Algorithmic Advancements and Massive Parallelism for Large-Scale Datasets in Phylogenetic Bayesian Markov Chain Monte Carlo

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

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

genehmigte

Dissertation

von

Andre Jakob Aberer

aus München

Tag der mündlichen Prüfung:	7. Dezember 2015
Erster Gutachter:	Prof. Dr. Alexandros Stamatakis Karlsruher Institut für Technologie
Zweiter Gutachter:	Prof. Dr. Bernhard Misof Universität Bonn

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.

Frankfurt, 24. April 2016 ______________________________
Andre J. Aberer

Zusammenfassung

Ein bedeutendes Forschungsgebiet im Bereich der Bioinformatik ist die Phyloinformatik. In der Phyloinformatik werden gegenwärtig statistische Modelle verwendet um auf evolutionäre Verwandschaftsbeziehungen noch bestehender Spezies zu schließen. Anders gesagt, es wird versucht den Baum des Lebens auf der Basis von genetischen Eingabedaten *in silico* zu berechnen. Im Verlauf des letzten Jahrzehnts hat der technologische Fortschritt bei der Sequenzierung im Nasslabor zu Datensätzen bislang unbekannten Ausmaßes geführt. Gegebene analytische Methoden auf stetig wachsenden Datensätzen anzuwenden, stellt eine Herausforderung für die Informatik dar. Somit wird die Skalierbarkeit von Software so wichtig wie die kontinuierliche Weiterentwicklung der zugrunde liegenden Methoden und Algorithmen.

Die bayesianische Statistik bietet ein etabliertes Verfahren in der Phyloinformatik, um evolutionäre Bäume unter komplexen Modellen mittels der Markov-Ketten-Monte-Carlo-Methode (MCMC) zu berechnen. Neuartige Datensätze, die ganze Genome umfassen, haben Hauptspeicher- und Laufzeitanforderungen, die nur mit Hilfe großer Computer-Cluster und Supercomputer bewältigt werden können. Die Skalierbarkeit bestehender Anwendungen beschränkt sich jedoch auf höchstens einen Computerknoten mit geteiltem Hauptspeicher (der somit mehrere CPUs umfasst). In dieser Arbeit wird ein benutzerfreundliches Softwarepaket vorgestellt, das über eine dreistufige MPI-Pthread-Hybrid-Parallelisierung verfügt. Bei ausreichenden Computerressourcen können damit dem Stand der Technik entsprechende Analysen auf Datensätzen von fast unbeschränktem Ausmaß durchgeführt werden. Die Software integriert bestehende Optimierungen (zum Beispiel zur Speicherreduzierung), die für die Maximum-Likelihood-Methode entwickelt wurden. Darüber hinaus setzt die Software technische Verbesserungen um, die spezifisch für Berechnungen mittels bayesianischer Statistik entwickelt wurden. Es werden weiterhin Optimierungen eingeführt, die von generellem Interesse für Anwender der MCMC-Methode sind. Konkret handelt es sich hierbei einen nicht-blockierenden Algorithmus für die Metropolis-Kopplung von Markov-Ketten.

Zur methodischen Verbesserung des bayesianischen MCMC-Verfahrens in der Phyloinformatik wird ein Aspekt untersucht, der ungefähr 50% der Gesamtlaufzeit der Methode in Anspruch nimmt: Hierbei handelt es sich um das Sampling von Kantenlängen des evolutionären Baumes. Zunächst wird eine neuartige Charakterisierung der A-posteriori-Verteilung von evolutionären Kantenlängen vorgenommen. Daraufhin wird ein Vorschlagsmechanismus für das MCMC-Verfahren abgeleitet, der wesentlich effizienter ist als herkömmliche Kantenlängen-Vorschlagsmechanismen. Dies wird erreicht, indem Kantenlängen gemäß einer Γ-Verteilung vorgeschlagen werden, welche die A-Posteriori-Verteilung der Kantenlängen unter Zuhilfenahme des Newton-Raphson-Verfahrens approximiert. Im Zuge der Newton-Raphson-Optimierung liegen optimale Kantenlängen und die Ableitungen der unkorrigierten logarithmischen A-posteriori-Wahrscheinlichkeit vor, die den Ausgangspunkt der Approximation darstellen. Des Weiteren wird untersucht, wie sich die A-posteriori-Wahrscheinlichkeiten von Kantenlängen ändern, wenn eine neue Topologie vorgeschlagen wird. Hierauf aufbauend, werden Hybridvorschlagsmechanismen evaluiert, die Topologie und Kantenlängen gleichzeitig vorschlagen. Es ist darauf zu schließen, dass für eine vorgeschlagene Topologie suboptimale Kantenlängen kein essentielles Hindernis für die Effizienz gängiger Topologievorschlagsmechanismen darstellen.

Schlussendlich wird das so-genannte Schurken-Taxon-Problem behandelt, in dem es um Spezies geht, deren Position sich im Baum des Lebens nicht eindeutig festlegen lässt. In einer Bootstrap-Analyse (zur Ermittlung von Konfidenzwerten für evolutionäre Verwandtschaftsbeziehungen) können Schurken-Taxa verschiedene evolutionäre Positionen in den zugrunde liegenden

Bootstrap-Replikat-Bäumen annehmen. Der Informationsgehalt der Statistik einer Bootstrap-Analyse lässt sich erhöhen, indem Schurken-Taxa aus den Bootstrap-Bäumen entfernt werden. Die Identifikation einer Menge von Schurken-Taxa, die den Informationsgehalt optimiert, ist $\mathcal{NP}$-hart. In dieser Arbeit wird ein exakterer Algorithmus zur Lösung diesen Problems eingeführt (dem jedoch immer noch ein Greedy-Verfahren zugrunde liegt). Die effiziente Implementierung dieses Algorithmus liefert Laufzeitverbesserungen von bis zu drei Größenordnungen im Vergleich zu einem zuvor publizierten Algorithmus. Der Algorithmus übertrifft einen ähnlichen wesentlich stärker heuristisch orientierten Algorithmus sowohl in Laufzeit als auch bezüglich der Qualität (das heißt dem Informationsgehalt) des Ergebnisses. Im Rahmen der resultierenden parallelisierten Methode wird auch ein Webserver (mit Visualisierung) zur Verfügung gestellt, der es Benutzern erlaubt, die Ergebnisse verschiedener algorithmischer Parametrisierungen zu vergleichen. Somit können Benutzer auf interaktive Weise in ihren Datensätzen andernfalls verborgene Hypothesen empirisch sichtbar machen.

