, while the times for HMMALIGN remain almost constant. We assume that the run time differences are mainly caused by the newer gcc version (presumably the naïve implementation of the alignment core in PaPaRa 1.0 gives the newer gcc version more room for optimization than the well-tuned HMMALIGN kernel). Generally, PaPaRa 2.0 is 12–15 times faster than PaPaRa 1.0 on these data sets. For data sets in this size range, the run time of PaPaRa 2.0 is in the same order of magnitude as HMMALIGN. For larger data sets (in terms of taxa contained in the RA), the program run time of PaPaRa 2.0 increases at a higher rate than for HMMALIGN. Evidently, this is because of the considerably higher time complexity of PaPaRa (both versions) compared to HMMALIGN [17].

The speedup of more than a factor of 12 for the vectorized PaPaRa 2.0 code is larger than expected, given that the SIMD vector width is only 8. Also the alignment kernel of PaPaRa 2.0 is slightly more complex than that of PaPaRa 1.0, because of the affine gap penalties on the RA side, which introduce an additional maximization as well as additions for calculating the $I^{i,j}$ values in Equation 5.4. The unexpectedly large performance gain is most likely due to the revised implementation of the maximization operation and conditional statements of Equation 5.4. The vectorized implementation deploys a specialized maximization operation (e.g., `_mm_max_epi16`) and bit masks, while PaPaRa 1.0 used explicit branches (i.e., `if/else`). As a result, the inner loop of the vectorized implementation is very compact and, more importantly, contains no conditional jumps.

### 6.5.6 Run Time Performance of the coupled CPU-GPU system

We assessed the performance of the PaPaRA GPU implementation (see Section 5.4) as well as the coupled CPU-GPU system in [6]. The following is a summary of the CPU-GPU system run time evaluation we did for [6]. The experiments were performed on a system consisting of an Intel core i7-2600

Table 6.15: System performance of the hybrid CPU-GPU PaPaRa system in terms of GCUPS (giga cell updates per second).

| data set | $T_{scoring}$(s) | $T_{all}$(s) | $GCUPS_{CPU}$ | $GCUPS_{GPU}$ | $GCUPS_{all}$ |
|---|---|---|---|---|---|
| 1604.PRANK | 227.21 | 273.12 | 13.38 | 20.04 | 33.42 |
| 16S.B.ALL | 12943.9 | 13111.4 | 13.87 | 20.00 | 33.87 |

CPU and a NVidia GeForce GTX 560 GPU. The CPU part of the coupled system uses all 4 CPU cores of the core i7-2600. We performed the run time evaluation on two data sets, 1604.PRANK, which was already used for the accuracy evaluation in Sections 6.5.2 and 6.5.4 and 16S.B.ALL from [65]. This data set consists of 13,822 RS of length 6857 and 13,820 QS of lengths that vary between 29 and 483.

Table 6.15 shows the performance of the hybrid CPU-GPU algorithm on the two data sets. Row $T_{scoring}$ provides the runtime for the scoring phase. Column $T_{all}$ shows the overall runtime for the whole algorithm, including pre-processing of input files and generating the actual alignments via back-tracking. These pre- and post-processing steps are unoptimized sequential tasks that are performed on the CPU. As a metric for our performance comparison we use the number of dynamic programming matrix cell calculations per second (giga cell updates per second; GCUPS). The overall CPU performance is shown in column $GCUPS_{CPU}$, which reflects the accumulated performance on 4 CPU cores. Overall GPU performance is provided in column $GCUPS_{GPU}$. On both data sets, the relative contibution of the CPU cores and of the GPU are very similar; the CPU and GPU contribute 40% and 60% respectively of overall GCUPS to the accumulated CPU-GPU system performance which is indicated in the last column $GCUPS_{all}$. All GCUPS values refer to sustained GCUPS, that is, the values include the overhead induced by load-imbalance between the CPU and the GPU. Load imbalance is observed when either one of the CPU threads or the GPU finish last, and require the other computational resources to wait. This evaluation shows that, by leveraging the combined computing power of CPUs and GPUs, it is possible to more than double overall performance with respect to an already highly optimized stand-alone CPU implementation. On the other hand, it is also clear that the performance advantage of the GPU over an optimized CPU implementation is not as high as often claimed (for a thorough discussion on this subject see [54]).

## Summary

This chapter addressed the performance evaluation of the novel algorithms introduced in Chapters 3, 4 and 5. For the EPA we demonstrated that its placements are – in most cases – more accurate than placements based on pure sequence similarity. When additional heuristics are used, EPA run-time performance is comparable to sequence similarity-based placement approaches such as BLAST. Also, the parallel implementation allows for using the EPA on large data sets. Furthermore, we show that the EPA can be applied to non-molecular sequence data. In combination with automatic site-weight calibration this enables fast and accurate placement of taxa with morphological data only (e.g., fossils). Finally we show that the PaPaRa algorithm, when used as a pre-processing step, can fundamentally improve the accuracy of subsequent EPA-based short read placements. By deploying technical optimizations (vectorization and multi-threading), we show that the run-time performance of PaPaRa is competitive with respect to non phylogeny-aware alignment procedures.

