Models, Optimizations, and Tools for Large-Scale Phylogenetic Inference, Handling Sequence Uncertainty, and Taxonomic Validation

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

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

genehmigte

Dissertation

von

Oleksii Kozlov

aus Cherson, Ukraine

Tag der mündlichen Prüfung:	17.01.2018
Erster Gutachter:	Prof. Dr. Alexandros Stamatakis
Zweiter Gutachter:	Prof. Dr. David Posada

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.

Heidelberg, 21.11. 2017 ________________________

Oleksii Kozlov

Zusammenfassung

Das Konzept der Evolution ist in der modernen Biologie von zentraler Bedeutung. Deswegen liefert die *Phylogenetik*, die Lehre über die Verwandschaften und Abstammung von Organismen bzw. Spezies, entscheidende Hinweise zur Entschlüsselung einer Vielzahl biologischer Prozesse. Phylogenetische Stammbäume sind einerseits für die Grundlagenforschung wichtig, da sie in Studien über die Diversifizierung und Umweltanpassung einzelner Organismengruppen (z.B., Insekten oder Vögel) bis hin zu der großen Herausforderung, die Entstehung und Entwicklung aller Lebensformen in einem umfassenden evolutionären Baum darzustellen (der sog. *Tree of Life*) Anwendung finden. Andererseits werden phylogenetische Methoden auch in praxisnahen Anwendungen eingesetzt, um beispielsweise die Verbreitungsdynamik von HIV-Infektionen oder, die Heterogenität der Krebszellen eines Tumors, zu verstehen.

Den aktuellen Stand der Technik in der Stammbaumrekonstruktion stellen Methoden Maximum Likelihood (ML) und Bayes'sche Inferenz (BI) dar, welche auf der Analyse molekularer Sequenzendaten (DNA und Proteine) anhand probabilistischer Evolutionsmodelle basieren. Diese Methoden weisen eine hohe Laufzeitkomplexität auf (NP-schwer), welche die Entwicklung effizienter Heuristiken unabdingbar macht. Hinzu kommt, dass die Berechnung der Zielfunktion (sog. Phylogenetic Likelihood Function, PLF) neben einem hohen Speicherverbrauch auch eine Vielzahl an Gleitkommaarithmetik-Operationen erfordert und somit extrem rechenaufwendig ist.

Die neuesten Entwicklungen im Bereich der DNA-Sequenzierung (Next Generation Sequencing, NGS) steigern kontinuierlich den Durchsatz und senken zugleich die Sequenzierungskosten um ein Vielfaches. Für die Phylogenetik hat dies zur Folge, dass die Dimensionen der zu analysierenden Datensätze alle 2–3 Jahre, um eine Grössenordnung zunhemen. War es bisher üblich, einige Dutzend bis Hunderte Spezies anhand einzelner bzw. weniger Gene zu analysieren (Sequenzlänge: 1–10 Kilobasen), stellen derzeit Studien mit Tausenden Sequenzen *oder* Genen keine Seltenheit mehr dar. In den nächsten 1–2 Jahren ist zu erwarten, dass die Analysen Tausender bis Zehntausender vollständiger Genome bzw. Transkriptome (Sequenzlänge: 1–100 Megabasen und mehr) anstehen. Um diesen Aufgaben gewachsen zu sein, müssen die bestehenden Methoden weiterentwickelt und optimiert werden, um vor allem Höchstleistungsrechner sowie neue Hardware-Architekturen optimal nutzen zu können.

Außerdem führt die sich beschleunigende Speicherung von Sequenzen in öffentlichen Datenbanken wie NCBI GenBank (und ihren Derivaten) dazu, dass eine hohe Qualität der Sequenzannotierungen (z. B. Organismus- bzw. Speziesname, taxonomische Klassifikation, Name eines Gens usw.) nicht zwangsläufig gewährleistet ist. Das hängt unter anderem auch damit zusammen, dass eine zeitnahe Korrektur durch entsprechende Experten nicht mehr möglich ist, solange ihnen keine adäquaten

Software-Tools zur Verfügung stehen.

In dieser Doktroarbeit leisten wir mehrere Beiträge zur Bewältigung der oben genannten Herausforderungen.

Erstens haben wir `ExaML`, eine dedizierte Software zur ML-basierten Stammbaumrekonstruktion für Höchstleistungsrechner, auf den Intel Xeon Phi Hardwarebeschleuniger portiert. Der Xeon Phi bietet im Vergleich zu klassischen x86 CPUs eine höhere Rechenleistung, die allerdings nur anhand architekturspezifischer Optimierungen vollständig genutzt werden kann. Aus diesem Grund haben wir zum einen die PLF-Berechnung für die 512-bit-Vektoreinheit des Xeon Phi umstrukturiert und optimiert. Zum anderen haben wir die in `ExaML` bereits vorhandene reine MPI-Parallelisierung durch eine hybride MPI/OpenMP-Lösung ersetzt. Diese hybride Lösung weist eine wesentlich bessere Skalierbarkeit für eine hohe Zahl von Kernen bzw. Threads innerhalb eines Rechenknotens auf (>100 HW-Threads für Xeon Phi).

Des Weiteren haben wir eine neue Software zur ML-Baumrekonstruktion namens `RAxML-NG` entwickelt. Diese implementiert, bis auf kleinere Anpassungen, zwar denselben Suchalgorithmus wie das weit verbreitete Programm `RAxML`, bietet aber gegenüber `RAxML` mehrere Vorteile: **(a)** dank den sorgfältigen Optimierungen der PLF-Berechnung ist es gelungen, die Laufzeiten um den Faktor 2 bis 3 zu reduzieren **(b)** die Skalierbarkeit auf extrem großen Eingabedatensätzen wurde verbessert, indem ineffiziente topologische Operationen eliminiert bzw. optimiert wurden, **(c)** die bisher nur in `ExaML` verfügbaren, für große Datensätze relevanten Funktionen wie Checkpointing sowie ein dedizierter Datenverteilungsalgorithmus wurden nachimplementiert **(d)** dem Benutzer steht eine größere Auswahl an statistischen DNA-Evolutionsmodellen zur Verfügung, die zudem flexibler kombiniert und parametrisiert werden können **(e)** die Weiterentwicklung der Software wird aufgrund der modularen Architektur wesentlich erleichtert (die Funktionen zur PLF-Berechnung wurden in eine gesonderte Bibliothek ausgeglidert).

Als nächstes haben wir untersucht, wie sich Sequenzierungsfehler auf die Genauigkeit phylogenetischr Stammbaumrekonstruktionen auswirken. Wir modifizieren den `RAxML` bzw. `RAxML-NG` Code dahingehend, dass sowohl die explizite Angabe von Fehlerwahrscheinlichkeiten als auch die automatische Schätzung von Fehlerraten mittels der ML-Methode möglich ist. Unsere Simulationen zeigen: **(a)** Wenn die Fehler gleichverteilt sind, kann die Fehlerrate direkt aus den Sequenzdaten geschätzt werden. **(b)** Ab einer Fehlerrate von ca. 1% liefert die Baumrekonstruktion unter Berücksichtigung des Fehlermodells genauere Ergebnisse als die klassische Methode, welche die Eingabe als fehlerfrei annimmt.

Ein weiterer Beitrag im Rahmen dieser Arbeit ist die Software-Pipeline `SATIVA` zur rechnergestützten Identifizierung und Korrektur fehlerhafter taxonomischer Annotierungen in großen Sequenzendatenbanken. Der Algorithmus funktioniert wie

folgt: für jede Sequenz wird die Platzierung im Stammbaum mit dem höchstmöglichen Likelihood-Wert ermittelt und anschließend geprüft, ob diese mit der vorgegeben taxonomischen Klassifikation übereinstimmt. Ist dies nicht der Fall, wird also eine Sequenz beispielsweise innerhalb einer anderen Gattung platziert, wird die Sequenz als falsch annotiert gemeldet, und es wird eine entsprechende Umklassifizierung vorgeschlagen. Auf simulierten Datensätzen mit zufällig eingefügten Fehlern, erreichte unsere Pipeline eine hohe Identifikationsquote (>90%) sowie Genauigkeit (>95%). Zur Evaluierung anhand empirischer Daten, haben wir vier öffentliche rRNA Datenbanken untersucht, welche zur Klassifizierung von Bakterien häufig als Referenz benutzt werden. Dabei haben wir je nach Datenbank 0.2% bis 2.5% aller Sequenzen als potenzielle Fehlannotierungen identifiziert.

Abstract

Evolution is a central paradigm of current biology, and thus *phylogenetics*, the study of evolutionary relationships between organisms, is essential for understanding biological processes. Phylogenetic trees are as important for fundamental research (e.g, reconstructing the *Tree of Life*) as for numerous practical applications (e.g., studying HIV transmission dynamics or tumor heterogeneity in cancer). State-of-the-art phylogenetic inference methods, such as Maximum Likelihood (ML) and Bayesian Inference (BI), rely on probabilistic models of molecular sequence evolution. This makes them computationally expensive both, in theory (NP-hardness), and in practice (extensive use of floating-point arithmetics and high memory requirements). At the same time, recent advances in DNA sequencing technology have dramatically increased the data generation pace, creating a demand for reconstructing ever larger trees from ever longer sequences (whole genomes or transcriptomes). Additionally, the fast accumulation of sequences in public databases has led to concerns about metadata quality, as human curation often lags behind the data tsunami. In this thesis, we make several contributions which address different aspects of the aforementioned challenges.

First, we adapt `ExaML`, an MPI-parallelized ML inference code for genome-scale alignments, to run efficiently on Intel Xeon Phi hardware accelerators. To this end, we optimize likelihood computations for the 512-bit wide vector unit, and implement a hybrid MPI/OpenMP parallelization approach which can better handle the high level of intranode parallelism that characterizes the Xeon Phi (>100 threads/card).

Then, we introduce `RAxML-NG`, a novel, fast, scalable, and flexible ML tree inference tool. It implements roughly the same basic tree search heuristic as the existing and widely-used program `RAxML`, but offers improvements in accuracy, speed, scalability, and user-friendliness. Moreover, `RAxML-NG` is substantially more flexible with respect to the evolutionary model that can be specified and used. Finally, the code is easier to maintain and extend because of its modular design.

Further, we explore the effects of sequencing error and sequence uncertainty metrics on phylogenetic inference. In simulation, we show that: **(a)** an uniform error rate can be reliably estimated from the molecular input data, and **(b)** using an explicit error model improves the accuracy of phylogenetic inference from noisy data.

Finally, we developed a software pipeline for semi-automatic identification and correction of taxonomically mislabeled sequences based on phylogenetic inference and phylogenetic placement approaches. We evaluate our approach on simulated datasets where it attains high accuracy (>95% precision and >90% recall). Then, we apply it to re-validate the taxonomic labels in four widely-used reference rRNA databases. We find 0.2% to 2.5% potentially mislabeled sequences in those databases.

Acknowledgements

Above all, I want to thank my supervisor, Prof. Dr. Alexandros Stamatakis for his invaluable assistance. He was always extremely supportive in both scientific and non-scientific matters, and I felt lucky to work under his guidance.

I am furthermore grateful to my co-advisor Prof. Dr. David Posada, who kindly agreed to review this thesis. I also appreciate our ongoing collaboration on cancer cell phylogeny inference, and I would like to thank Prof. Posada for hosting me during my research visit to Vigo.

My former and current colleagues Fernando Izquierdo-Carrasco, Jiajie Zhang, Tomas Flouri, Paschalia Kapli, Andre Aberer, Kassian Kobert, Diego Darriba, Lucas Czech, Pierre Barbera, Sarah Lutteropp, Benoit Morel, Rudolf Biczok and Dora Serdari were always friendly, collaborative and willing to share their knowledge. I always enjoyed spending my time with them, be it in the lab, on the Neckarwiese, or at one of the inventive cocktail parties organized by Lucas. I was also happy to share my workplace and to exchange ideas with our short- and long-term visitors Mark Holder, Emily Jane McTavish, Rebecca Harris, Nikos Psonis, Khouloud Madhbouh, and Laura Rubinat. I am grateful to my collaborators Pelin Yilmaz, Karen Meusemann, Ralph Peters, Oliver Niehuis, Manuela Sann, Sara Bank, Oliver Ratmann, and Micah Dunthorn for productive teamwork and interesting discussions. Furthermore, I would like to express my gratitude to Nick Goldman, Adam Leache and Deren Eaton for their helpful hints with respect to sequence uncertainty modeling.

Beyond work, I am sincerely thankful to all my friends from Karlsruhe, Heidelberg and Ladenburg, who accompanied me in countless leisure activities during these four years. In particular, I would like to acknowledge Felix and Dasha for being my faithful team fellows in the ’What?Where?When?’ quiz game, and also for giving me (maybe a bit too) ample diversion opportunities during the final writing phase. Speaking of which, nothing helped me to concentrate on the writing like the music of *God is an Astronaut*. And *Scar Symmetry* (among many other great metal bands) yielded a speedup of at least 2× for coding and cluster job submission tasks.

Finally and importantly, I am grateful to the *Klaus Tschira Foundation* for funding my position, and to the *Heidelberg Institute for Theoretical Studies* for providing an excellent work environment.

Contents

Chapter 1

Introduction

1.1 Motivation

Phylogenetic trees represent evolutionary relationship between biological species. They play an important role in both basic [62, 92, 163] and applied research [43, 51, 125]. Nowadays, phylogenetic tree inference is mainly based on molecular data (DNA and protein sequences), which generally provide a substantially more reliable evolutionary signal compared than phenotypic traits [53]. State-of-the-art tree reconstruction methods such as Maximum Likelihood (ML) and Bayesian Inference (BI), employ a probabilistic model of sequence evolution. They rely on computing the so-called phylogenetic likelihood function (PLF), which is both, compute-, and memory-intensive.

Thanks to the recent technological advances, the throughput of DNA sequencing has dramatically increased, while prices have dramatically decreased at the same time. In phylogenetics, this resulted in a shift from classical marker-based studies involving one or few conserved genes towards *phylogenomics*, that is, using whole-transcriptome or whole-genome to infer phylogenies. For instance, the 1KITE project (`http://1kite.org/`) we are involved in, aims to reconstruct the evolutionary history of more than 1500 phylogenetically diverse insect species from transcriptomic data. This entails the analysis of huge datasets comprising thousands of genes and millions of aminoacid and/or nucleotide characters. Similar, even more ambitious projects, are underway for other organism groups such as birds [1], plants [101], or bacteria [78]. Recently, plans have been announced [105] to sequence the full genomes of *all* ≈1.7 million eukaryotic species that have been described to date.

Clearly, analyzing these enormous data volumes requires excessive computing resources, which turns computational phylogenomics into an emerging supercomputing application area. Hence, tree reconstruction software has to be optimized for emerging hardware architectures and scale on large cluster systems as well as supercomputers. In particular, parallelization is becoming increasingly important not only for supercomputers, but also for desktop systems. Since the early 2000s, CPU clock rates have been stalling after hitting the *power wall*. This means, that although the number of transistors per chip is still increasing according to the Moore's law, performance improvements in the new CPU generations now mainly stem from the increased level of parallelism. Modern processor chips comprise multiple cores and make extensive use of data-level parallelism (often in form of *SIMD*, *Single Instruction Multiple Data*). For instance, the latest Intel *Skylake* CPUs feature up to 28 cores and support *AVX512* SIMD instructions, which operate on 512-bit vectors and can thus process 16 single-precision floating point numbers in one cycle. This constitutes a remarkable increase in parallelism compared to, for instance, the single-core Pentium III (released in 1999) that supported a 128-bit wide *SSE* vector instruction set. Specialized hardware accelerators such as GPUs and the Intel *Xeon Phi*s exhibit even higher core numbers and/or vector unit widths. Hence, phylogetic inference codes have to be adapted to such architectures to fully leverage the capabilities of modern hardware.

Another direct result of the sequence data avalanche is the challenge to maintain the correctness of annotations in public sequence databases such as NCBI GenBank and its more specialized derivatives (e.g., SILVA [109]). In particular, the organism name and its taxonomic affiliation represent important metadata fields as they are used to classify unknown sequences in downstream applications (e.g., in metagenetic studies). However, the quality of these taxonomic annotations can be compromised due to multiple reasons that range from human error (sample misidentification, misspelling) to inherent problems of modern taxonomy (synonyms, phylogenetically inconsistent ranks). With growing database sizes, manual re-validation and correction becomes less and less feasible. This calls for the development of (semi-)automatic methods to support taxonomic curation, which are do currently not exist, albeit there are a few notable exceptions [2, 90, 119].

1.2 Contribution and Overview

In this thesis, we make several contributions to the field of computational phylogenetics that address some of the challenges outlined above.

Firstly, we optimize PLF kernels for the Intel Xeon Phi hardware accelerators. Subsequently, we integrate the optimized kernels in `ExaML`, a ML-based phylogenetic inference tool for analyzing genome-scale datasets. The results of this work have been presented in the HICOMB2014 workshop (Phoenix, AZ, USA), and published in the respective conference proceedings [73]. version 3 of `ExaML` that includes Xeon Phi support was described in a *Bioinformatics* application note [72].

Further, we introduce RAxML-NG, a complete re-design of a widely-used ML inference tool RAxML [133]. RAxML-NG offers improvements in accuracy, speed, flexibility, user-friendliness, and integrates several HPC-related features previously only available in ExaML. Most importantly, RAxML-NG is much easier to support and extend thanks to its modular structure. In particular, the functionality shared between RAxML-NG and other phylogenetic tools developed in our group, has been encapsulated in two C libraries: libpll (PLF kernels and other low-level routines) and pll-modules (high-level routines for numerical optimization, tree moves etc.). This substantially simplifies the development of new features as well as optimizations, and facilitates faster integration into the relevant applications. The development of these libraries represents a separate project that was led by Tomas Flouris and Diego Darriba, although the author of this thesis made substantial contributions to both libraries (see **Section** 4).

Next, we extend the RAxML and RAxML-NG codes by novel methods to handle sequence uncertainty and sequence errors. Under simulation, we find that an uniform error rate can be reliably estimated from the sequence data using ML optimization methods. Furthermore, we show that, accounting for sequence uncertainty can improve the accuracy of phylogenetic inference. Then, we propose and evaluate a dedicated 2-parameter error model for diploid genotype data, which accounts for both, sequencing error (nucleotide substitutions), and so-called *allelic dropout* (failure to amplify one of the alleles). On simulated datasets, we show that, incorporating this error model into the RAxML-NG search algorithm yields more accurate trees than using the standard, error-agnostic evolutionary models.

A further contribution of this thesis is a software pipeline SATIVA for identification and correction of taxonomically mislabeled sequences in large databases. This tool yields savings in time and human labour by short-listing suspicious sequences warranting further investigation. We used SATIVA to re-validate the taxonomic labels in four widely-used rRNA marker databases (Greengenes [90], LTP [161], RDP [24], and SILVA [109]). The results of this analysis were published [74] in *Nucleic Acids Research*. The SILVA database curators, who co-authored this paper, used our findings to improve taxonomic annotations in their database.

All aforementioned software tools (ExaML, RAxML-NG, and SATIVA) are open-source and publicly available on GitHub. It is worth noting that, a beta version of RAxML-NG released in March 2017 has already been downloaded more than $1,300$ times (as of November 2017).

In the course of this thesis, we also contributed to several empirical data analysis projects which resulted in peer-reviewed journal publications. In particular, as part of the aforementioned 1KITE consortium we conducted large-scale phylogenetic inferences on whole-transcriptome data, to resolve the evolutionary relationships of *Hymenoptera* [8, 106]. Further, we participated in a large simulation study that evaluated several phylogenetic inference methods with respect to their ability to reconstruct HIV transmission dynamics from incomplete sequence data [111]. Finally, we contributed to a metagenetic study of soil protist diversity in Neotropical rainforests [88]. We do not include the results of the aforementioned studies into this manuscript.

The rest of this thesis is structured as follows. First, we introduce the general concepts of computational phylogenetics and ML tree inference in **Chapter** 2. Then, we describe the adaptation of `ExaML` to the Intel Xeon Phi in **Chapter** 3. In **Chapter** 4, we provide implementation details about `RAxML-NG`, and compare it to existing codes (`RAxML`/`ExaML`) in terms of speed and accuracy. In **Chapter** 5, we explore the topic of modeling sequencing error and uncertainty in phylogenetic inference. **Chapter** 6 is devoted to taxonomic validation and our `SATIVA` pipeline. Finally, we conclude and discuss aspects of future work in **Chapter** 7.

Chapter 2

General Concepts

2.1 Evolution

Since the early days of humanity, philosophers were intrigued by the great diversity of life forms found on Earth: why are there so many different kinds of plants and animals? how did they emerge? have they always existed in their present form, or did they undergo changes over time? At first glance, there is little empirical evidence for evolution: a dog will always give birth to a puppy and not a kitten, and an acorn, if planted, will grow into an oak tree; this reproduction cycle repeats for many generations without visible changes. It is therefore not surprising that a *fixed species* hypothesis was predominant for a long time in history: very much like the flat Earth model, it appeared to be more consistent with the observations.

Nevertheless, evolutionary ideas can already be found in the works of ancient Greek, Roman, and Chinese philosophers [7, 115]. In the Middle Ages, Christian scholars such as Thomas Aquinas argued that *Genesis* should be interpreted allegorically, and that the numerous species could have evolved gradually through natural processes, albeit following a divine master plan [20].

However, it was not before Charles Darwin, that the theory of evolution became generally accepted in the scientific community. In his famous book *On the Origin of Species* (1859) Darwin introduced the idea of a branching pattern of evolution from common descent (see **Figure** 2.1a). He furthermore suggested *variation* and *natural selection* as the driving forces of this evolutionary process. Whereas the general concept of evolution was adopted relatively quickly, its underlying mechanism continued to be a matter of scientific debate for many decades. Alternative theories including *Lamarckism* (inheritance of acquired characteristics) and *orthogenesis* (innate tendency to evolve towards higher complexity) provided seemingly plausible explanations for the observed evolution patterns. At the same time, these theories were free from 'random' and 'unethical' aspects of Darwin's natural selection, which made them more appealing to some scientists. In the early 20th century, a plethora of new empirical data from paleontology, ecology, and population genetics had been accumulated and needed to be explained. This led to the *modern synthesis* theory [57], which combined evidence from different biological disciplines, and re-established natural selection as the main evolutionary mechanism.

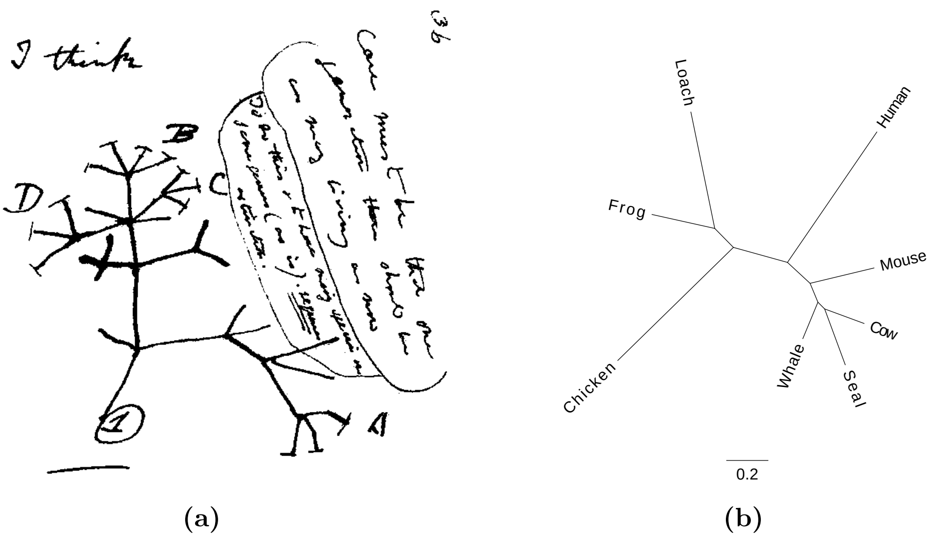
Between 1940 and 1970, major advances in biochemistry and biophysics elucidated the molecular basis of evolution. In particular, desoxyribonucleic acid (DNA) was identified as the carrier of genetic information. In the process of protein synthesis, DNA is first *transcribed* into the ribonucleic acid (RNA) intermediate, which is then *translated* into the actual protein. This information transmission chain (DNA → RNA → protein) is known as the *central dogma of molecular biology.*

DNA, RNA, and proteins are all long polymers, that is, compound molecules built from a sequence of a few basic elements (monomers). DNA consists of four monomers (also called *nucleotides*): adenine (`A`), cytosine (`C`), guanine (`G`) and thymine (`T`). RNA also contains adenine, cytosine and guanine, but thymine is replaced by uracil (`U`). Finally, proteins have 20 'standard' amino acids (AA) as their basic elements. Each amino acid is encoded by one or several specific triplet(s) of DNA/RNA nucleotides (also known as *codons*); this translation table is called *genetic code.*

DNA forms a double helix with a determined nucleotide pairing in complementary strands: `A` pairs with `T` and `C` pairs with `G`. This redundant structure allows for 'proofreading' and damage repair, which is crucial for DNA stability and hence, its role as permanent storage of genetic information. Despite proofreading mechanisms, DNA replication is not absolutely error-free. *Mutations* resulting from non-repaired DNA copying errors represent one source of genetic variation.

RNA and protein molecules are single-stranded, and fold into a sequence-specific 3D-structure (*secondary structure*) due to physical interactions between individual monomers of the same strand (and the environment). Secondary structure is important for the protein/RNA function, and it thus imposes an important constraint on sequence evolution: mutations that do not affect the secondary structure are likely to be neutral (or at least not harmful), and hence they are usually observed more often.

Remarkably, the major molecular components of heredity (DNA/RNA alphabet, AA alphabet, and the genetic code) are virtually identical in all living species, from bacteria to humans. This provides yet another strong evidence for evolution from common descent, as it is highly unlikely that such a complex system could have emerged multiple times independently *and* in exactly the same form.



(a) (b)

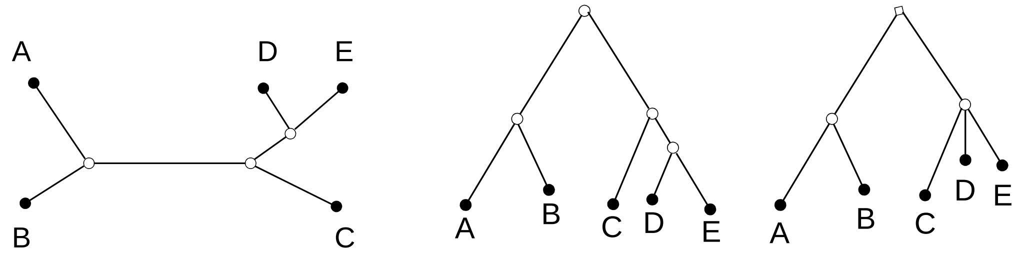
Figure 2.1: Sample phylogenetic trees: (a) A sketch from Darwin's 1837 notebook (source: Wikipedia) (b) A present-day phylogenetic tree (visualized with FigTree v1.4.3 [110]).

2.2 Phylogenetic Trees

Following the Darwin's original idea, the evolutionary history of species (or *taxa*[1]) is commonly represented by a *phylogenetic tree*, also called a *phylogeny* (**Figure** 2.1b). The outer tree nodes (*leaves* or *tips*) correspond to the living (*extant*) species, and the inner nodes – to the *speciation* events, when two novel species arose from a common hypothetical extinct ancestor. Tip nodes are *labeled*, that is, they are assigned unique identifiers (e.g., species names), whereas inner nodes are usually anonymous. A subtree within a phylogeny represents a group of related organisms and called *clade* or *lineage*.

From a biological point of view, phylogenetic trees are always *rooted* (**Figure** 2.2b) since they represent directed evolution from common descent. However, many phylogenetic inference methods (see **Section** 2.4) generate *unrooted* trees (**Figure** 2.2a), mainly for reasons of algorithmic or computational convenience. In these cases, a phylogenetic tree can be rooted *a posteriori*. One popular rooting method relies on so called *outgroups*, that is, one or a few taxa which are known to be closely related to (but not part of) the actual group of interest (*ingroup*). Under these assumptions, it is clear that the root must be placed on the branch separating the outgroup from the ingroup in the inferred phylogeny.

[1] Please note, that tips of a phylogenetic tree can represent species, strains, individuals, or even cells of a multicellular organism. Throughout this thesis, we will use an generic term *taxon* (plural: *taxa*) to refer to all of the above.



(a) **(b)** **(c)**

Figure 2.2: Different kinds of phylogenetic trees: (a) unrooted bifurcating, (b) rooted bifurcating, (c) rooted multifurcating

Phylogenetic trees are generally bifurcating, that is, every inner node has exactly two child nodes. However, multifurcating *consensus trees* represent a common way to summarize contradicting relationships from several phylogenies. Further, multifurcations can be used to represent a series of speciation events that happened within a short period of time and hence, can not be reliably resolved with the available data (*fast radiation*).

It is worth noting that the number of distinct tree topologies N grows over-exponentially with respect to the number of taxa n: $N(n) = \prod_{i=3}^{n}(2i-5)$ [40]. This has important implications for the tree search algorithms (see next chapter): a *brute force* approach for finding the best tree under a given criterion is impractical for virtually all empirical datasets, as the search space grows explosively with the number of taxa n. For instance, there are almost 8 trillion possible trees for just 15 taxa.

Each branch in a tree, if removed, gives two subtrees with two complementary sets of tip labels L and $\overline{L}$. Hence, we say that each branch defines a unique *bipartition* or *split* $(L|\overline{L})$ of the tree. Splits induced by the branches adjacent to the tip nodes are *trivial*, since they are present in all possible topologies. Therefore, only the *non-trivial* splits induced by the inner branches are of interest. In particular, a set of all non-trivial splits uniquely defines the tree topology: for instance, the tree in **Figure** 2.2a can be represented by its two non-trivial splits `B =` $\{$`(AB|CDE), (ABC|DE)`$\}$. A split is the elementary 'building block' of a tree. Therefore, a split-set representation is convenient whenever we intend to compare or summarize multiple trees (e.g., when computing distances, see **Section** 2.3).

While tree topology of a phylogeny defines the relationships among taxa, the branch lengths, if specified, are proportional to the evolutionary degree of change between nodes in the tree. If the evolutionary rate along the tree (from the root to the tips) is assumed to be constant (the *molecular clock* hypothesis), then the branch length between two nodes represents the *relative* amount of time between the respective speciation events. The *absolute* time of the evolutionary events can be obtained via *molecular dating* approaches, which typically involve a calibration with the fossils of (presumably) known age.

2.3 Distance Metrics for Trees

In many applications, it is important to have a measure of (dis)similarity among trees, for instance, to compare the results of different inference programs to each other and/or to the true reference tree in simulation studies. The most popular pairwise dissimilarity metric is the symmetric split distance, also known as *Robinson-Foulds (RF) distance* [113], which is defined as follows:

$$RF(T_1, T_2) = \mid B_1 \cup B_2 \mid - \mid B_1 \cap B_2 \mid \tag{2.1}$$

where B_1 and B_2 are the sets of non-trivial splits in T_1 and T_2, respectively. Since an unrooted tree with n taxa has $n-3$ internal branches, it is easy to see that the maximum RF distance (if no splits are shared) is $2(n-3)$. We can therefore define the *relative* or *normalized* RF distance nRF as

$$nRF(T_1, T_2) = \frac{RF}{2(n-3)} \tag{2.2}$$

Kuhner and Felsenstein introduced a generalization of the RF distance that takes into account tree branch lengths under the name *branch score* [76]. The Kuhner-Felsenstein (KF) branch score for a pair of trees is defined as a sum of square differences between the *corresponding* branch lengths. For the branches (splits) that are missing in one of the trees, the length is set to 0. The square root of the KF branch score yields a distance metric which we will call the *KF distance*:

$$KF(T_1, T_2) = \sqrt{\sum_{s \in B_1 \cup B_2} [b_1(s) - b_2(s)]^2} \tag{2.3}$$

where B_1 and B_2 are the sets of non-trivial splits in T_1 and T_2, respectively, and $b_i(s), i = 1 \ldots 2$ is the length of the branch in T_i corresponding to the split s if $s \in B_i$, and 0 otherwise.

2.4 Phylogenetic Tree Inference

A phylogenetic tree for a set of taxa is usually inferred based on the features or *traits* of these taxa. Historically, mostly morphological and physiological traits were in used (size of the respective bones, Gram staining, lactose metabolism etc.). In the last decades, the focus has shifted towards molecular data (AA and DNA sequences), which generally provide more abundant and less biased phylogenetic signal [53]. Importantly, the traits used for comparison must be *homologous*, that is, they must have evolved from the same trait in the common ancestor. For instance, wings of bats and arms of primates are homologous, as they represent different adaptations of the vertebrate forelimbs. Conversely, wings of insects and wings of birds are *analogous*, that is, despite functional similarity they evolved independently (this phenomenon is called *convergent evolution*). For molecular data, character homology is usually inferred computationally in a process called *sequence alignment*.

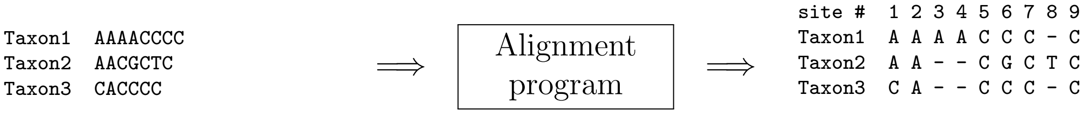
2.4.1 Sequence Alignment

In bioinformatics, it is common to represent DNA, RNA, and amino acid polymers as character strings (or *sequences*). Although this notation ignores physical and chemical properties of the respective molecules, it has proved to be very helpful in many applications, since it allows to leverage existing string processing algorithms. Individual characters are usually referred to as *bases*, and sequence length is measured in *base pairs* or *bp* [2]. For DNA and RNA, the alphabet contains four nucleotides (`A`, `C`, `G`, and `T` resp. `U`), and for proteins – 20 standard amino acids (`A`, `R`, `N`, `D`, `C`, `Q`, `E`, `G`, `H`, `I`, `L`, `K`, `M`, `F`, `P`, `S`, `T`, `W`, `Y`, `V`). In addition, a special *gap* character (`-`) represents the absence of a base at a certain position, for instance, due to deletion. Sometimes, a different character (`?` or `N`) is used to encode *missing data*, that is, absence due to technical reasons such as sequencing failure or fuzzy signal.

For many sequence analysis problems, phylogenetics included, it is important to know which characters are homologous, that is, which characters have evolved from the same character in the ancestral sequence. However, because of base insertion and deletion events (*indels*) that occur in the process of evolution, this information about character homology is not readily available. Hence, sequences need to be *aligned* by inserting the gap characters at the appropriate positions, such that all homologous characters are located in the same column (**Figure** 2.3). This yields a $n \times m$ matrix, where n is the number of taxa (rows), and m is the number of *sites* (columns). This matrix is called *multiple sequence alignment* or *MSA*. Please note, that an MSA can contain multiple identical columns. For instance, in the MSA from **Figure** 2.3 sites 3 and 4 both contain the same *pattern* `A--`, and sites 5, 7, and 9 contain pattern `CCC`. Therefore, we say that this MSA comprises 9 sites, but only 6 *unique patterns*. In some cases, columns with identical patterns can be compressed and considered as a single MSA column with a corresponding weight

[2] The word *pair* refers to the double-helix structure of DNA and does *not* mean that sequence length is measured in duplets. For instance, the length of the sequence `ACGTA` is 5 bp.

(see **Section** 2.6).



```
Taxon1  AAAACCCC
Taxon2  AACGCTC
Taxon3  CACCCC
```

```
site #  1 2 3 4 5 6 7 8 9
Taxon1  A A A A C C C - C
Taxon2  A A - - C G C T C
Taxon3  C A - - C C C - C
```

Figure 2.3: Sequence alignment: by inserting the gap character in the presumed *indel* positions, raw sequences (*left*) are converted into the multiple sequence alignment (MSA) matrix (*right*). Each MSA column only contains homologous characters(or gaps).

Sequence alignment can be formulated as an optimization problem under a given optimality criterion. The SP-score (sum-of-pairs) is a commonly used alignment quality measure, which is computed as a sum of similarity scores for all possible pairs of characters that belong to the same MSA column. Pairwise character similarity is usually defined as some positive constant if both characters are identical (*match*), and some negative constant if they differ (*mismatch*) or if one of the characters is missing (*gap*).

For the special case of pairwise sequence alignment ($n = 2$), efficient algorithms based on dynamic programming exist: the Needleman-Wunsch algorithm [96] can compute the exact solution in $\mathcal{O}(m_1 \cdot m_2)$, where m_1 and m_2 are the lengths of the sequences being aligned. Furthermore, heuristic methods such as BLAST [6] and USEARCH [34] allow to obtain millions of near-optimal sequence alignments in acceptable time.

On the other hand, the MSA problem was shown to be NP-hard [64]. This makes finding the optimal solution impossible for most empirical datasets and led to the development of numerous heuristic approaches. CLUSTAL [52] was the first MSA tool that gained popularity in the 1990s. Nowadays, more rapid and accurate alternatives such as MUSCLE [33], MAFFT [65], and ClustalOmega [127] are used.

It is easy to see that MSA and phylogenetic tree inference are interrelated problems as they share the same underlying goal: reconstruction of the evolutionary history of sequences/taxa. In fact, most MSA heuristics use a 'draft' phylogenetic tree (*guide tree*) to build an initial alignment, which is then used to improve the tree. This iterative process is continued until some convergence condition is fulfilled. Conversely, most modern phylogenetic inference methods require an MSA as input. Therefore, alignment-phylogeny co-estimation is a natural and theoretically appealing approach. Unfortunately, existing implementations [142] do not scale to large datasets, although there are some promising recent developments [102].

2.4.2 Distance-based Methods

Given a matrix of pairwise distances between sequences, classical hierarchical clustering methods can be used to build a tree. In phylogenetics, two algorithms became particularly popular: *Unweighted Pair Group Method with Arithmetic Mean* (UPGMA [128]) and *Neighbor Joining* (NJ [120]). Both algorithms are agglomerative, that is, they first assign each taxon to its own cluster, and then work 'bottom-up' by sequentially joining the *most similar* cluster pairs. The main distinction between UPGMA and NJ lies in the cluster similarity measures they employ. UPGMA uses the average distance between pairs of elements in the clusters being merged. In other words, it strives to *maximize* the similarity *within* a cluster. NJ additionally accounts for the distances to *all other* clusters, which allows to *minimize* similarity *between* clusters. UPGMA is usually considered inferior to NJ due to the fact that it generates *ultrametric* trees (that is, all distances from the root node to the tips are equal). It is equivalent to assuming a constant evolutionary rate across lineages (so called *molecular clock*), which is regarded as biologically unrealistic. Conversely, NJ can produce non-ultrametric trees and hence it does not rely on the molecular clock assumption.

Hierarchical clustering methods are relatively fast: although they have a theoretical time complexity of $\mathcal{O}(n^3)$, efficient implementations have substantially better performance on average (approximately $\mathcal{O}(n^2)$). Furthermore, distance-based methods do not (necessarily) require an MSA, since the input distance matrix can be also computed from pairwise alignments or even unaligned sequences. For these reasons, UPGMA and NJ are often used for datasets that are too large to be processed by more accurate and computationally more expensive methods and/or if the MSA is challenging.

2.4.3 Maximum Parsimony

A conceptually different approach to phylogenetic inference consists in finding the best-scoring tree with respect to a certain optimality criterion.