Abstract

One of the prominent research areas in the field of bioinformatics is phyloinformatics. Current phyloinformatics employs statistical models for unraveling evolutionary relationships among extant species. In other words, we strive to infer the tree of life *in silico* based upon genetic input data. Over the past decade, advances in wet-lab sequencing technology (namely, next-generation sequencing and third-generation sequencing) have given rise to datasets of unprecedented dimensions. These datasets induce a high computational burden for analytical methods. Thus, improving scalability is as important as continued methodological and algorithmic improvements.

Bayesian inference is a well-established method in phyloinformatics for reconstructing evolutionary trees under complex evolutionary models using the Markov chain Monte Carlo (MCMC) method. Emerging genome-sized datasets have memory and runtime requirements that can only be met by large clusters and supercomputers, yet scalability of existing applications is limited to at most one shared-memory computing node (i.e., several CPUs). We introduce a user-friendly software package employing a three-tier MPI-Pthreads hybrid parallelization that (given enough resources) allows conducting state-of-the-art analyses on datasets of almost arbitrary size. This tool integrates existing optimizations (e.g., memory saving techniques) that have been developed for maximum likelihood inference. Furthermore, it contains technical improvements that are specific to Bayesian inference. We introduce optimizations that are of general interest to practitioners of MCMC, such as a non-blocking algorithm for Metropolis-coupled chains.

For the methodological improvement of phylogenetic Bayesian MCMC, we examine an aspect that accounts for approximately 50% of the overall runtime in Bayesian phylogenetic inference: sampling the branch lengths of evolutionary trees. First, we present a novel characterization of the posterior distribution of evolutionary branch lengths. Then, we derive a proposal mechanism for the MCMC that is substantially more efficient than traditional branch length proposals. We achieve this by proposing new branches according to a Γ distribution that approximates the branch length posterior via the Newton-Raphson method. Thereby, we obtain optimized branch lengths and their derivatives of the uncorrected logarithmic posterior probability that form the basis of our approximation. Moreover, we examine how branch length posteriors change when new tree topologies are proposed and evaluate hybrid proposals that propose a topology and new branch lengths simultaneously. We find that, for popular topological proposals, branch lengths that are suboptimal for the proposed topology do not affect MCMC efficiency substantially.

Finally, the so-called rogue taxon problem revolves around species with an unclear position in the tree of life. In a bootstrap analysis (which tests statistical support for evolutionary relationships), rogue taxa may assume different positions in the bootstrap replicate topologies. Excluding these species from the tree set results in more informative summary statistics. However, identifying a set of taxa that maximizes informativeness is $\mathcal{NP}$-hard. In this thesis, we present a more exact graph-based algorithm for determining rogue taxa (that however still is greedy in nature). Its efficient implementation results in a runtime improvement of more than three orders of magnitude compared to a previously published naïve algorithm. It outperforms a related strongly heuristic algorithm both in terms of runtime and quality (i.e., informativeness) of the result. The resulting parallelized method is accompanied by a web server (including visualization) that allows users to compare the results of various algorithm parametrizations. Thus, users can interactively explore and uncover hypotheses about otherwise hidden evolutionary relationships in their datasets.

Acknowledgments

First and foremost, I would like to thank Prof. Alexandros Stamatakis for several years of excellent scientific supervision, for sharing his expertise and for providing me with guidance and freedom in equal parts for accomplishing this thesis.

I am grateful to Prof. Bernhard Misof for his interest in my research and for agreeing to review this thesis. Furthermore, I would like to express my sincere gratitude to Prof. Fredrik Ronquist for hosting my research stay at the *Naturhistoriska riksmuseet* in Stockholm, many discussions on Bayesian matters and the intensive research experience on Öland.

On a personal note, I am thankful for the consistent support of my parents Jakob and Anita and my brother Dominik. In particular, my deepest gratitude goes to Sabrina with whom I could share the ups and downs of my scientific development like with no other person.

As a research venue, the Exelixis lab excels by bringing together a talented group of people with diverse backgrounds. Thus, a thanks for many hours of interesting discussions and collaboration goes to Fernando Izquierdo-Carrasco, Simon Berger, Nikos Alachiotis, Pavlos Pavlidis, Kassian Kobert, Jiajie Zhang, Solon Pissis, Tomáš Flouri, Diego Darriba, Paschalia Kapli, Alexey Kozlov and Lucas Czech. Moreover, I am happy that my paths crossed with Mark Holder, Emily McTavish, Will Pearse and Christian Goll.

Finally, I would like to express my gratitude to Klaus Tschira, founder of the *Heidelberg Institute for Theoretical Studies* which provided the funding for my position.

Contents

1 Introduction

1.1 Motivation

The theory of evolution through natural selection as popularized by Charles Darwin is not merely the very foundation to evolutionary biology as it is known and taught today. Ideas that go along with this theory comprise (i) the concept that all life can be traced back to a single common universal ancestor, (ii) that species adapt to their environment in a process driven by the struggle for existence, and (iii) that traits of individuals within a species can vary and are passed down to offsprings. Darwin implied that these natural laws have to apply to the *Homo sapiens* as well. The impact that these ideas had on science, society, and human self-perception can not be overrated.

Today, the idea that the branching patterns of species give rise to the tree of life (TOL) is widely acknowledged. For inferring evolutionary relationships among species, early evolutionary biologists were restricted to using the morphological traits of species as their primary data source. After the discovery of the structure of the desoxyribonucleic acid (DNA) by James Watson and Francis Crick, DNA sequencing arose in the 1970ies and provided molecular data as an alternative data source. Through continued technological progress, the growth of the amount of available sequence data is analogous to the exponential growth of available computing resources. With the introduction of second-generation techniques, the growth of available sequence data even surpasses the increase in available computing power. Wet-lab sequencing progress culminated in the sequencing of the human genome in 2003. Today, more than ten years later, datasets have become available that comprise the genomes of several species. These datasets are assembled to provide definite answers about the evolutionary relationship among the species under examination.