CHAPTER 7

# Conclusion and Future Work

This chapter provides a conclusion of the work an presents possible future research directions.

## 7.1 Conclusion

### 7.1.1 EPA

We have presented accurate and scalable algorithms for phylogeny-aware analysis of short-reads. A phylogeny-aware approach has methodological advantages over standard, pair-wise, sequence similarity-based approaches and the EPA is freely available for download as open source code, as a web-service and as a GUI. We demonstrate that our approach is substantially more accurate than standard techniques used for analyzing microbial communities for example. More importantly, we demonstrate that achieving better accuracy does not require longer inference times and that our approach is as fast as a simple BLAST-based search when additional heuristics are used.

The EPA is also relatively straight-forward to parallelize via multi-grain parallelization [82]. On a multi-core system with 32 cores and 64GB of main memory, we were able to place 100,627 QS in parallel into a RT with 4,874 taxa within only 1.5 hours. The application of the EPA is not limited to molecular data only, and we have used the EPA for the placement of fossil taxa onto a (molecular) RT of extant species.

### 7.1.2 Fossil Placement

We have conducted an assessment of fossil placement accuracy using morphological data under the ML criterion on simulated and real-world data sets based on a well-established RT. In addition, we have developed a statistical weight calibration mechanism that is able to identify morphological sites, that exhibit a phylogenetic signal which is congruent to that of the RT. By using accordingly calibrated integer weights we can improve upon the absolute and relative placement accuracy by 20% on simulated data sets and by 25% on real-world data sets.

Moreover, we find that, despite the partially high incongruence between ML trees obtained from the morphological and molecular data partitions, the achieved accuracy under ML is sufficient for reliably placing fossils. Two biological case studies with real fossil taxa reveal that we can obtain reasonable biological results using the weight calibration and fossil placement algorithms.

The statistical weight calibration procedure as well as the phylogenetic placement algorithm have been integrated into RAxML which is a freely available and widely used tool for phylogenetic inference.

### 7.1.3 PaPaRa

We have conducted an experimental evaluation of methods for aligning short QS against a fixed RT and RA in the context of likelihood-based evolutionary QS placement methods. We also introduced PaPaRa 1.0, a novel phylogeny-aware method for this purpose. On short QS and large RAs, PaPaRa performs better than the currently best phylogeny-agnostic method (HMMALIGN). For longer QS and small RAs the performance of the current PaPaRa implementation is relatively poor. Apparently, the more powerful probabilistic model in HMMALIGN, is beneficial, if the RA is small enough to be represented by a single flat profile. For larger RAs, PaPaRa has the advantage of sampling different signals from different parts of the associated RT and performs well, despite using a simple model for ancestral states and an ‘ad-hoc’ scoring scheme. We intend to introduce additional heuristics for reducing the total number of ancestral state vectors against which individual QS need to be aligned.

Based on the proof-of-concept implementation of PaPaRa 1.0, we created PaPaRa 2.0, a substantially improved and accelerated version of the algorithm. The source code of the algorithm is available at `https://github.com/sim82/papara_nt.git`. The new version requires less empirical ad hoc rules for extracting information about the indel distribution in the RA by replacing them with a straight-forward statistical model. This improvement, combined with a better-tuned set of remaining ad hoc alignment scoring parameters improves the overall alignment quality on the data sets used for assessing PaPaRa 1.0. While the new algorithm performs slightly worse than PaPaRa 1.0 on a small number of data sets, it now consistently outperforms HMMALIGN. Performance-wise, the slightly more complex scoring scheme of PaPaRa 2.0 is alleviated by the new vectorized alignment core. Despite its high time complexity, PaPaRa 2.0 scales well to large data sets. PaPaRa 2.0 can also be executed on multi-core systems. We recently also ported the alignment kernel of PaPaRa 1.0 to general purpose graphics processing units (GPUs) using OpenCL. By combining the GPU implementation with a vectorized *and* multi-threaded CPU implementation, we designed a hybrid GPU/CPU system. On a typical desktop system with an Intel core i7-2600 CPU (4 cores) and a NVidia GTX560 GPU, the hybrid approach increases the maximum performance (in terms of cell updates per second) by a factor of 2.4 over the stand-alone vectorized CPU version [6]. Finally, we implemented an easy-to-use GUI frontend for PaPaRa 2.0. The frontend is tightly integrated with the core algorithm, which allows to assess the impact of the alignment parameters on the output alignment almost in real-time (e.g., on an exemplary small data set with 1885 short-reads and 25 reference sequences, computing the alignment takes roughly 3 seconds on a 4 core desktop CPU. Depending on the size of the data set, there can be a considerable delay). This feature of the GUI frontend allows users (i.e., biologists) to interactively tune alignment parameters for their specific data and based on their own expertise.