Maximum parsimony (MP) is an optimality criterion which is based on the Occam's razor principle: it favors the tree(s) that can explain the observed tip sequences (MSA) with the minimum number of mutations/substitutions. For a given topology, the parsimony score of a single MSA site can be computed via a dynamic programming algorithm [123], which finds the minimum number of required mutations across all possible assignments of characters to the internal tree nodes. The overall MSA parsimony score is simply the sum of the per-site parsimony scores. Please note that, since the parsimony score is an integer value, there may be multiple equally parsimonious trees.

For a MSA with n taxa and m sites, the parsimony score computation requires

$\mathcal{O}(mn)$ operations. However, the constant factor here is very small if an efficient implementation based on bitwise arithmetic and vector instructions is used. On the other hand, a polynomial-time algorithm for finding MP-optimal tree is unlikely to exist [45], and an exhaustive search is prohibitive due to enormous tree space (see **Section** 2.2). Hence, in practice one usually resorts to greedy search heuristics based on, e.g., stepwise addition or topological rearrangements (see **Section** 2.4.4).

The main disadvantage of the MP method is that it can be statistically inconsistent under certain conditions [38]. In particular, there is the well-known phenomenon of *long branch attraction* (LBA): distantly related taxa with highly diverged sequences tend to be grouped together in the inferred tree regardless of their true evolutionary history.

2.4.4 Maximum Likelihood

Maximum parsimony is a simplistic model that imposes artificial constraints on the evolutionary process. For instance, it does not allow multiple substitutions to occur along the same branch ($\mathtt{A} \rightarrow \mathtt{T} \rightarrow \mathtt{C}$). Modelling sequence evolution as a stochastic process can thus better represent the biological reality (see **Section** 2.5).

In the probabilistic framework, we can compute the (phylogenetic) likelihood as

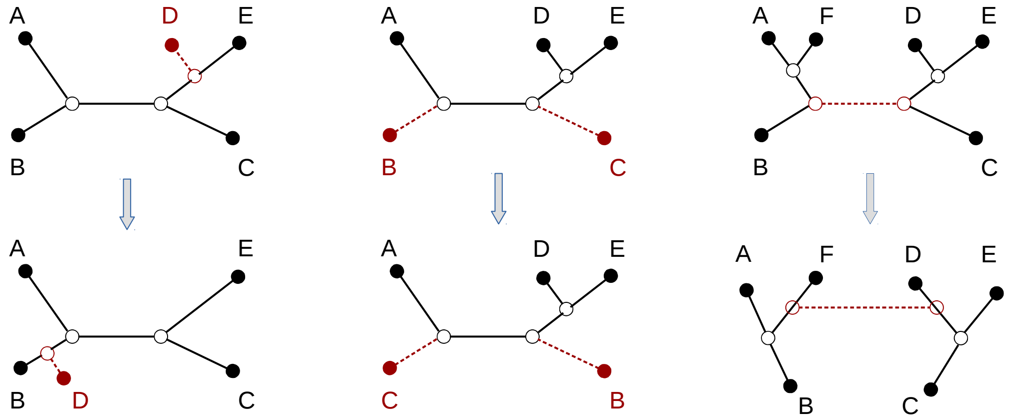
$$L(MSA \mid T, \bar{b}, M, \bar{\theta}) \tag{2.4}$$

that an observed MSA was generated by a tree (topology) T with branch lengths $\bar{b}$ under a certain evolutionary model M with parameters $\bar{\theta}$. Once the evolutionary model M is fixed (see **Section** 2.5.3 for the note on model selection), we can express the likelihood as a function of parameters T, $\bar{b}$, and $\bar{\theta}$. This function is commonly known as the *phylogenetic likelihood function (PLF)*. We can then apply the maximum likelihood estimation (MLE) approach to find the parameter values (including the tree topology) which maximize the PLF, and thus provide the best explanation for the observed data. This method of phylogenetic tree inference is known as *Maximum Likelihood*, or *ML* for short.

Finding the best-scoring tree topology T is a discrete optimization problem that was proved to be NP-hard under the ML criterion [22]. Furthermore, PLF evaluation is computationally expensive because it involves a large amount of floating point operations. This led to the development of numerous ML inference tools that offer different search heuristics and highly optimized PLF implementations. The general approach consists of two steps. First, one or more *starting tree(s)* are generated, typically using NJ or *randomized stepwise addition* (either fully random or parsimony-based, see below). Subsequently, a sequence of topological rearrangements (also called *moves*, see **Figure** 2.4) is applied in order to improve the likelihood.

Randomized stepwise addition works as follows. We start by building a minimal tree from 3 taxa randomly selected from the MSA. Then, we iteratively extend this tree by inserting all remaining taxa in random order. If we intend to generate a random tree, then the insertion branch for every taxon is selected at random. Alternatively, we can select the insertion branch that maximizes the parsimony score of the resulting tree (that is, we apply a greedy heuristic to obtain the MP tree). In the following, starting trees generated by the randomized stepwise addition algorithm with parsimony-based and random insertion strategy will be called simply *parsimony* and *random* starting trees, respectively.

For instance, RAxML [133] and ExaML [72] start from a single random or MP tree, and use greedy hill-climbing based on SPR moves (Subtree Pruning and Regrafting, see **Figure** 2.4a) to find a better topology. Here, 'greedy' means that only the moves that increase the likelihood are applied (*accepted*). PhyML [48] also relies on SPRs since version 3.0, and additionally employs fast parsimony-based prescoring to eliminate the least promising moves. IQTree [98] performs NNI moves on a *set* of candidate trees. In order to avoid local optima, score-decreasing moves are accepted with a certain probability. GARLI [166] uses a genetic search algorithm, that is, it simulates the classical evolutionary forces (random mutation, recombination, reproduction according to the fitness function) acting on a tree population over many generations.



(a) SPR **(b)** NNI **(c)** TBR

Figure 2.4: Commonly used tree moves: a) Subtree Pruning and Regrafting (SPR), b) Nearest Neighbour Interchange (NNI), c) Tree Bisection and Reconnection

For a fixed tree topology T, MLEs for branch lengths $\bar{b}$ and model parameters $\bar{\theta}$ can be obtained with general-purpose numerical optimization methods. In particular, the Newton-Raphson method is often used for branch length optimization, since derivatives of the PLF can be easily computed. For model parameter optimization, Brent [17] and Broyden–Fletcher–Goldfarb–Shanno (BFGS [41]) methods are commonly used.

2.4.5 Bayesian Inference

An alternative probabilistic approach, *Bayesian inference (BI)*, relies on Bayes' theorem to calculate the *posterior distribution* of the relevant evolutionary parameters:

$$p(\bar{\theta} \mid \bar{x}) = \frac{p(\bar{x} \mid \bar{\theta})p(\bar{\theta})}{p(\bar{x})} \tag{2.5}$$

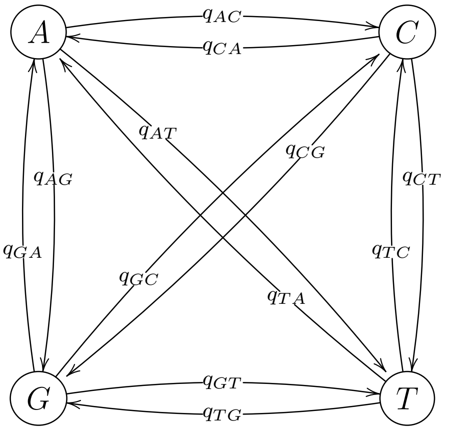
where

- $\bar{x}$ is the observed data (in phylogenetic inference: MSA)
- $\bar{\theta}$ is the parameter vector (tree topology, branch lengths, evolutionary model parameters)
- $p(\bar{\theta} \mid \bar{x})$ is the *posterior* distribution of the parameter values (*given* the data)
- $p(\bar{x} \mid \bar{\theta})$ is the *likelihood* of the data given the parameter vector $\bar{\theta}$ (cf. (2.4))
- $p(\bar{\theta})$ is the *prior* distribution of the parameter values (*without* the data)
- $p(\bar{x})$ is the probability of observing the data vector $\bar{x}$ (integrated over all possible parameter values $\bar{\theta}$)

In practice, the posterior distribution is usually approximated via Markov Chain Monte Carlo (MCMC) sampling methods such as the Metropolis-Hastings algorithm [50].

Despite similarities between ML and BI approaches, there are also several key differences. Firstly, BI allows to incorporate empirical knowledge by providing priors, whereas ML implicitly assumes a flat prior distribution for *all* parameters. Moreover, BI yields posterior probability distributions and not simply point estimates for parameter values. BI thereby provides a natural way for quantifying result uncertainty. Although the above properties are often considered to be theoretically appealing, their practical implications are debated [36].

Some widely-used BI tools include `MrBayes` [116], `BEAST` [30], `ExaBayes` [3], and `PhyloBayes` [80]. BI methods are generally more computationally demanding than ML, in part due to a potentially slow convergence of the sampling process. However, recent advances in hardware and software implementations make BI on very large datasets feasible (e.g., [106]).



(a)

$$Q = \begin{pmatrix} -q_A & q_{AC} & q_{AG} & q_{AT} \\ q_{CA} & -q_C & q_{CG} & q_{CT} \\ q_{AG} & q_{CG} & -q_G & q_{GT} \\ q_{AT} & q_{CT} & q_{GT} & -q_T \end{pmatrix}$$

$$-q_i = -\sum_{j \neq i} q_{ij}, \; i = \{A, C, G, T\}$$

(b)

Figure 2.5: Markov Chain Model of Nucleotide Substitutions: a) States and transitions b) Substitution rate matrix (Q-matrix).

Although Bayesian inference is not covered in this thesis, we should note that BI methods heavily depend on PLF computations (80%–90% of total run-time). Therefore, they will benefit from the optimizations presented in **Sections** 3 and 4.

2.5 Probabilistic Models of Molecular Sequence Evolution

2.5.1 Markov Chain Model of Substitutions

Molecular sequence evolution can be modeled as a continuous-time Markov chain (MC). In case of DNA sequences, the MC has four states `A`, `C`, `G`, and `T` that correspond to the nucleotides. State transitions correspond to nucleotide substitutions (**Figure** 2.5a). The substitution process is defined by an instantaneous transition rate matrix Q, where q_{ij} is the *rate* of transition from state i to state j. The diagonal elements q_{ii} are obtained by the requirement that each row must sum to 0 (**Figure** 2.5b). For a stationary process, the respective transition *probabilities* $p_{ij}(t)$ for a given time t can be obtained by exponentiating the Q matrix (e.g., [159]):

$$P(t) = e^{Qt} \tag{2.6}$$

If all transition rates are positive ($q_{ij} > 0, \forall i \neq j$), then the Markov chain will ultimately (after run long enough) reach the unique *stationary* distribution $\Pi = (\pi_A, \pi_C, \pi_G, \pi_T)$, where π_i is the proportion of time spent in the state i. Alternatively, if the Markov process is interpreted to produce the DNA sequences in the MSA, then Π is the *equilibrium* base composition of the MSA, and π_i are the *equilibrium* or *stationary base frequencies* for this MSA (given that, the process ran long enough to reach the equilibrium).

Most models of DNA evolution assume that Markov process is time-reversible, that is, $\pi_i q_{ij} = \pi_j q_{ji},\ \forall i \neq j$. Although this assumption is not meaningful biologically (as evolution *does* have a direction), it allows to simplify computations. For a time-reversible model, the Q matrix can be formulated as the product of a *symmetric* rate matrix $R = \{r_{i \leftrightarrow j}\}$ and a diagonal matrix containing the stationary base frequencies:

$$Q = R \cdot diag(\pi_i) = \begin{pmatrix} -q_A & r_{A\leftrightarrow C} \cdot \pi_C & r_{A\leftrightarrow G} \cdot \pi_G & r_{A\leftrightarrow T} \cdot \pi_T \\ r_{A\leftrightarrow C} \cdot \pi_A & -q_C & r_{C\leftrightarrow G} \cdot \pi_G & r_{C\leftrightarrow T} \cdot \pi_T \\ r_{A\leftrightarrow G} \cdot \pi_A & r_{C\leftrightarrow G} \cdot \pi_C & -q_G & r_{G\leftrightarrow T} \cdot \pi_T \\ r_{A\leftrightarrow T} \cdot \pi_A & r_{C\leftrightarrow T} \cdot \pi_C & r_{G\leftrightarrow T} \cdot \pi_G & -q_T \end{pmatrix} \tag{2.7}$$

The matrix above describes the so-called general time-reversible model or *GTR* model [146]. This is the most generic model of DNA evolution as all 6 substitution rates $r_{i\leftrightarrow j}, i \neq j$ and 4 base frequencies π_i can potentially be different. Since base frequencies must sum up to 1, and since substitution rates are often normalized by setting $r_{G\leftrightarrow T} = 1.0$, the *GTR* model has 8 free parameters (5 substitution rates + 3 base frequencies). Apart from *GTR*, there are more restrictive model with fewer free parameters. For instance, Jukes-Cantor model (*JC69* [63]) assumes equal substitution rates $r_{i\leftrightarrow j} = 1, i \neq j$ and equal base frequencies $\pi_i = 1/4$. Thus, *JC69* has no free parameters. The *K80* model [66] adds a transition-transversion ratio κ to distinguish between two types of mutations which are not equally likely from a biological perspective: $r_{A\leftrightarrow C} = r_{G\leftrightarrow T} = \kappa \cdot r_{A\leftrightarrow G} = \kappa \cdot r_{A\leftrightarrow T} = \kappa \cdot r_{C\leftrightarrow G} = \kappa \cdot r_{C\leftrightarrow T}$. Finally, the *HKY85* model [49] combines the transition-transversion ratio with unequal base frequencies, and has thus 4 free parameters.

In case of protein data, the state space of the Markov process becomes significantly larger (20 amino acids vs. 4 nucleotides). Consequently, the *GTR* model for protein data has as much as $(400-20)/2-1+19 = 208$ free parameters. This could easily lead to over-parametrization and over-fitting if the MSA being analyzed is not large enough to obtain reliable model parameter estimates. It is therefore common to use so-called *empirical* AA models, which comprise substitution rates and equilibrium base frequencies that were pre-estimated on very large reference MSA collections. Some of the popular empirical AA models include DAYHOFF [29], WAG [152], and LG [82], among many others.

2.5.2 Models of Rate Heterogeneity among Sites

As mentioned in **Section** 2.1, certain regions of DNA or AA molecules are under higher evolutionary pressure, for instance, due to their functional importance. It is therefore reasonable to assume that some MSA sites evolve faster than others. In order to account for this phenomenon in phylogenetic inference, several models of *rate heterogeneity among sites* (RHAS) have been proposed.

Proportion of invariable sites

The simplest model assumes that a certain proportion of alignment sites are absolutely conserved (i.e., they are identical in all taxa), whereas the remaining sites evolve at the same constant rate. Under this model (which we will abbreviate as *P-inv* henceforth), the likelihood of an alignment site s is computed as follows:

$$L_{P\text{-}inv}(s) = p \cdot L(s, 0) + (1 - p) \cdot L(s, r) \tag{2.8}$$

where $p \in [0, 1]$ is the proportion of invariable sites and is usually estimated from the data. Please note, that p is the only free parameter of the *P-inv* model. We do *not* explicitly classify sites as invariable or variable. Instead, we compute *both* $L(s, 0)$ and $L(s, r)$ for *every* alignment site, and then calculate the final site likelihood as the weighted sum of those two values.

Γ model

A more elaborate Γ RHAS model proposed by Yang [156] postulates that substitution rate is a random variable X that is distributed with the probability density function (PDF):

$$f_\Gamma(x, \alpha, \beta) = \frac{\beta^\alpha x^{\alpha-1} e^{-x\beta}}{\Gamma(\alpha)} \tag{2.9}$$

where $\Gamma(\alpha)$ is the gamma function, α is the shape parameter, and β is the inverse scale parameter. In order to obtain likelihoods that are comparable to the models without rate heterogeneity, we require mean substitution rate to be 1. Therefore, we set $\beta := \alpha$, such that $E[X] = \beta/\alpha = 1$, $X \sim \Gamma(\alpha, \beta)$. Depending on the α parameter value, the shape of the Γ distribution shape changes from an exponential-like ($\alpha < 1$, high rate heterogeneity) to a normal-like ($\alpha > 10$, low rate heterogeneity). This property of the Γ distribution allows to fit different rate heterogeneity profiles by optimizing just a single free parameter (α).

In practice, the continuous Γ distribution is approximated by a discrete distribution with K rate categories. The full range of possible rates $r : [0, \infty)$ is divided into K intervals with equal probability mass. For each category, the representative rate r_i is computed as either mean or median rate in the corresponding interval (see [156] for details). Finally, the site likelihood is obtained by averaging over all rate categories:

$$L_\Gamma(s) = \frac{1}{K} \sum_{i=1}^{K} L(s, r_i) \tag{2.10}$$

It is easy to see that for each site, under the Γ RHAS model we need to compute likelihood multiple times, namely for all per-category rates $r_1, r_2, \ldots r_K$. Consequently, the amount of computations and the required memory grow linearly with the number of categories K, resulting in a trade-off between accuracy of the discretization and computational efficiency. Following the recommendation in the original paper by Z.Yang [156], many implementations only use four Γ rate categories by default.

FreeRate model

Albeit flexible, the Γ distribution is not universally applicable: for instance, it cannot adequately represent multimodal distributions. This problem is solved by the *FreeRate* model [158] that does not require evolutionary rates to be drawn from any pre-defined distribution. Instead, both category rates $r_1, r_2, \ldots r_K$ and weights $w_1, w_2, \ldots w_K$ are treated as free parameters. Their values are estimated from the data under two normalization constraints: **(i)** $\sum_{i=1}^{K} w_i = 1$ and **(ii)** $\sum_{i=1}^{K} w_i r_i = 1$. Again, the site likelihood is computed as the weighted sum of the per-category likelihoods:

$$L_R(s) = \sum_{i=1}^{K} w_i L(s, r_i) \tag{2.11}$$

Using a sufficient number of categories, the *FreeRate* model can accurately approximate any distribution of evolutionary rates in the alignment. However, estimating $2(c - 1)$ free parameters is computationally expensive and requires ample input data to avoid over-fitting.

PSR model

The *PSR* model (*per-site rates*; originally named *CAT*) was proposed by Stamatakis [129] as a compute- and memory-efficient approximation of the Γ model.

Partition	P1				P2					P3					
Model	*JC69*				*HKY85*+Γ					*GTR*+*FreeRate*					
Site #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Taxon 1	A	A	-	G	C	A	C	C	T	A	G	G	A	A	G
Taxon 2	A	C	C	G	C	A	C	-	T	A	A	G	A	-	-
Taxon 3	A	A	C	-	C	A	T	C	G	C	A	G	T	C	G

Figure 2.6: An example MSA divided into three partitions: partition P1 has 4 sites and is assigned the Jukes-Cantor model (*JC69*) without RHAS, partition P2 has 5 sites and is assigned the Hasegawa-Kishino-Yano model (*HKY85*) with the Γ RHAS model, and partition P3 has 6 sites and is assigned the *GTR* model with the *FreeRate* model of rate heterogeneity. For the description of individual models, please see **Sections** 2.5.1 and 2.5.2.

Unlike the Γ model and other mixture models as described above, in the *PSR* model every alignment site s is explicitly assigned only one rate category $c(s)$, and its likelihood is calculated as follows:

$$L_{PSR}(s) = L(s, r_{c(s)}), \quad c(s) \in (1, K) \tag{2.12}$$

This solution eliminates the need to compute multiple likelihood values per site, which allows to analyze larger alignments and/or increase the number of rate categories (e.g., RAxML uses 25 categories by default). On the downside, *PSR* likelihood scores obtained with different site-to-category assignments are not comparable. Furthermore, *PSR* branch lengths do not represent the average number of substitution per site as it the case with other models (although this can be corrected for, and a respective normalization was implemented in RAxML later on). Therefore, tree inferences under the *PSR* model are usually combined with a final branch length optimization and likelihood re-evaluation under the Γ model on the final tree.

2.5.3 Alignment Partitioning

Not only evolutionary rates, but also substitution patterns can differ among MSA sites. In order to account for this, an MSA can be split into multiple *partitions*, where each partition is assigned its own model of evolution (that is, substitution matrix, stationary frequencies, RHAS model etc. – see **Figure** 2.6). The partitioning scheme is often devised based on empirical biological knowledge about the MSA at hand. In particular, each gene of a large multi-gene MSA and/or each codon position can be assigned to a separate partition. Furthermore, software tools such as PartitionFinder [79] can be used to optimize the partitioning scheme and to select an appropriate evolutionary model for each partition. Those tools usually rely on statistical model selection criteria to determine the optimal trade-off between model fit (that is, likelihood of the data) and the number of free parameters. This approach can be used both for partition scheme optimization (for instance, by merging certain empirical paratitions to prevent over-parametrization) and for assigning an evolutionary model to each partition. Commonly used model selection criteria include Akaike Information Criterion (AIC [4]) and Bayesian Information Criterion (BIC [126]).

In a partitioned analysis, there are several ways to estimate the branch lengths of a tree:

linked branches All partitions share the same branch lengths which are jointly estimated for the whole MSA.

unlinked branches Branch lengths are estimated independently for each partition using the corresponding subset of alignment sites. This approach yields multiple per-partition trees that share the same topology, but have different branch lengths. Due to to the large number of estimated parameters (np, where n is the number of branches and p is the number of partitions), unlinked branch lengths model can be prone to over-fitting and phylogenetic terraces [122].

proportional branches Branch lengths are estimated jointly as in the *linked* mode, but each partition has an additional scaling factor c, by which all global branch lengths are multiplied in order to obtain the per-partition branch lengths. This is an intermediate solution: it accounts for evolutionary rate differences among genes/partitions, while introducing only one extra parameter per partition.

Phylogenetic inference tools usually allow users to select among branch length estimation modes. For instance, `RAxML` supports linked and unlinked branch lengths, and `IQTree` supports all three modes.

2.6 Computation of Phylogenetic Likelihood Function and its Derivatives

Felsenstein's *pruning algorithm* [39] offers a practical way to compute the PLF. It works by traversing the tree in post-order from the tips towards the (virtual) root and recursively computes the so-called *conditional likelihood vectors* (CLVs) at each inner node (**Section** 2.6.2). At the root, two child CLVs are used to compute the final tree likelihood (**Section** 2.6.3).

Here, we will describe the likelihood computation as implemented in `RAxML` and in `libpll`, since those are the implementations relevant for this thesis. In both codes, the computation is split into several subroutines (*PLF kernels*), which we will introduce in this section and use throughout the remaining text. For the sake of simplicity, we will use the 4-state DNA model in the explanations below. The generalization to an arbitrary number of states is straight-forward.

2.6.1 P-matrix

The *P-matrix* ($P(t)$) defines a transition probability between any pair of states i and j after a certain time t. As shown in **Section** 2.5.1, the P-matrix can be obtained by exponentiating the Q-matrix: $P(t) = e^{Qt}$. While analytical solutions exist for simple models such as *JC69* or *K80*, in the general case (*GTR* model) eigendecomposition of the Q matrix is required to perform the exponentiation. In other words, we need to find an invertible matrix U and a diagonal matrix $\Lambda = diag(\lambda_i)$ such that $Q = U\Lambda U^{-1}$. Then, we obtain:

$$P(t) = e^{U\Lambda t U^{-1}} = Ue^{\Lambda t}U^{-1} = U \cdot diag(e^{\lambda_i t})_{i=1..4} \cdot U^{-1} \tag{2.13}$$

where λ_i are the eigenvalues of Q and U is the matrix of the corresponding eigenvectors. Hence, the elements of the P-matrix can be computed as follows:

$$P_{i,j}(t) = \sum_{k=1}^{4} e^{\lambda_k \cdot t} \cdot U_{i,k} \cdot U_{k,j}^{-1} \tag{2.14}$$

In the above formula, the time t is proportional to the branch length b (more specifically, $t = b \cdot r$, where r is the rate scaling factor, see below). Therefore, an individual P-matrix has to be computed for every branch of the tree, and it has to be updated whenever the corresponding branch length is modified (e.g., during and after branch length optimization). We call the corresponding subroutine `updateP()`.

2.6.2 Conditional Likelihood Vectors

In a sense, each CLV summarizes the subtree below the corresponding node, as it stores the conditional likelihood of *every* state given the respective subalignment (tip sequences), subtree topology *and* branch lengths. In case of DNA data, for instance, a CLV contains (at least) 4 elements per site: `CL(A)`, `CL(C)`, `CL(G)`, and `CL(T)`. If a mixture RHAS model such as Γ or `FreeRate` is used (**Section** 2.5.2), the CLV size grows to $4K$ elements per site, where K is the number of rate categories (**Figure** 2.7b).

In principle, CLVs at the tip nodes can be initialized with the actual likelihoods of observing `A`, `C`, `G` or `T` at the corresponding alignment position (cf. **Chapter** 5). However, in most cases MSA do not contain an uncertainty specification, and thus tip nodes only have 'pseudo-CLVs': e.g., given a nucleotide `A` in the alignment, we set `CL(A)` = 1.0 and `CL(C)` = `CL(G)` = `CL(T)` = 0.0 in the respective tip CLV entry.

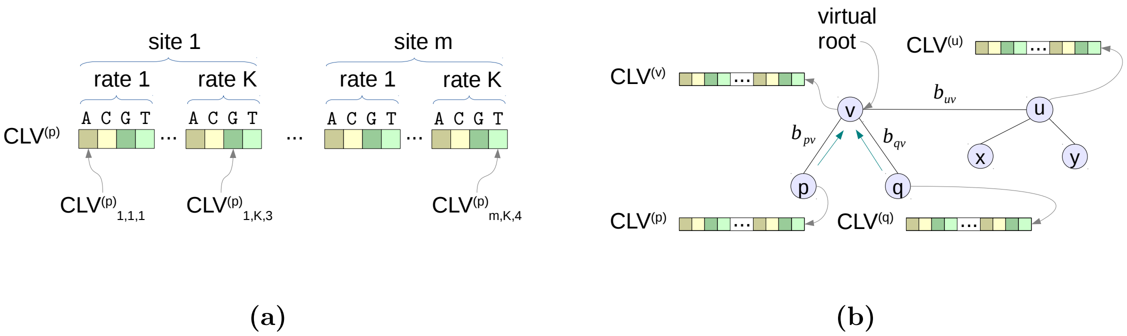


Figure 2.7: a) Conditional likelihood vector (CLV) structure for a DNA alignment with m sites when using a RHAS mixture model with 4 rate categories. b) Tree likelihood computation with Felsenstein's pruning algorithm. When computing the likelihood of an unrooted tree, we can place a *virtual root* at an arbitrary inner node (see **Section** 2.6.3).

The CLV of an inner node v can be computed recursively given the CLV of its two children p and q (**Figure** 2.7b):

$$CLV^{(v)}_{s,c,i} = \left(\sum_{j=1}^{4} P_{i,j}(r_c b_{pv}) \cdot CLV^{(p)}_{s,c,j} \right) \left(\sum_{k=1}^{4} P_{i,k}(r_c b_{qv}) \cdot CLV^{(q)}_{s,c,k} \right) \tag{2.15}$$

where

- $s = 1 \dots m$ is an MSA site,
- $c = 1 \dots K$ is a RHAS rate category
- $i = 1 \dots 4$ is a model state (in the following order: `A`, `C`, `G`, `T`),
- r_c is an evolutionary rate for rate category c
- b_{pv} and b_{qv} are the lengths of the branches between nodes $(p,\ v)$ and $(q,\ v)$, respectively, and
- $P(r_c b_{pv})$ and $P(r_c b_{qv})$ are the P-matrices for the branch lengths b_{pv} and b_{qv}, respectively.

In `libpll`, the CLVs are computed and stored as shown above, whereas in `RAxML` they are additionally multiplied with U^{-1}:

$$\widehat{CLV}^{(v)}_{s,c,k} = \sum_{i=1}^{4} U^{-1}_{k,i} \cdot CLV^{(v)}_{s,c,i} \tag{2.16}$$

The latter representation allows to simplify the computations at the root and during branch length optimization (see below) at the expense of increased overhead in the CLV kernel (henceforth denoted as `updateCLV()`).

2.6.3 Likelihood Evaluation at the Root

The likelihood of an unrooted tree can be computed by placing a *virtual root* into any of its branches. Due to the time-reversibility of the model, the likelihood is invariant to the virtual root placement. Moreover, we can freely 'slide' the virtual root along the branch, without changing the likelihood. In particular, it can coincide with one of the adjacent nodes to simplify the computations (**Figure** 2.7b). Then, likelihood of an MSA site s for rate category c (see **Section** 2.5.2) can be computed as follows:

$$L_{s,c} = \sum_{i=1}^{4} \sum_{j=1}^{4} CLV_{s,c,i}^{(u)} \cdot \pi_i \cdot P_{i,j}(r_c b_{uv}) \cdot CLV_{s,c,j}^{(v)} \tag{2.17}$$

$$L_{s,c} = \sum_{i=1}^{4} \sum_{j=1}^{4} CLV_{s,c,i}^{(u)} \cdot \pi_i \cdot \sum_{k=1}^{4} e^{\lambda_k \cdot r_c \cdot b_{uv}} \cdot U_{i,k} \cdot U_{k,j}^{-1} \cdot CLV_{s,c,j}^{(v)} \tag{2.18}$$

After changing the summation order we obtain:

$$L_{s,c} = \sum_{k=1}^{4} e^{\lambda_k \cdot r_c \cdot b_{uv}} \cdot \left(\sum_{i=1}^{4} \pi_i \cdot U_{i,k} \cdot CLV_{s,c,i}^{(u)} \cdot \sum_{j=1}^{4} U_{k,j}^{-1} \cdot CLV_{s,c,j}^{(v)} \right) \tag{2.19}$$

Considering (2.16) and since $\pi_i \cdot U_{i,k} = U_{k,i}^{-1}$ we can re-write the likelihood calculation for `RAxML`-style $\widehat{CLV}$ vectors as follows:

$$L_{s,c} = \sum_{k=1}^{4} e^{\lambda_k \cdot r_c \cdot b_{uv}} \cdot \widehat{CLV}_{s,c,k}^{(u)} \cdot \widehat{CLV}_{s,c,k}^{(v)} \tag{2.20}$$

Next, the per-site likelihoods are computed according to the specific RHAS model (see **Section** 2.5.2). For instance, for the *FreeRate* model we have:

$$L_s = \sum_{c=1}^{K} w_c L_{s,c} \tag{2.21}$$

Finally, we compute the overall likelihood for the entire MSA. It is generally assumed that alignment sites evolve independently, and thus for an MSA with m sites, the overall likelihood is computed as $L = \prod_{s=1}^{m} L_s$. However, this product is usually a very small number, which can lead to numerical underflow. Hence it is more convenient to work with the logarithm of the likelihood (*log-likelihood*). We therefore take a natural logarithm of each individual per-site likelihood, and compute the sum across all alignment m sites:

$$logL = \ln L = \sum_{s=1}^{m} \ln L_s \tag{2.22}$$

Please note, that MSA sites with identical patterns (see **Section** 2.4.1) will yield exactly the same likelihood. Thus, we do not need to repeat the likelihood computation for those duplicated sites. Instead, we build a compressed MSA comprising only $m' < m$ unique alignment patterns, and store the number of occurrences for each pattern in a *weight vector* ω. The likelihood on a compressed MSA is then computed as follows:

$$logL = \ln L = \sum_{s=1}^{m'} \omega_s \ln L_s \tag{2.23}$$

The PLF kernel that performs the computation described in this subsection is denoted as `computeLH()`.

2.6.4 Likelihood Derivatives with respect to the Branch Length Parameter

As mentioned before (**Section** 2.4.4), branch length optimization often relies on iterative methods such as Newton-Raphson, which require PLF derivatives. Here, we explain how those derivatives are computed in `RAxML` and `libpll`.

We assume that the virtual root is already placed on the branch that is being optimized, and that *CLV*s to the left and right of this branch have been updated accordingly. Then, we can write the *logL* as the function of the (root) branch length b:

$$logL(b) = \sum_{s=1}^{m} \ln L_s(b) \tag{2.24}$$

We derive $logL(b)$ analytically, which gives:

$$logL'(b) = \sum_{s=1}^{m} \frac{L'_s(b)}{L_s(b)} \tag{2.25}$$

$$logL''(b) = \sum_{s=1}^{m} \frac{L''_s(b) \cdot L_s(b) - [L'_s(b)]^2}{[L_s(b)]^2} \tag{2.26}$$

Now, we need to obtain the per-site likelihood $L_s(b)$ and its derivatives $L'_s(b)$ and $L''_s(b)$. From (2.17) and (2.20) it is easy to see that the product of both CLVs (resp. $\widehat{CLV}$s) is constant with respect to the branch length value being optimized. We can therefore pre-compute this product and store it in a temporary array to speedup the derivative computation (`derivativeInit()` kernel):

$$\begin{aligned} S_{s,c,k} &= \widehat{CLV}^{(u)}_{s,c,k} \cdot \widehat{CLV}^{(v)}_{s,c,k} \\ &= \left(\sum_{i=1}^{4} \pi_i \cdot U_{i,k} \cdot CLV^{(u)}_{s,c,i} \right) \cdot \left(\sum_{j=1}^{4} U^{-1}_{k,j} \cdot CLV^{(v)}_{s,c,j} \right) \end{aligned} \tag{2.27}$$

Then, in every Newton-Raphson iteration we can compute the per-site likelihood as well as its first and second derivative as follows (`derivativeCore()` kernel):

$$L_s(b) = \sum_{c=1}^{K} w_c \sum_{k=1}^{4} e^{\lambda_k \cdot r_c \cdot b} \cdot S_{s,c,k} \tag{2.28}$$

$$L'_s(b) = \sum_{c=1}^{K} w_c \sum_{k=1}^{4} \lambda_k \cdot r_c \cdot e^{\lambda_k \cdot r_c \cdot b} \cdot S_{s,c,k} \tag{2.29}$$

$$L''_s(b) = \sum_{c=1}^{K} w_c \sum_{k=1}^{4} \lambda_k^2 \cdot r_c^2 \cdot e^{\lambda_k \cdot r_c \cdot b} \cdot S_{s,c,k} \tag{2.30}$$

Finally, we plug in (2.28) – (2.30) into (2.25) and (2.26) to obtain the derivatives of $logL$.

Chapter 3

Efficient Likelihood Computation on Intel Xeon Phi Accelerators

This chapter is based on two peer-reviewed publications:

- **AM Kozlov**, A Stamatakis, C Goll. ”Efficient Computation of the Phylogenetic Likelihood Function on the Intel MIC Architecture.” In: *Proceedings of HICOMB workshop, held in conjunction with IPDPS 2014, Phoenix, Arizona, May 2014*
- **AM Kozlov**, AJ Aberer, A Stamatakis. ”ExaML Version 3: A Tool for Phylogenomic Analyses on Supercomputers.” In: *Bioinformatics* (2015) 31 (15): 2577-2579.

Contributions: Alexey Kozlov designed the SIMD vectorization and parallelization approaches for Xeon Phi, integrated them into the production version of `ExaML`, and conducted the performance evaluation. Alexandros Stamatakis and Andre Aberer developed the original `ExaML` code, and helped to write the paper. Christian Goll installed and configured the Xeon Phi cards on the HITS institutional cluster.

As stated in **Section** 1.1, there is a clear need for further performance improvements in phylogenetic inference in order to handle the constantly growing empirical datasets. Besides algorithmic advancements, efficient use of novel hardware is a key factor in achieving this goal. In the last decade, hardware accelerators have attracted a lot of attention in the HPC community. Although the original “1000x speedup” hype was debunked, accelerators can still yield significant gains in runtime and energy-efficiency for a plethora of scientific applications. Nowadays, it is common for Top500 systems to include accelerators such as NVIDIA GPUs or Intel Xeon Phis, and many popular scientific codes have already been ported to these architectures. In particular, likelihood computation was previously implemented on FPGAs [5, 11, 165] and GPUs [59, 141]. Continuing this line of work, I ported and optimized Phylogenetic Likelihood Function (PLF) kernels on the Intel Xeon Phi.

3.1 Intel Knights Corner: Platform Overview

The first generation of Intel Xeon Phi coprocessors, codenamed Knights Corner (KNC), became commercially available in late 2012. KNC is a PCIe add-on card equipped with 57 to 61 physical cores running at ∼1GHz (model-dependent) and delivers about 1 TFLOPS peak double-precision performance. Each core has a dedicated 512KB L2 cache, and can access. the caches of all other cores via the ring interconnect. Additionally, each card has 6 to 16GB of high-bandwidth GDDR5 memory that is shared among all cores. The instruction set (referred to as `AVX-KNC` henceforth) was inherited from the x86 architecture and extended by 512-bit wide vector operations. This allows to process 8 DP (double precision) or 16 SP (single precision) floating point values simultaneously, that is, twice as much as with AVX/AVX2 on Sandy/Ivy Bridge and Haswell CPUs.

In terms of architecture and programming model, the Xeon Phi is substantially more similar to regular CPUs than other accelerators such as GPUs. More specifically, there are several features worth to be noted:

1. Like most modern CPUs, KNC employs thread-based parallelization across cores as well as instruction-level SIMD parallelism within each core. This approach offers increased flexibility compared to SIMD-only hardware, since each thread is absolutely independent and can follow its own program execution path. Moreover, multiple kernels or program instances can be executed in parallel, each using a subset of available KNC cores.

2. KNC cache memory is transparent to the programmer. There is no need to take direct control over the memory hierarchy. Instead, the programmer is expected to write cache-efficient code (e.g., preserve access locality) and rely on the hardware and/or compiler to decide, how to cache data. The only mechanism for influencing this process is to place explicit prefetch hints in the code.

3. KNC is compatibile with the standard x86 development toolchain (icc, pthreads, OpenMP, MPI) and does not require dedicated frameworks such as CUDA or OpenCL.

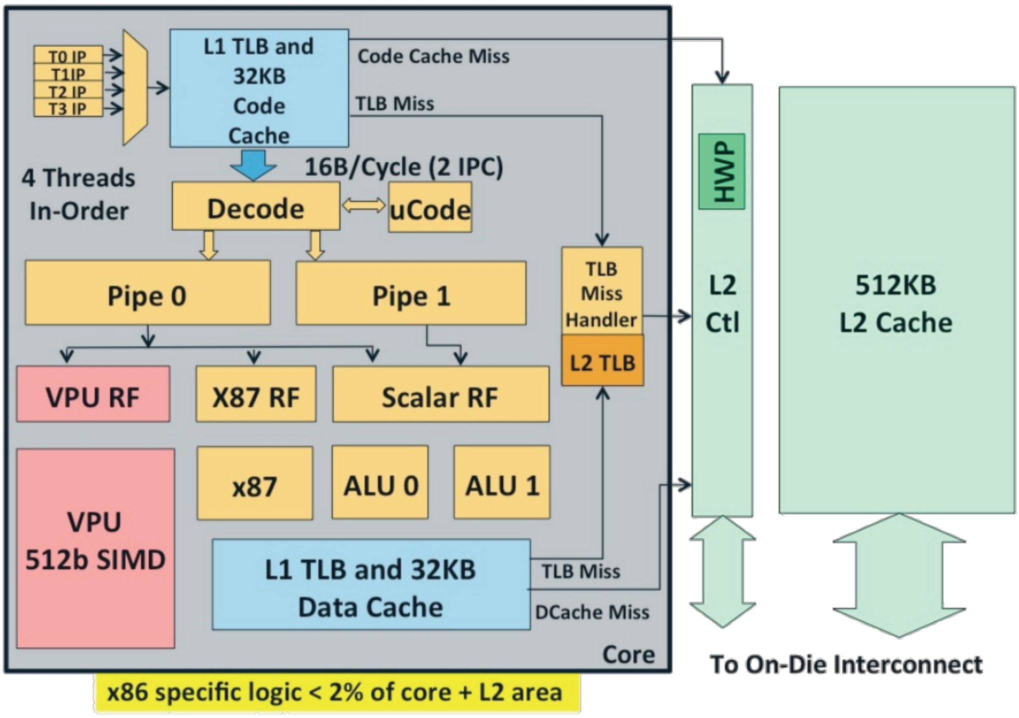


Figure 3.1: Architecture of the Intel Xeon Phi (Knights Corner). Source: `http://semiaccurate.com/2012/08/28/intel-details-knights-corner-architecture-at-long-last/intel_xeon_phi_core/`

4. Finally, KNC offers two different program execution modes. In the *offload* mode, the main program runs on the host and ’offloads’ compute-intensive code blocks (kernels) to the coprocessor. Conversely, in *native* mode, the entire program is executed exclusively on the coprocessor, without any host involvement. This is possible, because each KNC card executes a Linux-based micro-OS, thus, turning it into a self-contained ’host’ system. Furthermore, the coprocessors obtain IP addresses and can communicate through the simulated network interface, allowing to use MPI for KNC-KNC and KNC-Host communication.