Involved statistical models have been developed for inferring evolutionary trees from sequence data. While computational inference under these models initially was deemed unfeasible, technological progress and algorithmic advancements lead to the inception of the field of phyloinformatics as a sub-discipline of bioinformatics and scientific computing. Tree inference using Bayesian statistics is highly popular, however no high-performance computing (HPC) software is available that allows to harness supercomputing resources for evolutionary analyses of challenging datasets. Methodologically, a key component of Bayesian estimation of the TOL are so-called proposal mechanisms. The lack of efficiency of these proposal functions often renders the Bayesian method unfeasible for datasets with weak signal. Finally, even the post-analysis of the output of evolutionary inference tools can become computationally prohibitive. For instance, Bayesian inference can produce output with millions of tree samples. Computing a consensus tree (i.e., a summary statistic of sampled trees) and refining the result using a so-called rogue taxa analysis can be computationally prohibitive for existing methods.

1.2 Scientific Contribution

The central contribution of this thesis is the development of a user-friendly production-level HPC software package (called ExaBayes) for the Bayesian inference of evolutionary trees from datasets of almost arbitrary size (given sufficient computing resources) using state-of-the-art statistical models. First and foremost, ExaBayes features a novel three-tier MPI/thread hybrid parallelization (at data-level, chain-level and run-level). We demonstrate, that ExaBayes scales up to 32,768 CPU cores. Thus, ExaBayes is capable of reducing the runtime of a small analysis from 1 day 4 hours to 43.6 seconds (whereas short refers to an analysis parameter that usually must be chosen much larger in order to achieve more accurate results). For chain-level parallelism, ExaBayes introduces a non-blocking algorithm that increases parallel efficiency from 50% to 60% in the best case. Furthermore, as a HPC tool, ExaBayes implements adapted memory saving techniques that have previously been developed for maximum likelihood inference [56] and also implements an improved load balance algorithm developed by Kobert *et al.* [60] Also, ExaBayes introduces a load balance mechanism that is specific to Bayesian MCMC on highly partitioned datasets (i.e., distinct evolutionary parameters are assumed for different parts of the input data). We also report on first experiences with analyzing whole-genome datasets using ExaBayes.

The Bayesian framework used in ExaBayes requires a plethora of proposal functions. Several improvements to existing proposal functions (which are implemented in ExaBayes) are discussed in the thesis. Apart from that, we introduce a novel branch length proposal that uses a highly accurate approximation of the target branch length posterior distribution to efficiently propose branch lengths. This proposal is unique in the sense that, it allows to propose branch lengths *de novo*, that is, without relying on previous branch lengths. The design of this proposal required to analyze and characterize branch lengths in the Bayesian framework in-depth, an effort that has not been conducted before in that way. In addition, we examine how inferred branch lengths change under topological proposals and use this information to design novel hybrid proposals that update the topology and the branch lengths simultaneously.

In this thesis, we also describe a novel algorithm (called RogueNaRok) for identifying unstable taxa (so-called *rogue taxa*) in a set/sample of evolutionary trees that have been inferred by ExaBayes for instance. The state-of-the-art algorithm preceding RogueNaRok employs several heuristic assumptions that limit the quality of the identified set of rogue taxa. While still being a greedy algorithm, RogueNaRok is exact in its assessment of how the removal of species from a tree set will influence the summary statistics of interest. Algorithmic engineering substantially improves the runtimes of RogueNaRok. Thus, RogueNaRok allows to identify rogue taxa that are more harmful than those identified by previously employed algorithms, while being substantially faster. In consequence, RogueNaRok is particularly suited for extremely large datasets in terms of both, the number of trees, and species.

ExaBayes [1] has been published in *Molecular Biology and Evolution*, a prestigious peer-reviewed evolutionary biology journal (impact factor: 9.1 as of 2015). RogueNaRok, the stand-alone version, and webservice have been published in *Systematic Biology* (impact factor: 14.4 as of 2015). The implementation and empirical evaluation of the novel load balance algorithm is a minor contribution to a paper by Kobert *et al.* [60] that has been published at a peer-reviewed bioinformatics workshop. A third paper that deals with posterior distributions of the branch lengths on a tree, describes the novel branch length proposal as well as its application to existing topological proposals has also been published in *Systematic Biology.*

Several research projects that have been conducted during the course of the thesis do not form part of this dissertation. A major contribution is AnA-FiTS, a highly efficient forward-in-time simulator for population genetic datasets. It improves runtimes over existing methods via a graph-based algorithm and low-level technical optimizations [4]. A plethora of empirical data analyses on supercomputers resulted in a publication about the evolutionary relationship of the major bird orders that was published in *Science* [57]. Rogue taxon analyses on empirical dataset with RogueNaRok resulted in contributions to several peer-reviewed journal publications [27, 87, 111] including a letter published in *Science* [77]. The implementation of the Message Passing Interface (MPI) parallelism in the phylogenetic likelihood library (PLL) represents further contributions to a peer-reviewed journal publication [39] and to a workshop publication [24]. Runtime analyses, I/O and startup optimizations of ExaML (a tool for HPC maximum likelihood inference) contributed to a peer-reviewed HPC conference paper [110] and a journal publication [62].

1.3 Thesis Structure

In **Chap.** 2, we discuss several concepts that are essential for inferring the TOL. In particular, the statistical models are introduced that underlie TOL inference using the maximum likelihood method and the Bayesian framework. Subsequently, **Chap.** 3 provides a detailed description of how Bayesian inference of the TOL can be accomplished. Furthermore, essential methodological variations and improvements that have been developed for ExaBayes are discussed in this Chapter. In **Chap.** 4, we start with a detailed statistical analyses of branch lengths in the Bayesian framework. Based upon these insights, novel methods for proposing branch lengths are introduced and subsequently used to develop and evaluate a new class of hybrid proposals that update the topology and the branch lengths simultaneously. **Chap.** 5 highlights the implementation and parallelization of ExaBayes. Initially, the sequential performance of ExaBayes is compared to the state-of-the-art software MrBayes. Subsequently, the various levels of parallelization in ExaBayes are described and evaluated. This Chapter also discusses improvements to load balancing and memory reduction techniques that are implemented in ExaBayes. Finally, ExaBayes is run on the SuperMUC supercomputer to infer a tree from a simulated **whole-genome** dataset that comprises 200 species. **Chap.** 6 covers aspects pertaining to the post-analysis of trees that have been inferred (resp., sampled) using software such as ExaBayes. The quality of the consensus of a set of trees can be confounded by the presence of unstable taxa (so-called rogue taxa). This Chapter describes a highly efficient algorithm for the identification of these problematic taxa. In **Chap.** 7, we conclude and discuss open questions for future research based on the results of this dissertation.