## 7.2 Future Work

The current version of the EPA has been well tested and supports the broad range of evolutionary models and input-data types available in RAxML. The multi-grain parallel version as well as the additional heuristics, make it suitable for analyzing large data sets. Therefore, there is not much room for further technical improvements. It is also worth noting that, being part of the standard RAxML code base, it can potentially benefit from future improvements in RAxML. This has already been the case, with the SSE3 vectorization described in Section 3.4, which was mainly implemented to accelerate the tree search algorithm in RAxML. Similarily, there are currently ongoing efforts to further accelerate RAxML using more modern 256-bit wide SIMD instruction sets like AVX and FMA, as well as by using GPUs and FPGSs.

PaPaRa, on the other hand, is in a more experimental state, and offers more room for future improvements. Currently, PaPaRa can generate several 'candidate' alignments per QS (it can happen that the current method generates multiple alignments with equal scores per QS because of the discrete scoring scheme). All of those different per-QS alignments with identical scores for a single QS could be passed as input to the EPA, to select the best alignment by its placement score. An interesting topic to explore will be to assess methods that rely on a purely probabilistic representation of the ancestral sequence profiles. The probabilistic gap-signal in PaPaRa 2.0 shows that such a probabilistic representation can increase accuracy. More importantly, this would allow for removing the ad-hoc scoring parameters and potentially yield the algorithm more adaptive to different data sets and indel rates, without any need for parameter tuning. The current default parameters used in PaPaRa may be biased towards good performance on small subunit rRNA MSAs, which is the most abundant gene in our collection of test data sets. In the tradition of BLAST and other widely-used alignment programs, the scoring parameters are user-defined parameters, which can and *should* be adapted to the data set at hand. Nonetheless, designing a parameter-free, adaptive alignment extension tool represents a challenging task.

Finally, simultaneously estimating the tree topology and the multiple sequence alignment remains one of the big goals in computational molecular evolution. Tighter integration of the EPA and PaPaRa could provide an important step towards addressing this problem: Using PaPaRa to align new sequences against and existing tree/MSA pair followed by an EPA placement can already be used for incremental extension of a phylogeny and an alignment. This represents a straight forward way to perpetually update a large phylogeny with new sequence data. Optionally, standard global tree optimization strategies (e.g., using SPR moves) could periodically be applied to the comprehensive tree, to prevent the greedy extension algorithm from 'getting trapped' in local maxima of the likelihood space. Such an algorithm could also keep track of multiple alternative alignments and tree topologies.

# List of Figures

# List of Tables

# List of Acronyms

| | |
|---|---|
| AS | - Ancestral State |
| BS | - Bootstrap Support |
| CGAP | - Constant Gap |
| CQS | - Candidate Query Sequence |
| ELW | - Expected Likelihood Weights |
| EPA | - Evoluationary Placement Algorithm |
| FPGA | - Field Programmable Gate Array |
| GPL | - GNU General Public License |
| GTR | - Generalized Time Reversible |
| HMM | - Hidden Markov Model |
| LSR | - Lazy Subtree Rearrangement |
| ML | - Maximum Likelihood |
| MPI | - Message Passing Interface |
| MP | - Maximum Parsimony |
| MSA | - Multiple Sequence Alignment |
| ND | - Node Distance |
| NED% | - Normalized Edge Distance (percentage) |
| NGS | - Next Generation Sequencing |
| NNI | - Nearest Neighbor Interchange |
| PaPaRa | - Phylogeny Aware Parsimony based Short Read Alignment |
| PFG | - Parallel Fine-Grain |
| PLK | - Phylogenetic Likelihood Kernel |
| PMG | - Parallel Multi-Grain |
| QS | - Query Sequence |
| RA | - Reference Alignment |
| RF | - Robinson-Foulds (Distance) |
| RS | - Reference Sequence |
| RT | - Reference Tree |
| SIMD | - Single Instruction Multiple Data |
| SPR | - Subtree Pruning and Regrafting |

# Bibliography

[1] Intel Corporation. Intel 64 and IA-32 Architectures Software Developer's Manual. Volume 1: Basic Architecture; http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html (visited june 2012).

[2] Khronos OpenCL Working Group. The OpenCL Specification. http://www.khronos.org/registry/cl/specs/opencl-1.2.pdf (visited june 2012).

[3] F. Ababneh, L. S. Jermiin, C. Ma, and J. Robinson. Matched-pairs tests of homogeneity with applications to homologous nucleotide sequences. *Bioinf.*, 22:1225–1231, 2006.

[4] N. Alachiotis, **S. A. Berger**, and A. Stamatakis. Efficient PC-FPGA Communication over Gigabit Ethernet. In *2010 10th IEEE International Conference on Computer and Information Technology*, pages 1727–1734. IEEE, June 2010.

[5] N. Alachiotis, **S. A. Berger**, and A. Stamatakis. Accelerating Phylogeny-Aware Short DNA Read Alignment with FPGAs. In *2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines*, pages 226–233. IEEE, May 2011.

[6] N. Alachiotis, **S. A. Berger**, and A. Stamatakis. Coupling SIMD and SIMT Architectures to Boost Performance of a Phylogeny-aware Alignment Kernel. *BMC Bioinformatics*, 13:196, 2012.