The above features facilitate reuse of developer’s knowledge and yield it relatively straightforward to port existing HPC codes to the KNC. Nevertheless, manual tuning is usually required to achieve optimal performance, and this effort should not be underestimated (see below).

3.2 Likelihood Kernel Optimization

In theory, getting an existing C code to run on the Intel KNC might be as simple as recompiling it with the `-mmic` compiler option. In practice, however, a program 'ported' in this way, will most probably exhibit suboptimal performance. In fact, in some cases it might even execute slower than the original CPU version (see e.g., [117]). Hence, to fully exploit the hardware capabilities, one still needs to invest significant effort to optimize, adapt, and tune the code. In the following, we will describe several optimization techniques, and how we deployed them to improve PLF kernel efficiency on the KNC.

Vectorization

To attain optimal performance on KNC, the code must be vectorized; there are several ways to achieve this. First, the Intel compiler offers automatic loop vectorization. Multiple loop iterations or arithmetic operations can be combined and replaced by a single vectorized instruction. For automatic vectorization to be successful, several conditions must hold, most importantly: the loop in question must be the innermost loop, all vectors must be properly aligned (see below), and there should be no data dependencies between input and output vectors. In difficult cases, the programmer can provide hints to the compiler by using `ivdep` (no data dependency) and `vector aligned` pragmas. Alternatively, it is possible to write vectorized code by hand, using so called *compiler intrinsics*, that is, pseudo-functions which are usually mapped directly to the corresponding processor instructions. Although this approach offers the highest level of control, it produces less readable and more error-prone code (see Figure 3.2). For this reason, we used automatic compiler vectorization whenever possible, while resorting to intrinsics in non-trivial cases only.

Memory alignment

Most vector instructions operate on values stored in the corresponding vector registers. Those registers can be efficiently initialized from memory addresses that are aligned to the vector size boundary (64 bytes or 512 bits in case of KNC). Loading data from unaligned memory locations incurs a significant performance penalty. This property has several design implications:

- First, all arrays must start at addresses which are a multiple of 64. This can be ensured by using appropriate memory allocation functions (e.g., `__mm_malloc` or `memalign`).

- Second, all array sizes (in bytes) must be multiples of 64. We solve this through padding, that is, by adding empty trailing elements at the end of each array.

```
int l;
#pragma ivdep
#pragma vector aligned
for (l = 0; l < 16; l++)
{
 pr[l] = le[l] * ri[l];
}
```

(a)

```
__m512d l1 =
  _mm512_load_pd(&le[0]);
__m512d l2 =
  _mm512_load_pd(&le[8]);

__m512d r1 =
  _mm512_load_pd(&ri[0]);
__m512d r2 =
  _mm512_load_pd(&ri[8]);

__m512d s1 =
  _mm512_mul_pd(l1, r1);
__m512d s2 =
  _mm512_mul_pd(l2, r2);

_mm512_store_pd(&pr[0], s1);
_mm512_store_pd(&pr[8], s2);
```

(b)

```
vmovapd (%rsp,%r10,1), %zmm0

vmovapd 0x40(%rsp,%r10,1), %zmm1


vmovapd (%rcx,%rdi,8), %zmm2

vmovapd 0x40(%rcx,%rdi,8), %zmm3


vmulpd %zmm4, %zmm0, %zmm2

vmulpd %zmm5, %zmm1, %zmm3


vmovapd %zmm4, (%rsi,%rdi,8)
vmovapd %zmm5, 0x40(%rsi,%rdi,8)
```

(c)

Figure 3.2: Loop vectorized using pragmas *(a)* and compiler intrinsics *(b)*. In both cases, the generated assembly code is the same *(c)*.

- Finally, one has to ensure that all accesses to the array elements are also aligned. That is, for a `double` array, all offsets must be a multiple of 8. Most kernels operate on CLV elements, which have a size of $states \times rates$ `doubles`, where *rates* is the number of rate categories per site. Currently, `ExaML-KNC` only supports the Γ model of rate heterogeneity [156] with 4 discrete rate categories. Hence, a CLV element constains 16 DP numbers, and all array accesses are aligned. However, in order to implement the CAT model of rate heterogeneity [129] which only has one rate per site, special care must be taken to keep accesses aligned.

Re-organizing loops

As part of conditional likelihood computations in `updateCLV()`, the CLV vector of a child node has to be multiplied with the transition probability matrix P. The dimension of this matrix is equal to the number of states (e.g., DNA or AA characters). For DNA data we therefore multiply a 1×4 vector with a 4×4 matrix. For this operation, the innermost loop executes 4 iterations, which is smaller than the vector unit width on the KNC (8 doubles). Therefore, the loop can not be vectorized efficiently without changes. Note that, under the Γ model with 4 discrete rates, we actually need to perform 4 such vector-matrix multiplications for each alignment site. If we execute these multiplications simultaneously, we obtain 16 iterations in the innermost loop, which is sufficient for vectorization. Since all iterations must access contiguous memory locations, we need to re-arrange the input arrays accordingly. Then, the inner loop can be calculated by two fused-multiply-add (FMA) vector operations.

Site blocking

Computing derivatives in `derivativeCore()` can be split into two phases: for each alignment site, 16 elements of a vector are preprocessed, then several scalar operations are applied to obtain the final result. Obviously, the first phase can be easily vectorized, but the scalar operations pose a problem. To this end, we re-organized the main loop that iterates over single alignment sites to process alignment sites in groups of 8. This allows for executing one single vector operation replacing the 8 problematic scalar operations.

Streaming stores

When writing to a memory cell, the old contents of the corresponding cache line have to be loaded first. If our intention, however, is to overwrite the entire cache line (64B), this reading operation is not required. KNC introduces a special *streaming store* instruction, which allows to avoid this unnecessary read and associated time penalty. Even though the compiler implements some heuristics to automatically generate streaming store instructions, the programmer can enforce these by placing the `#pragma vector nontemporal` directive in front of the loop. We make use of this feature in the `updateCLV()` and `derivativeSum()` kernels when writing results to the parent CLV and the summation buffer, respectively.

Manual prefetching

Prefetching is an optimization technique for hiding memory access latency. It relies on predicting which data elements will be processed by the program in the near future and fetching those elements into the cache *in advance*. Thereby, the actual data access is carried out on the low-latency cache without further delays. On KNC, prefetching can be controlled by the hardware, by the compiler, or manually by the programmer (using `#pragma prefetch` or `_mm_prefetch`). In the latter case, it is up to the programmer to determine the best *prefetch distance*, that is, for how many loop iterations ahead in the future, the prefetch instruction shall be issued. While the optimal value is mainly influenced by memory latency and the amount of computations per iteration, a direct estimation is complicated by other confounding factors (e.g., number of threads per core). Thus, in practice one often has to resort to an empirical tuning approach.

Despite some previous studies reporting near-optimal performance with automatic prefetching [75], we observed notable speedups by manually inserting prefetching instructions into our code. This can be explained by the streaming access patterns of our kernels. They linearly read input vectors from memory (summing buffers in `derivativeCore()` and CLVs in other functions), perform relatively few computations, and then write results to the output vector. In this setting, memory access latencies dominate runtimes and therefore an optimal prefetching strategy is crucial for performance.

Offload vs. native mode

At first glance, it seems natural to offload compute-intensive PLF kernels to the Xeon Phi, and invoke them from the main tree search algorithm running on the host processor. Although host↔KNC data transfers are costly, they can easily be avoided by allocating CLVs in coprocessor memory and only sending the node indices – a solution previously suggested in [59]. However, initial experiments with the offloading-based version showed that KNC kernel calls induce a substantial overhead, even if no data transfer is involved. In fact, kernel call latency is comparable to and partially exceeds the time required for the actual computation. This seems to be an inherent limitation of the offloading approach: both hardware (initiating the data transfer over PCIe) and software (calling the offload runtime) components induce a certain latency [97], which can not easily be alleviated. Since ML inference algorithms perform thousands of kernel invocations per second, even for small trees, the offload latency becomes *the* major bottleneck.

For this reason, we then explored the *native* execution model. After minor modifications, we were able to compile the entire ExaML program on the KNC platform and execute it on the coprocessor without any host involvement. In this native version, the kernel invocations are simple function calls with negligible latency. Thus, we observed a speedup exceeding a factor of two compared to the initial offloading-based version. Moreover, the code became significantly simpler, because allocating and orchestrating separate CLVs on the coprocessor is not required any more. In fact, the only major differences between the CPU and the native KNC implementations are in the kernel codes.

3.3 Hybrid MPI/OpenMP Parallelization

Originally, `ExaML` provided MPI-based parallelization only. So, in order to exploit intra-node parallelism, multiple MPI processes had to be started (e.g., 1 process per CPU core). However, as our tests have shown, this approach does not fit the Xeon Phi well, since hundreds of MPI processes per card would need to be started. To circumvent this problem, we initially implemented an *ad hoc* OpenMP solution, where the main loop over alignment site patterns was parallelized in each kernel individually [73]. Despite performing better than the pure MPI approach, the parallel efficiency of this initial solution deteriorates with an increasing number of partitions.

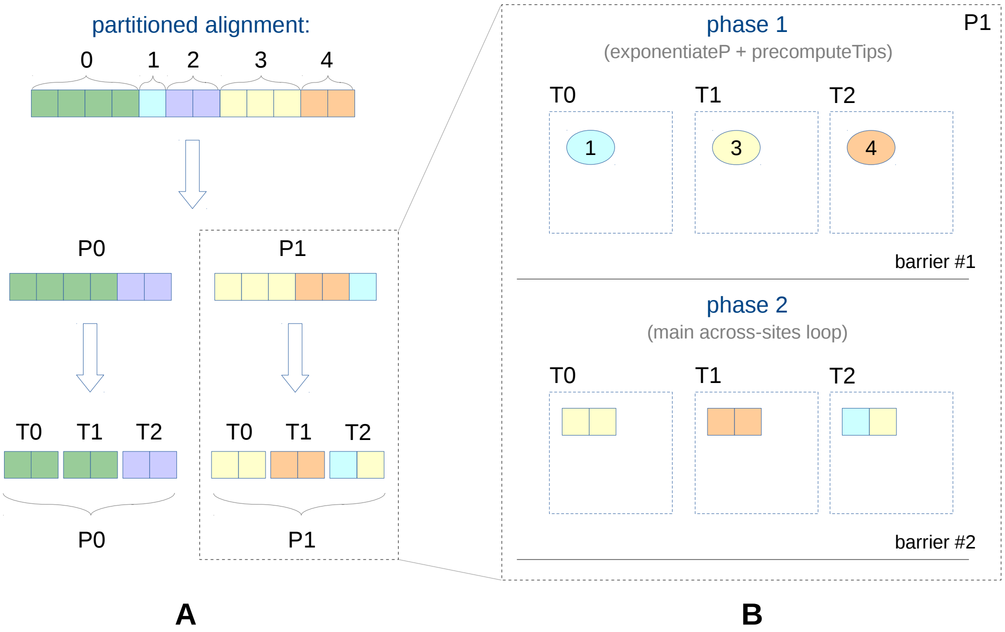


Figure 3.3: Hybrid MPI/OpenMP parallelization scheme: **A**. Partitions are distributed among MPI processes, and then among OpenMP threads. **B**. The two-phase PLF evaluation procedure allows for better load balance and less synchronization overhead.

Note that, analyses with hundreds or thousands of partitions represent the standard `ExaML` use case (e.g., [62, 92]). The are several reasons for the above inefficiency:

1. *Excessive amount of synchronization*: For practical reasons, each partition is processed via a separate kernel call. Consequently, for p partitions there will be p distinct `for`-loops over site patterns. Moreover, since every KNC thread is calculating per-site likelihoods for every partition, synchronization after each per-partition loop is required.

2. *Sequential overhead*: Each partition requires some constant amount of computations (with respect to the partition size). For instance, the Q matrix exponentiation and the pre-computation of conditional likelihoods at the tips fall into this category. Since these computations are conducted outside the parallelized `for`-loops, they are executed sequentially and limit performance according to Amdahl's law.

3. *Reduced data locality*: Due to the OpenMP loop-based parallelization employed, there is no fixed assignment of alignment site patterns to specific threads, so cache efficiency decreases because of lack of data locality.

Because of these shortcomings, we designed a novel OpenMP parallelization approach from scratch for `ExaML` 3.0 (see **Figure** 3.3). In particular, we use the algorithm described in [68] to distribute partitions and alignment site patterns not only among MPI processes, but also (in a second step) among OpenMP threads. In other words, we now perform two-level load balancing. Initially, partitions (or regions thereof) are assigned to MPI processes. Then, the partitions (or parts thereof) of a MPI process are assigned to individual threads within this process using the same algorithm once more. Thereby, we attain a fixed thread-to-alignment pattern assignment and improve data locality.

To deal with the remaining two issues, we introduce a two-phase PLF calculation:

- In phase 1, we parallelize over partitions: all partitions are distributed evenly among threads, and each thread performs the constant part of computational work (mentioned above) for the partition(s) assigned to it.
- In phase 2, we parallelize over sites: each thread performs PLF computations on its individual part of the alignment.

With this approach, we only require *two* synchronization points (one after *Phase 1* and one after *Phase 2*) to perform the PLF computation. Thus, the amount of barriers required is independent of the number of partitions. In addition, work is now evenly distributed among the KNC threads in both phases which eliminates the sequential bottleneck.

3.4 Evaluation

3.4.1 Test System

We performed all test runs on the SuperMIC cluster, which is part of the SuperMUC supercomputer (Leibniz Rechenzentrum, Garching, Germany). On SuperMIC, each compute node is equipped with 2 Ivy-Bridge host processors (Xeon E5-2650 v2, 2 × 8 cores @ 2.6 GHz) and 2 Intel KNC coprocessors (Xeon Phi 5110P, 2 × 60 cores @ 1.05 GHz). The nodes are connected via Mellanox Infiniband FDR14 using Mellanox OFED 2.2. Further details on hardware and software configuration of the test systems are given in **Table** 3.1.

3.4.2 Kernel-level Performance

Experimental setup

To assess the speedups of individual PLF kernels, we instrumented both CPU and KNC codes such that we can measure the total time spent in the corresponding

System	IvyBridge	KNC	KNL
CPU model	2S Xeon E5-2650 v2	Xeon Phi 5110P	Xeon Phi 7210
CPU architecture	IvyBridge	Knights Corner	Knights Landing
Cores	16 @ 2.60 GHz	60 @ 1.05 GHz	64 @ 1.3 GHz
Memory size	32GB DDR3	8GB GDDR5	16GB MCDRAM
Memory bandwidth	120 GB/s	320 GB/s	400 GB/s
Peak DP performance	346 GFLOPS	1011 GFLOPS	2662 GFLOPS
TDP	190 W	225 W	215 W
Software stack	SUSE SLES 11 Intel C Compiler (icc) 15.0.4 Intel MPI 5.0.1.035		SUSE SLES 12.2 icc 17.0.2 Intel MPI 2017.2

2S = dual slot, DP = double precision, TDP = thermal design package

Table 3.1: Hardware and software specifications of test systems used for performance evaluation.

functions during one complete program run (tree search). In this way, we registered per-kernel runtimes and computed relative speedups on a range of simulated DNA alignments (generated with `INDELible` v1.03 [42]). According to our previous observations, only alignment width (number of sites) but not the number of taxa would influence the relative PLF kernel performance. Hence, all simulated alignments contain 15 taxa while the sequence length is varying between 10,000 (10 Kilobases or Kbp) and 4,000,000 (4,000 Kbp) sites.

Based on our experience, PLF computation does not benefit from using the Hyper-Threading feature of the CPUs. Therefore, we always allocated one thread per physical CPU core in our test runs. In particular, we used 16 threads on the `IvyBridge` system. On KNC, however, the situation is slightly different: due to the specific design of the instruction pipeline, at least 2 threads per core are *essential* to fully utilize the hardware. Moreover, 4 threads per core can be used with Hyper-Threading, although this doesn't result in further performance improvements for PLF computation (similar to CPUs). Apart from this, it is often recommended to reserve one physical core for system task (e.g., running the built-in Linux OS). Given these considerations and since our Xeon Phi 5110P card has 60 physical cores (see **Table** 3.1), we used $(60 - 1) * 2 = 118$ threads on the *KNC* system.

Results

We measured the highest speedup of 2.8× for the `derivativeInit()` kernel (see **Figure** 3.4). This is expected given that `derivativeInit()` performs a simple element-wise multiplication of CLV entries (see **Eq.** (2.27)), which can be efficiently vectorized. The other kernels exhibit a less favorable mixture of numerical operations. Hence, the speedups for these kernels are at most a factor of two. Furthermore, there is a clear trend towards higher speedups on longer alignments which holds for all PLF kernels. In particular, KNC version was even slower than the CPU baseline

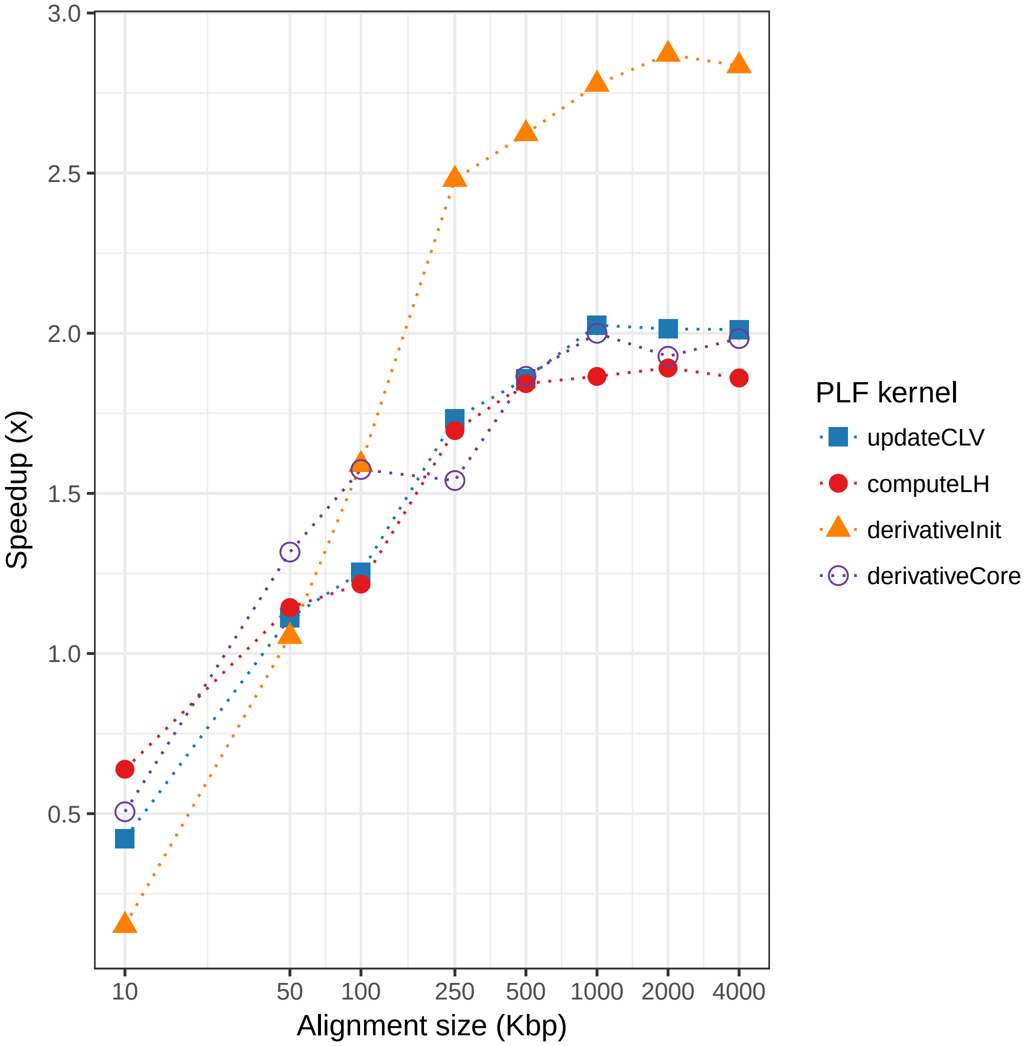


Figure 3.4: Speedups of the individual PLF kernels on `KNC` relative to the `IvyBridge` baseline.

on 10 Kbp alignment, and reached optimal performance on alignments of 1000 Kbp and longer. These findings are in line with the previously published results for GPU implementation of PLF [59].

3.4.3 Application-level Performance on a Single Node

Experimental setup

We used `INDELible` v1.03 [42] to simulate 10 alignments (5 DNA, 5 AA) with representative dimensions as observed for empirical datasets (see **Table** 3.2 for details):

- short: single-gene alignment (e.g., classical 16S rRNA analysis),
- medium: multi-gene alignment with a moderate number of genes,
- large: whole-genome alignment or concatenation of hundreds of genes.

Due to the limited size of KNC on-card memory (8 GB), large alignments comprise only 20 taxa, whereas medium and short alignments consist of 200 and 2000 taxa, respectively.

For the medium and large alignments, we tested 2 partitioning schemes:

- unpartitioned (AA) or partitioned by codon positions (DNA),
- fine-grained partitioning schemes analogous to schemes generated by automatic partitioning tools (e.g., `PartitionFinder` [79]).

To assess `ExaML` performance on the Intel KNC, we measured the execution times of one full tree search under the following 4 configurations:

- *host*: 16 MPI ranks are placed on the host CPUs only (reference for speedup calculation),
- *1xKNC*: 1 MPI rank on a single KNC card, 118 OpenMP threads (on `short` datasets, a smaller number of threads was used; see discussion below),
- *2xKNC*: 1 MPI rank with 118 OpenMP threads on each of the two KNC cards,
- *hybrid*: 16 MPI ranks on the host CPUs and 30 MPI ranks (× 4 threads) on each of the two KNC cards.

Results

The experimental results are summarized in **Table** 3.2, relative speedups are shown in **Figure** 3.5.

As we have already observed in the PLF kernels tests (see **Section** 3.4.2), the KNCs perform better on longer alignments, where synchronization and sequential overhead are better amortized. On the other hand, distributing several thousand kilobases among >100 threads proved to be inefficient. This is in line with our empirical observations for the `ExaML` CPU version, where one process per 100–500 alignment site patterns should be used for attaining good parallel efficiency. Hence, we only used 30 and 40 threads for the dna_short_1p and aa_short_1p datasets, respectively. Analyzing such extremely short alignments on the KNC only makes sense when a 'multiplexing' strategy is applied. If multiple ML searches are executed (e.g., with different starting trees), one can start several (independent) `ExaML` instances in parallel, so that all KNC cores are being used (e.g., 4 independent instances with 30 threads each). In such a 'multiplexing' configuration, a single KNC card is on par with the performance of the host cores alone. Thus, more than two-fold speedups are expected if both KNC cards *and* the host cores are used.

On the medium and long datasets, speedups on DNA data are higher than on AA data (1.34×–1.66× vs. 1.04×–1.37×). Also, despite our specific optimizations for partitioned alignments, the KNC version shows a 5–20% performance decrease (compared to the unpartitioned analysis) if the number of partitions is large.

In *hybrid* mode, we attained up to four-fold speedups have been achieved on large DNA alignments. Since in this configuration the CPU and KNC cores work together, load balancing becomes essential. Given that ExaML distributes alignment patterns and partitions evenly among MPI ranks, one can fine-tune the ratio between MPI ranks on host CPUs and KNCs to improve load balance. For example, in our experiments we used 30 KNC ranks and 16 host CPU ranks, which yields a ratio of 1.875. Intuitively, this ratio should be close to the KNC/host speedup. Thus, lower values might yield better results on smaller alignments (e.g., 24/16 = 1.5).

Dataset properties						Execution time, s				Speedup to host, ×		
Data type	Code	# taxa	# sites	# patterns	# part.	Host	1×KNC	2×KNC	Hybrid	1×KNC	2×KNC	Hybrid
DNA	dna_short_1p	2000	6K	5322	1	6776	29010	NA	NA	0.23	NA	NA
	dna_medium_3p	200	300K	293780	3	9579	6643	4073	3251	1.44	2.35	2.95
	dna_medium_50p	200	300K	293158	50	9505	7092	4264	3444	1.34	2.23	2.76
	dna_long_3p	20	4000K	1940554	3	1591	908	482	389	1.75	3.30	4.09
	dna_long_500p	20	4000K	3084131	500	2542	1528	825	680	1.66	3.08	3.74
AA	aa_short_1p	2000	2k	2044	1	41741	132019	NA	NA	0.32	NA	NA
	aa_medium_1p	200	60K	56655	1	18911	16304	9070	7423	1.16	2.08	2.55
	aa_medium_50p	200	60K	56966	50	22098	21169	12165	9534	1.04	1.82	2.32
	aa_long_1p	20	600K	442153	1	3402	2477	1265	1019	1.37	2.69	3.34
	aa_long_500p	20	600K	544580	500	4334	3711	1912	1497	1.17	2.27	2.90

Table 3.2: `ExaML` execution times and speedups on the host cores (Host), on the single (1×KNC) and dual (2×KNC) Xeon Phi coprocessor(s), and in hybrid mode where host cores and both KNCs are used simultaneously (Hybrid). Results for several nucleotide (DNA) and protein (AA) alignments with different dimensions and partitioning schemes are shown. The execution times are medians of 3 independent runs.

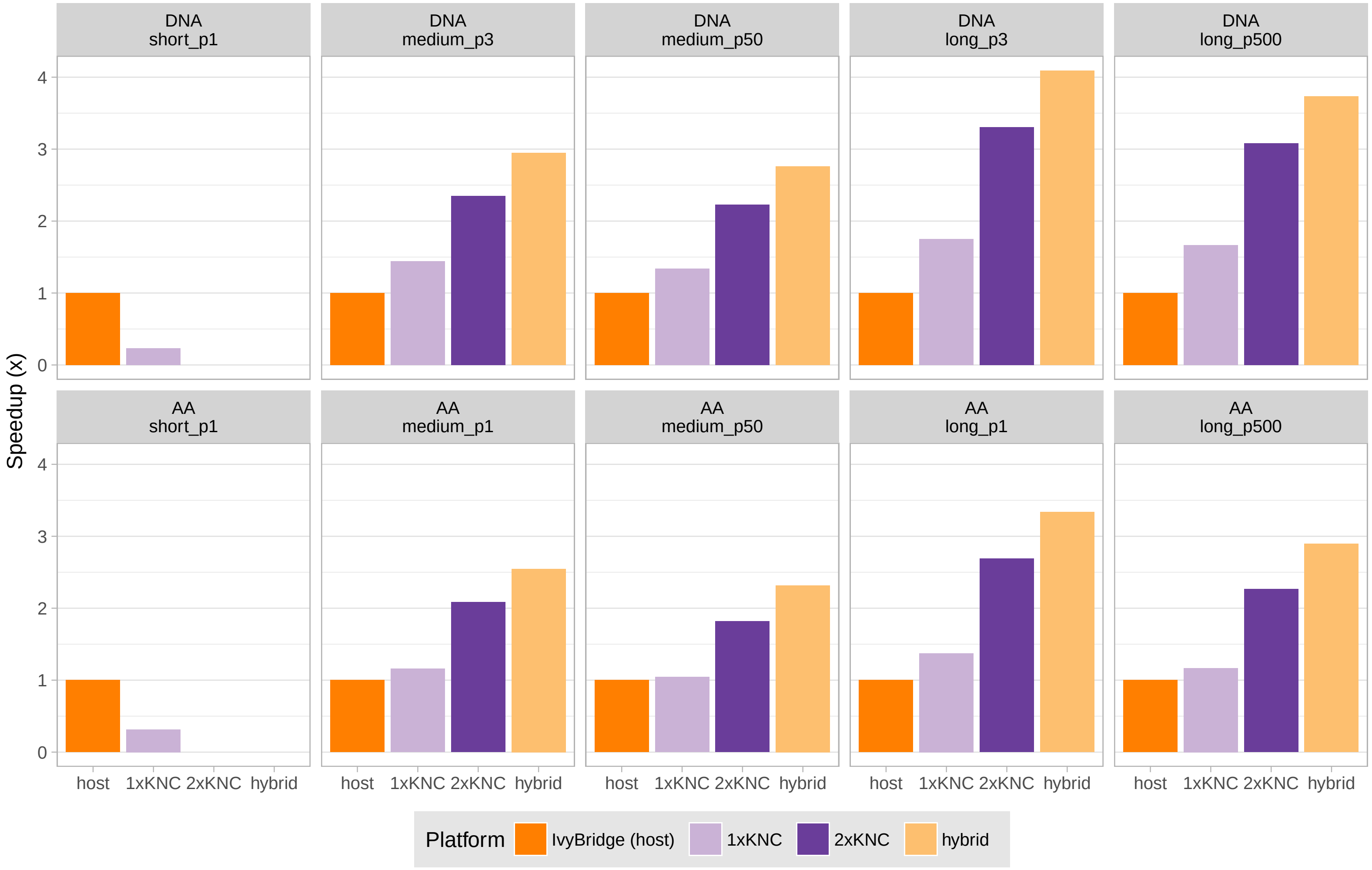


Figure 3.5: `ExaML-KNC` speedups on DNA and AA alignments

3.4.4 Scalability Analysis

In scalability analyses, one usually distinguishes between *strong* and *weak* scaling. Both metrics assess the runtime improvement that can be achieved by using more cores for solving the problem in parallel. However, strong scaling is calculated based on a *fixed* problem (dataset) size, whereas for weak scaling the problem size is increased with the number of cores (i.e., the working set size is fixed *per core*).

More formally, strong and weak scaling efficiency are calculated as follows:

$$S_{strong} = \frac{t_{1,1}}{N * t_{1,N}} * 100\% \tag{3.1}$$

$$S_{weak} = \frac{t_{N,1}}{t_{N,N}} * 100\% \tag{3.2}$$

where N is the number of cores and $t_{1,1}$, $t_{1,N}$, $t_{N,1}$, $t_{N,N}$ are the runtimes for the respective dataset size (1 or N working units, first index) and the number of cores (second index).

On the Intel KNC accelerators, strong scaling of `ExaML` is limited by two factors:

1. The maximum size of an alignment which can be analyzed on a *single* card is constrained by the amount of on-board memory (8 GB for the Xeon Phi 5110).

2. As we distribute an alignment of *fixed* size over multiple co-processors, the number of site patterns per card and per core decreases, and so does efficiency.

In most practical cases, it will therefore be suboptimal to run an analysis on multiple cards, if one single card can handle the data in terms of memory requirements. On the other hand, one might *have to* use multiple *KNC*s if the dataset does not fit into the memory of a single card. To account for this scenario, we evaluated the *weak* scaling behavior of `ExaML-KNC` on large datasets.

As **Figure** 3.6 shows, our implementation scales reasonably well: it attains a parallel efficiency of about 80% on 16 *KNC* cards and about 70% on 32 *KNC* cards. The difference in scalability between DNA and AA alignments amounts to only 1–3% and is therefore negligible.

3.5 Outlook: Intel Knights Landing

In 2016, Intel introduced the second generation of Xeon Phi accelerators codenamed Knights Landing (KNL). Compared to the *KNC*, they offer up to 3× higher peak

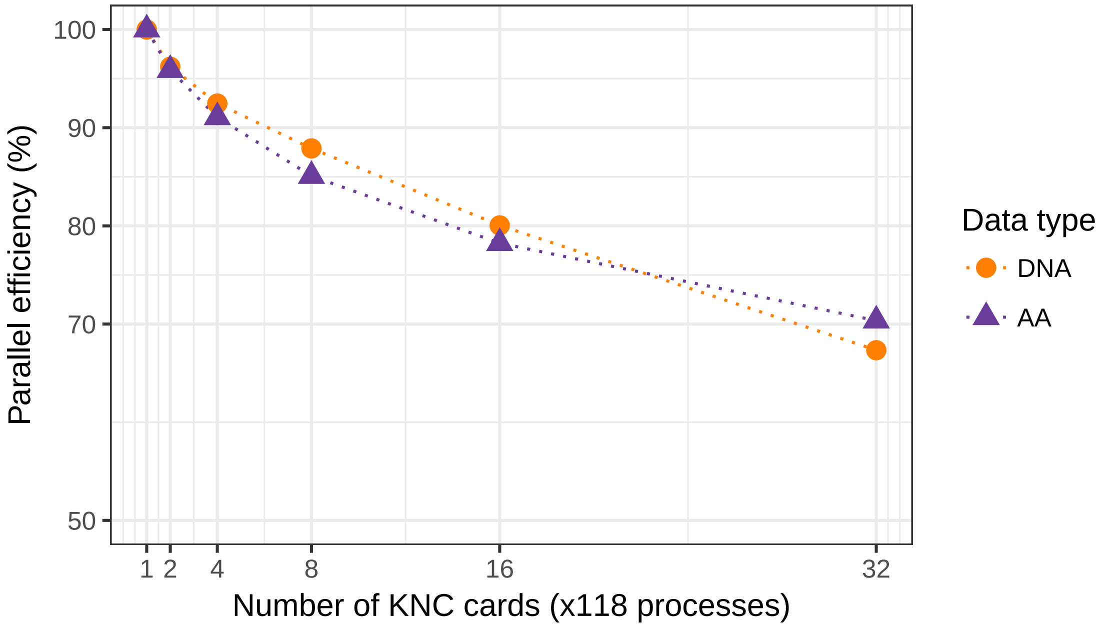


Figure 3.6: Weak scaling of `ExaML-KNC`. Each *KNC* card has been assigned a part of an alignment comprising 50 taxa and 1000k DNA sites or 200k AA sites, divided into 100 partitions.

performance while providing a very similar developer-friendly programming model. In particular, the *KNL* vector unit has the same width as on the *KNC* (512 bit), and the corresponding SIMD instrunction set (*AVX512*) is to a large extent compatible with that of the KNC. Moreover, the *AVX512* instruction set will be used in the recently announced Intel CPUs based on *Skylake-X*/*Skylake-SP* microarchitecture. Therefore, existing *KNC* codes can be easily adapted to these new platforms.

Indeed, `ExaML-KNC` required only minimal modifications to compile and run on *KNL*. Apart from changing the compiler flags, there was only a single vector intrinsic (`_mm512_reduce_gmax_pd`) which was missing in *AVX512* and thus had to be replaced. Our preliminary performance evaluation showed that `ExaML-KNC` runs 2.5× – 4.2× faster on the *KNL* than on the *KNC* (**Figure** 3.7). This is very close to the theoretical peak performance ratio of the two Xeon Phi cards being used (*KNL*: 2662 GFLOPS vs. *KNC*: 1011 GFLOPS, see **Table** 3.1). We therefore conclude that `ExaML-KNC` attains near-optimal performance on the *KNL* even without further specific optimizations.

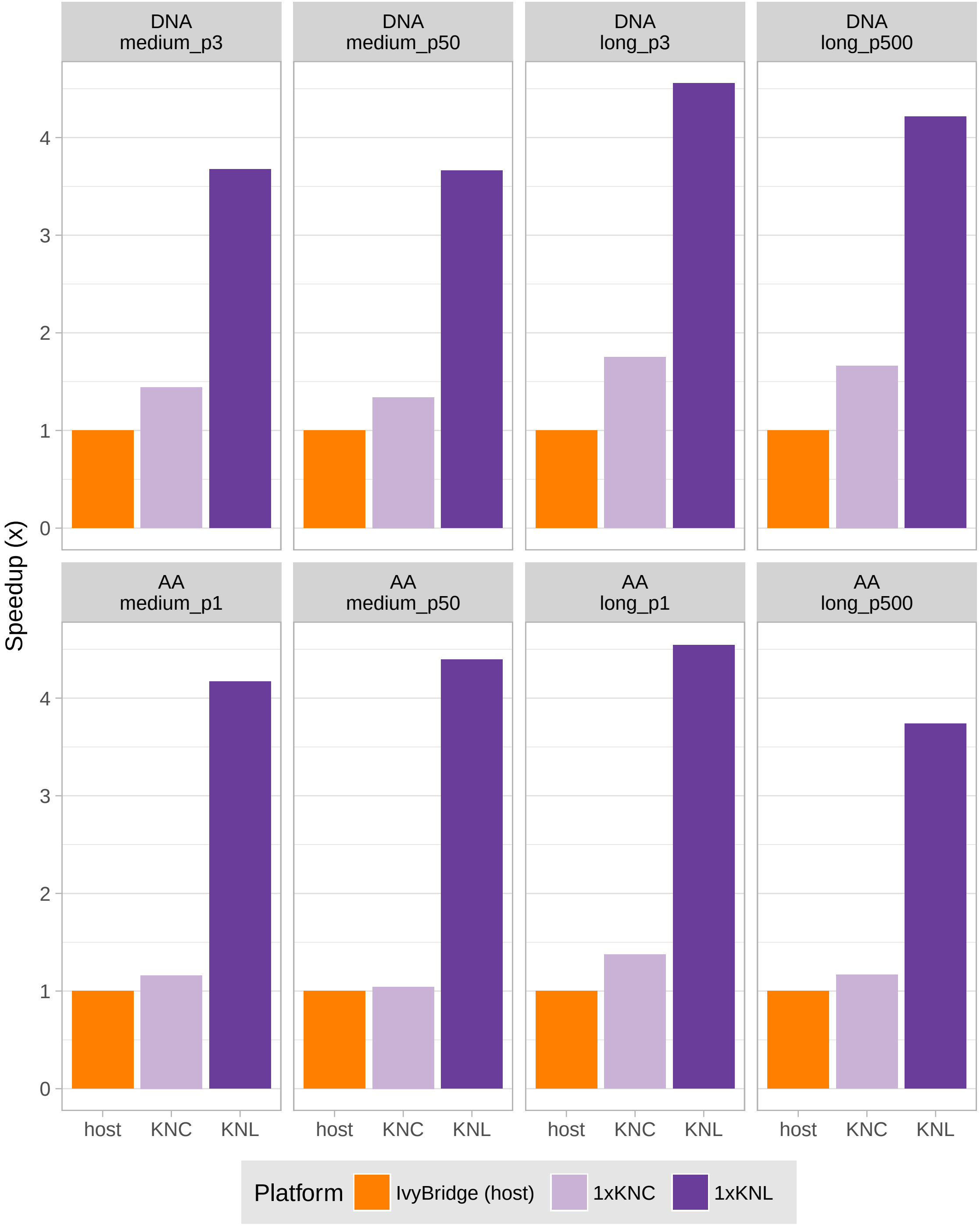


Figure 3.7: Preliminary evaluation of `ExaML-KNC` performance on Intel Knights Landing (KNL) accelerators.