2 Concepts in Evolutionary Bioinformatics

2.1 Evolution

Evolution is the continuous change of inheritable information over subsequent generations of a population. Thus, the term evolution does not apply to individuals but to a population as a whole. According to a simplistic definition (that fits sexually reproducing individuals), a *species* is the largest set of individuals that is capable of producing offspring. In biology, the primary evolutionary information source is the desoxyribonucleic acid (DNA), respectively the ribonucleic acid (RNA). The DNA and RNA are two similar polymers that are often denoted as the blueprint of life. The entirety of the inheritable DNA of a species is referred to as *genome*. Depending on the species, the genome may be partitioned into several *chromosomes*, where a chromosome is defined as a continuous sequence of DNA/RNA monomers (i.e., a small number of comparably simple molecules that can form a higher-complexity molecule that is called polymer). Genes are small stretches of the genome that are transcribed into messenger RNA by specific types of polymerases. The messenger RNA in turn is translated into proteins [124]. DNA triplets are translated into an amino acid (AA) each and thus proteins emerge as linear polymers of AAs. A strand of DNA forms a double helix with a reverse complementary copy of itself. For practical reasons, we usually only consider single-stranded DNA. Essentially, it is a sequence of the four nucleobases: adenine (`A`), cytosine (`C`), thymine (`T`) and guanine (`G`). In a double helix, `A` preferentially bonds with `T` and `C` preferentially bonds with `G`. Single elements of a DNA sequence often are referred to as base pairs (bps).

While the redundant structure of the double helix allows for error correction during DNA replication, the replication process is error-prone by design. The erroneous replication of DNA is referred to as *mutation*. Thus, a mutation manifests itself either as a replacement (i.e., substitution) of one base (resp., bp) or the deletion (resp., insertion) of a bp into the sequence. Mutations are the primary source of diversity (if inherited to the succeeding generation) and thus the central driving force behind evolution. Many mutations do not influence the ability of an individual's offspring to reproduce. In other words, they are *neutral* and do not impact the *fitness* of the individual (that is determined by its number of offspring). On the other hand, some mutations may cause fitness disadvantages such as Mendelian diseases (e.g., cystic fibrosis). Furthermore, the cumulative effect of mutations influences continuous traits such as the body height of an individual. The idea that the change induced by a mutation has an effect on the ability of an individual to reproduce is referred to as *selection* and represents another important evolutionary force.

A further concept that is specifically studied in population genetics, is *genetic drift*: if we assume random mating among individuals of a population, there is a chance that the number of carriers of a positively selected mutation decreases over generations, because individuals do not reproduce as expected. Thus, even the frequency of an allele (i.e., a genetic variant) with deleterious effect may increase over time. Once, all individuals carry the deleterious mutation, the only way to remove the mutation from the gene pool is the unlikely event of a backwards mutation or the reintroduction of the extinct allele via individuals of another population of the same species (i.e., via *migration* of individuals). Specifically, for asexually reproducing species, the accumulation of such deleterious and *fixated* mutations (i.e., a mutation carried by every individual in a population) can ultimately lead to a mutational meltdown [72] that drives the species into extinction.

The manifestation and genetic stabilization of two distinct sexes in sexually reproducing species as well as sexual reproduction itself comes with substantial risks and resource requirements. An initial attempt to explain the phenomenon of distinct sexes was the theory that *recombination* makes mutational meltdown less likely [81]. Recombination is a process that occurs during reproduction and creates mixed DNA sequences that are composed of DNA sequences of either parent. Sexually reproducing species have two or more sets of chromosomes (not counting sex chromosomes), that means every cell in an organism contains n copies of each chromosome. Thus, in case of a *diploid* species (i.e., two sets of chromosomes), each individual inherits one set from each parent. A single or multiple cross-over of DNA strands from each of the parents during reproduction (resp., the creation of cells used for reproduction) thus creates a mixed descendant strand.

Long-standing genetic isolation of populations of the same species eventually results in the two populations becoming less compatible and eventually, two new species emerge from a common ancestor. Specifically, the genetic code for translating triplets of RNA into AA that form proteins corroborates the hypothesis, that all life on earth originated from a single common ancestor [113]. However, exchange of genetic material among species is still possible after a speciation event and was apparently not uncommon in early times of evolution [28]. Specifically, bacterial species are capable of *horizontal gene transfer*, that is, genetic information can be exchanged among individuals of a population. Thus, it is still a subject of debate, to which extent the relationship among all extant species resembles a *tree* or *network* of life. The evolutionary forces discussed in this Section, primarily shape the genetic diversity of populations. However, these population genetic effects have an impact on the evolutionary relationship among closely related species. For instance, there is a high chance that two or more gene lineages co-exist in an ancestor species that later diverged into distinct species. If one of these lineages gave rise to a gene lineage that diverged into a separate species, then the evolutionary history for this gene is different from the evolutionary history of the species. This effect is known as incomplete lineage sorting (ILS) or *deep coalescence*. ILS is the reason, why part of the human genome is more closely related to the orangutan than to the chimpanzee, whereas the latter is assumed to be the closest evolutionary relative of man [51].

2.2 Sequence Alignment

Because of the aforementioned evolutionary forces the genomes of distant species (also referred to as *taxa*) exhibit substantial differences. Even individuals within homogeneous diploid populations are typically noticeably genetically different (with the exception of identical twins). The linear polymer structure of proteins and DNA provide the essential input data for many problem settings in bioinformatics: sequence data. In case of DNA, the alphabet of the sequence consists of the four previously mentioned nucleic bases: `A`, `C`, `G` and `T`, whereas in case of RNA, uracil (`U`) replaces `T`. The alphabet of AA sequences typically consists of 20 amino acids, not counting species-specific exceptions such as selenocysteine.