[7] S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic acids research*, 25(17):3389–402, Sept. 1997.

[8] F. E. Angly, D. Willner, A. Prieto-Davó, R. A. Edwards, R. Schmieder, R. Vega-Thurber, D. A. Antonopoulos, K. Barott, M. T. Cottrell,

C. Desnues, E. A. Dinsdale, M. Furlan, M. Haynes, M. R. Henn, Y. Hu, D. L. Kirchman, T. McDole, J. D. McPherson, F. Meyer, R. M. Miller, E. Mundt, R. K. Naviaux, B. Rodriguez-Mueller, R. Stevens, L. Wegley, L. Zhang, B. Zhu, and F. Rohwer. The GAAS metagenomic tool and its estimations of viral and microbial average genome size in four major biomes. *PLoS computational biology*, 5(12):e1000593, Dec. 2009.

[9] S. Balzer, K. Malde, A. Lanzén, A. Sharma, and I. Jonassen. Characteristics of 454 pyrosequencing data–enabling realistic simulation with flowsim. *Bioinformatics (Oxford, England)*, 26(18):i420–5, Sept. 2010.

[10] R. M. D. Beck, H. Godthelp, V. Weisbecker, M. Archer, and S. J. Hand. Australia's Oldest Marsupial Fossils and their Biogeographical Implications. *PLoS ONE*, 3(3):e1858+, Mar. 2008.

[11] **S. A. Berger**, N. Alachiotis, and A. Stamatakis. An Optimized Reconfigurable System for Computing the Phylogenetic Likelihood on DNA Data. In *IEEE RAW workshop (in conjunction with IPDPS 2012)*, 2012.

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

[13] **S. A. Berger** and A. Stamatakis. PaPaRa 2.0: A Vectorized Algorithm for Probabilistic Phylogeny-Aware Alignment Extension; Exelixis-RRDR-2012-5; http://sco.h-its.org/exelixis/pubs/Exelixis-RRDR-2012-5.pdf. Technical report.

[14] **S. A. Berger** and A. Stamatakis. Accuracy and Performance of Single versus Double Precision Arithmetics for Maximum Likelihood Phylogeny Reconstruction. In *Proceedings of PBC09, Parallel Biocomputing Workshop*. Springer LNCS, 2009.

[15] **S. A. Berger** and A. Stamatakis. Accuracy of Morphology-based Phylogenetic Fossil Placement under Maximum Likelihood. In *Proceedings of IEEE/ACS International Conference on Computer Systems and Applications (AICCSA-10); Hammamet, Tunisia, 2010*, May 2010.

[16] **S. A. Berger** and A. Stamatakis. Assessment of barrier implementations for fine-grain parallel regions on current multi-core architectures.

In *2010 IEEE International Conference On Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS)*, pages 1–8. IEEE, Sept. 2010.

[17] **S. A. Berger** and A. Stamatakis. Aligning short Reads to Reference Alignments and Trees. *Bioinformatics (Oxford, England)*, 27(15):2068–2075, June 2011.

[18] **S. A. Berger**, A. Stamatakis, and R. Lücking. Morphology-based phylogenetic binning of the lichen genera Allographa and Graphis via molecular site weight calibration; Exelixis-RRDR-2010-6; http://sco.h-its.org/exelixis/pubs/Exelixis-RRDR-2010-6.pdf. Technical report.

[19] **S. A. Berger**, A. Stamatakis, and R. Lücking. Morphology-based phylogenetic binning of the lichen genera Graphis and Allographa (Ascomycota: Graphidaceae) using molecular site weight calibration. *Taxon*, 60(5):8, 2011.

[20] O. R. P. Bininda-Emonds, S. G. Brady, M. J. Sanderson, and J. Kim. Scaling of accuracy in extremely large phylogenetic trees. In *Pacific Symposium on Biocomputing 2001 (R. B. Altman, A. K. Dunker, L. Hunter, K. Lauderdale, and T. E. Klein eds.). World Scientific, River Edge, New Jersey*, pages 547–558, 2000.

[21] A. Brady and S. L. Salzberg. Phymm and PhymmBL: metagenomic phylogenetic classification with interpolated Markov models. *Nature Methods*, 6:673–676, 2009.

[22] S. Brakmann. Single-molecule analysis: A ribosome in action. *Nature*, 464(7291):987–8, Apr. 2010.

[23] S. Chakravorty, D. Helb, M. Burday, N. Connell, and D. Alland. A detailed analysis of 16S ribosomal RNA gene segments for the diagnosis of pathogenic bacteria. *J. Microbiol. Meth.*, 69(2):330–339, 2007.

[24] B. Chor and T. Tuller. Maximum likelihood of evolutionary trees: hardness and approximation. *Bioinformatics*, 21(1):97–106, 2005.