Chapter 4

RAxML-NG: a Next Generation Phylogenetic Inference Tool

This chapter is based on yet unpublished work by Alexey Kozlov, Diego Darriba, and Tomas Flouri. The respective codes are freely available on Github:

- **RAxML-NG:** `https://github.com/amkozlov/raxml-ng`
- **pll-modules:** `https://github.com/ddarriba/pll-modules`
- **libpll:** `https://github.com/xflouris/libpll`

Contributions: Alexey Kozlov designed, implemented and tested the `RAxML-NG` program. Diego Darriba and Alexey Kozlov developed the `pll-modules` library (numerical optimization methods, tree topology operations, SPR-based tree search primitives). Tomas Flouri, Diego Darriba and Alexey Kozlov developed the `libpll` library (low-level PLF computation kernels, MSA and tree I/O utilities). Additional contributions to `libpll` and `pll-modules` (not covered in this chapter) were made by Pierre Barbera and Benoit Morel. Alexandros Stamatakis designed the original tree search algorithm (implemented in `RAxML`) and provided guidance on its re-implementation as well as PLF computation techniques.

4.1 Background and Motivation

`RAxML` [130, 134] is a widely-used software for ML-based phylogenetic inference. The four main papers on `RAxML` have been cited more than 20,000 times according to Google Scholar (as of November 2017). More recently, we released `ExaML` [72, 135], a light-weight version of RAxML that has been specifically optimized to process large phylogenomic alignments and attain high scalability on supercomputers. In particular, ExaML features more efficient parallelization and data distribution approaches. At also supports binary input file format and checkpointing. On the other hand, many important functions of classical RAxML such as bootstrapping and ascertainment bias correction [83, 85] are not implemented in ExaML. At the same time, the old code-base of RAxML (largely inherited by ExaML) that had been growing for over 15 years hampered the implementation of new features. Furthermore, the maintenance overhead (i.e., identifying and fixing bugs) became excessively high.

This motivated us to re-write the software from scratch in a clean and modular way. In a first step, the core likelihood computation kernel was encapsulated in libpll, a flexible, yet efficient library with a well-defined interface (not part of this thesis). Subsequently, we leveraged libpll to re-implement the modified version of the RAxML tree search algorithm in a new tool called RAxML-NG (for *RAxML Next Generation*). In the process of re-implementation, several known bottlenecks were eliminated (**Section** 6.2.3) and the functionality was extended in multiple ways (**Section** 4.2). Moreover, several high-level tree search primitives (e.g., enumerating and scoring SPR moves) were transferred to the pll-modules library, making them available for other phylogenetic codes as well.

4.2 Improvements over RAxML

4.2.1 Flexibility and User-friendliness

New DNA models with flexible parametrization

In RAxML, only the *GTR* model of DNA substitution (see **Section** 2.5.1) was fully supported. Although limited support for *JC69* [63] and *K80* [66] models was added in the latest versions, it was impossible to assign different models to distinct partitions (for instance, *GTR* for partition 1 and *K80* for partition 2).

RAxML-NG supports all 22 'classical' *GTR*-derived models [40], and allows for arbitrary combinations thereof in partitioned analyses. Furthermore, all model parameters (substitution rates, equilibrium frequencies, proportion of invariant sites etc.) can be fixed to user-specified values.

Rate heterogeneity across sites

In addition to the Γ and the *P-inv* (**Section** 2.5.2) RHAS models available in RAxML, RAxML-NG now also supports the *FreeRate* model. The *PSR* model will also be implemented soon.

In RAxML, the number of Γ rate categories was hard-coded and fixed to 4. In RAxML-NG, the number of categories for both the Γ and the *FreeRate* models can be specified by the user, although 4 categories are still the default. In partitioned analyses, RAxML-NG allows to set individual per-partition RHAS models, whereas in RAxML all partitions needed to have the same RHAS model (albeit with independent parameters).

Vectorization and parallelization

RAxML supports multiple SIMD instruction sets (scalar, SSE3, AVX, AVX2) and several parallelization approaches (sequential, PThreads, MPI, and hybrid with MPI *and* PThreads). However, a specific vectorization and parallelization scheme has to be selected at compile-time, by using the respective Makefile. Albeit the compilation process is described in the RAxML manual, for many users without a strong computational background, it proved to be challenging to select the optimal version for their system. Moreover, even if multiple versions were pre-installed in the cluster environment, we repeatedly observed users running the 'default' binary (for instance, the single-threaded non-vectorized one) and thereby wasting computing time and energy.

In order to prevent such inefficient usage and to improve user experience, we implemented runtime SIMD auto-detection in RAxML-NG. This guarantees that the most advanced instruction set provided by the CPU will be used. Furthermore, it allows to build a statically-linked portable binary which can be used across different Unix/Linux systems and x86-64 CPU models. Also, we automatically detect the number of CPU cores and set the default number of threads accordingly. RAxML-NG parallelization is fully configurable in runtime, so all running modes from single-threaded to hybrid MPI/PThreads are available with the same executable (if compiled with MPI support).

4.2.2 Performance and Scalability

Hybrid MPI/PThreads parallelization

RAxML uses the fork-join parallelization model, which requires a synchronization after each PLF kernel call. This approach becomes inefficient when the number of processes is large and/or the synchronization cost is high, as it is the case in large distributed systems. This is much less of concern for RAxML, since it implements the fine-grained parallelization (across alignment sites) only within a single node (with PThreads), whereas the inter-node parallelization with MPI is coarse-grained (across starting trees and bootstrap replicates). The latter requires virtually no communication between processes and is therefore very efficient for small- to medium-size datasets.

However, scalability of this coarse-grained approach to large phylogenomic alignments is limited, not only due to longer execution times, but also because of the high memory requirements for the PLF computation *on the whole alignment*, which can easily exceed the available amount of RAM on a single node. Therefore, `ExaML` implements MPI-based parallelization across alignment sites using an alternative approach [135]. Each thread (or MPI rank) executes its own consistent copy of the tree search algorithm, and there are only two synchronization points: evaluating the tree likelihood at the virtual root (see **Section** 2.6.3), and computing the PLF derivatives during the branch length optimization with the Newton-Raphson method (see **Section** 2.6.4). In both cases of synchronization, a single *MPI_Allreduce* operation is performed (parallel sum reduction followed by the resulting sum broadcast to all MPI ranks). This solution does not only avoid unnecessary, potentially costly synchronizations, but also substantially reduces code complexity.

In `RAxML-NG`, we implement `ExaML`-style fine-grained parallelization, but in a hybrid MPI/PThreads setting. Hence, the parallel reduction is performed in three steps: **(i)** first, all threads of a rank synchronize between themselves, then, **(ii)** an *MPI_Allreduce* operation is performed to compute a global sum across all MPI ranks and distribute it to all ranks, and finally, **(iii)** this global sum is broadcasted to all threads.

In our experience, hybrid MPI/PThreads approach yields better scaling to thousands of CPU cores (see [72] and **Figure** 4.5). Nevertheless, `RAxML-NG` can also be configured to run in pure MPI parallelization mode by setting the number of threads per MPI rank to 1.

Per-rate CLV scalers

When evaluating the likelihood of large trees with Felsenstein's pruning algorithm, there is a risk of numerical underflow during the CLV computation at the inner nodes (see **Section** 2.6.2). In `RAxML/ExaML`, this problem is alleviated by applying *numerical scaling*: if *all* CLV entries for a certain alignment site (and node) are below a threshold $\epsilon > 0$, they are multiplied with a large constant M to prevent underflow. We furthermore keep track of the total number of scaling multiplications being made in a dedicated scaling vector σ that is associated with every inner node. The length of this vector is equal to the number of alignment sites m, and its elements $\sigma_s, s = 1 \dots m$ store the *scaling factors* (or *scalers*) for the CLV entries at site s. During the final likelihood computation at the virtual root, we use scalers to correct for the CLV multiplications.

Although this scaling approach is sufficient to prevent overflow for most datasets, it is still prone to underflow under the Γ RHAS model on very large trees with thousands of taxa [60]. This is due to the fact that the scaling (that is, the multiplication) is applied on a *per-site* basis. In other words, only if *all* CLV entries for *all* rate categories drop below the threshold, the scaling is applied to *all* CLV entries simultaneously. Hence, if conditional likelihoods vary substantially among individual Γ rate categories (which is often the case for low values of the α parameter, that is, for datasets with high rate heterogeneity), then CLV entries for one or more categories can underflow. This is not (yet) a problem, since at least one rate category is guaranteed to have non-zero CLV entries. However, computing the tree likelihood at the virtual root involves the multiplication of two CLVs associated with the respective adjacent nodes, *left* and *right* from the virtual root (see **Section** 2.6.3). Consequently, if for a certain alignment site, there is no single category for which *both* left and right CLVs did not underflow, then the site likelihood computed according to the equations (2.17) – (2.21) will be equal to zero. Obviously, this is an invalid likelihood value resulting from the information loss due to underflow in CLVs. In practice, this prohibits the application of the Γ model of rate heterogeneity (as implemented in `RAxML/ExaML`) to some important empirical datasets (for instance, large marker gene databases such as LTP [161] and SILVA [109], see **Section** 6.4.1).

We solved this problem by introducing the *per-category* scaling mode in `libpll`. In this mode, we perform the underflow check and scaling for each Γ rate category individually. We also extend the scaling vector to the length mK (where K is the number of rate categories), such that it can store all individual per-category scalers $\sigma_{s,c}, s = 1 \ldots m, c = 1 \ldots K$. The log-likelihood at the virtual root can then be computed as follows:

$$\log L_s = \log \left(\frac{1}{K} \sum_{c=1}^{K} \left(L_{s,c} M^{-\sigma_{s,c}} \right) \right) \tag{4.1}$$

For computational convenience, we do not apply the above formula directly, but simplify it in the following way. Let us define the minimum scaler across all categories $\mu_s = \min_c\{\sigma_{s,c}\}$ and the relative per-category scalers $\rho_{s,c} = \sigma_{s,c} - \mu_s$. Then we can re-write (4.1) as:

$$\log L_s = \log \left(M^{-\mu_s} \cdot \frac{1}{K} \sum_{c=1}^{K} \left(L_{s,c} M^{-\rho_{s,c}} \right) \right) \tag{4.2}$$

and further simplify it to:

$$\log L_s = \mu_s \cdot \log(M^{-1}) + \log \left(\frac{1}{K} \sum_{c=1}^{K} \left(L_{s,c} M^{-\rho_{s,c}} \right) \right) \tag{4.3}$$

Finally, we truncate the relative scaler $\rho_{s,c}$ at a certain value (currently: 4), since the overall likelihood contribution of categories with larger scaling factors will be negligible.

Obviously, per-category scaling incurs a certain overhead in terms of both computation and memory consumption (for storing the larger scaling vectors). Therefore, this mode is optional in `RAxML-NG`, but it is enabled by default on datasets with more than 2,000 taxa, where this underflow is likely to occur.

Checkpointing

In a cluster or supercomputer environment, the maximum execution time of user jobs is usually limited (e.g., 24 or 48 hours limits are very common). Therefore, long-running analyses can only be executed if the program has an ability to save its intermediate execution state (*checkpoint*) in a file, and use this information later on to restart from that stage. Checkpointing was supported by `ExaML`, but standard `RAxML` lacked this important feature. In `RAxML-NG`, we implement checkpointing in a more efficient and user-friendly way: checkpoint files are more compact than for `ExaML`, and restarting from a checkpoint is fully transparent to the user (that is, no command line modifications are needed). The latter is particularly convenient in some cluster systems, where the job scheduler is configured to automatically restart user jobs that have been interrupted due to external reasons (hardware failure, maintenance etc.).

4.2.3 Search Algorithm Modifications

In general, `RAxML-NG` implements the same hill-climbing search heuristic as `RAxML` (see [137] and [131] for details). The basic building block of this heuristic is a so-called *SPR round*: all possible subtrees are subsequently removed from the currently best tree T_b, re-insertions into all neighboring branches up to a specified *rearrangement distance* are performed, and corresponding induced topologies T_b' are scored. There are two scoring methods: in the *fast* SPR mode, the log-likelihood of each induced topology with *original* branch length values is computed, whereas in the *slow* mode three branches adjacent to the insertion point are additionally optimized using the Newton-Raphson method. If any of the induced topologies T_b' yields a higher log-likelihood score than the currently best tree, the corresponding SPR move is immediately applied, and we continue the optimization process with the new best tree $T_b := T_b'$.

Briefly, the complete search algorithm of `RAxML` and `RAxML-NG` comprises the following steps:

1. Initial optimization of all branch lengths and evolutionary model parameters

on the starting topology (using log-likelihood improvement threshold of $\varepsilon = 10.0$)

2. Determination of the best rearrangement distance: we perform SPR rounds in the *fast* mode with increasing rearrangement distance to detect the optimal distance value for the dataset at hand. More specifically, we start with a distance of 5, and increase it in steps of 5 until no log-likelihood improvement can be achieved, or until the maximum rearrangement distance is reached. The last (and hence, the highest) rearrangement distance that yielded a log-likelihood improvement is selected as the 'best' distance D_b and is subsequently used in the fast SPR iterations.

3. Intermediate model parameter optimization ($\varepsilon = 5.0$)

4. Fast SPR iterations. We perform SPR rounds in the *fast* mode with a fixed rearrangement distance D_b determined above. We maintain a list BN of 50 (`RAxML`) or 60 (`RAxML-NG`) best-scoring *subtrees* (that is, pruning *nodes*). At the end of the round, every subtree in BN is pruned again, re-inserted into neighboring branches up to a distance of D_b, and scored in the *slow* mode. Now, we maintain another list BT of 20 best-scoring *topologies* resulting from the slow re-insertions. Finally, all trees in BT undergo a full branch length optimization, and a tree with the highest log-likelihood is selected as the new best tree T_b. The iteration process terminates when no better topology was found after an SPR round.

5. Intermediate model parameter optimization ($\varepsilon = 1.0$)

6. Slow SPR iterations. We perform SPR rounds in the *slow* mode, and maintain a list BT of 20 best-scoring topologies. At the end of each SPR round, all trees in BT undergo a full branch length optimization, and a tree with the highest log-likelihood is selected as the new best tree T_b. The rearrangement distance is initially set to 5, and is increased in steps of 5 if no better tree was found after an SPR round. Conversely, if a better tree was found after an SPR round, the rearrangement distance is reset to 5. The iteration process terminates when the maximum rearrangement distance is reached (default: 25).

7. Final model parameter optimization (user-specified threshold, default:$\varepsilon = 0.1$)

In `RAxML-NG`, several implementation details have been changed compared to `RAxML`:

- **Subtree enumeration**: As described above, each SPR round involves pruning and regrafting all possible subtrees of the current best tree. In `RAxML-NG`, we use a simple approach to subtree enumeration: we iterate over a list of *inner nodes* of the tree, and then prune and regraft three subtrees induced by three branches adjacent to each inner node. Conversely, RAxML (implicitly) iterates over *branches*, and performs (at most) two SPRs per each branch (that is, it prunes left and right subtrees). Most importantly, the selection of best-scoring topologies for the *BT* list and nodes for the *BN* list (see above) is also performed on a *per-branch* basis. In other words, at most one node of each adjacent pair can be added to the best-scoring list, and can thus undergo a thorough evaluation at the end of the SPR round (see above). Hence, if *both* adjacent nodes (and respective subtrees) have promising SPRs, one of them will be missed. Most likely, this inefficiency of RAxML is the main reason why it failed to find the best-known ML tree in some of our tests (see **Section** 4.3).

- **Best rearrangement distance**. In RAxML, the *minimum* rearrangement distance was always set to 1, and only the *maximum* distance was increasing. In other words, in the first SPR round, the re-insertions into all branches at the distance between 1 and 5 nodes from the pruning point were attempted, in the second round – at the distance between 1 and 10, and so on. In RAxML-NG, we increase the minimum rearrangement distance as well: that is, the distance interval is 1 to 5 in the first SPR round, 5 to 10 in the second SPR round, and so on. We think that this new approach better reflects the original idea of distance probing, as we want to set a larger best distance (e.g., 10) only if the improvement can not be reached by SPRs with a shorter range (e.g., 5).

- **Branch length optimization**. While libpll and RAxML operates with the actual branch length values $b \in [0, \infty)$, RAxML converts them into a normalized form $\hat{b} = e^{-b/c} \in (0, 1]$, where c is a scaling constant. Obviously, this representation affects the branch length optimization process, and thus can contribute to the discrepancy between the results of RAxML and RAxML-NG. However, it is currently, unclear whether the choice of representation has any systematic effect.

4.2.4 Modularization

Unlike its predecessors RAxML and ExaML, RAxML-NG is designed in a modular way and heavily relies on the functionality provided by two software libraries, libpll and pll-modules (see **Figure** 4.1). In particular, libpll encapsulates an efficient implementation of parsimony score and PLF computation, with specifically optimized kernels for the most frequently used input data types (DNA and protein) and SIMD instruction sets (*SSE3*, *AVX*, and *AVX2*). It furthermore provides a basic tree structure with corresponding operations (for instance, tree traversals) as well as parsers

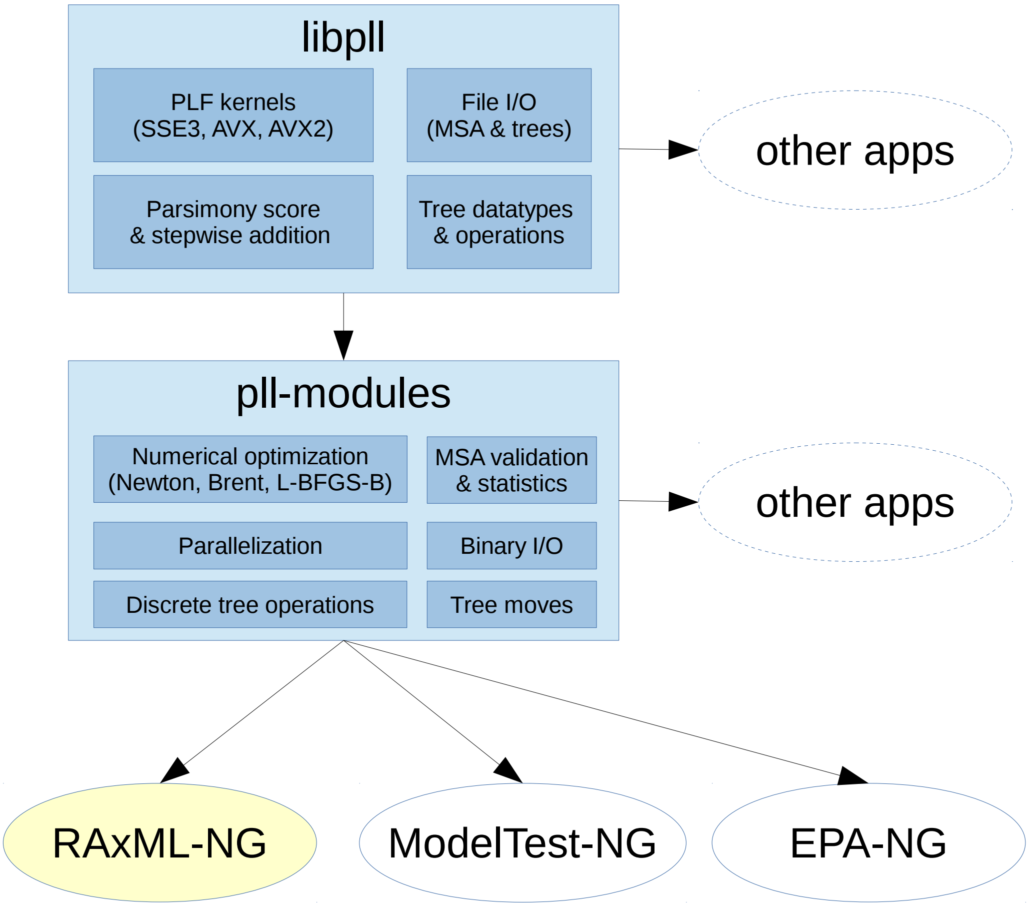

libpll
PLF kernels (SSE3, AVX, AVX2)
File I/O (MSA & trees)
Parsimony score & stepwise addition
Tree datatypes & operations
other apps
pll-modules
Numerical optimization (Newton, Brent, L-BFGS-B)
MSA validation & statistics
Parallelization
Binary I/O
Discrete tree operations
Tree moves
other apps
RAxML-NG
ModelTest-NG
EPA-NG

Figure 4.1: `RAxML-NG` in the context of the phylogenetic software ecosystem at our lab. Encapsulation of shared functionality in `libpll` and `pll-modules` libraries allows for code reuse, efficient testing, and faster integration of new features as well as optimizations.

for several common MSA and tree datatypes (FASTA, PHYLIP, and Newick). The `pll-modules` library builds on top of `libpll` and provides high-level functionality that includes MSA processing utilities, discrete tree operations (consensus tree building, branch support calculation), and numeric optimization methods that are required for model and branch length optimization (currently: Brent, Newton, and L-BFGS-B).

Apart from `RAxML-NG`, two other phylogenetic codes developed in our group rely on `libpll` and `pll-modules` libraries: **(a)** `ModelTest-NG` [28] by Diego Darriba (best-fit model selection for DNA and AA alignments), and **(b)** `EPA-NG` [9] by Pierre

Barbera (efficient and scalable re-implementation of the evolutionary placement algorithm, see **Section** 6.1.3).

This thesis contributed multiple functions that are essential for RAxML-NG, but also relevant for other applications, to the libpll and pll-modules libraries:

(a) numerous low-level optimizations of PLF kernels

(b) support for per-category rate CLV scalers (see **Section** 4.2.2)

(c) flexible parallelization support with callback functions for parallel all-reduce operations

(d) high-level methods for tree likelihood evaluation, model parameter and branch length optimization on partitioned alignments

(e) basic tree search capabilities (enumeration and scoring of all possible SPRs for a given topology, which is the central part of the RAxML/RAxML-NG search algorithm)

(f) MSA validation and statistics calculation

In addition to the benefits for testing and maintenance, the modular structure of RAxML-NG also facilitates integration of new features. This was exemplified by the recent implementation of the *site repeats* optimization technique [69] in libpll (carried out by Benoit Morel and not covered in this thesis). At least three applications developed in our group – ModelTest-NG, EPA-NG, and RAxML-NG– immediately profited in form of reduced memory consumption and/or running times. Another example is the SPR-based tree search functionality which is now also used in the ModelTest-NG code.

4.3 Evaluation

4.3.1 Experimental Setup

Test system configuration

We performed all benchmarks cluster at HITS, which consists of 224 compute nodes with dual-slot Intel Haswell CPUs (see **Table** 4.1). We compared RAxML-NG to three other state-of-the-art ML tree inference tools: IQTree, RAxML, and ExaML (see **Table** 4.2).

System	HITS cluster		
Hardware		Software	
CPU model	2 × Xeon E5-2630 v3	OS	CentOS Linux release 7.2.1511
CPU architecture	Haswell		
Cores	16 @ 2.40 GHz	Compiler	GCC 5.4.0
Memory size	64GB DDR4	MPI	Open MPI 1.10.3

Table 4.1: Hardware and software specifications of test system used for performance evaluation.

Tool	Version	Release date	References
`ExaML`	3.0.19	May 2017	[72, 135]
`IQTree`	1.5.5	June 2017	[21, 98]
`RAxML`	8.2.10	March 2017	[134]
`RAxML-NG`	0.5.1	November 2017	[71]

Table 4.2: ML inference tools used for benchmarking.

Datasets

For the `RAxML-NG` evaluation, we picked 10 empirical protein and DNA datasets with varying number of taxa, alignment sites and partitions (**Table** 4.3). In particular, we included:

(a) four short alignments downloaded from `TreeBase` [121] that were analyzed without partitioning (dna_M7024, dna_M8385, aa_M8630, and aa_M10372),

(b) four medium-size partitioned alignments recently used to benchmark fast ML-based tools by Zhou *et al.* [164] (dna_WickD3b, dna_PrumD6, aa_NagyA1 and aa_WhelA7), and

(c) two genome-scale datasets with thousands of partitions from two recent phylogenomic studies (dna_hymeALL [106] and aa_bird4M [62]).

The number of taxa in our test datasets varies from 48 in aa_bird4M to 767 in dna_M7024. For the partitioned datasets, we used the original partitioning scheme and models as provided in the respective references. For the unpartitioned datasets, we used the *GTR* model for DNA data and the *LG* model for protein data (combined with the Γ model of RHAS). Although we did not perform the formal model testing, *GTR*+Γ and *LG*+Γ models are known to fit most empirical datasets reasonably well [82]. Moreover, since we only compare tree search efficiency and speed (see discussion below), the choice of evolutionary models is of secondary importance.

The memory requirements of the two largest datasets (dna_hymeALL and aa_bird4M) exceed the amount of RAM available on a single node (64GB). Hence, we were unable to analyze them with RAxML and IQTree, as these tools do not support fine-grain parallelization across multiple nodes (see **Section** 4.2.2)

Designator	Data type	# taxa	# alignment sites	# unique patterns	# partitions	Reference
dna_M7024	DNA	767	5,814	3492	1	[108]
dna_M8385	DNA	212	19,972	11,673	1	[154]
aa_M8630	AA	50	21,154	15,022	1	[56]
aa_M10372	AA	169	22,426	18,663	1	[25]
dna_WickD3b	DNA	103	290,718	277,375	8	[153]
dna_PrumD6	DNA	200	394,684	236,674	75	[107]
aa_NagyA1	AA	60	172,073	156,312	594	[94]
aa_WhelA7	AA	70	59,725	58,419	210	[151]
dna_hymeALL	DNA	174	3,011,099	2,248,590	4,116	[106]
aa_bird4M	AA	48	4,432,759	2,341,493	8,000	[62]

Table 4.3: Characteristics of the datasets used for the RAxML-NG evaluation.

Evaluation strategy

For each dataset and ML inference tool, we ran 20 independent tree searches. We used RAxML to generate 20 starting trees per dataset: 10 fully random trees and 10 trees using parsimony-based randomized step-wise addition. We then used those starting trees in the respective replicate runs of RAxML, ExaML, and RAxML-NG. Unlike other evaluated methods, IQTree is designed to use a collection of starting trees (99 parsimony tree + 1 NJ tree by default). Therefore, we did not force IQTree to use a single starting tree, but instead performed all 20 tree searches with the same (default) parameters, but with different random seeds.

We compared RAxML-NG to alternative methods with respect to three performance metrics: search efficiency (using the log-likelihood of the inferred trees as a proxy), inference speed, and scalability to a large number of cores or compute nodes.

The evaluation of the tree inference tools is complicated by the fact that the true evolutionary history is usually unknown, and thus no reference trees are available for empirical datasets. Simulated alignments, on the other hand, do not necessarily reflect the full complexity of real-world data, since they are usually generated under the same model of evolution that is subsequently used for the tree inference. In our evaluation, we use empirical datasets and consider the log-likelihood scores of the inferred trees as a *relative* performance measure for tree search algorithms. Please note that ML inference is based on the theoretical assumption that the tree with the highest log-likelihood represents the biologically most plausible hypothesis. In practice, however, this assumption can be invalidated by issues such as model misspecification, noisy sequence data, and/or insufficient phylogenetic signal in the alignment. In fact, even on simulated data with a known evolutionary model and no sequencing or alignment errors, we repeatedly observed that the true reference tree had a *lower* log-likelihood score than certain alternative – and thus incorrect – tree topologies. This phenomenon is best explained by a lack of phylogenetic signal, since we observed it mainly in short alignments with thousands of taxa from [72] and [86]. Short alignments are generally problematic, since the ML criterion has been shown to be statistically consistent only if the number of alignment sites goes to infinity [157]. These methodological issues, albeit important, are not relevant for the benchmarks we perform in this chapter. Here, we merely compare the *efficiency of search heuristics* implemented in the respective tools.

Another practical complication is that log-likelihood scores reported by different ML inference programs can not be directly compared due to, for instance, different thresholds for minimum and maximum parameter values (e.g., branch lengths, α shape parameter etc.). Therefore, we re-evaluated all trees with `ExaML` (using the `-f E` option) to be sure that the log-likelihood scores are only affected by the tree topology and not by program-specific implementation details.

4.3.2 Results

Search efficiency

We plot the obtained *normalized* log-likelihood scores for all tools and replicate searches in **Figure** 4.2 (small datasets), **Figure** 4.3 (medium-size datasets), and **Figure** 4.4 (large datasets). For better readability, we normalize the log-likelihoods score by subtracting the 'best-observed' log-likelihood score for the respective dataset (among all tools and replicates). More formally, we for every inference tool i and replicate j, we compute the normalized log-likelihood score $nLL_{i,j}$ as follows:

$$nLL_{i,j} = \log L_{i,j} - \max\{\log L_{t,r}\}_{r=1..20}^{t=\{IQTree,ExaML,RAxML,RAxML-NG\}} \tag{4.4}$$

Clearly, $nLL_{i,j} \leq 0$, and the best-observed topology has a normalized log-likelihood score of zero.

Overall, RAxML-NG showed the best search performance, finding the best-known ML tree at least once for all 10 datasets. It is closely followed by IQTree (7 out of 8, two largest datasets not analyzed). RAxML and ExaML searches converged to a local optimum more often, which resulted in a failure to discover the best-known topology for 3 out 8 (RAxML, two largest datasets not analyzed) and 4 out of 10 (ExaML) test datasets. It should be noted, however, that on the aa_bird4M dataset, ExaML on average returned better topologies, and recovered the best-known tree in more replicate searches than RAxML-NG did (5 vs. 3 out of 20).

On two datasets, aa_M8630 and aa_WhelA7, all methods converged to the best-known topology in all 20 replicate searches. These alignments only contain moderate number of taxa (50 and 70), relatively long sequences (>20,000 AA sites), and thus exhibit a very strong phylogenetic signal with a clear, presumably global, optimum. On the contrary, several other datasets (aa_M10372, aa_NagyA1, dna_WickD3b, and dna_PrumD6) seem to exhibit multiple local optima, in which methods searching from a single starting tree – RAxML, ExaML and, to a lesser extent, RAxML-NG– can be easily trapped. These results provide further evidence for the well-known rule-of-thumb: it is important to use multiple starting trees, and include both random and parsimony-based starting topologies.

Finally, on short DNA alignments (dna_M7024 and dna_M8385), all four methods show a substantial variability in results. This is characteristic of the 'rough' likelihood surface without clear optima, which is typical for the datasets with this shape (i.e., many taxa and relatively few alignment sites).

Inference speed

Apart from tree search efficiency, we also benchmarked IQTree, ExaML, RAxML, and RAxML-NG with respect to their computational speed. Since all programs in our evaluation support parallelization, we measured and compared parallel runtimes using 16 threads (1 compute node) for small and medium datasets, and 512 threads (32 compute nodes, only ExaML and RAxML-NG) for the two largest datasets.

RAxML-NG was the fastest program on all datasets but one: on an extremely short DNA alignment dna_M7024 (<3500 patterns), RAxML and ExaML were 1.5× faster, but also returned worse trees than RAxML-NG. For the remaining alignments, the relative performance of the programs exhibits substantial variation. RAxML-NG yields the highest speedups on long AA alignments: for instance, it is 4.5× faster than RAxML on aa_WhelA7, and 2.9× faster than ExaML on aa_bird4M. On DNA data, however, RAxML-NG speedups are much more modest (up to 1.2× on dna_hymeALL compared to ExaML). This discrepancy is not surprising, as PLF computation for DNA data was already highly optimized in RAxML, and thus, additional optimizations in libpll mainly focused on PLF kernels for AA data. On the other hand, the moderate RAxML-NG speedups on DNA dataset are compensated (and partially explained) by the fact that it inferred better trees than RAxML and ExaML.

Finally, RAxML-NG is between 1.2× (dna_M8385) and 2.6× (aa_M8630) faster than the IQTree program, which showed similar tree search efficiency in our analyses. Of note, our findings are consistent with the results of independent evaluation recently carried out by Zhou *et al.* [164, Table S6]. According to their measurements, RAxML-NG was ≈ 2× faster than IQTree, and between 2× (DNA data) and 3× (AA data) faster than RAxML on single-gene alignments.

Please note, that we measured RAxML-NG run times without enabling the recently integrated site repeats optimization [69] (available since version 0.5.0). In our preliminary tests, this optimization yielded additional speedups of 1.1× – 1.6×, and was particularly efficient for DNA alignments with many taxa.

Scalability

We also tested the scalability of RAxML-NG and ExaML on larger number of cores (up to 2048) on two large phylogenomic datasets (dna_hymeALL and aa_bird4M). Because of the high memory requirements of these alignments, they could not be analyzed on a single node of our cluster. Therefore, we have to use the run time on 32 cores (2 compute nodes) as the reference value for assessing scaling efficiency. Scaling efficiency is the percentage of the ideal (linear) speedup that a given program attains in practice. For instance, a speedup of 3 on 4 cores has a scaling efficiency of 75% (see also **Section** 3.4.4).

Overall, RAxML-NG shows better scaling than ExaML (**Figure** 4.5). This can be attributed to the hybrid MPI/PThread parallelization, which already proved to be better suited for large core counts in our previous experiments (**Section** 3.3). Moreover, we obtained superlinear speedups of up to 115% and 140% on the dna_hymeALL and the aa_bird4M datasets, respectively. This effect is in line with our previous observations [3, 72], and can be explained by the improved cache efficiency: when every core has to process a smaller alignment chunk, the respective CLVs still fit in cache, thus reducing the memory access latency. Additionally, increased (accumulated) memory bandwidth of multiple compute nodes can also contribute to the observed parallel efficiency improvements. Despite those positive effects of parallelization, there is a clear optimum (512 cores for dna_hymeALL and 64-128 cores for aa_bird4M), after which using more cores yields lower parallel efficiency. This is expected, since growing communication and synchronization overhead can not be amortized anymore by decreasing amount of computation performed per core.

4.4 Conclusion and Outlook

As we show, RAxML-NG offers improvements in tree inference accuracy and speed over its predecessors RAxML and ExaML. It also compares favorably to the IQTree program, which employs a conceptually different search algorithm. Moreover, due to to the hybrid parallelization approach RAxML-NG scales better than ExaML. Apart from this, with RAxML-NG we resolve several long-standing shortcomings and limitations of RAxML, such as numerical underflow with the Γ model on large datasets, or lack of flexibility with respect to the DNA and RHAS model specification for partitioned alignments. Finally, we replace an unmaintainable legacy code by a modular software that will be easy to maintain and extend.

RAxML-NG is currently under active development. Firstly, we plan to re-implement the most widely-used features of the RAxML program such as topological constraints, bootstopping [104], and the *PSR* model [129] of rate heterogeneity. Secondly, we are now working on the integration with the terraphast library [14] for detection of phylogenetic terraces [122]. Other potential future features include the implementation of fast support measures [84] and advanced mixture models (for instance, PhyloBayes' CAT [81] model).

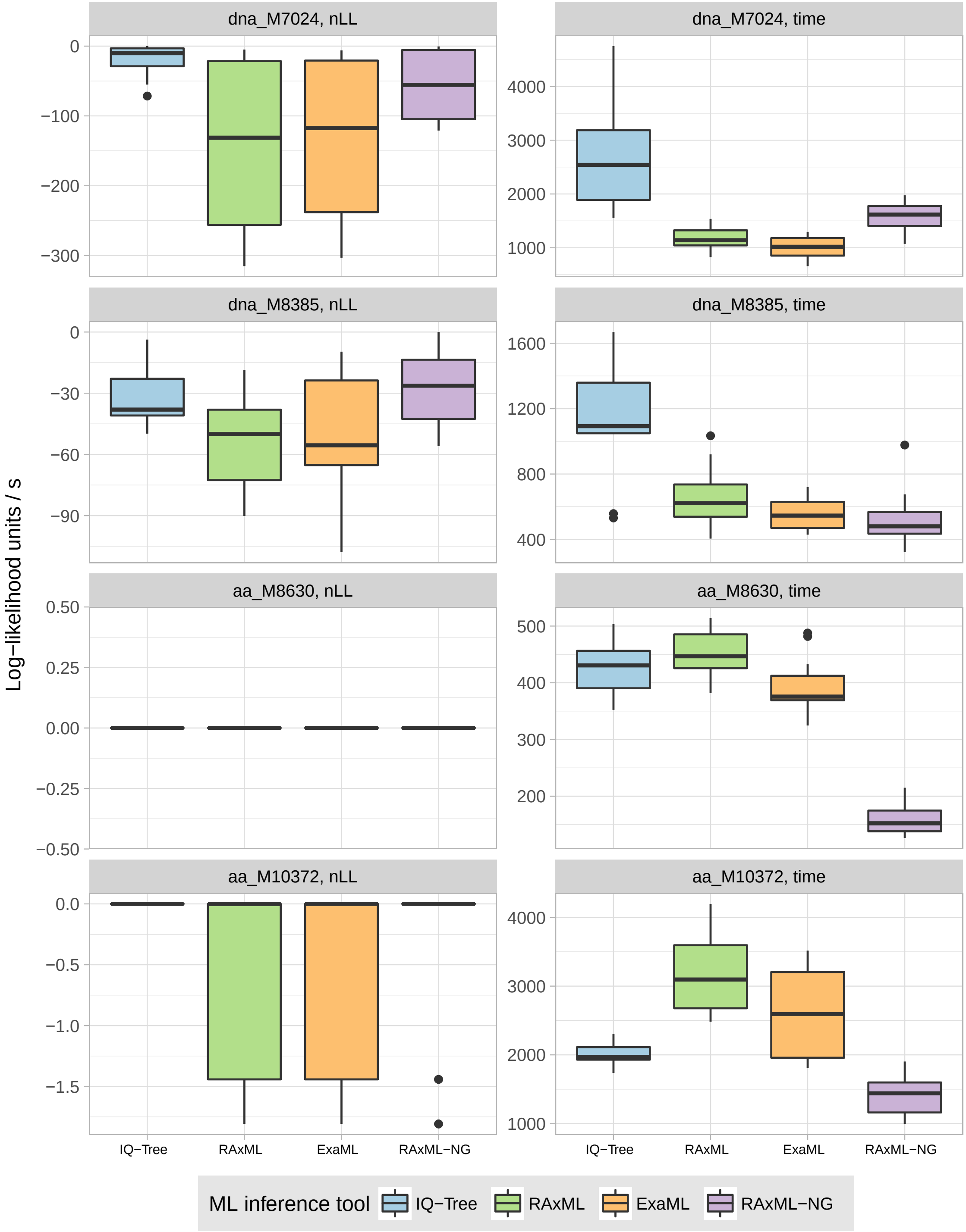


Figure 4.2: Tree search efficiency and speed of `IQTree`, `ExaML`, `RAxML`, and `RAxML-NG` on *small* unpartitioned datasets. *Left:* Distance to the best-observed tree in log-likelihood units (normalized log-likelihood score). *Right:* Wall-clock execution time with 16 threads (1 compute node).