An important step in many *phyloinformatic* analyses is the alignment of two or more *orthologous* regions of the genome. Orthologs are sequences that are similar by descent from a common ancestor (see **Fig. 2.1**). In contrast, *homologous* regions can be similar because of orthology or because of genetic duplication events. Because of mutations and insertions, respectively deletions of subsequences, the sequence identity of orthologous regions can be low, depending on the time that has passed between the divergence of taxa and the mutation rate of the species. Thus, for both types of data, an additional symbol is included in the alphabet to either represent missing data (represented via `?` or `N`) or the absence of an orthologous bp (represented via `-`).

For a given scoring scheme (penalizing sequence substitution and insertion/deletion operations), the *Needleman-Wunsch* algorithm [82] allows us to determine an optimal alignment of two sequences with sequence lengths m and n in $\mathcal{O}(m \cdot n)$ using dynamic programming. Formally, an alignment is defined as a matrix $\mathcal{A} = a_{i,j}$, where the i-th row represents sequence data of taxon i and the j-column is referred to as its j-th character (see **Fig. 2.1**).

Using a simple modification of the original algorithm [103], we can identify sub-regions in the two sequences that yield an optimal alignment score. These *local* alignments marked an important step towards querying a sequence against a sequence database (resp., the concatenation of all sequences in a database). However, with technological improvement of sequencing methodology, the size of sequence databases soon reached a level for which exact database queries became prohibitive. For instance, one of the most popular sequence databases (GenBank) grew from 680,338 bps as of December 1982 to 165,722,980,375 bps as of August 2014 [see online statistics of 14]. Blast [8] is a popular heuristic algorithm for local alignments that trades sensitivity against runtime. In brief, the Blast algorithm avoids the computation of an entire dynamic programming matrix by focusing on small high-scoring sub-sequences (*words*). High-scoring regions are identified in the database and subsequently extended. The algorithm also accounts for the possibility of encountering high scoring regions by chance and assesses the statistical significance of matches. Blast reduced the complexity of database searches to $\mathcal{O}(nw)$, where n is the number of bps in the database and w is the sub-sequence length [8].

Determination of an optimal alignment of more than 2 sequences becomes computationally prohibitive for an increasing number of sequences. In fact, the multiple sequence alignment (MSA) problem is $\mathcal{NP}$-complete [123] and an exhaustive solution for n se-

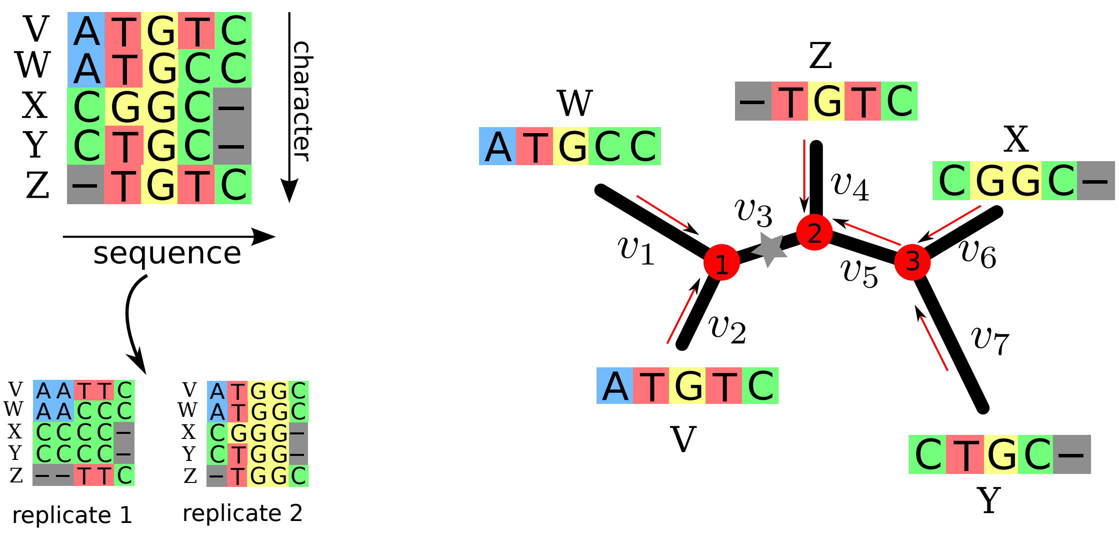


Figure 2.1: *Left:* an alignment data matrix comprising the sequence of 5 taxa and 5 characters. Two examples of possible alignment replicates (see **Sect.** 2.6.4) are depicted below the original alignment matrix. *Right:* An unrooted phylogenetic tree comprising 5 taxa and their sequences, along with branch lengths $v_1, \ldots, v_7$. Red arrows indicate an upward recursive traversal of the tree with inner nodes 1..3 towards a virtual root (star symbol on v_3).

quences with lengths $l_1, \ldots, l_n$ has runtime and space requirements in $\mathcal{O}\left(\prod_{i \in 1..n} l_i\right)$. Notice that, for the creation of a MSA, we need orthologous sequences as input (which are often obtained via Blast). Clustal [50] implements an approximate algorithm and was one of the first production-level tools for MSA. First, it creates pairwise alignments of all sequences in order to derive a guide tree from the similarity scores of the pairwise sequence alignments. This guide tree is subsequently used for creating the MSA by iteratively extending an initial pairwise alignment (progressive alignment). Since alignment quality is essential for downstream analyses, the complexity of alignment algorithms increased over time. Muscle [30] is a state-of-the-art tool that is based on the same underlying guide-tree principle as Clustal, but ultimately constructs a MSA in three phases: (i) use sub-sequence frequencies (i.e., k-mers) and a clustering procedure to create a draft guide tree for an initial progressive MSA, (ii) create an improved progressive MSA using a guide tree that is obtained via a more accurate alignment-based distance measure with subsequent clustering and (iii) several refinement iterations that revisit vertices of the guide tree and re-align the two MSAs defined by the subtrees on each side of a vertex.

Muscle is an example for the type of hybrid algorithms that is needed in evolutionary biology to obtain an acceptable result in a feasible amount of time. Further efforts to improve MSA quality focused on more accurately modeling of biological processes. For instance, Prank [71] models insertions and deletions as distinct evolutionary events and thereby tries to reduce the number of evolutionary events necessary to describe sequence evolution. Apart from that, special-purpose alignment tools like PaPaRa [17] solve domain-specific problems, like the efficient and accurate alignment of short subsequences to an existing reference alignment.