[25] T. Z. DeSantis, P. Hugenholtz, K. Keller, E. L. Brodie, N. Larsen, Y. M. Piceno, R. Phan, and G. L. Andersen. NAST: a multiple sequence alignment server for comparative analysis of 16S rRNA genes. *Nucleic Acids. Res.*, 34(Web Server issue):W394—-399, 2006.

[26] C. W. Dunn, A. Hejnol, D. Q. Matus, K. Pang, W. E. Browne, S. A. Smith, E. Seaver, G. W. Rouse, M. Obst, G. D. Edgecombe, M. V. Sorensen, S. H. D. Haddock, A. Schmidt-Rhaesa, A. Okusu, R. M. Kristensen, W. C. Wheeler, M. Q. Martindale, and G. Giribet. Broad phylogenomic sampling improves resolution of the animal tree of life. *Nature*, 452(7188):745–749, 2008.

[27] S. Eddy. Profile hidden Markov models. *Bioinformatics*, 14(9):755–763, 1998.

[28] R. C. Edgar. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucl. Acids Res.*, 32(5):1792–1797, 2004.

[29] M. Farrar. Striped Smith-Waterman speeds database searches six times over other SIMD implementations. *Bioinformatics (Oxford, England)*, 23(2):156–61, Jan. 2007.

[30] J. S. Farris. Phenetics in camouflage. *Cladistics*, 6(1):91–100, 1990.

[31] J. Felsenstein. Evolutionary trees from DNA sequences: a maximum likelihood approach. *J. Mol. Evol.*, 17:368–376, 1981.

[32] J. Felsenstein. Confidence limits on phylogenies: An approach using the bootstrap. *Evolution*, 39(4):783–791, 1985.

[33] N. Fierer, M. Hamady, C. L. Lauber, and R. Knight. The influence of sex, handedness, and washing on the diversity of hand surface bacteria. *Proc. Natl. Acad. Sci. USA*, 105(46):17994–17999, 2008.

[34] W. M. Fitch. Toward Defining the Course of Evolution: Minimum Change for a Specific Tree Topology. *Syst Biol*, 20(4):406–416, Dec. 1971.

[35] W. M. Fitch and E. Margoliash. Construction of phylogenetic trees. *Science*, 155(3760):279–284, 1967.

[36] N. Galtier and M. Gouy. Inferring pattern and process: Maximum-likelihood implementation of a nonhomogeneous model of DNA sequence evolution for phylogenetic analysis. *Mol. Biol. Evol.*, 15:871–879, 1998.

[37] L. Ganzert, G. Jurgens, U. Munster, and D. Wagner. Methanogenic communities in permafrost-affected soils of the Laptev Sea coast, Siberian Arctic, characterized by 16S rRNA gene fingerprints. *FEMS Microbiol. Ecol.*, 59(2):476–488, 2007.

[38] O. Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162(3):705–708, Dec. 1982.

[39] R. W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 26:147–160, 1950.

[40] M. Han and C. Zmasek. phyloXML: XML for evolutionary biology and comparative genomics. *BMC Bioinformatics*, 10(1):356, 2009.

[41] D. M. Hillis and J. J. Bull. An empirical test of bootstrapping as a method for assessing confidence in phylogenetic analysis. *Syst. Biol.*, 42(2):182, 1993.

[42] J. W. K. Ho, C. E. Adams, J. B. Lew, T. J. Matthews, C. C. Ng, A. Shahabi-Sirjani, L. H. Tan, Y. Zhao, S. Easteal, S. R. Wilson, and L. S. Jermiin. SeqVis: Visualization of compositional heterogeneity in large alignments of nucleotides. *Bioinf.*, 22:2162–2163, 2006.

[43] X. HUANG. A contig assembly program based on sensitive detection of fragment overlaps. *Genomics*, 14(1):18–25, Sept. 1992.

[44] D. Hudson, A. Auch, J. Qi, and S. Schuster. MEGAN analysis of metagenomic data. *Genome Res.*, 17:377–386, 2007.

[45] D. H. Huson, R. Rupp, and C. Scornavacca. *Phylogenetic Networks.* Cambridge University Press, Cambridge, 2010.

[46] V. Jayaswal, J. Robinson, and L. S. Jermiin. Estimation of phylogeny and invariant sites under the general markov model of nucleotide sequence evolution. *Syst. Biol.*, 56(2):155–162, 2007.

[47] L. S. Jermiin, S. Y. W. Ho, F. Ababneh, J. Robinson, and A. W. D. Larkum. The biasing Effect of compositional Heterogeneity on phylogenetic Estimates may be underestimated. *Syst. Biol.*, 53(4):638–643, 2004.

[48] T. Jukes and C. Cantor. *Evolution of protein molecules.*, chapter III, pages 21–132. Academic Press, New York, 1969.

[49] J. Karow. Survey: Illumina, SOLiD, and 454 Gain Ground in Research Labs; Most Users Mull Additional Purchases; http://www.genomeweb.com/sequencing/survey-illumina-solid-and-454-gain-ground-research-labs-most-users-mull-addition, 2010.