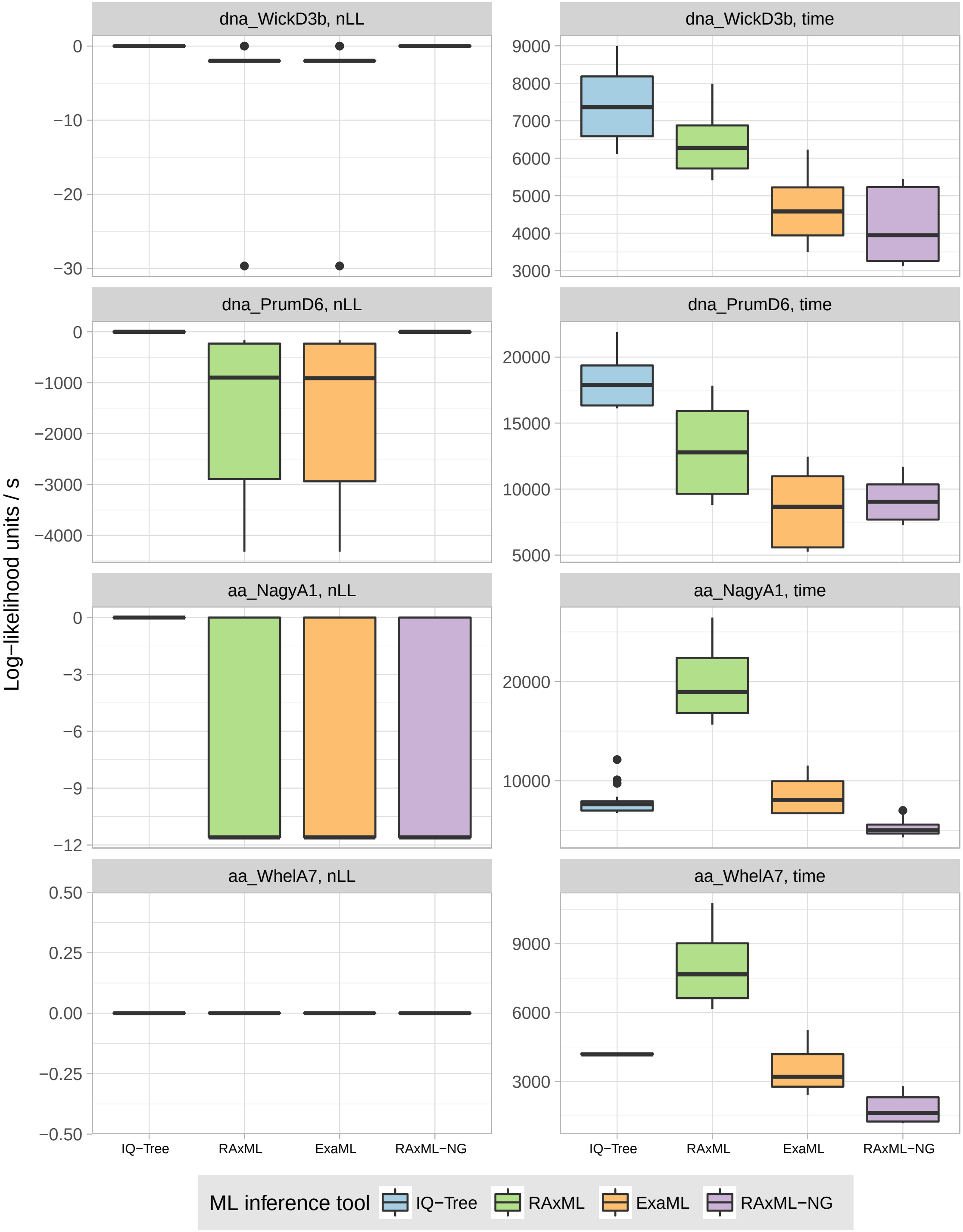


Figure 4.3: Tree search efficiency and speed of `IQTree`, `ExaML`, `RAxML`, and `RAxML-NG` on *medium-size* partitioned datasets. *Left:* Distance to the best-observed tree in log-likelihood units (normalized log-likelihood score). *Right:* Wall-clock execution time with 16 threads (1 compute node).

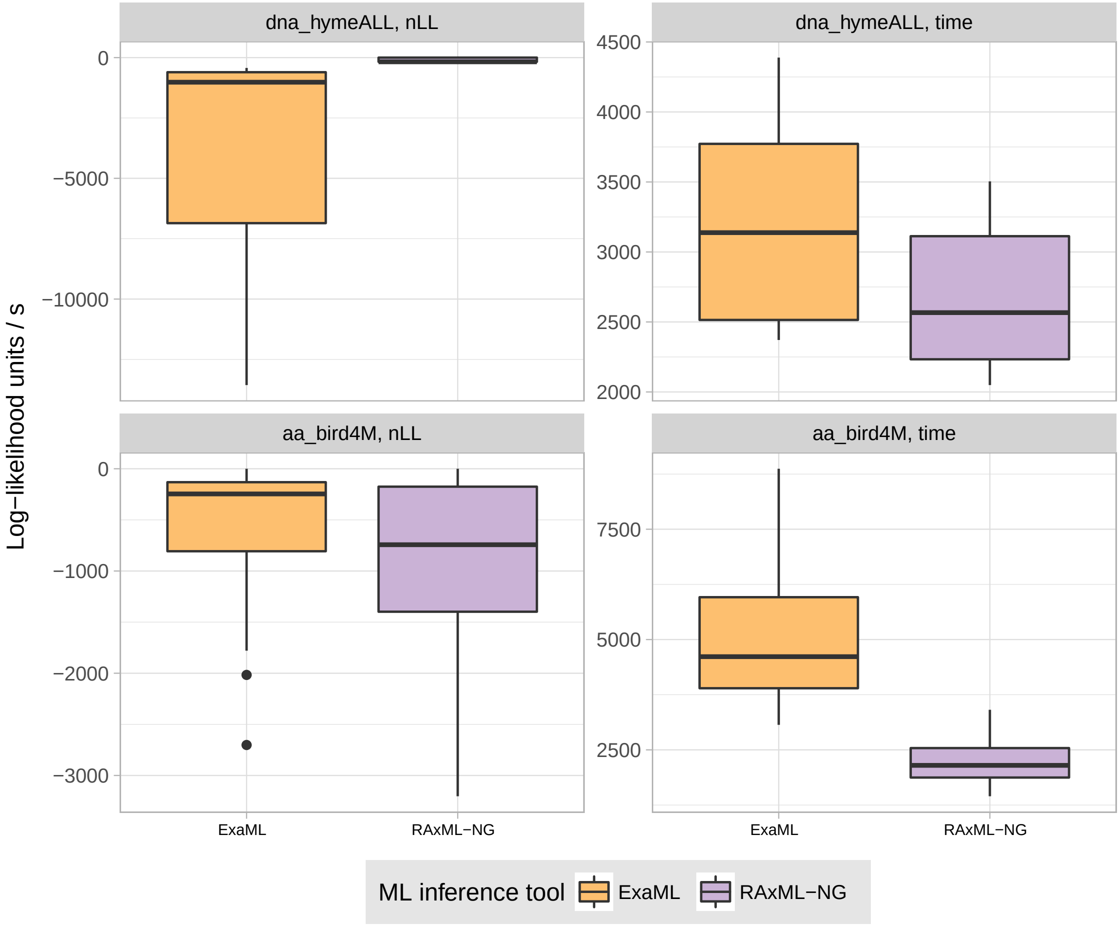


Figure 4.4: Tree search efficiency and speed of `ExaML` and `RAxML-NG` on *very large* partitioned datasets. *Left:* Distance to the best-observed tree in log-likelihood units (normalized log-likelihood score). *Right:* Wall-clock execution time with 512 threads (32 compute nodes).

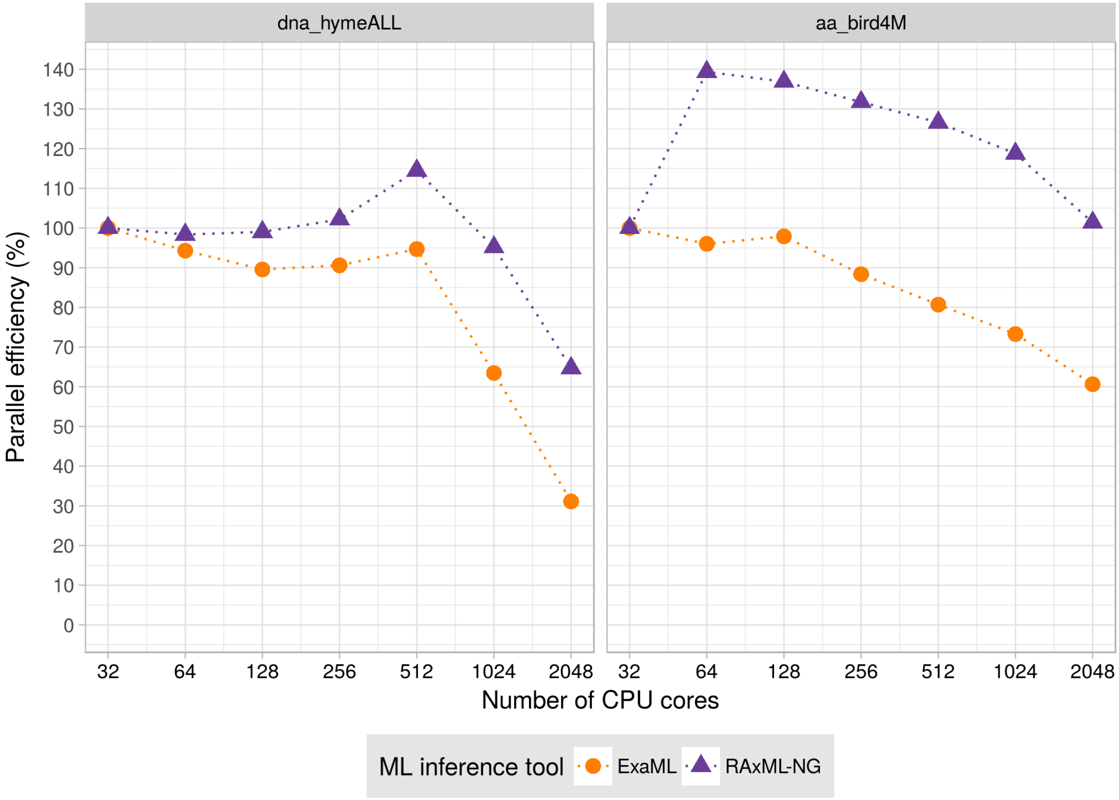


Figure 4.5: Strong scaling efficiency of RAxML-NG vs. ExaML on large phylogenomic datasets.

Chapter 5

Accounting for Sequence Uncertainty in Phylogenetic Inference

Part of the work described in this chapter was first presented in the following conference poster:

- **Alexey Kozlov**, Alexandros Stamatakis: ”Accounting for Sequence Uncertainty in Maximum Likelihood Phylogenetic Inference”, *Poster at Symposium of the SMBE, Vienna, Austria, July 2015.*

Contributions: Alexandros Stamatakis, Nick Goldman, David Posada and Alexey Kozlov designed the sequence uncertainty-aware phylogenetic inference models. Alexey Kozlov developed a proof-of-concept implementation of these models in `RAxML-QS` and `RAxML-NG`, and carried out the evaluation.

5.1 Background and Motivation

Traditionally, phylogenetic inference methods take a multiple sequence alignment (MSA) as input. This MSA is deterministic, that is, at each position the MSA contains a symbol from a discrete alphabet, coding for a specific base (nucleotide/amino acid) or a “gap” (insertion/deletion or missing data). Obviously, this is a simplification, since empirical sequence data characters always exhibit an associated uncertainty because of the sequencing technology deployed and respective induced errors. This uncertainty usually stems from sequencing, read mapping, and/or alignment error, but can also reflect the actual heterogeneity of the biological sample. The latter case is exemplified by alternative alleles of a diploid organism or nucleotide polymorphisms present in the individuals of a sequenced population (e.g., in viral quasi-species). In practice, researchers usually use ad-hoc filtering methods to reduce sequence uncertainty, for instance, by discarding low-quality reads or low-confidence alignment regions [18, 144]. However, such filtering could also remove potentially useful information, and thus become problematic for datasets with closely related sequences or high error rates [145].

These considerations are particularly relevant for single-cell sequencing (SCS) data. This emerging technology allows to elucidate the genomic variation at the level of individual cells, in contrast to classical 'bulk' sequencing which only determines the consensus genome sequence of a population of cells. Among other promising applications, SCS technology can deliver the data needed to reconstruct the evolutionary history of cancer cell populations, which will in turn help to understand patterns of tumor heterogeneity and mechanisms of cancer progression [54, 95, 150]. However, SCS data exhibits particularly high noise levels, making its analysis extremely challenging. Although data quality is expected to improve as the technology matures, it is unlikely to attain the accuracy levels of bulk tissue sequencing due to inherent issues associated with the SCS approach. In particular, genomic DNA is amplified from a single template, so there is no positive averaging effect. This leads to increased error rates and non-uniform read coverage along the genome [95]. Moreover, failure to amplify a stretch of DNA from one of the chromosomes on a pair results in a specific type of false negative error known as *allelic dropout* (*ADO*): a heterozygous genotype cannot be detected since it appears as being homozygous in the sequencing data. Phylogenetic tree inference from noisy single-cell data is a hot research topic, and several tools featuring specialized error correction models have recently been proposed [61, 118, 162].

Probabilistic models of evolution and the ML tree inference framework offer a natural way to incorporate sequence data uncertainty. The most straightforward method of uncertainty specification is the IUPAC ambiguity code [100], which can encode alternative nucleotides at the same alignment position (for instance, character `R` stands for 'either `A` or `G`'). Most ML inference programs support this type of sequence uncertainty. Another simple model of sequencing error that assumes a constant error rate has been suggested by Felsenstein [40]. Recently, Kuhner and McGill evaluated this model in a simulation study [77], concluding that such an error correction improves the branch length estimates, given that this constant error rate is known at least approximately. We extend this work by implementing a ML estimation of Felsenstein's error rate parameter, thus, making the model applicable to cases where the true error rate is unknown. We also suggest more elaborate sequence uncertainty models, which can take into account the error rate heterogeneity among alignment positions. Additionally, we develop a specialized error model for genotype data which models dropout events.

5.2 Implementation

5.2.1 Models of DNA Sequence Uncertainty

We implement three sequence uncertainty models for DNA data:

1. ***UFE* (uniform error)**. Here, we assume a single constant error rate ε, which applies to all alignment positions. Under this model, the likelihood of having a *true* state $x \in \{A, C, G, T\}$ at the alignment row $i \in [1, n]$ and column $j \in [1, m]$, given the *observed* state S_{ij}, can be computed as follows:

$$L_{ij}(x) = P(S_{ij} = y \mid x, \varepsilon) = \begin{cases} 1 - \varepsilon & \text{if } x = y \\ \varepsilon/3 & \text{if } x \neq y \end{cases} \tag{5.1}$$

   Please note, that this is exactly the model that was previously suggested in [40] and evaluated in [77].

2. ***PSE* (position-specific error)**. This is an extension of the previous model, which allows for variable error rates across alignment positions. In particular, each individual position at alignment row (taxon) $i \in [1, n]$ and site $j \in [1, m]$ is assigned an individual error rate ϵ_{ij}. Hence, the likelihood computation from (5.1) is modified as follows:

$$L_{ij}(x) = P(S_{ij} = y \mid x, \varepsilon_{ij}) = \begin{cases} 1 - \varepsilon_{ij} & \text{if } x = y \\ \varepsilon_{ij}/3 & \text{if } x \neq y \end{cases} \tag{5.2}$$

3. ***PSL* (explicit per-state likelihoods)**. In some applications, per-state likelihoods at every alignment position are provided by the upstream software (for instance, variant-calling tools such as GATK [91]). In this case, we can directly initialize per-position and per-state likelihoods with the values given in the corresponding input file (see **Section** 5.2.3):

$$L_{ij}(x) = S_{ijx} \tag{5.3}$$

   where S_{ijx} is the likelihood of state $x \in \{A, C, G, T\}$ at alignment row (taxon) $i \in [1, n]$ and site $j \in [1, m]$.

All three DNA error models (*UFE*, *PSE*, and *PSL*) were initially implemented in RAxML-QS, a modified version of RAxML (code available under https://github.com/amkozlov/raxml-qs). Later on, we ported the *UFE* and *PSL* models into the new RAxML-NG code (see **Chapter** 4). Although the *PSE* model is currently not supported by RAxML-NG, it can easily be integrated upon demand.

5.2.2 Models of Genotype Evolution and Sequence Uncertainty

Genotype data encoding

Most higher organisms are either diploid or polyploid, that is, their cells contain more than just one genome copy in multiple homologous chromosomes. For instance, humans are diploid, and so our cells carry two sets of chromosomes: one inherited from the mother (*maternal*) and another from the father (*paternal*). If both homologous chromosomes encode the same nucleotide at a certain position (e.g., A/A), we call this a *homozygous* site or allele. Conversely, a site where parental and maternal chromosomes differ (e.g., A/C), is called *heterozygous*. Usually, heterozygous sites constitute a very small fraction of the genome sequence (≈1/1000), and they can thus be ignored in classical species-level phylogenetic analyses. However, the signal from heterozygous sites becomes important for inferring relationships between individuals of the same species or even between single cells of the same individual, as in these cases the overall sequence variation is much lower. This motivates the development of evolutionary models for (diploid or polyploid) genotypes.

A genotype can be encoded by a pair of nucleotides x_1/x_2, $x_1, x_2 \in \{A, C, G, T\}$. If we know the assignment of x_1 and x_2 to their corresponding maternal and paternal chromosomes (*phasing*), then genotype is an ordered pair. Thus, we need 16 states to represent all possible *phased* genotypes. With current sequencing technology, however, the phasing is usually unknown. In this case, we cannot distinguish, for instance, between genotypes A/C (maternal A/paternal C) and C/A (maternal C/paternal A). Hence, we can only use 10 states to represent all possible unordered pairs (*unphased* genotypes): AA CC GG TT AC AG AT CG CT GT (for convenience, we will use the notation where nucleotide characters are ordered lexicographically within each pair).

Models of genotype evolution

Now, we can define Markov Chain models of genotype evolution in the same way as for DNA and protein data (see **Section** 2.5.1). In order to reduce the number of free parameters, we will only consider substitution matrices with identical nucleotide mutation rates for all genotypes. In other words, we assume $r_{AC \leftrightarrow AA} = r_{AC \leftrightarrow CC} =$

$r_{A\leftrightarrow C}$. Furthermore, we do not consider double mutations, that is, we set $r_{xx\leftrightarrow yy} = 0.\forall x \neq y$. Finally, we do not model recombination, since we focus on somatic cell evolution, where recombination does not occur. Under these assumptions, we can derive the 'genotype equivalents' for all GTR-based substitution models: for instance, *JC69*, *HKY85* etc.

Model of genotype sequence uncertainty

Here, we describe a genotype uncertainty model proposed by David Posada (pers. comm.), which we will call ***GTE* (genotype error)**. This model has two free parameters: sequencing error rate ε and allelic dropout rate δ. More specifically, ε is the probability that nucleotide x will be observed as another nucleotide $y \neq x$ (either due to amplification error or sequencing error). At the same time, δ is the probability that the amplification of one of the chromosomes has failed (and thus we observe the homozygous genotype defined by the nucleotide in the amplified chromosome). Furthermore, we do not consider multiple sequencing errors within one genotype: for instance, the situation `GT` (true) → `AG` → `AC` (observed) is considered as being impossible. Thus, we set $P(AC \mid GT) = 0$. Under these assumptions, the likelihood of a true genotype x, given the observed genotype y, is computed as follows:

$$L_{ij}(x) = P(S_{ij} = y \mid x, \varepsilon, \delta) = \begin{cases} 1 & \text{if } y = \texttt{'-'}\text{(gap)} \\ P_0(x) & \text{if } x = y \\ P_1(x,y) & \text{if } d(x,y) = 1 \\ \varepsilon\delta/6 & \text{if } d(x,y) = 2 \wedge y \in \mathcal{H} \\ 0 & \text{otherwise} \end{cases} \tag{5.4}$$

where

- $\mathcal{H} = \{\texttt{AA}, \texttt{CC}, \texttt{GG}, \texttt{TT}\}$ is a set of all homozygous genotypes
- $d(x,y)$ is a mutation distance between genotypes x and y:

$$d(x,y) = d(x_1x_2, y_1y_2) = \begin{cases} 0 & \text{if } x_1 = y_1 \wedge x_2 = y_2 \\ 2 & \text{if } x_1 \neq y_1 \wedge x_2 \neq y_2 \\ 1 & \text{otherwise} \end{cases} \tag{5.5}$$

- L_0 is the probability of observing the true genotype x:

$$P_0(x) = \begin{cases} 1 - \varepsilon + \varepsilon\delta/2 & \text{if } y \in \mathcal{H} \\ 1 - \varepsilon - \delta + \varepsilon\delta & \text{otherwise} \end{cases} \tag{5.6}$$

- L_1 is the probability of observing the genotype y with a mutation distance $d(x, y)$ of 1 from the true genotype x:

$$P_1(x, y) = \begin{cases} \varepsilon(1-\delta)/3 & \text{if } x \in \mathcal{H} \\ \delta/2 + \varepsilon/6 - \varepsilon\delta/3 & \text{if } x \notin \mathcal{H} \land y \in \mathcal{H} \\ \varepsilon(1-\delta)/6 & \text{otherwise} \end{cases} \tag{5.7}$$

The *GTE* model is currently implemented in an experimental branch of `RAxML-NG`.

5.2.3 Sequence Uncertainty Specification

For the *UFE* and *GTE* models, the uniform error rates (sequencing error and ADO) can be specified as command line parameters. Alternatively, they can also be directly estimated as free parameters from the alignment data (see **Section** 5.2.5).

For the *PSE* and *PSL* models, we use an enriched MSA file (which we will call *probabilistic MSA* or *pMSA*) to specify the position-specific error rates or per-state likelihoods, respectively. To the best of our knowledge, an established MSA file format which allows to accommodate sequence uncertainty information in appropriate way does not exist. There exist, however, similar formats in related areas of bioinformatics, which can be adapted to our purposes.

In particular, *FASTQ* [23] is a file format commonly used to represent per-base quality scores in read sequencing data. Unfortunately, this information is typically lost (or, more precisely, utilized and not propagated) in the process of read mapping and alignment. Still, we can technically use *FASTQ* to represent a probabilistic MSA, in analogy to FASTA that is commonly used to store conventional, deterministic alignments.

Further, the Variant Call Format (VCF [27]) is a widely used file format to represent genomic variation (in particular, single nucleotide variants or SNPs). Among many other attributes, it allows to specify the likelihood of every variant (genotype). While we have implemented an experimental support for *VCF* input files, it turned out that the interpretation of genotype likelihoods is not always consistent across different variant calling tools (e.g., due to different normalizations). Moreover, the *VCF* format is comparatively complicated, which makes it inconvenient for those users who intend to provide custom likelihoods and thus need to generate the input pMSA on their own. Therefore, we decided to adopt a substantially simpler *CATG* format for specifying per-state likelihood.

We will briefly describe the *FASTQ* and *CATG* file formats below.

```
@taxon1
TCGTCCATCCACGTAAGATCAAGTGACTGAGTATGCATATCTGTACACACCAGTCTAGTGCACTCAAATCCCTAGCTGTTTCTTTCCTAATTG
+
4=B;=@;=:<77687=3?=<<<229:@75;;:;9:81A;5@817924=5579:5848;97;:/<65986-59<54/;4/80+7,3158;3523
@taxon2
TCATCCATCCACGTTAGATCAACTGACTGAATATGCATATCTGTAAACACATGTCCAGTAAACTCAAATGCCTAGCTCTTCCTTCCCTAATTT
+
;70@=;7@6;;?79<<69<19/;4681199>;35A=9=?1/6<7601:::-<3<3=6838020B82979;7B5=>248793.3<084668465
@taxon3
TCATCCTTCCACGTTAGATCAACTGACTAAGTAAGCATATCTTTAAACACATGTCCAGTGAACTCAAATGGCTAGCTCTTTCCTCACTAATTT
+
/?858?9738:4.;/;>6><>7?771:</..6B69:?<:94;268.49=9>?96>447;0:753@B31;8<3B6860:=/285=3:?E/=@;@
@taxon4
TCGTGTAGCTATGTAGGATCAGCCGAGTATGTATGTATATCATTAATCACAGGCCCAGCGAATTGAAAGGCCTAGCTCTGTCCCCGGAAATGT
+
<788<89;?3@1.=<5=5>46,/6767=5@09845878/@==+:6699>?;2:3958:=3244=78<>-96B<77199339=67@249;76:8
@taxon5
TCTTCCCGCAACGTCGAATTAACGGAGTGAGTATGCGGACCTGTAAACACAGGTCCAGGGAATTGAAATACGTAGGTCTGTCTTCGGTTATTT
+
9;55<1.@8>979<445A88+-9390=3554758=@788525947::<>9942998E;?9:2@<5/593>769:5:=:>6277A7998687::
```

Figure 5.1: Sample FASTQ file

FASTQ file format

A *FASTQ* file contains four lines for every sequence (**Figure** 5.1). The first line starts with the special character '@' (commercial at) followed by a sequence identifier. The second line contains the actual sequence, and the third line is a delimiter (a single '+' symbol). Finally, the forth line contains per-base *Phred* quality scores [37] encoded as ASCII characters. The *Phred* quality score is computed as $Q := -10 \log_{10} \varepsilon$, where ε is the probability of the respective base being wrong. Thus, we can calculate the per-position error probability from the respective quality score as follows:

$$\varepsilon_{ij} = 10^{-\frac{Q_{ij}}{10}} \tag{5.8}$$

CATG file format

The *CATG* format is a simple text-based format for representing sequence uncertainty, which was originally proposed by Deren Eaton, the author of the PyRAD software [31]. The format is similar to 'transposed' PHYLIP (alignment sites are given in rows instead of columns), and allows to specify per-state likelihoods for each alignment position.

A *CATG* file starts with a two-line header: the first line contains the number of taxa (n) and alignment sites (m), and the second line – a tab-separated list of n taxon names. The following m lines contain the actual alignment data for sites

```
5 6
taxon1   taxon2   taxon3   taxon4   taxon5
TTTTT   0.1,0.1,0.3,0.5   0.1,0.3,0.2,0.4   0.3,0.3,0.0,0.4   0.0,0.2,0.1,0.7   0.3,0.3,0.0,0.4
TTTGG   0.0,0.0,0.3,0.7   0.2,0.2,0.1,0.5   0.1,0.3,0.1,0.5   0.3,0.0,0.5,0.2   0.1,0.1,0.4,0.4
TTTAT   0.1,0.3,0.1,0.5   0.3,0.3,0.1,0.3   0.2,0.0,0.3,0.5   0.5,0.0,0.1,0.4   0.2,0.2,0.2,0.4
TTTTT   0.1,0.0,0.2,0.7   0.0,0.1,0.3,0.6   0.3,0.2,0.1,0.4   0.1,0.3,0.2,0.4   0.0,0.3,0.2,0.5
TTTTG   0.2,0.2,0.1,0.5   0.2,0.2,0.2,0.4   0.1,0.2,0.3,0.4   0.3,0.0,0.2,0.5   0.0,0.1,0.7,0.2
TCCTG   0.3,0.0,0.3,0.4   0.0,0.6,0.0,0.4   0.3,0.4,0.1,0.2   0.3,0.1,0.0,0.6   0.0,0.3,0.5,0.2
```

Figure 5.2: Sample CATG file

1 to m. In these lines, columns 2 to n contain comma-separated lists of per-state likelihoods for the respective taxa (in the same order as given in the 2nd header line). The first column contains a consensus state for each taxon in the IUPAC encoding. Although this consensus information is redundant, it improves the readability of the file.

5.2.4 Internal Representation of Probabilistic sequences

In standard `RAxML`, only inner nodes have full CLV vectors associated to them. For the tip nodes, trivial 'pseudo-CLVs' can be computed on-the-fly from the respective discrete aligned sequence (**Figure** 5.3). For instance, given a character `C` in the alignment, the corresponding pseudo-CLV entry will contain a value of 1.0 for state `C` and zeros for all remaining states (`A`, `G`, and `T`).

In `RAxML-QS`, we also allocate CLV vectors for the tip nodes. These tip CLVs are initialized with the per-state likelihood values computed according to equations (5.1) – (5.3). If a RHAS model is used (see **Section** 2.5.2), then the tip likelihoods are replicated for all K rate categories:

$$\forall c \in [1, K] : CLV^{(i)}_{j,c,x} = L_{ij}(x), \quad x \in \{A, C, G, T\} \tag{5.9}$$

where $i \in [1, n]$ is the number of the tip (sequence) and $j \in [1, m]$ is the alignment site.

5.2.5 Estimating Uniform Error Rates

In order to obtain a ML estimate for the uniform sequencing error rate parameter, we apply Brent's root-finding method [17]. This method was already available in `RAxML`, where it is used to optimize the α shape parameter of the Γ distribution (in the Γ RHAS model).

After each iteration of the Brent's algorithm, we re-compute all per-state likelihoods according to equation (5.1) using the new value ε' of the error rate ε. Then, we update the CLVs according to equation (5.9).

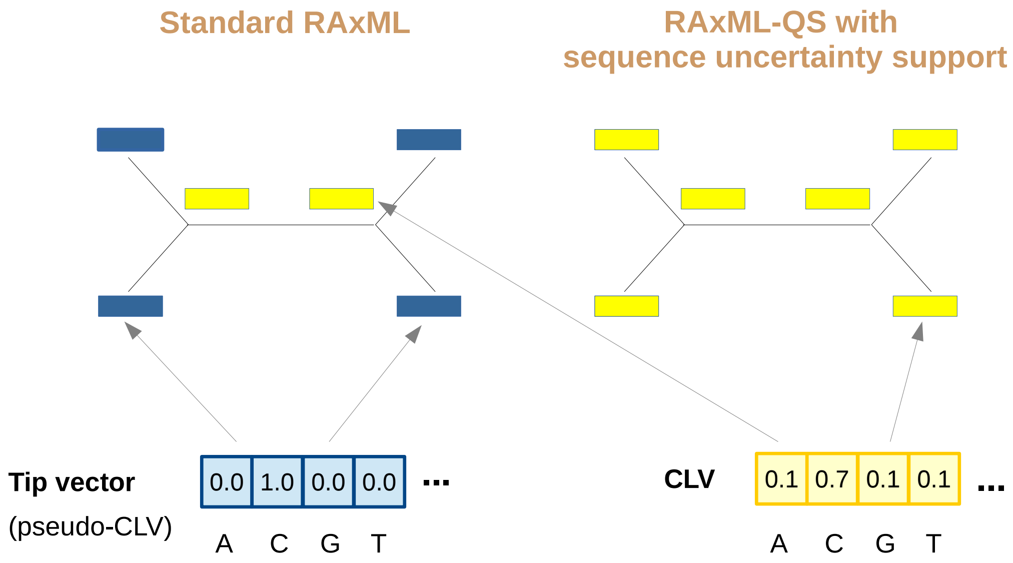


Figure 5.3: Tip CLV representation in standard `RAxML` *(left)* and in the modified version with sequence uncertainty support *(right)*.

For the *GTE* model, we use the same approach to estimate the ADO rate.

5.3 Evaluation

5.3.1 Experimental Setup

DNA data

We evaluate our sequence uncertainty models on five datasets, including both simulated and empirical data (**Table** 5.1).

In particular, we used `INDELible` [42] to simulate three datasets with varying characteristics:

(a) short, slow-evolving sequences (simSLOW, mutation rate = 0.34)

(b) short, fast-evolving sequences (simFAST, mutation rate = 1.00)

(c) longer, slow-evolving sequences (simLONG, mutation rate = 0.34)

All three datasets were simulated under the Kimura substitution model (*K80* [66]) with a transition/transversion ratio of $\kappa; = 2.0$. We chose this particular model because it was used in the aforementioned study by Kuhner and McGill [77]. Also, the *K80* model offers a good trade-off between model complexity and biological realism: it is not as restrictive as the Jukes-Cantor model, but also not as parameter-rich as the full *GTR* model and thus easier to explore in simulation. Furthermore, we simulated rate heterogeneity across sites according to the Γ model with the shape parameter $\alpha := 1.0$ (this value corresponds to the moderate level of rate heterogeneity commonly observed in empirical alignments). For each dataset, we generated 100 replicate alignments by running `INDELible` with varying random number seeds.

Furthermore, we included two empirical datasets: a small set of 23 aligned HIV *pol* region sequences (realHIV), and a larger concatenated alignment of 34 genes from 125 mammalian species (realMAM).

Designator	Organism group, gene	# taxa	# sites	Reference
simSLOW	(simulation, low mutation rate)	50	2,000	
simLONG	(simulation, low mutation rate)	50	20,000	
simFAST	(simulation, high mutation rate)	50	2,000	
realHIV	HIV-I virus, pol	23	2,841	[160]
realMAM	Mammals, multiple genes	125	29,149	[136]

Table 5.1: Characteristics of the datasets used for the sequence uncertainty model evaluation.

For each dataset and replicate, we generated modified alignments with randomly introduced errors at varying levels (mean error rate ε of 1%, 5%, and 10%) using the following procedure. For every alignment position, we draw an error probability ε_{ij} from the exponential distribution with the scale ε, that is, $\varepsilon_{ij} \sim Exp(\frac{1}{\varepsilon})$. In the modified alignment, the original base at the position (i, j) is substituted with probability ε_{ij}. We also convert the true position-specific error probabilities into *Phred* quality scores, and store them in a *FASTQ* file (see **Section** 5.2.3) for later use in the *PSE* model evaluation. We use the exponential distribution of error probabilities since it roughly reflects the empirical error profile of the modern sequence machines (e.g., Illumina [58, 147]). Still, this is a simplification which ignores multiple aspects of empirical data, such as non-uniform quality across positions in the read and effects of downstream processing (assembly/mapping and alignment).

In our evaluation, we ran `RAxML-QS` in four different modes:

(a) STD: standard ML tree inference without error correction

(b) eUFE: *UFE* model with ML estimate of the uniform error rate,

(c) **tUFE**: *UFE* model with the true error rate used to generate the alignment as input parameter,

(d) **tPSE**: *PSE* model with the true position-specific error rates used to generate the alignment (as stored in the *FASTQ* file).

In all cases, we used the *GTR* substitution model with empirical equilibrium base frequencies and the Γ RHAS model.

We assess the accuracy of phylogenetic inference by computing the Kuhner-Felsenstein (KF) distance (see **Section** 2.3) to the true tree (simulated datasets) or to the best-known ML tree (empirical datasets). We use the `DendroPy` library [143] for the KF distance calculation. Furthermore, we assess the accuracy of ML error rate estimates in the *eUFE* mode.

Genotype data

We used `coaltumor` (coalescent simlulator of tumor genealogies by David Posada, `https://github.com/dapogon/coaltumor`) to generate sequence data for 20 tumor cells under varying ADO rates (0%, 10%, 30%, 50%) and sequencing error levels (0%, 0.01%, 1%, 10%). We generated 200 replicate alignments for each condition. We used the HKY substitution model with $\kappa := 2.0$ and the Γ model of rate heterogeneity with $\alpha := 1.0$.

We evaluate the accuracy of both error rate estimation and phylogenetic inference under several GTR-based substitution models. In the following, we only report results for thee best-performing models, both without error correction (*JC69*, *K80*, and *HKY85*) and with the *GTE* error model (*JC69+E*, *K80+E*, and *HKY85+E*).

Finally, we compare the accuracy of our approach with the state-of-the-art method for cell phylogeny reconstruction, `SiFit` [162].

5.3.2 Results

Estimating the uniform error rate

On all DNA datasets tested, the ML optimization approach yielded an accurate approximation of the true error rate values (**Figure** 5.4). As expected, better estimates were observed for the simulated dataset with longer sequences (simLONG), as it contains more abundant signal for the estimation.

On the genotype data, ML estimates for both the sequencing error and the ADO rate were accurate for moderate error levels, but showed large deviations on the most distorted datasets with 10% sequencing error and 50% ADO, respectively (see **Figure** 5.5 and **Figure** 5.6). From the three evolutionary models tested, *K80+E* and *HKY85+E* were superior to the simplistic *JC69+E* model. Please note, that

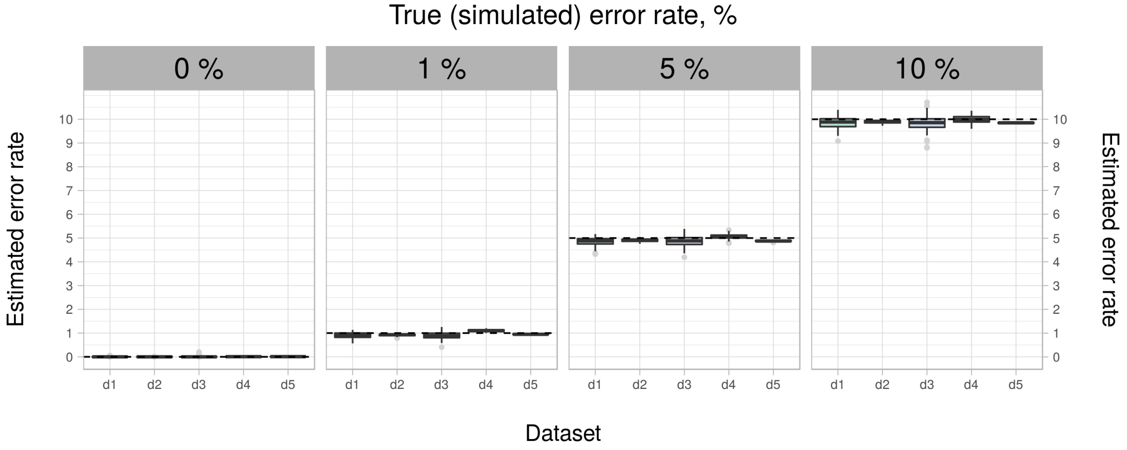


Figure 5.4: ML estimates of the uniform error rate from DNA alignment data. Black dashed line indicates the true (simulated) error rate.

the term 'error rate' is ambiguous for genotype datasets. In the simulations, we set the target *DNA sequence* error rate ε_{DNA} and not the genotype error rate. On the other hand, ML estimation does work on the genotype data and reports the *genotype sequence* error rate as $\varepsilon_{GT} = 2\varepsilon_{DNA}(1-\varepsilon_{DNA})$. Hence, for instance, a simulated DNA error of 10% corresponds to a true genotype error of 18%. In order to avoid the confusion, we indicate both the DNA and the genotype (GT) error rates in **Figure** 5.5.

Impact on phylogenetic inference accuracy

On the DNA datasets, we observe negligible topological deviations due to induced sequence error (data not shown). However, branch length estimates were substantially affected by the errors: on the simSLOW, the simLONG, and the realMAM datasets, the Kuhner-Felsenstein (KF) distance ([76], **Eq.** (2.3)) to the true tree increases from <0.1 on the original error-free alignments to 0.5–0.7 (5% induced error) to 0.8–1.4 (10% induced error). Although this effect is less pronounced on the simFAST and the realHIV datasets, the KF distance increases by a factor of 3 to 5 on alignments with 10% induced errors (as compared to 0% error).

Using the error model allows to revert this decrease in branch length accuracy to a large extent: even with 10% sequence error, deviations from the true branch lengths are only slightly above the respective values for the error-free alignments. Remarkably, the *UFE* model performs equally well regardless of whether the true error rate is specified or if it is directly estimated from the alignment data.

On the other hand, a more elaborate *PSE* model is only marginally better, and only so for some datasets and error levels (for instance, realMAM with 10% error). This can possibly be attributed to the simplistic error profile used in our simulations (exponentially distributed position-specific error rates). Hence, it remains to be seen whether the information about position-specific error probabilities can be still useful with the more complicated error profiles we suspect to occur in empirical data.