Alignment tools use a guide-tree to incorporate as much *phylogenetic* information (i.e., information regarding the evolution of the sequences) as possible into the MSA inference process, yet these phylogenetic methods typically are heuristics for reasons of runtime efficiency. Software packages such as BAli-Phy [92] try to avoid alignment bias via a simultaneous alignment and phylogenetic inference process at the expense of substantial runtime requirements.

2.3 Phylogenetic Trees

We define an unrooted *phylogenetic tree* comprising n taxa as an undirected graph $\tau = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V} = \{t_1, t_2, \dots, t_n, i_1, \dots, i_m\}$ is a set of vertices or nodes and $\mathcal{E} \subset \mathcal{V}^2$ is a set of edges or branches. We define the degree d of node k as

$$d(k) = \left|\{(k_i, k_j) \in \mathcal{E} \mid k_i = k \vee k_j = k\}\right| . \qquad (2.1)$$

A subgraph of a phylogenetic tree is called a *path* $p(k_1, k_n) = (\mathcal{V}', \mathcal{E}')$ from k_1 to k_n, if $\mathcal{V}' \subset \mathcal{V}, \mathcal{E}' \subset \mathcal{E}$ and $\mathcal{E}' = \{k_1, k_2, k_3, \dots, k_{i-1}, k_i\}$ and $\mathcal{V}' = \{(k_1, k_2), (k_2, k_3), \dots, (k_{i-1}, k_i)\}$. A graph is an unrooted phylogenetic tree, if there exists exactly one path between any two nodes k_i and k_j. For an illustration of a phylogenetic tree, see **Fig. 2.1**.

In each phylogenetic tree comprising n taxa, we have n nodes $t_1, \dots, t_n$ with degree 1 that are called *outer* nodes or tip nodes and represent taxa, respectively the observed sequences (e.g., $W, \dots, Z$ in **Fig. 2.1**). Inner nodes $i_1, \dots, i_m$ (e.g., $1, \dots, 3$ in **Fig. 2.1**) are hypothetical common ancestors of the taxa under examination. We call a phylogenetic tree binary or fully resolved, if there exists no node $k_i \in \mathcal{V}$ with $d(k_i) > 3$. Thus, a fully resolved (i.e., binary) phylogenetic tree contains $n-2$ inner nodes and $2 \cdot n - 2$ nodes in total.

Some phylogenetic models – specifically those that assume a molecular clock [132] – require rooted trees. In rooted trees, edges typically are directed. For the remainder of this thesis, we will focus on unrooted trees only. While unrooted undirected trees assume evolution to be symmetric, in reality time can be considered as being the "direction" of evolution: branches are then directed from ancestor node to descendant node. However, the symmetry assumption in an unrooted tree yields desirable properties for mathematical models of evolution (see **Sect.** 2.6). For rooting an unrooted tree after the tree inference, practitioners usually include an evolutionary distant group of taxa in the analyses (also referred to as *outgroup*). The position of the outgroup in the phylogenetic tree then serves as a root.

Phylogenetic trees are leaf-labeled, that is, inner nodes are indistinguishable from each other. There exist 3 distinct phylogenetic trees (resp., *topologies*) for a set of 4 taxa, which is the smallest biologically meaningful tree. Since a fully resolved tree contains $2 \cdot n - 3$ branches, we have $2 \cdot n - 3$ possibilities to insert an additional taxon into an existing tree with n taxa. Accounting for duplicates yields the following equation [35] for obtaining the number of possible phylogenetic trees depending on the number of taxa n:

$$\frac{(2n-5)!}{2^{n-3}(n-3)!}, \quad n \geq 3. \tag{2.2}$$

This means, for instance, that for 20 taxa, there exist almost 2.22×10^{20} trees that explain the evolutionary history of these taxa. This immense number of combinatorial possibilities is the main challenge for phylogenetic inference methods as well as phylogenetic post-processing alike. Assume, we want to compare τ_1 and τ_2 which have been inferred from an identical alignment, but using different inference methods. While efficient algorithms for identifying common subtrees [40] exist, there is a high chance that subtrees with small differences are similarly optimal for given criteria. Thus, common subtrees in τ_1 and τ_2 possibly do not comprise large sets of taxa.

Instead, many methods in phylogenetic post-processing revolve around the branches $\mathcal{E}$ of a tree. Two nodes t_1 and t_2 are called connected, if there exists a path $p(t_1, t_2)$. The removal of a branch $b \in \mathcal{E}$, divides a tree into two separately connected subgraphs $\gamma' = (\mathcal{E}', \mathcal{V}')$ and $\gamma'' = (\mathcal{E}'', \mathcal{V}'')$, such that $\tau = (\mathcal{V}' \cup \mathcal{V}'', \mathcal{E}' \cup \mathcal{E}'' \cup \{b\})$. Inner nodes are unlabeled and can be neglected, thus we say that the removal of b partitions the set of outer nodes $t_1, \ldots, t_n$ into two complementary sets $T = \{t_1, \ldots, t_i\}$ and $\overline{T} = \{t_{i+1}, \ldots, t_n\}$. We call this partition of T a *split* or a *bipartition* and denote it as $(T|\overline{T}) = (t_1, \ldots, t_i|t_{i+1}, \ldots, t_n)$. Bipartitions with $|T| = 1$ or $|\overline{T}| = 1$ are contained in every tree with an identical set of outer nodes and thus are called *trivial*.

Each branch $b \in \mathcal{E}$ defines a bipartition and each phylogenetic tree is equivalent to a set of bipartitions. The opposite is not necessarily true, since a set of bipartitions may contain conflicting bipartitions that can not occur in the same tree. A bipartition $(T \,|\, \overline{T})$ can be considered as the smallest unit of evolutionary information and identifies the hypothesis that the taxa in T are more closely related to each other than the taxa in $\overline{T}$. For example, consider removing of branch $(1, 2)$ in the tree of **Fig. 2.1**. We obtain the bipartition $(VW \,|\, XYZ)$, which in turn means that V and W are related more closely to each other than to any of the taxa X, Y or Z (ignoring the fact that in a biological context trees must be rooted).