[50] J. J. Kasianowicz, E. Brandin, D. Branton, and D. W. Deamer. Characterization of individual polynucleotide molecules using a membrane channel. *Proceedings of the National Academy of Sciences of the United States of America*, 93(24):13770–3, Nov. 1996.

[51] K. Katoh, K. Kuma, H. Toh, and T. Miyata. MAFFT version 5: improvement in accuracy of multiple sequence alignment. *Nucl. Acids Res.*, 33:511–518, 2005.

[52] A. Kluge and J. Farris. Quantitative phyletics and the evolution of anurans. *Syst. Zool.*, 18:1–32, 1969.

[53] L. B. Koski and G. B. Golding. The closest BLAST hit is often not the nearest neighbor. *J. Mol. Evol.*, 52(6):540–542, 2001.

[54] V. W. Lee, P. Hammarlund, R. Singhal, P. Dubey, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, and S. Chennupaty. Debunking the 100X GPU vs. CPU myth. In *Proceedings of the 37th annual international symposium on Computer architecture - ISCA '10*, volume 38, page 451, New York, New York, USA, June 2010. ACM Press.

[55] P. O. Lewis. A likelihood approach to estimating phylogeny from discrete morphological character data. *Systematic Biology*, 50(6):913–925, 2001.

[56] R. E. Ley, F. Bäckhed, P. Turnbaugh, C. A. Lozupone, R. D. Knight, and J. I. Gordon. Obesity alters gut microbial ecology. *Proc. Natl. Acad. Sci. USA*, 102(31):11070–11075, Aug. 2005.

[57] R. E. Ley, J. K. Harris, J. Wilcox, J. R. Spear, S. R. Miller, B. M. Bebout, J. A. Maresca, D. A. Bryant, M. L. Sogin, and N. R. Pace. Unexpected diversity and complexity of the Guerrero Negro hypersaline microbial mat. *Appl. Environ. Microbiol.*, 72(5):3685–3695, May 2006.

[58] R. E. Ley, C. A. Lozupone, M. Hamady, R. D. Knight, and J. I. Gordon. Worlds within worlds: evolution of the vertebrate gut microbiota. *Nat. Rev. Microbiol.*, 6(10):776–788, 2008.

[59] A. Loytynoja and N. Goldman. Phylogeny-aware gap placement prevents errors in sequence alignment and evolutionary analysis. *Science*, 320(5883):1632, 2008.

[60] C. Lozupone and R. Knight. UniFrac: a new phylogenetic method for comparing microbial communities. *Appl. Environ. Microbiol.*, 71(12):8228–8235, 2005.

[61] W. Ludwig, O. Strunk, R. Westram, L. Richter, H. Meier, Yadhukumar, A. Buchner, T. Lai, S. Steppi, G. Jobb, W. Forster, I. Brettske, S. Gerber, A. W. Ginhart, O. Gross, S. Grumann, S. Hermann, R. Jost, A. Konig, T. Liss, R. Lussmann, M. May, B. Nonhoff, B. Reichel, R. Strehlow, A. Stamatakis, N. Stuckmann, A. Vilbig, M. Lenke, T. Ludwig, A. Bode, and K.-H. Schleifer. ARB: a software environment for sequence data. *Nucl. Acids Res.*, 32(4):1363–1371, Feb. 2004.

[62] P. S. Manos, P. S. Soltis, D. E. Soltis, S. R. Manchester, S.-H. Oh, C. D. Bell, D. L. Dilcher, and D. E. Stone. Phylogeny of Extant and Fossil Juglandaceae Inferred from the Integration of Molecular and Morphological Data Sets. *Systematic Biology*, 56(3):412–430, 2007.

[63] F. Matsen, R. Kodner, and E. V. Armbrust. pplacer: linear time maximum-likelihood and Bayesian phylogenetic placement of sequences onto a fixed reference tree. *BMC Bioinformatics*, 11(1):538, 2010.

[64] A. C. McHardy, H. G. Martin, T. Aristotelis, P. Hugenholtz, and I. Rigoutsos. Accurate phylogenetic classification of variable-length DNA fragments. *Nature Methods*, 4:63–72, 2007.

[65] S. Mirarab, N. Nguyen, and T. Warnow. SEPP: SATe-Enabled Phylogenetic Placement. In *Proceedings of the 2012 Pacific Symposium on Biocomputing (PSB 2012).*, pages 247–258, 2012.

[66] B. M. E. Moret, U. Roshan, and T. Warnow. Sequence-Length Requirements for Phylogenetic Methods. In *Proceedings of 2nd Intl Workshop on Algorithms in Bioinformatics (WABI 02) 2002 (R. Guigo and D. Gusfield eds.). Lecture Notes in Computer Science 2452, Springer*, pages 343–356, 2002.

[67] K. Munch, W. Boomsma, J. P. Huelsenbeck, E. Willerslev, and R. Nielsen. Statistical Assignment of DNA Sequences Using Bayesian Phylogenetics. *Syst Biol*, 57(5):750–757, Oct. 2008.

[68] R. Nielsen and M. Matz. Statistical approaches for DNA barcoding. *Syst. Biol.*, 55(1):162–169, 2006.