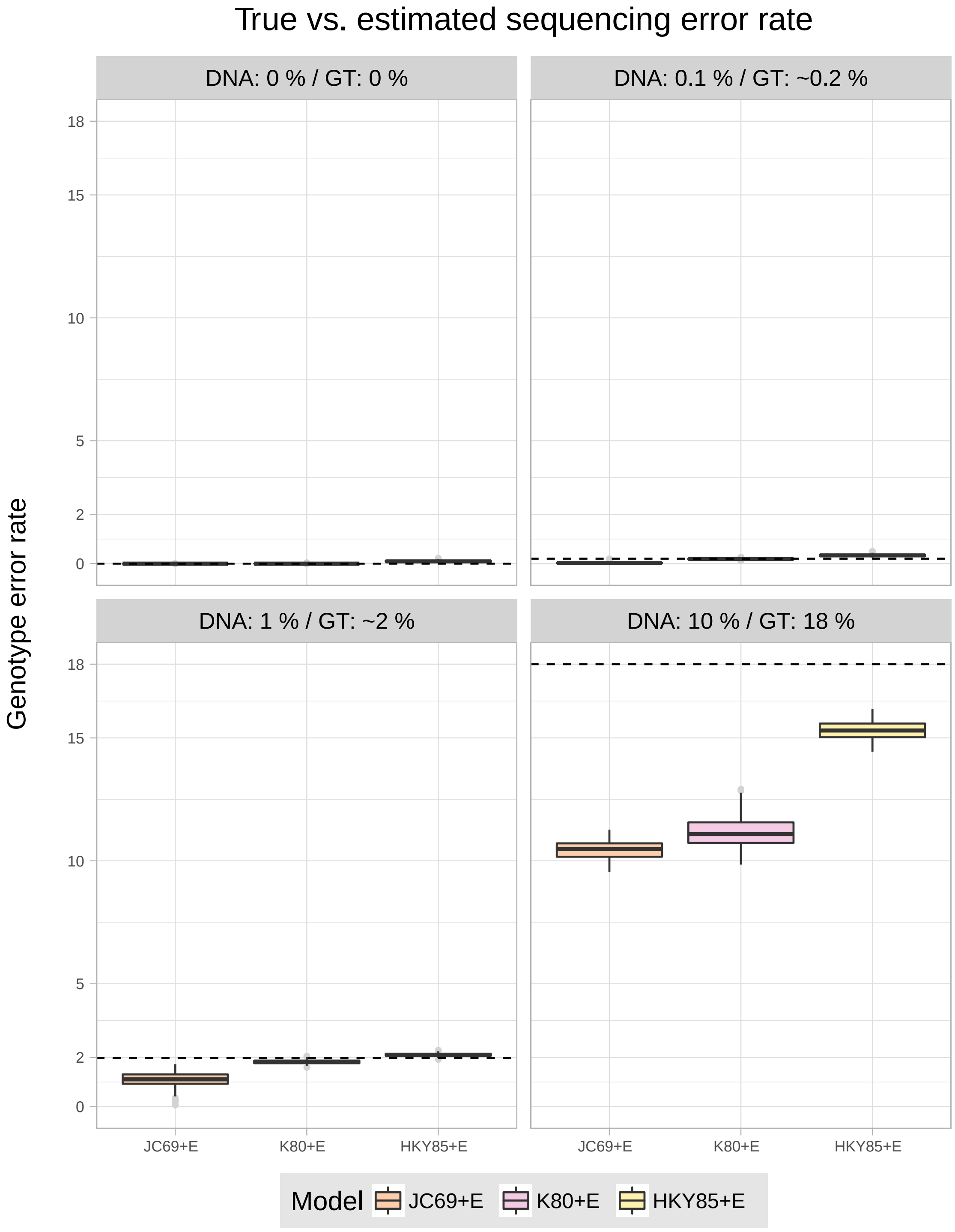


Figure 5.5: ML estimates of the sequencing error rate from genotype alignment data. Black dashed line shows the true (simulated) error rate.

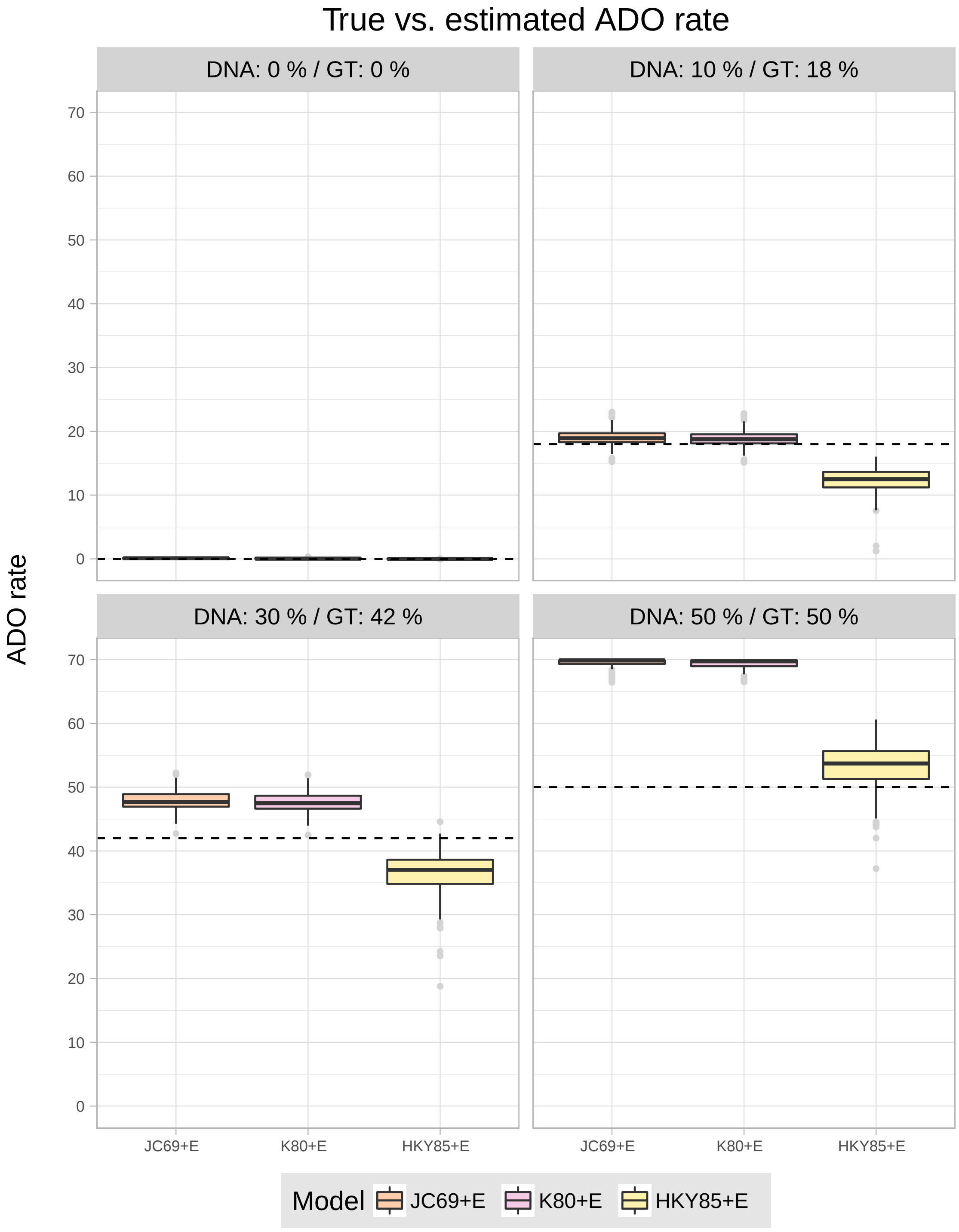


Figure 5.6: ML estimates of the ADO rate from genotype alignment data. Black dashed line shows the true (simulated) ADO rate.

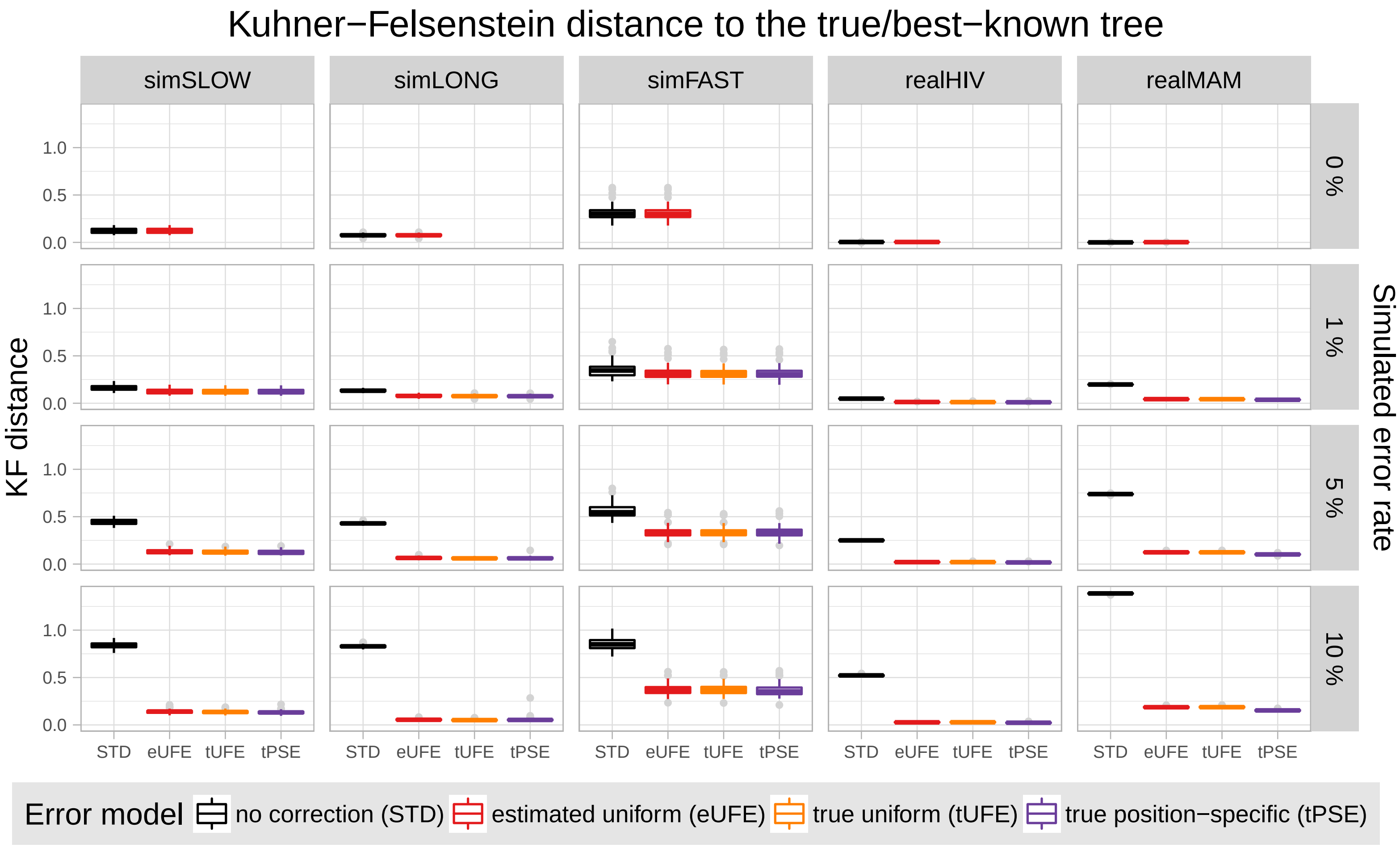


Figure 5.7: Phylogenetic inference accuracy with and without accounting for sequence uncertainty (DNA data).

On the more challenging genotype alignments, tree inference under the *GTE* error model is clearly superior in terms of topological accuracy (see **Figure** 5.8 and **Figure** 5.9). This effect holds for all respective model pair comparisons (that is, *JC69* vs. *JC69+E*, *K80* vs. *K80+E*, and *HKY85* vs. *HKY85+E*) as well as for both modelled error types (sequencing error and ADO). More specifically, we assessed the tree inference accuracy by computing the normalized RF distance (nRF, see **Section** 2.3) between the true tree used for simulation and the respective inferred tree.

On error-free alignments, all models yield very similar results, meaning that error-aware models do not decrease accuracy in the absence of errors. As the error rate increases, models with *GTE* attain better results, although this effect is more accentuated for ADO than for sequencing error. For instance, with 30% ADO the *HKY85+E* model yields an average nRF of 0.14 (averaged over all 200 replicates), whereas the corresponding model without error correction, *HKY85*, has an average nRF of as much as 0.29.

Of the three substitution matrices tested, *K80+E* shows the best results. It is somewhat surprising that it even outperforms the *HKY85+E* model, that is, the true model under which the data was simulated. One possible explanation of this phenomenon is that the empirical genotype frequencies computed from the alignment represent a very poor estimate of the actual equilibrium frequencies, since the evolutionary process has not yet reached stationarity. Therefore, *K80+E* model which assumes equal genotype frequencies is more realistic in this setting.

Finally, all three models incorporating sequence uncertainty consistently outperformed `SiFit`, but the difference only becomes pronounced for the high error rates. For instance, with 50% ADO *HKY85+E* has an average nRF of 0.25 as compared to 0.32 for `SiFit`.

5.4 Conclusion and Outlook

In this chapter, we propose and evaluate several error models which account for uncertainty in DNA and genotype sequences. We show that, although ML tree inference is generally robust to noise, and can recover the correct topology under moderate error levels, explicit error modeling can nonetheless improve the accuracy of branch length estimation. In this respect, we corroborate previous findings [77], and generalize them to more datasets, models, and inference tools. At the same time, we show that on noisy single-cell sequencing data explicit error models can potentially yield more accurate tree topologies. Moreover, in our preliminary tests, our approach outperforms the most recently published tool for phylogenetic inference from singe-cell data (`SiFit` [162]).Finally, we integrate our sequence uncertainty models in the widely used phylogenetic inference software (`RAxML`/`RAxML-NG`), thus making them available to a broad research community.

In general, we consider cell tree inference from noisy single-cell sequencing data as the most promising application for sequence uncertainty models. Despite encouraging preliminary results, there are multiple potential improvements to our approach. For instance, using genotype likelihoods provided by variant callers such as GATK [91] or OncoNEM [118] represents a potentially better alternative to ML estimation of uniform error rates. Furthermore, applying nonstationary Markov models [67] appears justified in this setting, since equilibrium state is unlikely to be reached within the short timespan of cancer cell evolution.

Apart from that, position-specific error models can be used in the context of phylogenetic placement (see **Section** 6.1.3). In this scenario, the gene assembly step is missing, and raw reads are directly mapped to the reference alignment. Therefore, the original *FASTQ* quality scores remain readily available in the MSA and can thus be exploited for estimating sequence uncertainty.

Another potential research direction is to exploit the alignment quality scores provided by probabilistic aligners. For instance, HMMER [32] can output per-column alignment confidence scores, and FSA [16] estimates the probabilities of pair-wise homology for all candidate character pairs. Incorporating this information into ML-based phylogenetic inference methods will allow for 'smoothing' or 'down-weighting' unreliable or ambiguous alignment regions. This approach will provide a reasonable alternative to the 'hard' filtering methods commonly used today, which exclude such regions based on mostly arbitrary thresholds and criteria [145].

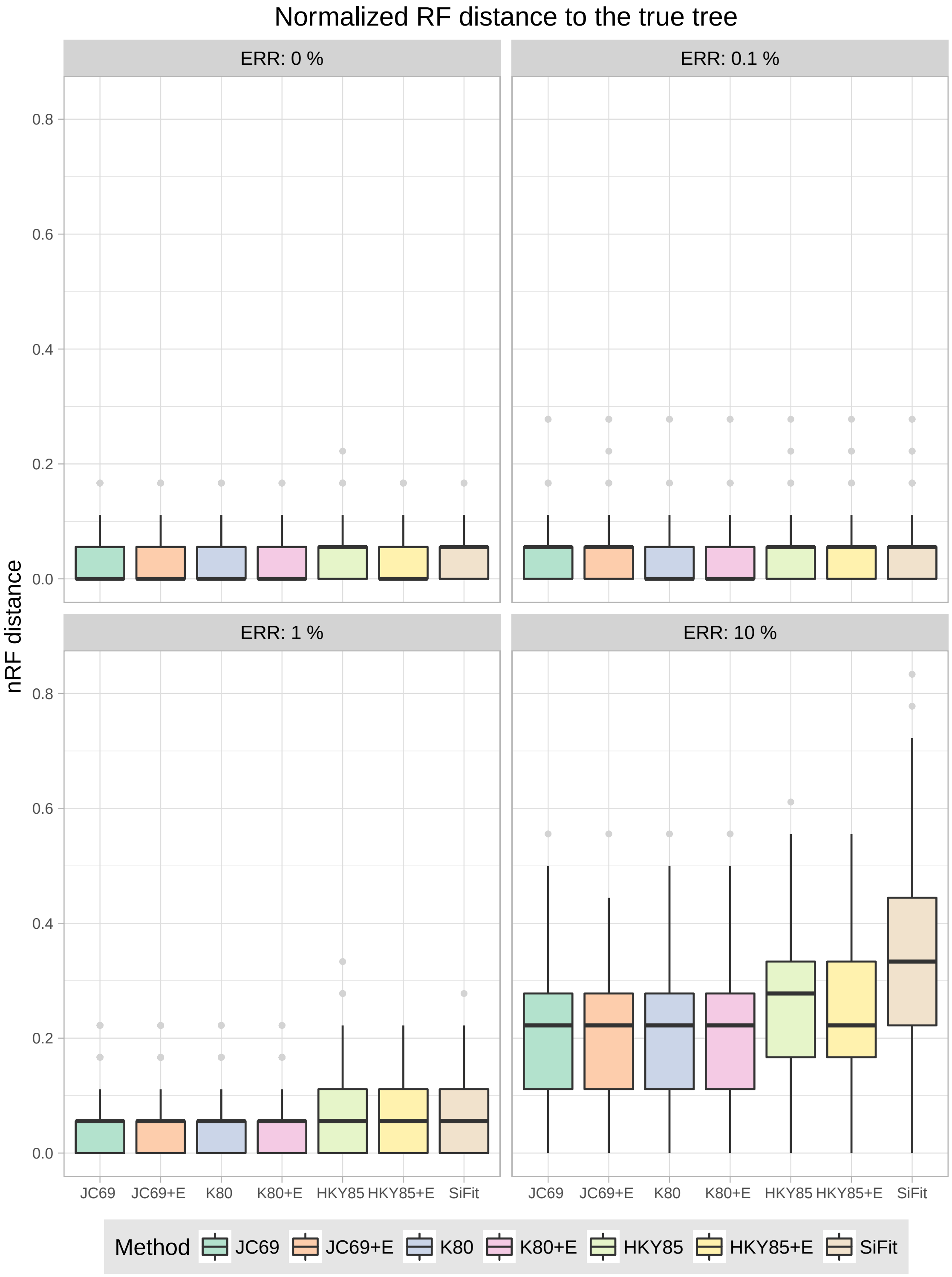


Figure 5.8: Phylogenetic inference accuracy of evaluated methods at varying sequencing error levels (genotype data).

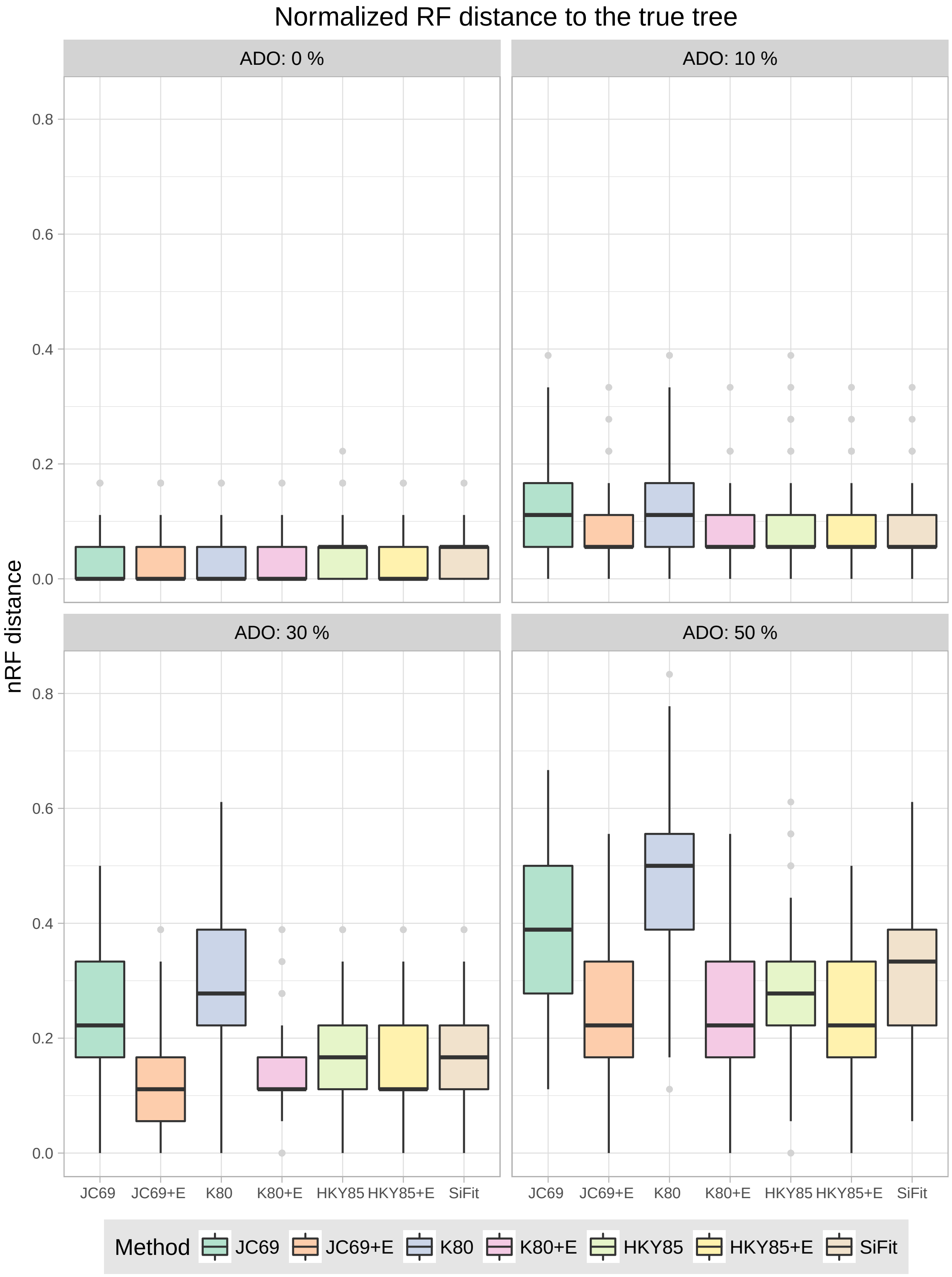


Figure 5.9: Phylogenetic inference accuracy of evaluated methods at varying ADO levels (genotype data).

Chapter 6

Phylogeny-aware detection of taxonomically mislabeled sequences

This chapter is based on the following peer-reviewed publication:

- **AM Kozlov**, J Zhang, P Yilmaz, FO Glöckner, A Stamatakis. ”Phylogeny-aware identification and correction of taxonomically mislabeled sequences.” In: *Nucleic Acids Res 2016; 44 (11): 5022-5033*

Contributions: Jiajie Zhang devised the original algorithmic idea and generated the simulated datasets for the publication. Alexey Kozlov and Jiajie Zhang further developed the algorithm and implemented it as a Python pipeline. Alexey Kozlov made necessary modifications in the `RAxML` code, integrated the pipeline into the `ARB` workbench, designed and performed the evaluation on large empirical reference databases. Pelin Yilmaz tested and evaluated the tool on Cyanobacteria taxonomy (not covered in this chapter). All co-authors contributed to writing the manuscript, with Alexey Kozlov and Pelin Yilmaz taking the lead.

In this Chapter, we address the problem of taxonomic curation, that is, identifying and maintaining valid taxonomic annotations in public sequence databases (**Section** 6.1.2). We propose a method called `SATIVA` (**S**emi-**A**utomatic **T**axonomy **I**mprovement and **V**alidation **A**lgorithm) that automatically detects putatively misannotated sequences, thereby assisting taxonomic curation efforts. We describe the `SATIVA` pipeline in **Section** 6.2, followed by a performance assessment on simulated (**Section** 6.3) and empirical data (**Section** 6.4). Finally, we discuss limitations of our method as well as possibilities for future improvements (**Section** 6.5).

6.1 Preliminaries

6.1.1 Background

Taxonomy is the science of classifying and naming groups of organisms, usually based on shared characteristics and/or presumed natural relatedness. Although first attempts to classify living organisms can be traced back to antiquity (e.g., Aristotle), modern taxonomy has its origin in the work of Carl Linnaeus. His unique binomial system standardized species naming across all domains of life, from bacteria to animals. Although Linnaeus' naming system is still being used today, methods of taxonomic classification have witnessed a paradigm change over the last decades. Historically, taxonomic classification was largely based on organism *phenotype*, that is, on visible morphological or physiological similarities. Obviously, this approach is better suited for macroscopic organisms such as animals and plants, whose morphological traits can be readily observed. Still, there is a serious risk of misinterpretation due to so called convergent evolution: similar traits can emerge independently in multiple distant lineages. Phenotypic characterization is even more problematic for microbial organisms (Bacteria, Archaea, and microscopic Eukaryota) as they are often difficult to cultivate [26] and have only a limited number of morphological traits due to their simple anatomy. Therefore, it was a major breakthrough when advances in molecular biology and bioinformatics allowed to characterize organism's *genotype*. Despite certain limitations, phylogenetic relationships as inferred from molecular data (DNA or amino acid sequences) provided a solid basis for modern taxonomy.

For the practical reasons (sequencing cost, computational feasibility), phylogenetic analysis is often based on just one or few genes that are well-conserved and ubiquitous in the organism group of interest, so called *marker genes* or *markers*. The most universal and widely-used marker is the small subunit ribosomal RNA (SSU, often referred to as 16S rRNA for Bacteria and Archaea and 18S rRNA for Eukaryota). Being part of the ribosome, SSU rRNA is present in all living organisms, and, thanks to its conserved secondary structure, SSU sequences from evolutionary distant lineages can still be aligned. Notably, Carl Woese's unified classification of life into three domains (Bacteria, Archaea and Eukaryota) was based on a phylogenetic tree inferred from SSU rRNA sequences [155]. For studies focusing on specific organism groups, other marker genes offer better resolution than SSU. For instance, the *interal transcribed spacer (ITS)* has been established as a marker for fungi, and the *cytochrome c oxidase I (COI)* is commonly used for animals.

Another key advancement was the development of the environmental PCR method by Norman R. Pace and colleagues [103]. It enabled the amplification of rRNAs directly from environmental samples and thereby opened a way for assessing microbial diversity at a molecular scale [44, 124]. Following this research avenue, recent studies correlated changes in the gut microbial composition with human conditions such as obesity, diabetes, and inflammatory bowel disease [15, 46, 70]. The prerequisite for carrying out such environmental studies is the availability of a reliable taxonomic classification of the environmental sequences. In turn, this requires a reference sequence database with a stable and well-curated taxonomy.

6.1.2 Motivation

Over the last decades, millions of marker gene sequences have been accumulated in *primary* sequence databases such as NCBI's GenBank and the European Nucleotide Archive (ENA). Unfortunately, the usefulness of these invaluable data resources is hampered by partially unreliable sequence metadata, including important attributes such as species name and taxonomic classification. As sequence annotation is usually conducted by the submitting author, its quality depends on his/her qualification and willingness to invest time and effort. Furthermore, primary database maintainers are generally not allowed to modify sequence metadata; only the original submitter can perform an amendment. Due to this policy, even well-known annotation errors often remain unrectified in GenBank entries.

In order to alleviate this problem and to provide easy access to reliable reference sequence collections, several *secondary* gene-specific databases have recently emerged. These databases are regularly updated by querying a primary database (e.g., GenBank) for the relevant marker sequences, which are then subjected to quality filtering and taxonomic validation. Some databases also allow for direct sequence submission and/or support community-driven taxonomic curation, often combined with some sort of post-submission checks. The most widely-used secondary databases include BOLD [112] (mostly animal COI), UNITE [2] (fungal ITS), PR2 [47] (protist 18S), SILVA [109] (prokaryotic 16S and eukaryotic 18S), RDP-II [24] and Greengenes [90] (prokaryotic 16S).

Even though the annotation quality in secondary databases is considerably higher than in GenBank, they still do contain (taxonomic) errors. This is in part due to imperfection of the algorithms used for quality-filtering and classification. Another problem is that of error propagation: given the iterative update procedure used by most secondary databases, potentially incorrect annotations of existing sequences are used to classify new sequences, which will in turn obtain an erroneous label. Of course, such mistakes can be eliminated by means of manual curation and continuous re-assessment of old classifications based on the new data. However, growing database sizes make this approach less practical: some erroneous annotations might escape the curator's attention, and thus persist in the database and propagate to future releases (see example in Box 4.1).

Box 4.1: A case of error propagation

In SILVA release 119, a new sequence with the NCBI accession `KF053060` was added to the database. It was initially annotated as *Alcaligenes faecalis* (order *Burkholderiales*, phylum *Proteobacteria*) in NCBI GenBank, and SILVA adopted this classification as well. Later on, the sequence record in GenBank was amended. First, the organism name was changed to *Bacillus sp.*, and the taxonomic path was adjusted accordingly (order *Bacillales*, phylum *Firmicutes*). Then, the sequence itself was also corrected, and the new revision obtained accession number `KF053060.2` (revision history: `https://www.ncbi.nlm.nih.gov/nuccore/KF053060.2?report=girevhist`).

In the subsequent release of SILVA (r123), sequence data for this accession was updated to the latest version (KF053060.2), but the original taxonomic path was preserved. Furthermore, six sequences highly similar to `KF053060` were added, and they were also classified as members of the *Alcaligenes* genus (see Table S2). This was in disagreement with the GenBank annotation, where they were classified as *Bacillus*. Presumably, the erroneous taxonomy was derived from an existing misclassified sequence (`KF053060`), which was highly similar to the new sequences.

`SATIVA` suggested to re-classify all seven aforementioned sequences into the *Bacillus* genus. This is consistent with the current taxonomic annotation in NCBI GenBank and RDP-II. Further, this re-classification was carried out in the subsequent release of SILVA (r128, September 2016).

NCBI accession	NCBI organism name	NCBI taxonomy	SILVA taxonomy
Added in SILVA 119			
KF053060	Bacillus sp. BAB-2669	Bacteria; Firmicutes; Bacilli; Bacillales; Bacillaceae; Bacillus	Bacteria; Proteobacteria; Betaproteobacteria; Burkholderiales; Alcaligenaceae; Alcaligenes
Added in SILVA 123			
JX093131	uncultured Bacillus sp.	Bacteria; Firmicutes; Bacilli; Bacillales; Bacillaceae; Bacillus	Bacteria; Proteobacteria; Betaproteobacteria; Burkholderiales; Alcaligenaceae; Alcaligenes
JX093151	uncultured Bacillus sp.		
KC442332	Bacillus sp. CH6		
KF721697	uncultured Bacillus sp.		
KF722496	uncultured Bacillus sp.		
KF740382	Bacillus sp. ZYJ-39		

Therefore, we developed a novel phylogeny-aware method that can automatically identify putative mislabels in large databases and suggest new, corrected taxonomic annotations for them. In order to detect conflicts between taxonomic and phylogenetic assignments, our method relies on the Evolutionary Placement Algorithm (EPA [12]), which we will briefly outline in the following section.

6.1.3 The Evolutionary Placement Algorithm

The concept of *phylogenetic* or *evolutionary placement* was popularized by Matsen [89] and Berger and Stamatakis [12, 138] who independently developed highly efficient implementations of analogous algorithms between 2009 and 2011. In that time, metagenomic studies started to gain momentum, generating millions of short reads from multiple distinct species in the environment being surveyed. One possibility to analyze this sort of data would be to build a phylogenetic tree including (unlabeled) short reads as well as some 'reference' sequences with known annotations. However, *de novo* phylogenetic inference with classical state-of-the-art methods is practically impossible in this setting. Firstly, it is extremely challenging computationally, since phylogenetic inference under maximum likelihood is shown to be NP-hard [114]. Secondly, even with enough computing power, it is barely possible to resolve the relationships among a huge number of short sequences due to the lack of phylogenetic signal therein [93, 139]. Because of these limitations, *evolutionary placement* was proposed as a 'lightweight' alternative to phylogenetic inference: it does not aim to obtain the fully-resolved tree, but can determine the most likely placement of the (millions of) query sequences (e.g., short reads) on the (much smaller) annotated reference tree.

The evolutionary placement workflow consists of the following steps:

1. Annotated, quality-filtered, full-length sequences for the organism group of interest are used to build a *reference alignment.*

2. Classical phylogenetic methods (e.g., ML inference) are applied to obtain a *reference tree.*

3. Query sequences (QS) are aligned to the reference alignment.

4. Each query sequence q_i is grafted to *every* branch b_k of the reference tree *independently*, and the likelihood score of each respective extended tree $LH(q_i, b_k)$ is evaluated.

5. For each branch b_k of the reference tree its *expected likelihood weight (ELW)* [140, 148] is calculated as the ratio of $LH(q_i, b_k)$ and the sum of likelihoods for all possible QS placements:

$$ELW(q_i, b_k) = \frac{LH(q_i, b_k)}{\sum\limits_j LH(q_i, b_j)} \tag{6.1}$$

6. The branch with the maximum ELW score is considered to represent the most likely QS placement. Alternatively, a set of best-scoring branches whose sum

of *ELW* scores is above a certain threshold (e.g., 0.95) can be reported and considered for downstream analyses.

In our mislabel detection method, we will use the evolutionary placement algorithm implemented in `RAxML` ([12], called `RAxML-EPA` henceforth).

6.2 Implementation

6.2.1 SATIVA pipeline

We implemented our taxonomic curation method in a Python pipeline called `SATIVA` (`https://github.com/amkozlov/sativa`), which is based on a modified versions of `RAxML` and `RAxML-EPA` (see **Section** 6.2.3). It also employs ETE library [55] for tree topology manipulation and alignment parsing.

`SATIVA` requires two input files: a multiple sequence alignment (in FASTA or PHYLIP format) and a list of taxonomic annotations with matching sequence identifiers. The output is a tab-delimited text file which contains the putative mislabel identifiers as well as the original and suggested taxonomic annotations including confidence values.

`SATIVA` provides several parameters which allow to set the trade-off between runtime and the thoroughness of mislabel detection. First, the user can set the number of `RAxML` tree searches (with different starting trees) that `SATIVA` will conduct to attain the best-scoring ML reference tree. While using multiple starting trees can potentially yield a better reference tree, it also leads to an about linear increase in tree inference time. Further, `SATIVA` can be configured to run in the 'fast' mode, in which a topological convergence criterion [132] is used to stop the tree search earlier (this corresponds to `-D` option of `RAxML`). Conversely, in the 'thorough' mode (default) `SATIVA` uses a likelihood-based stopping criterion, which is typically 1.5 to 2.1 times slower [132].

In the following text, we give the detailed step-by-step description of the pipeline (**Figure** 6.1).

Building a taxonomically-labeled reference tree.

Using the input set of taxonomic annotations, we initially build a rooted, multifurcating tree that represents the underlying taxonomy. In this tree (which we henceforth call *taxonomic tree*), leaf nodes correspond to the sequences, and inner nodes to higher taxonomic ranks such as genus or family. Then, we perform a Maximum Likelihood (ML) tree inference with `RAxML` [134], using the input sequence alignment and the taxonomic tree as a topological constraint. Thereby, we obtain a strictly bifurcating tree (a *reference tree*) that is fully congruent with the original taxonomic

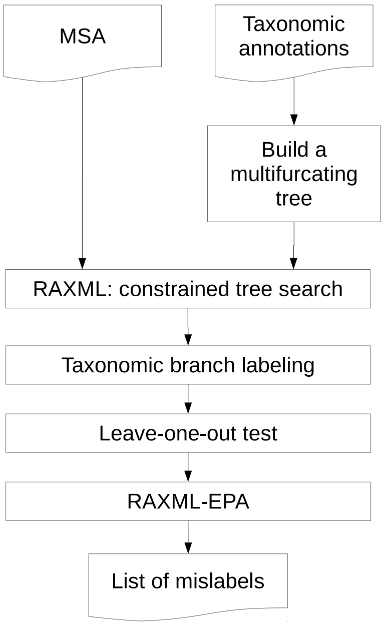

MSA
Taxonomic annotations
Build a multifurcating tree
RAXML: constrained tree search
Taxonomic branch labeling
Leave-one-out test
RAXML-EPA
List of mislabels

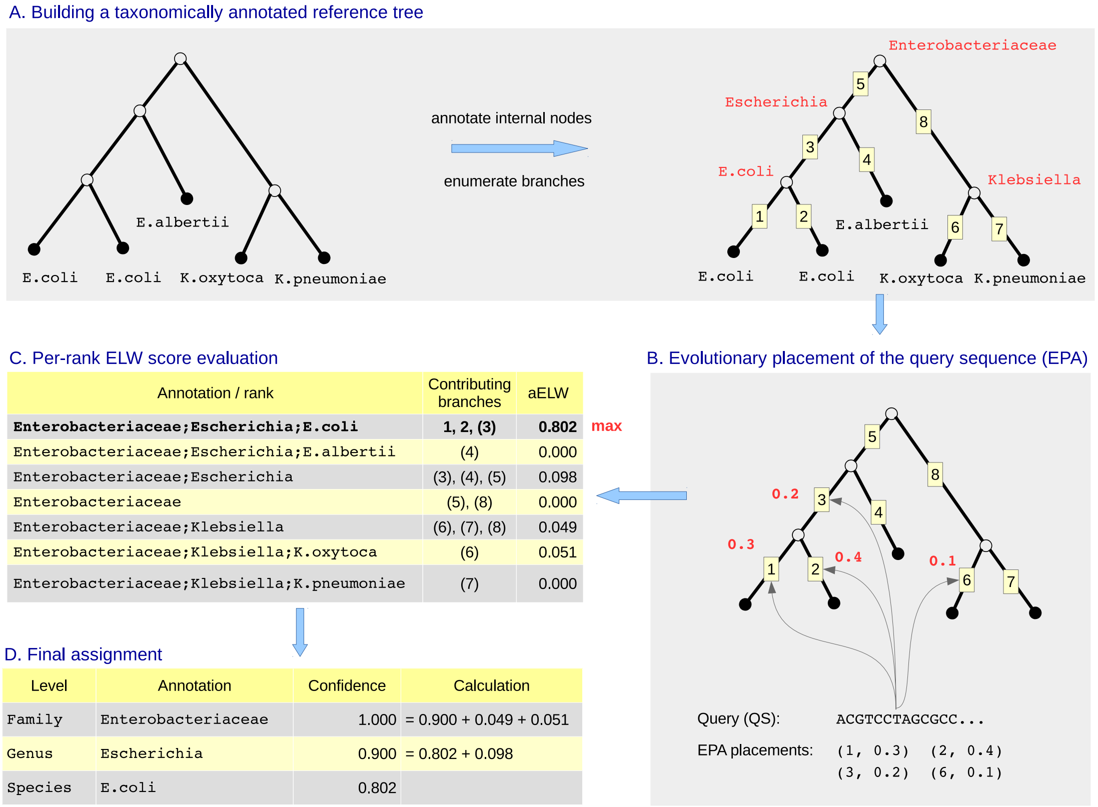
Figure 6.1: SATIVA processing workflow.

tree. Further, we label each inner node of this strictly bifurcating reference tree by the lowest common rank of its corresponding child nodes (see **Figure** 6.2, A). For instance, given annotations *('Escherichia', 'E.coli')* and *('Escherichia', 'E.albertii')* at the child nodes, the parent node will be labeled as *('Escherichia')*.