We can now define a distance measure between phylogenetic trees τ_1 and τ_2 that is based on the number of shared bipartitions. Given the sets of non-trivial bipartitions B_{τ_1} and B_{τ_2} of τ_1 and τ_2, we define the Robinson-Foulds (RF) distance [95] as

$$RF(\tau_1, \tau_2) = \frac{1}{2} \cdot \left|\{b \mid (b \in B_{\tau_1} \wedge b \notin B_{\tau_2}) \vee (b \notin B_{\tau_1} \wedge b \in B_{\tau_2})\}\right|. \tag{2.3}$$

The original authors of the RF-distance proofed that their distance measure is a metric on the space of trees. The quartet distance is an alternative distance measure for unrooted phylogenetic trees [21]. Here, we extract induced subtrees of each combination of 4 taxa from the phylogenetic tree and consider these quartets as smallest unit of phylogenetic information. An appealing property of this approach is, that for each quartet of taxa, there exist only 3 different topologies (see **Eq. 2.2**). The quartet distance between τ_1 and τ_2 is defined as the number of quartets that are unique to one of the two trees.

So far, our definition of a phylogenetic tree τ only specifies a topology, that is, it defines divergence events (i.e., the inner nodes) among taxa. If we extend the definition of τ to be a weighted graph, we can define a *branch length* as the weight v_i of branch b_i (see **Fig. 2.1**). A branch length v_i then specifies the evolutionary distance of a taxon or a subtree with respect to the remaining subtree connected by b_i.

2.4 Distance-Based Phylogenetic Inference

Phylogenetic inference methods use sequence data to infer a phylogenetic tree (see **Sect.** 2.3) that explains the evolutionary history of all taxa contained in that tree. One of the most straight-forward ways of obtaining a tree from sequence data are *distance*-based methods. The unweighted pair group method with arithmetic mean (UPGMA) is an agglomerative clustering algorithm [104]. Its application is not limited to the phylogenetic inference problem. For inferring a phylogeny via UPGMA, we first compute distances among sequences under examination and then cluster the sequences iteratively by their distance (where the distance of a newly added cluster towards the remaining clusters is recomputed each time). UPGMA yields a rooted ultrametric tree (i.e., the sums of branch lengths on the paths from the root to each taxon are identical). UPGMA is fast, but comes with the strong assumption of a molecular clock. Thus, UPGMA trees are mostly used as starting trees for more involved inference methods or in alignment algorithms that incorporate phylogenetic information (such as MUSCLE).

Another distance-based clustering algorithm is neighbor joining (NJ) [97]. As UPGMA, NJ is agglomerative, the main difference between the two approaches is the distance function used to calculate distances between clusters. Furthermore, NJ does not assume a molecular clock, thus we do not obtain an ultrametric tree.

2.5 Maximum Parsimony

The aforementioned distance-based methods typically offer a deterministic algorithm to infer the tree. In contrast to this, we can search for a tree that is optimal with respect to a criterion. A popular, biologically motivated criterion is maximum parsimony (MP) [37]. It relies on *Occam's razor*: among many hypothesis, the hypothesis with the lowest number of assumptions is preferable. In the phylogenetic setting, this means that the tree that can be explained by the least number of substitutions is considered optimal. Note that, there may be more than one optimal tree with respect to the parsimony criterion.

While there exist more involved scoring schemes (i.e., scoring different types of substitutions differently [100]), we focus on a simple parsimony cost-function, that puts the same weight on every substitution. In **Fig. 2.1**, we can calculate the parsimony score of the first character as follows: we can assume that the ancestral state of node 1 was `A`, since both of its descendants V and W have an `A` at this character position. Similarly, for node 3, we obtain a `C` as most parsimonious assumption. For node 2, we assume that a deletion occurred along the branch v_4 (from ancestor node 2 to species Z), so the internal state at node 2 can be assigned `C`. At a virtual root (depicted as a star symbol in **Fig. 2.1**), we then need an additional substitution from `A` at node 1 to `C` at node 2. Thus, we obtain an overall parsimony score of 2 for the first character. We obtain an identical parsimony score, if we assume instead, that a substitution `C` $\rightarrow$ `A` occurred on branch v_5. There exist explanations that require more substitutions. However, we refer to the parsimony score of a character as the one with the least number of substitutions.

We can use a dynamic programming algorithm [100], to compute the per-character parsimony score efficiently: for a given topology τ comprising $2 \cdot n - 2$ nodes and a sequence alignment a_{ij}, let $S_{i,j}(X)$ denote the parsimony score of observing state X at node i for character j. We assume that the alignment row indices i correspond to indices of outer nodes in τ. Assume that k and l are direct descendants of node i, then we can compute the score $S_{i,j}(X)$ as follows:

$$S_{i,j}(X) = \begin{cases} 0, & \text{if } d(i) = 1 \wedge a_{i,j} = X; \\ 1, & \text{if } d(i) = 1 \wedge a_{i,j} \neq X; \\ \min_Y \{S_{k,j}(Y) + S_{l,j}(Y) + \delta_{X,Y}\}, & \text{else;} \end{cases}$$

where $\delta_{i,j} = 0$, if $i = j$ and 1 otherwise.

We have to start the recursion at a virtual root node that can be placed onto an arbitrary branch of tree τ (since branches in τ are undirected and evolution is not assumed to be directional). The parsimony score of character j is $P_i = min_X \left\{S_{i,j}(X)\right\}$, where i is the virtual root node.

For a character to be informative under the parsimony criterion, we need at least two different states to occur at least twice. Thus, character 2 and 3 are called *parsimony-uninformative* and can be ignored when computing the parsimony score. The mutation from `G` to `T` occurs at a trivial bipartition, thus we obtain the same parsimony score for each topology (the same holds for character 3). Thus, we can compute the overall parsimony score of the alignment and tree depicted in **Fig. 2.1** as $P = P_1 + P_4 + P_5 = 2 + 2 + 1 = 5$. Character 4 suggests a close evolutionary relationship between V and Z. Indeed, a tree τ' that contains a bipartition $(VZ \mid WXY)$ instead of $(VW \mid XYZ)$ has a more optimal score of 4 and is the unique most parsimonious tree for this example. In other words, τ' explains the observed parsimony informative characters with only 4 substitutions.