[69] G. A. Petsko. Rising in the East. *Genome biology*, 11(1):102, Jan. 2010.

[70] K. D. Pruitt, T. Tatusova, and D. R. Maglott. NCBI reference sequences (RefSeq): a curated non-redundant sequence database of genomes, tanscrips and proteins. *Nucleic Acids. Res.*, 35 (Database Issue):D61–D65, 2007.

[71] D. F. Robinson and L. R. Foulds. Comparison of phylogenetic trees. *Math. Biosci.*, 53(1-2):131–147, 1981.

[72] T. Rognes. Faster Smith-Waterman database searches with inter-sequence SIMD parallelisation. *BMC bioinformatics*, 12(1):221, Jan. 2011.

[73] M. Ronaghi. Pyrosequencing sheds light on DNA sequencing. *Genome Res.*, 11(1):3–11, 2001.

[74] M. J. Sanderson, M. J. Donoghue, W. Piel, and T. Eriksson. TreeBASE: a prototype database of phylogenetic analyses and an interactive tool for browsing the phylogeny of life. *American Journal of Botany*, 81(6):183, 1994.

[75] D. Sankoff. Minimal mutation trees of sequences. *SIAM. J. Appl. Math.*, 28:35–42, 1975.

[76] M. Seeland, **S. A. Berger**, A. Stamatakis, and S. Kramer. Parallel structural graph clustering. In *ECML PKDD'11 Proceedings of the 2011 European conference on Machine learning and knowledge discovery in databases - Volume Part III*, pages 256–272, Sept. 2011.

[77] T. Smith. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, Mar. 1981.

[78] A. Stamatakis. Phylogenetic models of rate heterogeneity: a high performance computing perspective. In *Proceedings of 20th IEEE/ACM International Parallel and Distributed Processing Symposium (IPDPS2006); Rhodos, Greece. 2006*, volume 0, page 278, Los Alamitos, CA, USA, Apr. 2006. IEEE Computer Society.

[79] A. Stamatakis. RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics*, 22(21):2688–2690, 2006.

[80] A. Stamatakis, A. J. Aberer, C. Goll, S. A. Smith, **S. A. Berger**, and F. Izquierdo-Carrasco. RAxML-Light: A Tool for computing TeraByte Phylogenies. *Bioinformatics*, 28(15):2064–2066, May 2012.

[81] A. Stamatakis, P. Hoover, and J. Rougemont. A Rapid Bootstrap Algorithm for the RAxML Web Servers. *Syst. Biol.*, 57(5):758–771, 2008.

[82] A. Stamatakis, Z. Komornik, and **S. A. Berger**. Evolutionary Placement of Short Sequence Reads on Multi-Core Architectures. In *Proceedings of IEEE/ACS International Conference on Computer Systems and Applications (AICCSA-10); Hammamet, Tunisia, 2010*, May 2010.

[83] A. Stamatakis, T. Ludwig, and H. Meier. RAxML-III: A fast program for maximum likelihood-based inference of large phylogenetic trees. *Bioinformatics*, 21(4):456–463, 2005.

[84] A. Stamatakis and M. Ott. Efficient computation of the phylogenetic likelihood function on multi-gene alignments and multi-core architectures. *Phil. Trans. R. Soc. series B, Biol. Sci.*, 363:3977–3984, 2008.

[85] M. Stark, **S. A. Berger**, A. Stamatakis, and C. von Mering. ML-TreeMap - accurate Maximum Likelihood placement of environmental DNA sequences into taxonomic and functional reference phylogenies. *BMC Genomics*, 11(1), Aug. 2010.

[86] K. Strimmer and A. Rambaut. Inferring confidence sets of possibly misspecified gene trees. *Proceedings of the Royal Society B: Biological Sciences*, 269(1487):137–142, 2002.

[87] S. Struckmann, M. J. Arauzo-Bravo, H. R. Schoeler, R. A. Reinbold, and G. Fuellen. ReXSpecies - a tool for the analysis of the evolution of generegulation across species. *BMC evolutionary biology*, 8:111+, Apr. 2008.

[88] S. Tavaré. Some probabilistic and statistical problems in the analysis of DNA sequences. *Some mathematical questions in biology-DNA sequence analysis*, 17:57–86, 1986.

[89] K. Thiele. The Holy Grail of the Perfect Character: The Cladistic Treatment of Morphometric Data. *Cladistics*, 9(3):275–304, 1993.

[90] J. L. Thorne, H. Kishino, and J. Felsenstein. Inching toward reality: An improved likelihood model of sequence evolution. *J. Mol. Evol.*, 34(1):3–16, 1992.

[91] P. J. Turnbaugh, M. Hamady, T. Yatsunenko, B. L. Cantarel, A. Duncan, R. E. Ley, M. L. Sogin, W. J. Jones, B. A. Roe, J. P. Affourtit,

and Others. A core gut microbiome in obese and lean twins. *Nature*, 457(7228):480–484, 2008.

[92] 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.