Leave-one-out test

We prune *one sequence at a time* from the reference tree, and use the EPA algorithm to place it back into all branches of the remaining reference tree. Then, we use the taxonomy assignment approach described below to calculate a new taxonomic label for the pruned sequence. If there is a disagreement between the new and the original taxonomic label, we put the sequence into a *preliminary* mislabels list.

Note that, when comparing taxonomic labels, we do not consider missing rank annotations as disagreements. For instance, if the original annotation is (*'Enterobacteriaceae'*, *'Escherichia'*), new annotations (*'Enterobacteriaceae'*, *'Klebsiella'*) or (*'Enterobacteriaceae'*, *'Klebsiella'*, *'K. pneumoniae'*) will be reported as a mislabel, whereas (*'Enterobacteriaceae'*, *'Escherichia'*, *'E. coli'*) and (*'Enterobacteriaceae'*) will not.



A. Building a taxonomically annotated reference tree

B. Evolutionary placement of the query sequence (EPA)

Query (QS): ACGTCCTAGCGCC...

EPA placements: (1, 0.3) (2, 0.4) (3, 0.2) (6, 0.1)

C. Per-rank ELW score evaluation

Annotation / rank	Contributing branches	aELW	
Enterobacteriaceae;Escherichia;E.coli	**1, 2, (3)**	**0.802**	max
Enterobacteriaceae;Escherichia;E.albertii	(4)	0.000	
Enterobacteriaceae;Escherichia	(3), (4), (5)	0.098	
Enterobacteriaceae	(5), (8)	0.000	
Enterobacteriaceae;Klebsiella	(6), (7), (8)	0.049	
Enterobacteriaceae;Klebsiella;K.oxytoca	(6)	0.051	
Enterobacteriaceae;Klebsiella;K.pneumoniae	(7)	0.000	

D. Final assignment

Level	Annotation	Confidence	Calculation
Family	Enterobacteriaceae	1.000	= 0.900 + 0.049 + 0.051
Genus	Escherichia	0.900	= 0.802 + 0.098
Species	E.coli	0.802	

Figure 6.2: Example illustrating the taxonomic assignment method implemented in SATIVA. **A**. We start with a resolved, bifurcating reference tree which has taxonomic annotations at the tips (i.e., sequence annotations). First, we perform a post-order tree traversal and assign taxonomic labels to internal nodes (shown in red) by taking the longest common annotation of the respective child nodes. Then, we enumerate the branches and store the mapping between the branch numbers and the taxonomic annotations of their adjacent nodes. **B**. We use the RAxML-EPA algorithm to calculate the most likely placements of the QS on the reference tree. For our purposes, each placement is represented by a pair (branch number, likelihood weight). **C**. We compute the accumulated likelihood weight ($aELW$) for each taxonomic rank by summing over the weights of the corresponding branches. The branches in parentheses have two competing annotations, and their weights contribute partially to both respective $aELWs$ (see main text for details). **D**. We assign the QS to the taxonomic rank with the highest $aELW$. At each taxonomic level, we compute a confidence score by summing over $aELWs$ of all annotations which do not contradict the assigned one *at this level.*

Identification of mislabels

Once all sequences have been subjected to the leave-one-out test, we prune all sequences in the preliminary mislabels list from the reference tree *at once*. Then, we use EPA to independently place each of them back into the remaining tree and recalculate the taxonomic annotations. Once again, we compare the new annotations with the original ones, and put sequences in the *final* mislabels list if they differ. In addition, the calculated taxonomic annotation for each final mislabeled sequence will be reported as a suggested correction. Further, we report the *mislabel confidence score*, which is equal to the assignment confidence score (see above) at the *highest* taxonomic rank level for which the original and the new taxonomic labels differ.

This two-step approach allows to reduce the noise or misleading signal introduced by mislabels in the reference phylogeny. In particular, we observed a phenomenon which we call 'reciprocal' mislabels: if one out of two highly similar ('sibling') sequences was taxonomically mislabeled (and thus placed in a remote clade of the reference tree), both of them were incorrectly reported as mislabels in the leave-one-out test. For each sequence, a re-classification into the rank of its sibling was proposed. In other words, suggested corrections were reciprocal. This situation can be resolved by simultaneously pruning both sequences from the reference tree and placing them back independently.

Taxonomic assignment algorithm

We use the following approach to assign a taxonomic annotation to a so-called *query* sequence (QS), that is, a sequence which is not present in the reference tree. First, we use the `RAxML-EPA` to evaluate all possible QS placements and obtain the respective likelihood weights (ELWs) for all branches of the reference tree (see **Figure** 6.2, B).

Then, we use those likelihood weights to calculate the accumulated ELW ($aELW$) for each taxonomic rank (see **Figure** 6.2, C). In order to map branches to taxonomic ranks, we use a taxonomically-labeled reference tree constructed in the previous step. In particular, for each branch b_k, we analyze the taxonomic ranks r_u and r_l assigned to the both nodes *adjacent* to this branch. We distinguish two cases:

(a) If *both* taxonomic ranks are identical ($r_u = r_l = r_i$), then the entire likelihood weight of the branch will contribute to the $aELW$ of this ranks:

$$aELW(r_i) := aELW(r_i) + ELW(b_k) \tag{6.2}$$

(b) If the taxonomic ranks differ, then the likelihood weight is distributed between both ranks:

$$aELW(r_u) := aELW(r_u) + c \cdot ELW(b_k) \tag{6.3}$$

$$aELW(r_l) := aELW(r_l) + (1 - c) \cdot ELW(b_k) \tag{6.4}$$

where parameter c defines the distribution ratio. In the current implementation, we distribute weights (almost) equally by setting $c := 0.49$ (a small imbalance is needed to avoid ties).

Thereafter, we select the taxonomic annotation with the highest $aELW$ as new, phylogeny-aware annotation for the QS.

Note that 'nested' taxonomic annotations such as (*'Escherichia'*, *'E.coli'*) and (*'Escherichia'*) are considered as distinct at this step of our algorithm. This allows to directly compare competing annotations at *different* taxonomic levels to each other and select the most likely one according to the $aELW$.

Finally, we calculate an overall *assignment confidence score* for the QS by summing over the $aELW$s for *all* annotations that are in concordance with the *proposed* taxonomic annotation of the QS. So, if (*'Enterobacteriaceae'*, *'Escherichia'*) is selected as the most likely QS assignment, the $aELW$ for the annotation (*'Enterobacteriaceae'*, *'Escherichia'*, *'E.coli'*) will also contribute to the assignment confidence score at both, the family and genus level. At the same time, $aELW$s for (*'Enterobacteriaceae'*, *'Klebsiella'*) or (*'Enterobacteriaceae'*) annotations will contribute to the family assignment confidence score, but not to the genus level assignment confidence score (see **Figure** 6.2, D).

6.2.2 ARB integration

To make `SATIVA` easy-to-use for taxonomists, we integrated it with the `ARB` software [87], which is a widely used tool for maintaining and curating large rRNA databases. `ARB` is built around an efficient sequence storage engine, it provides a graphical workbench for editing sequences and associated metadata as well as import/export tools, and offers advanced tree visualization capabilities. `SATIVA` is currently available in the development version of `ARB`, which can be downloaded at `http://www.arb-home.de/downloads.html`. Within `ARB`, the user can invoke `SATIVA` by simply marking a subset of sequences in the workbench and selecting "Validate taxonomy" from the menu. The results of the analysis are written back to the `ARB` database fields, and putatively mislabeled sequences will be highlighted on the tree (see **Figure** 6.3). Furthermore, `ARB` can be used to easily visualize results and customize the visualization using `ARB` features such as "search by field value" and "set field value". For instance, mislabeled sequences can be highlighted with different colors based on their rank level incongruence (e.g., phylum, class etc.) or mislabel confidence value. Finally, `ARB` can also write `SATIVA` results to an external file.

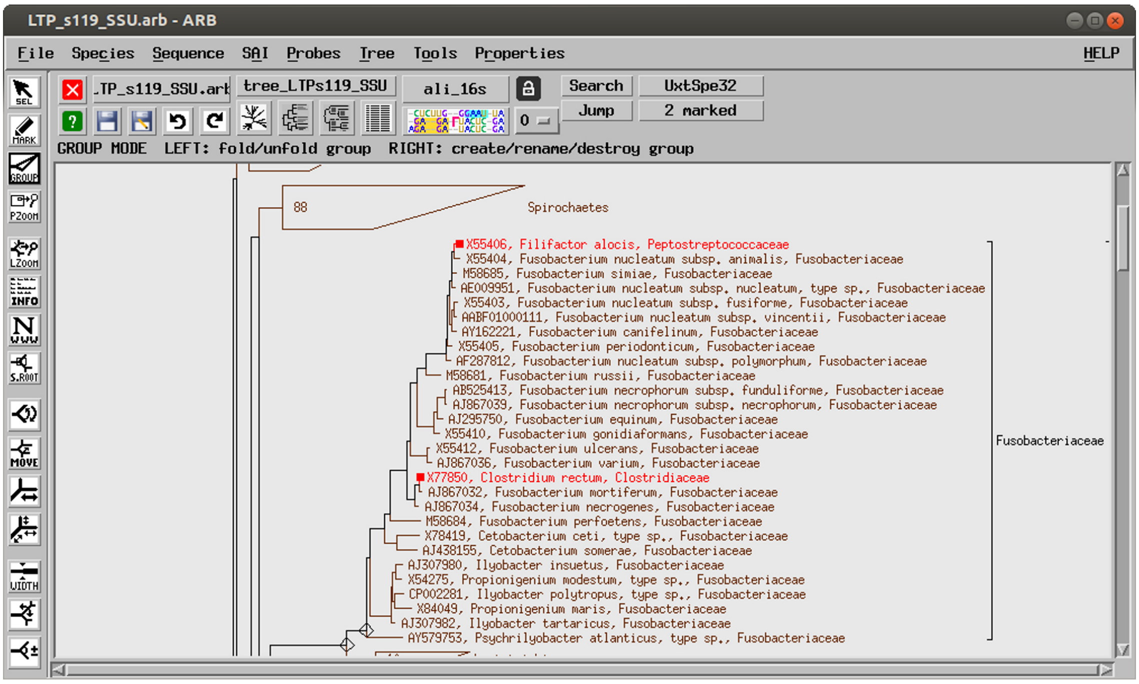

LTP_s119_SSU.arb - ARB
File
Species
Sequence
SAI
Probes
Tree
Tools
Properties
HELP
_TP_s119_SSU.arb
tree_LTPs119_SSU
ali_16s
Search
UxtSpe32
Jump
2 marked
0
GROUP MODE LEFT: fold/unfold group RIGHT: create/rename/destroy group
SEL
MARK
GROUP
P2OOM
LZOOM
INFO
WWW
S.ROOT
MOVE
WIDTH
88
Spirochaetes
X55406, Filifactor alocis, Peptostreptococcaceae
X55404, Fusobacterium nucleatum subsp. animalis, Fusobacteriaceae
M58685, Fusobacterium simiae, Fusobacteriaceae
AE009951, Fusobacterium nucleatum subsp. nucleatum, type sp., Fusobacteriaceae
X55403, Fusobacterium nucleatum subsp. fusiforme, Fusobacteriaceae
AABF01000111, Fusobacterium nucleatum subsp. vincentii, Fusobacteriaceae
AY162221, Fusobacterium canifelinum, Fusobacteriaceae
X55405, Fusobacterium periodonticum, Fusobacteriaceae
AF287812, Fusobacterium nucleatum subsp. polymorphum, Fusobacteriaceae
M58681, Fusobacterium russii, Fusobacteriaceae
AB525413, Fusobacterium necrophorum subsp. funduliforme, Fusobacteriaceae
AJ867039, Fusobacterium necrophorum subsp. necrophorum, Fusobacteriaceae
AJ295750, Fusobacterium equinum, Fusobacteriaceae
X55410, Fusobacterium gonidiaformans, Fusobacteriaceae
X55412, Fusobacterium ulcerans, Fusobacteriaceae
AJ867036, Fusobacterium varium, Fusobacteriaceae
X77850, Clostridium rectum, Clostridiaceae
AJ867032, Fusobacterium mortiferum, Fusobacteriaceae
AJ867034, Fusobacterium necrogenes, Fusobacteriaceae
M58684, Fusobacterium perfoetens, Fusobacteriaceae
X78419, Cetobacterium ceti, type sp., Fusobacteriaceae
AJ438155, Cetobacterium somerae, Fusobacteriaceae
AJ307980, Ilyobacter insuetus, Fusobacteriaceae
X54275, Propionigenium modestum, type sp., Fusobacteriaceae
CP002281, Ilyobacter polytropus, type sp., Fusobacteriaceae
X84049, Propionigenium maris, Fusobacteriaceae
AJ307982, Ilyobacter tartaricus, Fusobacteriaceae
AY579753, Psychrilyobacter atlanticus, type sp., Fusobacteriaceae
Fusobacteriaceae

Figure 6.3: SATIVA results displayed in the ARB workbench. Two sequences annotated as *Clostridiales* species in LTP123 have been identified as mislabels (marked in red). The suggested re-classification as *Fusobacterium* is consistent with the SILVA123 annotation (placement in the tree).

6.2.3 RAxML modifications

In order to handle large empirical datasets with thousands of taxa (see **Section** 6.4.1), we had to introduce several optimizations and extensions to the original `RAxML` code:

Leave-one-out test The aforementioned EPA-based leave-one-out test was implemented directly in the `RAxML` code. Although it would be possible to leverage the existing `RAxML-EPA` implementation and run it multiple times (once for every sequence in the dataset on the respective pruned reference tree), it would incur additional overhead due to re-loading/parsing the alignment, re-computing the CPVs etc. Since this overhead grows quadratically with the number of sequences in the alignment, it becomes a bottleneck on large datasets.

Checkpointing In order to facilitate long runs in the cluster environment with fixed job runtime limits, we implemented checkpointing in both, tree inference (analogous to `ExaML`) and EPA leave-one-out modes.

Topological operations The cost of bookkeeping operations such as comparing and saving/restoring tree topologies is usually negligible compared to likelihood computations. Therefore, the respective routines were not optimized in the original `RAxML` code. However, on very large trees (10,000 to 500,000 taxa) the naïve implementation of these bookkeeping operations led to a noticeable slowdown. We therefore carried out several additional optimization, for instance, we removed some excessive tree saving operation and substantially accelerated identification of duplicate topologies by using hash-based comparisons.

Per-rate scalers It is known that using the Γ model [156] of across-site rate heterogeneity with large trees can lead to numerical underflow problems [60]. We resolved this issue by introducing individual scalers per each Γ rate category (see **Section** 4.2.2 for details).

CAT model optimization When optimizing per-site rates under the CAT model of across-site rate heterogeneity [129], many expensive-to-compute exponentiation operations can be avoided by applying a pre-computation technique that takes into account the structure of the optimization algorithm. Once again, this part of the code is only relevant for large trees, as its overhead grows linearly with the number of taxa.

Applying the above optimizations yielded a cumulative speedup of more than an order of magnitude, and enabled `SATIVA` to analyze the largest empirical dataset with 536,224 sequences. Note that, these optimizations are also of great value to stand-alone `RAxML` analysis of very large datasets w.r.t. the number of taxa. We therefore made our modified version of `RAxML` publicly available on GitHub (`https://github.com/amkozlov/sativa`), and applied similar optimizations in the newly developed `RAxML-NG` code (see **Section** 4).

6.3 Evaluation on Simulated Data

6.3.1 Experimental Setup

We based our simulations on version 123 of the LTP database (LTP123). First, we used `RAxML` to infer a *constrainted*, fully bifurcating ML tree from the LTP123 reference alignment, using the LTP123 taxonomy as constraint tree. Next, we partitioned the alignment into 11 regions which reflect the known secondary structure of the 16S rRNA gene: we defined nine partitions for variable regions V1-V9, one for all conserved regions, and one for the 'flanking' regions not found in the *E.coli* reference sequence. We then estimated the GTR (General Time Reversible) model parameters and branch lengths for each partition individually. These parameters as well as the ML tree topology were subsequently used to simulate sequence alignments with `INDELible` [42]. We tuned the `INDELible` simulation parameters (insertion/deletion rate and sequence length at the root) such that for each partition, the length and proportion of gaps approximately match the corresponding region of the empirical alignment.

Clearly, the simulated alignment we is fully consistent with the original LTP123 taxonomy. In other words, we assume that the LTP taxonomy contains no mislabels in the simulated alignment. Therefore, we can deliberately introduce taxonomic mislabels by randomly changing the original ('true') sequence annotations to the new ('incorrect') ones. We used this approach to generate six test taxonomies: three replicates with 1% mislabels (`SIM1`) and three replicates with 5% mislabels (`SIM5`). The distribution of mislabels among the taxonomic rank levels was based on the proportions observed in the empirical LTP123 dataset (see **Table** 6.1).

Taxonomic rank	Dataset	
	`SIM1`	`SIM5`
	% mislabels	% mislabels
Phylum	0.05%	0.25%
Class	0.10%	0.50%
Order	0.15%	0.75%
Family	0.35%	1.75%
Genus	0.35%	1.75%
Total	**1.00%**	**5.00%**

Table 6.1: Simulated datasets used for `SATIVA` evaluation.

To the best of our knowledge, there are no established tools for automatic mislabel identification. Thus, a direct accuracy and performance comparison is not feasible at present. However, since with `SATIVA` we also introduce a novel method for taxonomic assignment of new sequences, we included two widely used taxonomic classification methods (UCLUST [35] and RDP [149]) into our performance evaluation. To this end, we implemented two methods denoted as `RDP-LO` and `UCLUST-LO`, which use a `SATIVA`-like leave-one-out approach to detect mislables, but rely on RDP and UCLUST, respectively, for taxonomic classification. More specifically, we remove one sequence at a time from a database with n sequences and use the remaining sequences and taxonomic labels as new reference. Then, we use RDP and UCLUST to assign a new taxonomic label to the removed sequence using the new reference with $n-1$ sequences. If the new taxonomic label is different from the original one, we consider it as mislabel and the inferred taxonomic label as the proposed new classification. For both RDP and UCLUST, we used the implementations that are available in the QIIME v1.8.0 [19] pipeline with default parameters.

6.3.2 Results

We deployed two metrics to quantify the ability of competing tools to identify mislabels on simulated data. Firstly, we used the accuracy of mislabel *identification*. To this end, we compared the output of each program to the true list of mislabels: each sequence was counted as *true positive (TP)* if it was present in both lists, and as *false negative (FN)* or *false positive (FP)* if it was missing from the inferred or ground truth list, respectively. Then, we used the standard formulas to calculate *precision* and *recall* values at each taxonomic level (s. **Table** 6.2). Secondly, we evaluated the *correction accuracy* by comparing the suggested taxonomic annotation for mislabels with the true one (s. **Table** 6.3). If a mislabel was not identified as such, we assumed its inferred annotation to be equal to the original, uncorrected one. In other words, such sequences were counted as *false positives* at taxonomic levels that were (deliberately) mislabeled in the respective simulations.

Since all three methods provide confidence values for taxonomic placements (RDP, UCLUST) or identified mislabels (`SATIVA`), it is possible to use a threshold to exclude results with low confidence. For each method, we empirically evaluated several confidence thresholds and chose the value which yielded the highest F-measure (that is, the best precision/recall trade-off). Specifically, we set the confidence threshold to 0.7 for `UCLUST-LO`, 0.8 for `RDP-LO` and 0.51 for `SATIVA`.

Among the three algorithms tested, `SATIVA` shows the best mislabel identification accuracy: at least 96.9% of all mislabels are recognized, while the false positive rate below than 9%. `RDP-LO` achieved similar recall values to `SATIVA` (e.g., 97.7% vs. 98.4% on the `SIM1` dataset with 1% mislabels). However, its precision was unacceptably low (12.0% on `SIM1` / 38.2% on `SIM5`). Finally, the `UCLUST-LO` algorithm shows higher precision, but lower recall than `RDP-LO`, and is clearly inferior to `SATIVA` in terms of both, precision and recall.

Our measurements of *precision* for `UCLUST-LO` and `RDP-LO` might appear to contradict earlier studies (e.g., [149]), where much higher values have been reported. Note that, here we measure the precision of *mislabel identification*, which is different from the precision of *taxonomic classification*. Specifically, in the latter case *all* sequences with correctly inferred taxonomic annotation are considered *true positives*. In our test, however, only those sequences that were *deliberately mislabeled* and correctly identified by the method are counted as *true positives*. All other, non-mislabeled sequences, which were recognized as such, represent *true negatives*. And since in our test datasets mislabeled sequences constitute only a small fraction of the data (1% or 5%), the impact of *false positives* on precision is substantially more pronounced. This also explains the significantly higher precision values for the `SIM5` dataset (5% 'true' mislabels) as compared to the SIM1 dataset (1% mislabels).

In the correction accuracy test, SATIVA and RDP-LO performed almost equally well, achieving precision and recall of around 95% on the SIM1 dataset, and slightly lower precision on the dataset with 5% mislabels (≈90% SATIVA/ ≈92% RDP-LO). UCLUST-LO showed higher recall (98.8% on SIM1 / 98.0% on SIM5), but this comes at the expense of substantially reduced precision (81.2% / 74.2%).

Level	Dataset / percentage of 'true' mislabeled sequences											
	`SIM1` / 1 %						`SIM5` / 5 %					
	Precision			Recall			Precision			Recall		
	`RDP-LO`	`UCLUST-LO`	`SATIVA`	`RDP-LO`	`UCLUST-LO`	`SATIVA`	`RDP-LO`	`UCLUST-LO`	`SATIVA`	`RDP-LO`	`UCLUST-LO`	`SATIVA`
Phylum	0.625	1.000	1.000	1.000	0.933	1.000	0.813	0.947	1.000	1.000	0.828	1.000
Class	0.675	0.958	0.900	1.000	0.852	1.000	0.785	0.896	0.983	0.989	0.793	1.000
Order	0.409	0.796	1.000	1.000	0.867	1.000	0.661	0.860	0.977	0.995	0.726	0.995
Family	0.311	0.657	0.965	0.973	0.811	0.991	0.605	0.833	0.971	0.976	0.745	0.987
Genus	0.054	0.130	0.893	0.965	0.832	0.965	0.217	0.416	0.822	0.949	0.757	0.928
Total	0.120	0.274	0.939	0.977	0.836	0.984	0.382	0.619	0.917	0.971	0.757	0.969

Table 6.2: Accuracy of mislabel identification on simulated data. Taxonomic levels are the levels where sequences were deliberatly misclassified in the ground truth. That is, a recall value of 0.974 at the family level means that 97.4 % of sequences with an incorrect family label were successfully identified.

Level	Dataset / percentage of 'true' mislabeled sequences											
	`SIM1` / 1 %						`SIM5` / 5 %					
	Precision			Recall			Precision			Recall		
	`RDP-LO`	`UCLUST-LO`	`SATIVA`	`RDP-LO`	`UCLUST-LO`	`SATIVA`	`RDP-LO`	`UCLUST-LO`	`SATIVA`	`RDP-LO`	`UCLUST-LO`	`SATIVA`
Phylum	1.000	0.997	1.000	1.000	1.000	1.000	1.000	0.991	1.000	1.000	1.000	1.000
Class	1.000	0.984	1.000	1.000	1.000	1.000	0.999	0.968	1.000	0.999	0.999	0.999
Order	1.000	0.965	1.000	1.000	1.000	1.000	0.997	0.931	0.999	0.999	0.999	0.999
Family	0.984	0.894	0.994	1.000	1.000	0.997	0.981	0.843	0.992	0.997	0.999	0.996
Genus	0.930	0.812	0.946	0.959	0.988	0.949	0.919	0.742	0.899	0.964	0.980	0.954

Table 6.3: Accuracy of the suggested taxonomic annotation for mislabels on simulated data. Note that, errors are propagated down along the taxonomy, that is, an incorrect family label also implies an incorrect genus label, etc.

6.4 Analysis of widely-used 16S Sequence Databases

6.4.1 Experimental Setup

We analyzed four established databases of taxonomically annotated 16S rRNA sequences of Bacteria and Archaea: RDP-II [24], Greengenes [90], the 'All-Species' Living Tree Project (LTP) [161], and SILVA [109]. These databases have different underlying taxonomies, and vary in their size and taxonomic composition. In particular, LTP includes bacterial and archaeal type strains only, and thus contains a moderate number of sequences (11,939 as of release 123, September 2015). RDP-II, Greengenes, and SILVA, however, contain all rRNA sequences available in public databases that passed a database-specific quality check. Hence, they contain orders of magnitude more entries (1.2–3 million sequences). Furthermore, SILVA and Greengenes provide so-called non-redundant reference datasets (referred to as 'NR99' henceforth), which exclude highly similar (¿99% sequence identity) and partial sequences.

In order to make results for different taxonomies comparable and to maximize the coverage of each individual database, we divided our analysis into two parts. First, we evaluate taxonomic annotations of type strains only, using the same sequence set and alignment (LTP v123) for all four databases (datatsets `GG13_T`, `LTP123_T`, `RDP11_T` and `SLV123_T` in **Table** 6.4). Second, we evaluate the NR99 subsets (that is, representatives of the 99% identity clusters) for Greengenes and SILVA (datasets `GG13_NR99` and `SLV123_NR99`), thereby also including environmental sequences into our analysis.

For the type strain datasets, we executed `SATIVA` in 'thorough' mode, using 10 `RAxML` runs to infer the reference tree. For the NR99 datasets, we used the 'fast' mode and 1 `RAxML` run, for computational reasons. The confidence cut-off was set to 0.51 for all datasets.

6.4.2 Results

We used our approach to assess the phylogenetic consistency and identify mislabels in four widely-used 16S databases. We ran `SATIVA` on the representative datasets (see **Table** 6.4) and evaluated the percentage of mislabels reported for each taxonomic rank (see **Figure** 6.4a) as well as for several major bacterial phyla (see **Figure** 6.4b).

Among type strain datasets, `GG13_T` exhibits by far lowest percentage of identified mislabels (0.27%), followed by `RDP11_T` (1.27%), `SLV123_T` (1.54%) and `LTP123_T` (2.52%). In all taxonomies but Greengenes, the vast majority of mislabels was detected at the genus level. Therefore, the estimated percentage of mislabels at higher taxomomic levels (family and above) is more similar across datasets: `GG13_T`

Dataset	Taxonomy	Alignment	# sequences
	Type strains only		
`GG13_T`	Greengenes 13.8	LTP v123	10 635
`LTP123_T`	LTP v123	LTP v123	11 939
`RDP11_T`	RDP v11	LTP v123	11 775
`SLV123_T`	SILVA v123	LTP v123	11 939
	Non-repetitive subsets (99 % identity filter)		
`GG13_NR99`	Greengenes 13.8	Greengenes 13.8	203 452
`SLV123_NR99`	SILVA v123	SILVA v123	536 224

Table 6.4: Empirical datasets derived from widely-used microbial 16S sequence databases

0.20%, `SLV123_T` 0.31%, `RDP11_T` 0.41% and `LTP23_T` 1.37%.

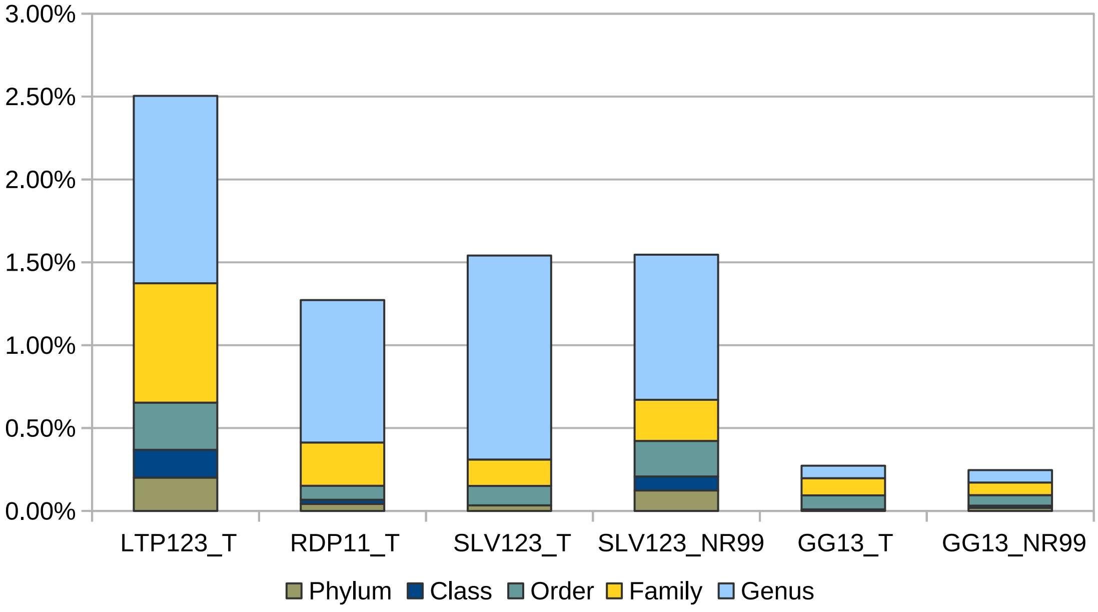
On the datasets which also include environmental sequences (`GG13_NR99` and `SLV123_NR99`), the Greengenes taxonomy shows less inconsistency (0.27% mislabels) compared to SILVA (1.55%). Again, the difference becomes less pronounced if genus-level mislabels are excluded (0.17% vs. 0.67%).

As **Figure** 6.4b shows, the distribution of mislabels among individual phyla is non-uniform. In all taxonomies, *Actinobacteria* and *Bacteroidetes* appears to contain less mislabels (0.15%–1.15% and 0%–1.92%, respectively) than *Proteobacteria* (0.38%–2.74%) and *Firmicutes* (0.34%–3.89%).

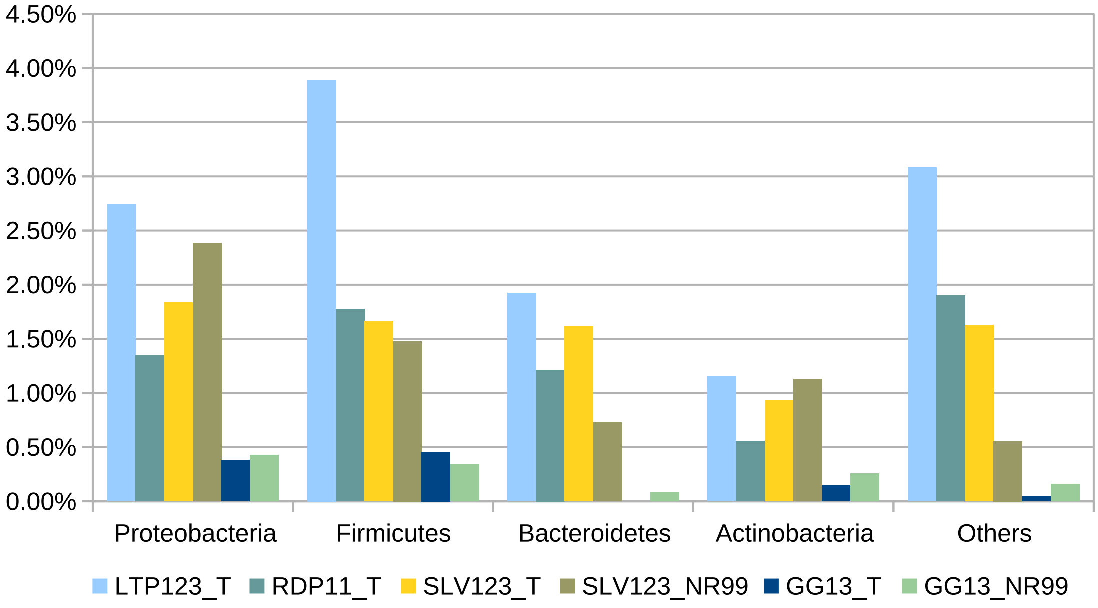
6.4.3 Discussion

According to our analysis, Greengenes, RDP, SILVA, and, to a lesser extent, LTP, are consistent at higher taxonomic levels. The few sequences proposed for reclassification into a different phylum or class are most probably due to an incorrect culture in the collection. Although putatively mislabeled sequences are more common at lower ranks (e.g., family or genus), their overall percentage is below 3% for all taxonomies. This implies that current taxonomic frameworks represent the phylogenetic signal of 16S rRNA well, but that there is nevertheless room for improvement.

We identified the highest amount of putative mislabels in the LTP taxonomy, especially at higher taxonomic ranks. This can be explained by the fact, that the LTP classification strictly follows Bergey's taxonomic outlines and LPSN. Conversely, the other three taxonomies adapt their classifications in order to better reflect the 16S tree topology, even if it involves changes that violate the formal rules of taxonomic and nomenclature code. For instance, non validly published names are widely used



(a) by taxonomic rank



(b) by phylym

Figure 6.4: Percentage of identified putative mislabels in the empirical datasets

to split non-monophyletic taxa (e.g., *'Clostridium III'* or *'Bacillaceae 1'*) or to represent uncertainty in classification (e.g., *'Clostridiales Incertae Sedis'*). Although such changes might be justified from the practical standpoint, they make direct comparison between LTP and other taxonomies impossible. Therefore, we suggest that LTP should be viewed as a 'baseline' in our comparison.

At the other end of the scale, the Greengenes taxonomy shows an extremely low percentage of mislabeled sequences (<0.3%). This suggests that this taxonomy is phylogenetically very consistent, most likely owing to the fact that it is based on a *de novo* phylogeny. On the other hand, the surprisingly few genus-level mislabels (0.08%) could be partially explained by the lack of annotation at this level. More specifically, as much as 29% of the sequences in `GG13_T` and 54% in `GG13_NR99` are not assigned to a genus. For comparison: just 0.04% of the sequences in `SLV123_T` and 16% in `SLV123_NR99` do not have a genus-level annotation.

Interestingly, the overall percentages of identified mislabels in full non-repetitive datasets (`GG13_NR99` and `SLV123_NR99`) and in the corresponding type strain datasets (`GG13_T` and `SLV123_T`) are highly similar. However, `SILVA_NR99` shows substantially more mislabels at higher taxonomic levels compared to `SLV123_T` (0.12% vs. 0.03% at the phylum level and 0.08% vs. 0% at the class level). This finding suggests that in the SILVA database, at least the most obvious mis-annotations for type strain sequences were fixed.

6.5 Conclusions and Future Directions

Public sequence databases, both primary and secondary, represent an invaluable resource for biological, medical and environmental research. It is therefore of fundamental importance to guarantee the highest possible quality of both data and metadata stored therein. In particular, incorrect species names and/or taxonomic affiliations pose a serious problem as they can affect downstream analyses (e.g., metagenomic studies). We contributed to solving this problem by developing a tool which can automatically analyze large datasets and short-list the putatively mislabeled sequences. It helps to increase the rate of taxonomic curation by eliminating the need for a visual tree inspection, an error-prone and labor-intensive process. As we have shown in this chapter, out tool attains high accuracy on simulated data, and can be applied to large empirical datasets. Moreover, our findings were used to improve taxonomic annotations in the latest release of the SILVA database (r128).

Despite those encouraging results, we realize that large-scale taxonomic curation is an extremely difficult endeavor, issuing many theoretical, organisational and technical challenges.

First, analyzing large marker databases with existing methods is computationally very demanding: running `SATIVA` on the largest empirical dataset in our study

(SLV123_NR99) took almost 20 days of wall-time (and $> 600,000$ CPU-hours) on a large cluster. To this end, we work on more efficient low-level optimizations, heuristics and parallelization techniques for both phylogenetic tree inference (see **Section** 4) and evolutionary placement [9, 10].

Further technical issues include chimeric and/or poor-quality sequences, synonymous taxonomic names, alignment of distant homologues (especially with fast-evolving markers such as ITS), to name just a few. Although most of them could be solved, at least to some extent, using appropriate tools and techniques, they do contribute to the uncertainty in the results of SATIVA, or any other sequence-based mislabel identification method.

From a theoretical standpoint, a conflict between taxonomic annotation and phylogenetic placement – that is, what we call a 'mislabel' throughout this chapter – could have several different explanations. In the simplest case, it could be an individual sequence misidentified due to, for instance, human data entry error or sample contamination. Such mislabels are usually easy to detect, and once identified, they could be trivially corrected by changing the sequence annotation in the database record. The situation becomes much more complicated, however, if the inconsistency between taxonomy and phylogeny is genuine: e.g., a species classified into a (phylogenetically) incorrect genus, or a higher taxonomic rank being paraphyletic or polyphyletic with respect to the evolutionary tree. In such cases, conflict resolution would require a formal amendment to the taxonomic classification, which constitutes a time-consuming and cumbersome process. Furthermore, such amendments are sometimes deliberately rejected by the community, for instance, due to an unwillingness to change historically established names of medically important organisms. One famous example is the *Shigella* bacterial genus: its species are grouped together with *Escherichia coli* in molecular phylogeny, but the existing name and genus rank were preserved to avoid confusion in the medical context. Some marker sequence databases try to circumvent this problem by introducing their own, 'unofficial' taxonomy which better reflects the evolutionary history of the given marker gene. This pragmatic solution has its merits, but it also complicates result interpretation by introducing name ambiguity: given the example above, it is not clear whether the *Escherichia* genus includes or excludes the *Shigella* species.

Finally, it is worth noting that rather than correcting the errors post hoc, it would be much more efficient to prevent suboptimal taxonomic annotations from being deposited in public databases. One way to deal with this problem would be to encourage submitters to follow the best practices in sequence quality control [99] and use specialized tools to check annotations prior to submission [119]. A more reliable solution could involve an automatic (and compulsory) validation enforced by the respective database. For organism groups without a well-established taxonomy, devising a phylogenetically consistent taxonomic framework from scratch is a viable option, as exemplified by the UniEuk project [13].

Chapter 7

Conclusion and Outlook

In this thesis, we embarked upon multiple challenges in the area of computational phylogenetics, and made multiple contributions to the field.

Firstly, we ported and optimized the `ExaML` code to run on the Intel *Knights Corner* hardware accelerators (1st generation of the Xeon Phi). Our `ExaML-KNC` implementation attained speedups which are comparable to other scientific codes ported to Xeon Phi, and it is freely available as production-level software. In the context of the 1KITE project, we also used `ExaML-KNC` ourselves to analyze large empirical datasets comprising millions of DNA and AA alignment sites.

Although some unfortunate features of the *Knights Corner* platform (low on-card memory amount, high MPI interconnect latency) limit its usefulness for large-scale phylogenetic analyses, our work on PLF kernel optimization for the 512-bit vector unit created a basis for future developments. In particular, we are currently working towards porting the `libpll` PLF kernels to the new *AVX512* SIMD instruction set available in the latest Intel processors (*Skylake-X* and *Skylake-SP*) and hardware accelerators (*Knights Landing*).

Further, we introduced `RAxML-NG`, which will supersede the existing and widely-used ML-based tree inference tools `RAxML` and `ExaML`. `RAxML-NG` outperforms these older tools in terms of tree search efficiency, speed and scalability, while also offering higher flexibility and user-friendliness. Importantly, `RAxML-NG` will also be easier to maintain and extend due to its modular design. With `RAxML-NG` we have developed a single code that scales from the laptop to supercomputer.

In the future, we want to explore new search strategies, in particular, using multiple starting trees and/or maintaining a population of trees in the course of the tree search. We expect that this approach will help to escape local optima, which still do constitute a problem of the `RAxML-NG` search heuristic. This conjecture is further supported by the performance of the `IQTree` program in our evaluation: it returned consistently high log-likelihoods with low variation, presumably since it does not rely on optimizing a single startingtree as `RAxML`/`ExaML`/`RAxML-NG` do.