[93] J. Wiens. Paleontology, Genomics, and Combined-Data Phylogenetics: Can Molecular Data Improve Phylogeny Estimation for Fossil Taxa? *Systematic Biology*, 58(1):87, 2009.

[94] J. Wiens, J. Fetzner, C. Parkinson, and T. Reeder. Hylid Frog Phylogeny and Sampling Strategies for Speciose Clades. *Systematic Biology*, 54(5):719–748, Oct. 2005.

[95] J. J. Wiens. Character analysis in morphological phylogenetics: problems and solutions. *Systematic Biology*, 50(5):689–699, 2001.

[96] J. J. Wiens, R. Bonett, and P. Chippindale. Ontogeny Discombobulates Phylogeny: Paedomorphosis and Higher-Level Salamander Relationships. *Systematic Biology*, 54(1):91–110, Feb. 2005.

[97] Z. Yang. Estimating the pattern of nucleotide substitution. *Journal of Molecular Evolution*, 39(1):105–11, July 1994.

[98] Z. Yang. Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites. *J. Mol. Evol.*, 39:306–314, 1994.

[99] Z. Yang. *Computational Molecular Evolution*. Oxford University Press, Oxford, 2006.

[100] A. Zharkikh. Estimation of evolutionary distances between nucleotide sequences. *Journal of molecular evolution*, 39(3):315–29, Sept. 1994.

## Simon Berger (Dipl.-Bioinf. Univ.)

| | |
|---|---|
| Address: | The Exelixis Lab,<br>Scientific Computing Group,<br>Heidelberg Institute for Theoretical Studies<br><br>Schloss-Wolfsbrunnenweg 35, D-69118, Heidelberg |
| WWW: | http://exelixis-lab.org/ |
| Phone: | +49-89-289-17338 (office)<br>+49-170-5293442 (mobile) |
| E-Mail: | simberger@gmail.com |
| Skype: | simberger |

## Education:

| | |
|---|---|
| 01/2009- Present | Ph.D. Student at the The Exelixis Lab (Exelixis Lab located at Technical University Munich 2009-2010 and moved to Heidelberg Institute for Theoretical Studies 2011-2012)<br><br>Project Working title: “Simultaneous Tree building and Alignment” (Prof. Dr. Alexandros Stamatakis) |
| 10/2003- 8/2008 | Study of Bioinformatics at the joint Bioinformatics Program of the LMU and TU Munich, Germany.<br>Diploma thesis: “Protonation State and Enzyme/Substrate Interaction” (Dr. Joannis Apostolakis, Prof. Dr. Ralf Zimmer), August 2008. [Grade 1.4]. |
| 09/1993- 05/2002 | Gymnasium Ottobrunn, Ottobrunn, Germany. Abitur 2002, specialized courses in English and Physics.<br>[Grade 1.9] |

## Research Activities:

|  |  |
|---|---|
| 01/2009 - present | Research on algorithms for phylogeny-aware alignment and placement of short sequence reads. (RAxML EPA, PaPaRa short read aligner) |
| 01/2012 - present | Collaboration on the GapMis sequence aligner (created the multi-threaded and SSE-vectorized alignment kernel) |
| 11/2009 - present | Research on methods for PC/FPGA communication using Gigabit-Ethernet and UDP/IP |
| 03/2011 - present | Collaboration with TU Munich on parallel structural graph clustering (created the multi-threaded and MPI C++ implementations) |
| 11/2007 - 08/2008 | Diploma project<br>Research on fast calculation of amino-acid protonation states in enzymes and the influence of protonation states on enzyme/substrate interaction. |
| 05/2006 - 04/2007 | Student Assistant, Cheminformatics Group, Chair of Bioinformatics, LMU München.<br>Research on electrostatics calculations for enzyme/substrate complexes. The project included a complete Java implementation of a highly efficient multigrid Poisson solver and protein solvent-accessibility functions. |

## Other activities:

|  |  |
|---|---|
| 1997-2004 | co-development of a 3d game-engine written in C/C++. A technical demonstration including network multiplayer gameplay and original design/artwork was released in 2000. This work helped to develop my interest and skills in general C/C++ programming/optimization, cross-platform programming (linux, win32), 2D/3D graphics (OpenGL, DirectX, X11), low-level network programming (UDP/IP) and sound programming (ALSA, OpenAL). |
| 09/2001 – 02/2002 | School Project as part of specialized physics course<br>Calculation and 3D-visualization of the electrostatic field induced by point-charges using Java and OpenGL. |

## Computer Languages:

C/C++ (including low-level and SSE optimization), Java (including performance optimization of numerical code), Ruby, Perl, JavaScript, Go, OCaml, Ada

## Interests and Hobbies:

Guitar Playing (classical/electric, 7 years of classical lessons), Music Recording/Production, Music Synthesizer Technology, Mountain Biking, Swimming, Alpine Skiing, Programming Languages and Techniques, Computer Games

## Languages:

| | |
|---|---|
| German | Fluent, mother tongue |
| English | Fluent |

Simon Berger, Munich, October 2012