Flexible parallelization is another possible direction of RAxML-NG improvement. The current fine-grain parallization approach requires at least 1,000–5,000 alignment sites per core to attain reasonable parallel efficiency, and is thus not suitable for taxa-rich alignments comprising only one or a few gene(s). Ironically, optimization of the time-critical per-site loops in the PLF kernels only aggravates this problem, since even more parallel workload is needed to amortize the sequential overhead (according to Amdahl's law). To this end, it will be worth exploring orthogonal parallelization schemes such as computing multiple CLVs and/or evaluating alternative SPR moves in parallel. However, these approaches also introduce additional complexity in terms of load balancing, data distribution, and increased memory footprint.

With respect to sequence error modeling, we showed that even simple models, that assume an uniform error distribution and have no prior information about the actual noise level, can improve branch length estimates compared to the classical error-agnostic models of evolution. It should be noted, however, that this positive effect is mainly observed for relatively high error rates (5%–10%), which are unlikely to occur in empirical sequence alignments obtained with traditional methods (NGS sequencing with high coverage). However, single-cell sequencing data exhibits substantially higher noise levels. Thus, an explicit modeling of sequence uncertainty can be beneficial. To this end, we implemented a dedicated error model for diploid genotype data, and compared its performance to SiFit [162], a state-of-the-art method specifically developed for analyzing noisy single-cell data. Although our method showed better performance compared to SiFit, the absolute tree inference accuracy from noisy data (10% sequencing error or 30% ADO) was still comparatively low (nRF distance to the true tree: 0.15–0.20). Therefore, we plan to explore more sophisticated error models and/or exploit external information about sequence uncertainty (e.g., incorporate genotype likelihoods reported by variant callers).

Support for sequence error models and diploid genotype data is currently being developed in the experimental branch of RAxML-NG, and this functionality will be transferred to the production version as the code matures.

With SATIVA, we introduced a practical way to detect incorrect taxonomic annotations in large sequence databases. It will help reference database curators to spot problematic sequences more easily, resulting in improved annotation quality and faster update cycles. On the other hand, individual researchers who rely on the sequences retrieved from GenBank and other public databases for their studies will benefit from the semi-automatic validation of their particular reference subset.

The scalability of the SATIVA pipeline to large datasets can be improved by upgrading its most performance-critical components, RAxML and RAxML-EPA, to their respective successors RAxML-NG and EPA-NG [9], which have been recently developed in our group. It will further benefit from the aforementioned coarse-grained parallelization in RAxML-NG.

List of Figures

List of Tables

List of Acronyms

AA Amino acid.

ADO Allelic DropOut.

AVX Advanced Vector Extensions.

bp base pair.

CLV Conditional Likelihood Vector.

CPU Central Processing Unit.

DNA Deoxyribonucleic acid.

EPA Evolutionary Placement Algorithm.

GB Gigabyte.

GFLOPS Giga (billions of) FLoating-point OPerations per Second.

GTR General Time Reversible.

KF Kuhner-Felsenstein (distance metric for trees).

KNC Knights Corner (Intel Xeon Phi accelerator).

KNL Knights Landing (Intel Xeon Phi accelerator).

MC Markov Chain.

ML Maximum Likelihood.

MP Maximum Parsimony.

MPI Message Passing Interface.

MSA Multiple Sequence Alignment.

NGS Next Generation Sequencing.

NJ Neighbour Joining.

NNI Nearest Neighbour Interchange.

PLF Phylogenetic Likelihood Function.

RF Robinson-Foulds (distance metric for trees).

RHAS Rate Heterogeneity Across Sites.

RNA Ribonucleic acid.

SIMD Single Instruction Multiple Data.

SPR Subtree Pruning and Regrafting.

TBR Tree Bisection and Reconnection.

Bibliography

[1] The bird 10,000 genomes (b10k) project. https://b10k.genomics.cn/index.html. Website. Accessed November 3, 2017.

[2] K. Abarenkov, R. Henrik Nilsson, K. H. Larsson, I. J. Alexander, U. Eberhardt, S. Erland, K. Hoiland, R. Kjoller, E. Larsson, T. Pennanen, R. Sen, A. F. Taylor, L. Tedersoo, B. M. Ursing, T. Vralstad, K. Liimatainen, U. Peintner, and U. Koljalg. The UNITE database for molecular identification of fungi– recent updates and future perspectives. *New Phytol*, 186(2):281–5, 2010.

[3] A. J. Aberer, K. Kobert, and A. Stamatakis. ExaBayes: Massively parallel bayesian tree inference for the whole-genome era. *Molecular Biology and Evolution*, 31(10):2553–2556, 2014.

[4] H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, Dec 1974.

[5] N. Alachiotis, E. Sotiriades, A. Dollas, and A. Stamatakis. Exploring FPGAs for accelerating the phylogenetic likelihood function. In *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–8, 2009.

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

[7] Aristotle. *Physics.* Hosted by MIT's Internet Classics Archive, retrieved 1 August 2017.

[8] S. Bank, M. Sann, C. Mayer, K. Meusemann, A. Donath, L. Podsiadlowski, A. Kozlov, M. Petersen, L. Krogmann, R. Meier, P. Rosa, T. Schmitt, M. Wurdack, S. Liu, X. Zhou, B. Misof, R. S. Peters, and O. Niehuis. Transcriptome and target DNA enrichment sequence data provide new insights into the phylogeny of vespid wasps (Hymenoptera: Aculeata: Vespidae). *Molecular Phylogenetics and Evolution*, 116(Supplement C):213 – 226, 2017.

[9] P. Barbera. EPA-NG: Massively parallel phylogenetic placement of genetic sequences. https://github.com/Pbdas/epa-ng, 2017. Website. Accessed November 20, 2017.

[10] P. Barbera, A. Kozlov, T. Flouri, D. Darriba, L. Czech, and A. Stamatakis. Massively parallel evolutionary placement of genetic sequences. In *ISC 2017 PhD Symposium*, Frankfurt am Main, Germany, June 2017.

[11] S. Berger, N. Alachiotis, and A. Stamatakis. An optimized reconfigurable system for computing the phylogenetic likelihood function on DNA data. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 352–359, 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, 2011.

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

[14] R. Biczok, P. Bozsoky, P. Eisenmann, J. Ernst, T. Ribizel, F. Scholz, A. Trefzer, F. Weber, M. Hamann, and A. Stamatakis. Two C++ libraries for counting trees on a phylogenetic terrace. *bioRxiv*, 2017.

[15] B. P. Boerner and N. E. Sarvetnick. Type 1 diabetes: role of intestinal microbiome in humans and mice. *Annals of the New York Academy of Sciences*, 1243(1):103–118, 2011.

[16] R. K. Bradley, A. Roberts, M. Smoot, S. Juvekar, J. Do, C. Dewey, I. Holmes, and L. Pachter. Fast statistical alignment. *PLOS Computational Biology*, 5(5):1–15, 05 2009.

[17] R. Brent. An algorithm with guaranteed convergence for finding a zero of a function. In *Algorithms for Minimization Without Derivatives*, Prentice-Hall series in automatic computation, chapter 4. Prentice-Hall, 1972.

[18] S. Capella-Gutiérrez, J. M. Silla-Martínez, and T. Gabaldón. trimAl: a tool for automated alignment trimming in large-scale phylogenetic analyses. *Bioinformatics*, 25(15):1972–1973, 2009.

[19] J. G. Caporaso, J. Kuczynski, J. Stombaugh, K. Bittinger, F. D. Bushman, E. K. Costello, N. Fierer, A. G. Pena, J. K. Goodrich, J. I. Gordon, G. A. Huttley, S. T. Kelley, D. Knights, J. E. Koenig, R. E. Ley, C. A. Lozupone, D. McDonald, B. D. Muegge, M. Pirrung, J. Reeder, J. R. Sevinsky, P. J. Turnbaugh, W. A. Walters, J. Widmann, T. Yatsunenko, J. Zaneveld, and R. Knight. QIIME allows analysis of high-throughput community sequencing data. *Nature Methods*, 7(5):335–336, 2010.

[20] W. E. Carroll. Creation, evolution, and Thomas Aquinas. *Revue des Questions Scientifiques*, 171(4):319–347, 2000.

[21] O. Chernomor, A. von Haeseler, and B. Q. Minh. Terrace aware data structure for phylogenomic inference from supermatrices. *Systematic Biology*, 65(6):997–1008, 2016.

[22] B. Chor and T. Tuller. Maximum likelihood of evolutionary trees is hard. In *Proceedings of the 9th Annual International Conference on Research in Computational Molecular Biology*, RECOMB'05, pages 296–310, Berlin, Heidelberg, 2005. Springer-Verlag.

[23] P. J. A. Cock, C. J. Fields, N. Goto, M. L. Heuer, and P. M. Rice. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Research*, 38(6):1767–1771, 2010.

[24] J. R. Cole, Q. Wang, J. A. Fish, B. Chai, D. M. McGarrell, Y. Sun, C. T. Brown, A. Porras-Alfaro, C. R. Kuske, and J. M. Tiedje. Ribosomal Database Project: data and tools for high throughput rRNA analysis. *Nucleic Acids Research*, 42(D1):D633–D642, 2014.

[25] R. E. Collins and P. G. Higgs. Testing the infinitely many genes model for the evolution of the bacterial core genome and pangenome. *Molecular Biology and Evolution*, 29(11):3413–3425, 2012.

[26] T. P. Curtis, W. T. Sloan, and J. W. Scannell. Estimating prokaryotic diversity and its limits. *Proceedings of the National Academy of Sciences*, 99(16):10494–10499, 2002.

[27] P. Danecek, A. Auton, G. Abecasis, C. A. Albers, E. Banks, M. A. DePristo, R. E. Handsaker, G. Lunter, G. T. Marth, S. T. Sherry, G. McVean, R. Durbin, and . The variant call format and VCFtools. *Bioinformatics*, 27(15):2156–2158, 2011.

[28] D. Darriba, D. Posada, and A. Stamatakis. ModelTest-NG: Best-fit evolutionary model selection. https://github.com/ddarriba/modeltest, 2017. Website. Accessed November 20, 2017.

[29] O. B. Dayhoff MO, Schwartz RM. A model of evolutionary change in proteins. In *Atlas of Protein Sequence and Structure.*, page 345–352. National Biomedical Research Foundation, Washington, DC, 1978.

[30] A. J. Drummond, M. A. Suchard, D. Xie, and A. Rambaut. Bayesian phylogenetics with BEAUti and the BEAST 1.7. *Molecular biology and evolution*, 29(8):1969–1973, 2012.

[31] D. A. R. Eaton. PyRAD: assembly of de novo RADseq loci for phylogenetic analyses . *Bioinformatics*, 30(13):1844–1849, 2014.

[32] S. R. Eddy. Multiple alignment using hidden Markov models. In *ISMB-95: Proceedings, Third International Conference on Intelligent Systems for Molecular Biology*, volume 3, pages 114–120. AAAI Press, 1995.

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

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

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

[36] P. Erixon, B. Svennblad, T. Britton, B. Oxelman, and J. Sullivan. Reliability of Bayesian posterior probabilities and bootstrap frequencies in phylogenetics. *Systematic Biology*, 52(5):665–673, 2003.

[37] B. Ewing and P. Green. Base-calling of automated sequencer traces using phred. ii. error probabilities. *Genome Research*, 8(3):186–194, 1998.

[38] J. Felsenstein. Cases in which parsimony or compatibility methods will be positively misleading. *Systematic Biology*, 27(4):401–410, 1978.

[39] J. Felsenstein. Evolutionary trees from DNA sequences: A maximum likelihood approach. *Journal of Molecular Evolution*, 17(6):368–376, 11 1981.

[40] J. Felsenstein. *Inferring phylogenies.* Sinauer Associates Sunderland, 2004.

[41] R. Fletcher. *Practical methods of optimization.* Number v. 1 in Wiley-interscience publication. Wiley, 1987.

[42] W. Fletcher and Z. Yang. INDELible: a flexible simulator of biological sequence evolution. *Molecular biology and evolution*, 26(8):1879–1888, 2009.

[43] D. Futuyma. The uses of evolutionary biology. *Science*, 267(5194):41–42, 1995.

[44] J. A. Gilbert, J. K. Jansson, and R. Knight. The Earth Microbiome project: successes and aspirations. *BMC Biology*, 12(1):1–4, 2014.

[45] R. Graham and L. Foulds. Unlikelihood that minimal phylogenies for a realistic biological study can be constructed in reasonable computational time. *Mathematical Biosciences*, 60(2):133–142, 1982.

[46] S. Greenblum, P. J. Turnbaugh, and E. Borenstein. Metagenomic systems biology of the human gut microbiome reveals topological shifts associated with obesity and inflammatory bowel disease. *Proceedings of the National Academy of Sciences*, 109(2):594–599, 2012.

[47] L. Guillou, D. Bachar, S. Audic, D. Bass, C. Berney, L. Bittner, C. Boutte, G. Burgaud, C. de Vargas, J. Decelle, J. Del Campo, J. R. Dolan, M. Dunthorn, B. Edvardsen, M. Holzmann, W. H. Kooistra, E. Lara, N. Le Bescot, R. Logares, F. Mahe, R. Massana, M. Montresor, R. Morard, F. Not, J. Pawlowski, I. Probert, A. L. Sauvadet, R. Siano, T. Stoeck, D. Vaulot, P. Zimmermann, and R. Christen. The Protist Ribosomal Reference database (PR2): a catalog of unicellular eukaryote small sub-unit rRNA sequences with curated taxonomy. *Nucleic Acids Reseach*, 41(Database issue):D597–604, 2013.

[48] S. Guindon, J.-F. Dufayard, V. Lefort, M. Anisimova, W. Hordijk, and O. Gascuel. New algorithms and methods to estimate maximum-likelihood phylogenies: assessing the performance of PhyML 3.0. *Systematic biology*, 59(3):307–321, 2010.

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

[50] W. K. Hastings. Monte Carlo sampling methods using Markov Chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[51] A. P. Hendry, M. T. Kinnison, M. Heino, T. Day, T. B. Smith, G. Fitt, C. T. Bergstrom, J. Oakeshott, P. S. Jørgensen, M. P. Zalucki, G. Gilchrist, S. Southerton, A. Sih, S. Strauss, R. F. Denison, and S. P. Carroll. Evolutionary principles and their practical application. *Evolutionary Applications*, 4(2):159–183, 2011.

[52] D. G. Higgins and P. M. Sharp. CLUSTAL: a package for performing multiple sequence alignment on a microcomputer. *Gene*, 73(1):237 – 244, 1988.

[53] D. Hillis and J. Wiens. Molecules versus morphology in systematics: conflicts, artifacts, and misconceptions. *Phylogenetic analysis of morphological data*, pages 1–19, 2000.

[54] Y. Hou, L. Song, P. Zhu, B. Zhang, Y. Tao, and X. Xu. Single-cell exome sequencing and monoclonal evolution of a JAK2-negative myeloproliferative neoplasm. *Cell*, 148, 2012.

[55] J. Huerta-Cepas, J. Dopazo, and T. Gabaldón. ETE: a Python Environment for Tree Exploration. *BMC bioinformatics*, 11(1):24, 2010.

[56] F. Husník, T. Chrudimský, and V. Hypša. Multiple origins of endosymbiosis within the Enterobacteriaceae (γ-Proteobacteria): convergence of complex phylogenetic approaches. *BMC Biology*, 9(1):87, Dec 2011.

[57] J. Huxley. *Evolution, the Modern Synthesis.* G. Allen & Unwin Limited, 1942.

[58] Illumina Inc. Quality scores for next-generation sequencing. techincal note. https://www.illumina.com/documents/products/technotes/technote_Q-Scores.pdf, 2011.

[59] F. Izquierdo-Carrasco, N. Alachiotis, S. Berger, T. Flouri, S. Pissis, and A. Stamatakis. A generic vectorization scheme and a GPU kernel for the phylogenetic likelihood library. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*, pages 530–538, 2013.

[60] F. Izquierdo-Carrasco, S. Smith, and A. Stamatakis. Algorithms, data structures, and numerics for likelihood-based phylogenetic inference of huge trees. *BMC bioinformatics*, 12(1):470, 2011.

[61] K. Jahn, J. Kuipers, and N. Beerenwinkel. Tree inference for single-cell data. *Genome Biol*, 17, 2016.

[62] E. D. Jarvis, S. Mirarab, A. J. Aberer, B. Li, P. Houde, C. Li, S. Y. Ho, B. C. Faircloth, B. Nabholz, J. T. Howard, et al. Whole-genome analyses resolve early branches in the tree of life of modern birds. *Science*, 346(6215):1320–1331, 2014.

[63] T. Jukes and C. Cantor. Evolution of protein molecules. In H. Munro, editor, *Mammalian Protein Metabolism*, pages 21–132. Academic Press, New York, USA, 1969.

[64] W. Just. Computational complexity of multiple sequence alignment with SP-score. *Journal of computational biology*, 8(6):615–623, 2001.

[65] K. Katoh and D. M. Standley. MAFFT multiple sequence alignment software version 7: Improvements in performance and usability. *Molecular Biology and Evolution*, 30(4):772–780, 2013.

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

[67] S. Klopfstein, L. Vilhelmsen, and F. Ronquist. A nonstationary Markov model detects directional evolution in Hymenopteran morphology. *Systematic Biology*, 64(6):1089–1103, 2015.

[68] K. Kobert, T. Flouri, A. Aberer, and A. Stamatakis. The divisible load balance problem and its application to phylogenetic inference. In D. Brown and B. Morgenstern, editors, *Algorithms in Bioinformatics*, volume 8701 of *Lecture Notes in Computer Science*, pages 204–216. Springer Berlin Heidelberg, 2014.

[69] K. Kobert, A. Stamatakis, and T. Flouri. Efficient detection of repeating sites to accelerate phylogenetic likelihood calculations. *Systematic Biology*, 66(2):205–217, 2017.

[70] A. D. Kostic, R. J. Xavier, and D. Gevers. The microbiome in inflammatory bowel disease: Current status and the future ahead. *Gastroenterology*, 146(6):1489 – 1499, 2014. The Gut Microbiome in Health and Disease.

[71] A. Kozlov, A. Stamatakis, D. Darriba, T. Flouri, and B. Morel. RAxML-NG: Next Generation tool for Maximum Likelihood phylogenetic inference. https://github.com/amkozlov/raxml-ng, 2017. Website. Accessed November 3, 2017.

[72] A. M. Kozlov, A. J. Aberer, and A. Stamatakis. ExaML version 3: a tool for phylogenomic analyses on supercomputers. *Bioinformatics*, pages 2577–2579, 2015.

[73] A. M. Kozlov, C. Goll, and A. Stamatakis. Efficient computation of the phylogenetic likelihood function on the Intel MIC architecture. In *Parallel Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, pages 518–527, May 2014.

[74] A. M. Kozlov, J. Zhang, P. Yilmaz, F. O. Glöckner, and A. Stamatakis. Phylogeny-aware identification and correction of taxonomically mislabeled sequences. *Nucleic acids research*, 44(11):5022–5033, 2016.

[75] R. Krishnaiyer, E. Kultursay, P. Chawla, S. Preis, A. Zvezdin, and H. Saito. Compiler-based data prefetching and streaming non-temporal store generation for the Intel(R) Xeon Phi(TM) coprocessor. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*, pages 1575–1586, 2013.

[76] M. K. Kuhner and J. Felsenstein. A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates. *Molecular Biology and Evolution*, 11(3):459–468, 1994.

[77] M. K. Kuhner and J. McGill. Correcting for sequencing error in maximum likelihood phylogeny inference. *G3: Genes, Genomes, Genetics*, 4(12):2545–2552, 2014.

[78] N. C. Kyrpides, P. Hugenholtz, J. A. Eisen, T. Woyke, M. Göker, C. T. Parker, R. Amann, B. J. Beck, P. S. G. Chain, J. Chun, R. R. Colwell, A. Danchin, P. Dawyndt, T. Dedeurwaerdere, E. F. DeLong, J. C. Detter, P. De Vos, T. J. Donohue, X.-Z. Dong, D. S. Ehrlich, C. Fraser, R. Gibbs, J. Gilbert, P. Gilna, F. O. Glöckner, J. K. Jansson, J. D. Keasling, R. Knight, D. Labeda, A. Lapidus, J.-S. Lee, W.-J. Li, J. MA, V. Markowitz, E. R. B. Moore, M. Morrison, F. Meyer, K. E. Nelson, M. Ohkuma, C. A. Ouzounis, N. Pace, J. Parkhill, N. Qin, R. Rossello-Mora, J. Sikorski, D. Smith, M. Sogin, R. Stevens, U. Stingl, K.-i. Suzuki, D. Taylor, J. M. Tiedje, B. Tindall, M. Wagner, G. Weinstock, J. Weissenbach, O. White, J. Wang, L. Zhang, Y.-G. Zhou, D. Field, W. B. Whitman, G. M. Garrity, and H.-P. Klenk. Genomic encyclopedia of bacteria and archaea: Sequencing a myriad of type strains. *PLOS Biology*, 12(8):1–7, 08 2014.

[79] R. Lanfear, B. Calcott, S. Y. W. Ho, and S. Guindon. PartitionFinder: Combined selection of partitioning schemes and substitution models for phylogenetic analyses. *Molecular Biology and Evolution*, 29(6):1695–1701, 2012.

[80] N. Lartillot, T. Lepage, and S. Blanquart. PhyloBayes 3: a bayesian software package for phylogenetic reconstruction and molecular dating. *Bioinformatics*, 25(17):2286–2288, 2009.

[81] N. Lartillot and H. Philippe. A Bayesian mixture model for across-site heterogeneities in the amino-acid replacement process. *Molecular Biology and Evolution*, 21(6):1095–1109, 2004.

[82] S. Q. Le and O. Gascuel. An improved general amino acid replacement matrix. *Molecular Biology and Evolution*, 25(7):1307–1320, 2008.

[83] A. D. Leaché, B. L. Banbury, J. Felsenstein, A. n.-M. de Oca, and A. Stamatakis. Short tree, long tree, right tree, wrong tree: New acquisition bias corrections for inferring SNP phylogenies. *Systematic Biology*, 64(6):1032–1047, 2015.

[84] F. Lemoine, J.-B. Domelevo Entfellner, E. Wilkinson, T. De Oliveira, and O. Gascuel. Boosting Felsenstein phylogenetic bootstrap. *bioRxiv*, 2017.

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

[86] K. Liu, C. R. Linder, and T. Warnow. RAxML and FastTree: Comparing two methods for large-scale maximum likelihood phylogeny estimation. *PLOS ONE*, 6(11):1–11, 11 2011.

[87] 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. *Nucleic Acids Research*, 32(4):1363–1371, 2004.

[88] F. Mahé, C. de Vargas, D. Bass, L. Czech, A. Stamatakis, E. Lara, D. Singer, J. Mayor, J. Bunge, S. Sernaker, et al. Parasites dominate hyperdiverse soil protist communities in Neotropical rainforests. *Nature Ecology & Evolution*, 1:0091, 2017.

[89] F. A. Matsen, R. B. 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, Oct 2010.

[90] D. McDonald, M. N. Price, J. Goodrich, E. P. Nawrocki, T. Z. DeSantis, A. Probst, G. L. Andersen, R. Knight, and P. Hugenholtz. An improved Greengenes taxonomy with explicit ranks for ecological and evolutionary analyses of bacteria and archaea. *The ISME journal*, 6:610–8, 2012.

[91] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kernytsky, K. Garimella, D. Altshuler, S. Gabriel, M. Daly, and M. A. DePristo. The Genome Analysis Toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Research*, 20(9):1297–1303, 2010.

[92] B. Misof, S. Liu, K. Meusemann, R. S. Peters, A. Donath, C. Mayer, P. B. Frandsen, J. Ware, T. Flouri, R. G. Beutel, et al. Phylogenomics resolves the timing and pattern of insect evolution. *Science*, 346(6210):763–767, 2014.

[93] B. M. Moret, U. Roshan, and T. Warnow. Sequence-length requirements for phylogenetic methods. *Lecture Notes in Computer Science*, 2452:343–356, 2002.

[94] L. G. Nagy, R. A. Ohm, G. M. Kovács, D. Floudas, R. Riley, A. Gácser, M. Sipiczki, J. M. Davis, S. L. Doty, G. S. De Hoog, et al. Latent homology and convergent regulatory evolution underlies the repeated emergence of yeasts. *Nature communications*, 5:4471, 2014.

[95] N. Navin. Cancer genomics: one cell at a time. *Genome Biol*, 15, 2014.

[96] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443 – 453, 1970.

[97] C. Newburn, R. Deodhar, S. Dmitriev, R. Murty, R. Narayanaswamy, J. Wiegert, F. Chinchilla, and R. McGuire. Offload compiler runtime for the Intel® Xeon Phi coprocessor. In J. M. Kunkel, T. Ludwig, and H. W. Meuer, editors, *Supercomputing*, volume 7905 of *Lecture Notes in Computer Science*, pages 239–254. Springer Berlin Heidelberg, 2013.

[98] L.-T. Nguyen, H. A. Schmidt, A. von Haeseler, and B. Q. Minh. IQ-TREE: A fast and effective stochastic algorithm for estimating maximum-likelihood phylogenies. *Molecular Biology and Evolution*, 32(1):268–274, 2015.

[99] R. H. Nilsson, L. Tedersoo, K. Abarenkov, M. Ryberg, E. Kristiansson, M. Hartmann, C. L. Schoch, J. A. A. Nylander, J. Bergsten, T. M. Porter, A. Jumpponen, P. Vaishampayan, O. Ovaskainen, N. Hallenberg, J. Bengtsson-Palme, K. M. Eriksson, K.-H. Larsson, E. Larsson, and U. Kõljalg. Five simple guidelines for establishing basic authenticity and reliability of newly generated fungal its sequences. *MycoKeys*, 4:37–63, sep 2012.

[100] Nomenclature Committee of the International Union of Biochemistry (NC-IUB). Nomenclature for incompletely specified bases in nucleic acid sequences: Recommendations 1984. *Proceedings of the National Academy of Sciences of the United States of America*, 83(1):4–8, 1986.

[101] D. Normile. Plant scientists plan massive effort to sequence 10,000 genomes. http://www.sciencemag.org/news/2017/07/plant-scientists-plan-massive-effort-sequence-10000-genomes, 2017. Website. Accessed November 3, 2017.

[102] M. Nute and T. Warnow. Scaling statistical multiple sequence alignment to large datasets. *BMC Genomics*, 17(10):764, Nov 2016.

[103] N. Pace. A molecular view of microbial diversity and the biosphere. *Science*, 276:734–740, 1997.

[104] N. D. Pattengale, M. Alipour, O. R. Bininda-Emonds, B. M. Moret, and A. Stamatakis. How many bootstrap replicates are necessary? *Journal of Computational Biology*, 17(3):337–354, 2010.

[105] E. Pennisi. Biologists propose to sequence the DNA of all life on earth. http://www.sciencemag.org/news/2017/02/biologists-propose-sequence-dna-all-life-earth, 2017. Website. Accessed November 3, 2017.

[106] R. S. Peters, L. Krogmann, C. Mayer, A. Donath, S. Gunkel, K. Meusemann, A. Kozlov, L. Podsiadlowski, M. Petersen, R. Lanfear, P. A. Diez, J. Heraty, K. M. Kjer, S. Klopfstein, R. Meier, C. Polidori, T. Schmitt, S. Liu, X. Zhou, T. Wappler, J. Rust, B. Misof, and O. Niehuis. Evolutionary history of the hymenoptera. *Current Biology*, 27(7):1013 – 1018, 2017.

[107] R. O. Prum, J. S. Berv, A. Dornburg, D. J. Field, J. P. Townsend, E. M. Lemmon, and A. R. Lemmon. A comprehensive phylogeny of birds (Aves) using targeted next-generation DNA sequencing. *Nature*, 526(7574):569–573, 2015.

[108] R. A. Pyron, F. T. Burbrink, G. R. Colli, A. N. M. de Oca, L. J. Vitt, C. A. Kuczynski, and J. J. Wiens. The phylogeny of advanced snakes (Colubroidea), with discovery of a new subfamily and comparison of support methods for likelihood trees. *Molecular Phylogenetics and Evolution*, 58(2):329 – 342, 2011.

[109] C. Quast, E. Pruesse, P. Yilmaz, J. Gerken, T. Schweer, P. Yarza, J. Peplies, and F. O. Glockner. The SILVA ribosomal RNA gene database project: improved data processing and web-based tools. *Nucleic Acids Research*, 41(Database issue):D590–6, 2013.

[110] A. Rambaut. FigTree v1.4.3, October 2016.

[111] O. Ratmann, C. Wymant, C. Colijn, S. Danaviah, M. Essex, S. D. Frost, A. Gall, s. gaiseitsiwe, M. Grabowski, R. Gray, et al. HIV-1 full-genome phylogenetics of generalized epidemics in sub-Saharan Africa: impact of missing nucleotide characters in next-generation sequences. *AIDS Research and Human Retroviruses*, 33(11):1083–1098, November 2017.

[112] S. Ratnasingham and P. D. N. Hebert. BOLD: The Barcode of Life Data system. *Molecular Ecology Notes*, 7(3):355–364, 2007.

[113] D. Robinson and L. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53(1):131 – 147, 1981.

[114] S. Roch. A short proof that phylogenetic tree reconstruction by maximum likelihood is hard. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 3(1):92, 2006.

[115] C. A. Ronan. *The Shorter Science and Civilisation in China: An Abridgement by Colin A. Ronan of Joseph Needham's Original Text.* Cambridge; New York: Cambridge University Press, 1995.

[116] F. Ronquist, M. Teslenko, P. Van Der Mark, D. L. Ayres, A. Darling, S. Höhna, B. Larget, L. Liu, M. A. Suchard, and J. P. Huelsenbeck. MrBayes 3.2: efficient Bayesian phylogenetic inference and model choice across a large model space. *Systematic biology*, 61(3):539–542, 2012.

[117] C. Rosales. Porting to the Intel Xeon Phi: Opportunities and challenges. 2013.

[118] E. M. Ross and F. Markowetz. OncoNEM: inferring tumor evolution from single-cell sequencing data. *Genome Biol*, 17, 2016.

[119] B. Rulik, J. Eberle, L. von der Mark, J. Thormann, M. Jung, F. Köhler, W. Apfel, A. Weigel, A. Kopetz, J. Köhler, F. Fritzlar, M. Hartmann, K. Hadulla, J. Schmidt, T. Hörren, D. Krebs, F. Theves, U. Eulitz, A. Skale, D. Rohwedder, A. Kleeberg, J. J. Astrin, M. F. Geiger, J. W. Wägele, P. Grobe, and D. Ahrens. Using taxonomic consistency with semi-automated data pre-processing for high quality DNA barcodes. *Methods in Ecology and Evolution*, 2017.

[120] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4):406–425, 1987.

[121] M. J. Sanderson, M. J. Donoghue, W. H. 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.

[122] M. J. Sanderson, M. M. McMahon, and M. Steel. Terraces in phylogenetic tree space. *Science*, 333(6041):448–450, 2011.

[123] D. Sankoff. Minimal mutation trees of sequences. *SIAM Journal on Applied Mathematics*, 28(1):35–42, 1975.

[124] S. Schloissnig, M. Arumugam, S. Sunagawa, M. Mitreva, J. Tap, A. Zhu, A. Waller, D. Mende, J. Kultima, J. Martin, K. Kota, S. Sunyaev, G. Weinstock, and P. Bork. Genomic variation landscape of the human gut microbiome. *Nature*, 493(7430):45–50, 2013.

[125] R. Schwartz and A. A. Schäffer. The evolution of tumour phylogenetics: principles and practice. *Nature Reviews Genetics*, 18(4):213–229, 2017.

[126] G. Schwarz. Estimating the dimension of a model. *Ann. Statist.*, 6(2):461–464, 03 1978.

[127] F. Sievers, A. Wilm, D. Dineen, T. J. Gibson, K. Karplus, W. Li, R. Lopez, H. McWilliam, M. Remmert, J. Söding, J. D. Thompson, and D. G. Higgins. Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega. *Molecular Systems Biology*, 7(1), 2011.

[128] R. R. Sokal and C. D. Michener. A statistical method for evaluating systematic relationships. *University of Kansas Science Bulletin*, 38:1409–1438, 1958.

[129] A. Stamatakis. Phylogenetic Models of Rate Heterogeneity: A High Performance Computing Perspective. In *Proc. of IPDPS2006*, HICOMB Workshop, Proceedings on CD, Rhodos, Greece, April 2006.

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

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

[132] A. Stamatakis. *Phylogenetic Search Algorithms for Maximum Likelihood*, pages 547–577. John Wiley & Sons, Inc., 2011.

[133] A. Stamatakis. RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics*, 30(9):1312–1313, 2014.

[134] A. Stamatakis. RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics*, 30(9):1312–1313, 2014.

[135] A. Stamatakis and A. Aberer. Novel parallelization schemes for large-scale likelihood-based phylogenetic inference. In *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 1195–1204, 2013.

[136] A. Stamatakis and N. Alachiotis. Time and memory efficient likelihood-based tree searches on phylogenomic alignments with missing data. *Bioinformatics*, 26(12):i132–i139, 2010.

[137] 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.

[138] M. Stark, S. A. Berger, A. Stamatakis, and C. von Mering. MLTreeMap - accurate maximum likelihood placement of environmental DNA sequences into taxonomic and functional reference phylogenies. *BMC Genomics*, 11(1):461, Aug 2010.

[139] M. A. Steel and L. A. Székely. Inverting random functions ii: Explicit bounds for discrete maximum likelihood estimation, with applications. *SIAM Journal on Discrete Mathematics*, 15(4):562–575, 2002.

[140] 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.

[141] M. A. Suchard and A. Rambaut. Many-core algorithms for statistical phylogenetics. *Bioinformatics*, 25(11):1370, 2009.

[142] M. A. Suchard and B. D. Redelings. BAli-Phy: simultaneous Bayesian inference of alignment and phylogeny. *Bioinformatics*, 22(16):2047–2048, 2006.

[143] J. Sukumaran and M. T. Holder. DendroPy: a Python library for phylogenetic computing. *Bioinformatics*, 26(12):1569–1571, 2010.

[144] G. Talavera and J. Castresana. Improvement of phylogenies after removing divergent and ambiguously aligned blocks from protein sequence alignments. *Systematic biology*, 56(4):564–577, 2007.

[145] G. Tan, M. Muffato, C. Ledergerber, J. Herrero, N. Goldman, M. Gil, and C. Dessimoz. Current methods for automated filtering of multiple sequence alignments frequently worsen single-gene phylogenetic inference. *Systematic biology*, 64(5):778–791, 2015.

[146] S. Tavaré. *Some Probabilistic and Statistical Problems in the Analysis of DNA Sequences*, volume 17, pages 57–86. Amer Mathematical Society, 1986.

[147] S. M. Utturkar, D. M. Klingeman, J. M. Bruno-Barcena, M. S. Chinn, A. M. Grunden, M. Köpke, and S. D. Brown. Sequence data for Clostridium autoethanogenum using three generations of sequencing technologies. *Scientific data*, 2, 2015.

[148] C. von Mering, P. Hugenholtz, J. Raes, S. Tringe, T. Doerks, L. Jensen, N. Ward, and P. Bork. Quantitative phylogenetic assessment of microbial communities in diverse environments. *Science*, 315(5815):1126–30, 2007.

[149] Q. Wang, G. M. Garrity, J. M. Tiedje, and J. R. Cole. Naïve Bayesian classifier for rapid assignment of rRNA sequences into the new bacterial taxonomy. *Applied and Environmental Microbiology*, 73(16):5261–5267, 2007.

[150] Y. Wang, J. Waters, M. L. Leung, A. Unruh, W. Roh, and X. Shi. Clonal evolution in breast cancer revealed by single nucleus genome sequencing. *Nature*, 512, 2014.

[151] N. V. Whelan, K. M. Kocot, L. L. Moroz, and K. M. Halanych. Error, signal, and the placement of Ctenophora sister to all other animals. *Proceedings of the National Academy of Sciences*, 112(18):5773–5778, 2015.

[152] S. Whelan and N. Goldman. A general empirical model of protein evolution derived from multiple protein families using a maximum-likelihood approach. *Molecular Biology and Evolution*, 18(5):691–699, 2001.

[153] N. J. Wickett, S. Mirarab, N. Nguyen, T. Warnow, E. Carpenter, N. Matasci, S. Ayyampalayam, M. S. Barker, J. G. Burleigh, M. A. Gitzendanner, et al. Phylotranscriptomic analysis of the origin and early diversification of land plants. *Proceedings of the National Academy of Sciences*, 111(45):E4859–E4868, 2014.

[154] B. M. Wiegmann, M. D. Trautwein, I. S. Winkler, N. B. Barr, J.-W. Kim, C. Lambkin, M. A. Bertone, B. K. Cassel, K. M. Bayless, A. M. Heimberg, B. M. Wheeler, K. J. Peterson, T. Pape, B. J. Sinclair, J. H. Skevington, V. Blagoderov, J. Caravas, S. N. Kutty, U. Schmidt-Ott, G. E. Kampmeier, F. C. Thompson, D. A. Grimaldi, A. T. Beckenbach, G. W. Courtney, M. Friedrich, R. Meier, and D. K. Yeates. Episodic radiations in the fly tree of life. *Proceedings of the National Academy of Sciences*, 108(14):5690–5695, 2011.

[155] C. R. Woese, O. Kandler, and M. L. Wheelis. Towards a natural system of organisms: proposal for the domains Archaea, Bacteria, and Eucarya. *Proceedings of the National Academy of Sciences of the United States of America*, 87(12):4576–4579, 1990.

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

[157] Z. Yang. Statistical properties of the maximum likelihood method of phylogenetic estimation and comparison with distance matrix methods. *Systematic Biology*, 43(3):329–342, 1994.

[158] Z. Yang. A space-time process model for the evolution of DNA sequences. *Genetics*, 139(2):993–1005, 1995.

[159] Z. Yang. *Molecular Evolution: A Statistical Approach.* OUP Oxford, 2014.

[160] Z. Yang, R. Nielsen, N. Goldman, and A.-M. K. Pedersen. Codon-substitution models for heterogeneous selection pressure at amino acid sites. *Genetics*, 155(1):431–449, 2000.

[161] P. Yarza, M. Richter, J. Peplies, J. Euzeby, R. Amann, K.-H. Schleifer, W. Ludwig, F. O. Glöckner, and R. Rosselló-Móra. The All-Species Living Tree project: A 16S rRNA-based phylogenetic tree of all sequenced type strains. *Systematic and Applied Microbiology*, 31(4):241 – 250, 2008.

[162] H. Zafar, A. Tzen, N. Navin, K. Chen, and L. Nakhleh. SiFit: inferring tumor trees from single-cell sequencing data under finite-sites models. *Genome Biology*, 18(1):178, Sep 2017.

[163] A. E. Zanne, D. C. Tank, W. K. Cornwell, J. M. Eastman, S. A. Smith, R. G. FitzJohn, D. J. McGlinn, B. C. O'Meara, A. T. Moles, P. B. Reich, et al. Three keys to the radiation of angiosperms into freezing environments. *Nature*, 506(7486):89–92, 2014.

[164] X. Zhou, X.-X. Shen, C. T. Hittinger, and A. Rokas. Evaluating fast maximum likelihood-based phylogenetic programs using empirical phylogenomic data sets. *bioRxiv*, 2017.

[165] S. Zierke and J. Bakos. FPGA acceleration of the phylogenetic likelihood function for Bayesian MCMC inference methods. *BMC Bioinformatics*, 11(1):184, 2010.

[166] D. J. Zwickl. *Genetic algorithm approaches for the phylogenetic analysis of large biological sequence datasets under the maximum likelihood criterion.* PhD thesis, The University of Texas at Austin, 2006.