For small trees (e.g., up to 10 taxa), we can exhaustively compute the score of every tree topology and thus determine all parsimony values. For larger trees, a typical strategy — that is implemented, for instance, in RAxML [108] — is to start with a randomized stepwise addition sequence parsimony tree. This means, we start out with a trivial tree of 3 taxa and then add one taxon after another in a random sequence. In each addition step, we compute the parsimony score of all possible insertions (i.e., at every branch for the current taxon) and choose the location with the lowest parsimony score for addition. The resulting tree strongly depends on the order of the taxa that are added. Once the tree contains the full set of taxa, we can improve the MP score of the tree by perturbing the topology (for examples using topological perturbations, see **Sect.** 3.3.1) and re-evaluating the parsimony score. General-purpose optimization algorithms, such as greedy or evolutionary algorithms can be used to search for an optimal tree.

A parsimony-specific search algorithm is the *parsimony ratchet* [84]. Here, perturbations of the topology are combined with modifications of the alignment matrix. We usually reduce an alignment matrix into a matrix of unique characters (referred to as *alignment patterns*) and assign a weight that equals the frequency of a character in a matrix. In the parsimony ratchet search strategy, we repeatedly switch back and forth between the original alignment and a reweighted alignment and execute the search algorithm (using topological perturbations). The underlying idea is, that this facilitates escaping islands of local optimality and we thus obtain more parsimonious trees.

2.6 Inference Under Probabilistic Models

The central disadvantage of distance- and parsimony-based methods is that multiple substitutions (or an unobservable forward-backward substitution) at the same site can not be modeled adequately. On simple 4 taxon trees, the parsimony criterion incorrectly favors topologies that assume an evolutionary close relationship among highly diverged taxa. This phenomenon has been termed long branch attraction (LBA) [32]. These shortcomings gave rise to inference methods that derive their optimality criterion from probabilistic models of sequence evolution. In the following, we describe the *likelihood* criterion for a model of DNA evolution [for details, see 127, chapter 1+3, 35, chapter 16].

2.6.1 Probabilistic Evolution

Probabilistic evolutionary models consider substitutions as probabilistic events and account for any combination of events that could have occurred along a branch in a tree (as depicted in **Fig. 2.1**). We can model the process of substitution as a continuous-time Markov chain (ctMC) with a state space $\mathcal{S} = \{A, C, G, T\}$. The Markov chain assumes a specific state and transitions to another state at rates that are given by a transition rate matrix Q that has dimensions $|\mathcal{S}| \times |\mathcal{S}|$ (see **Fig. 2.2**). Q is also called the chain's generator matrix. By definition, the rate at which the chain remains in its current state is $q_{ii} = -\sum_{j \neq i} q_{ij}$. As a result each row of Q sums up to 0. An important feature of a Markov chain is that transition probabilities only depend on the current state of the chain (a realistic assumption for the biological process that is modeled).

If a Markov chain is *irreducible* (i.e., all states are reachable) and *recurrent* (i.e., it can be in any state for an unbound number of times), then the probability that the chain is in a particular state converges against an equilibrium distribution (as time $t \to \infty$). This equilibrium state is called the *stationary distribution* of the Markov chain. For our biological model this means that if substitutions occur at random on a sequence at given rates, the frequency of the four bases will eventually reach a stable distribution $\{\pi_A, \pi_C, \pi_G, \pi_T\}$ for $t \to \infty$ regardless of the initial state. The process is illustrated on the right-hand side of **Fig. 2.2**: if we start in A, then after time t_1, the probabilities that the chain is in state $s \in \mathcal{S}$ are $\{\pi'_A, \pi'_C, \pi'_G, \pi'_T\}$. After an additionally (randomly chosen) time step t_2, the probability distribution that a chain is in state s is $\{\pi''_A, \pi''_C, \pi''_G, \pi''_T\}$, before it reaches the stationary distribution after an infinite amount of time.

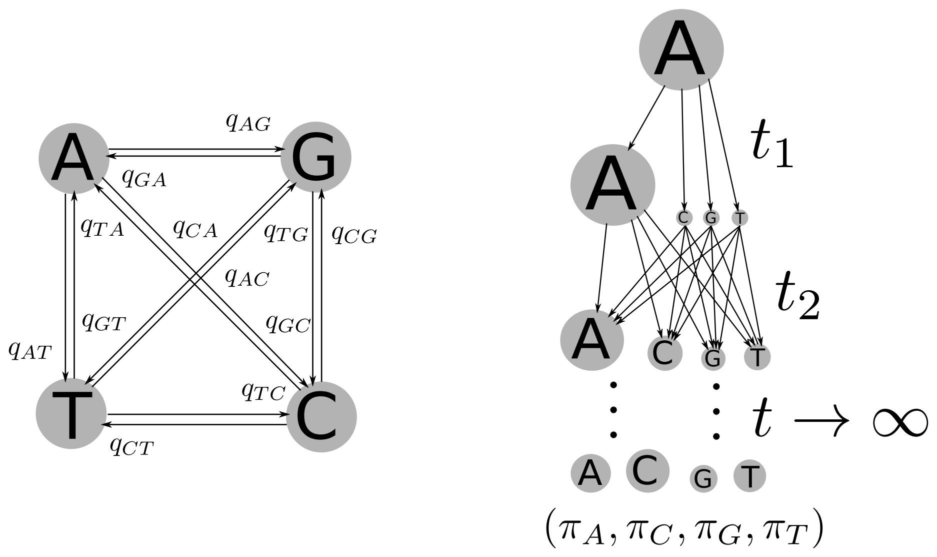


Figure 2.2: *Left:* Illustration for the six instantaneous rates q_{ij} of a ctMC for state changes between the four states in DNA sequences. *Right:* Illustration of state frequencies starting from an observed state A after time t_1, after time $t_1 + t_2$ and after reaching stationary frequencies $(\pi_A, \pi_C, \pi_G, \pi_T)$ after an infinite amount of time. Arrows indicate transition probabilities. For instance, the arrow from A to C at the Section of t_1 represents the probability $p_{AC}(t_1)$ for a transition from A to C in time t_1.

A large number of DNA substitution models have been developed. The Jukes-Cantor [58] model comes with the strong assumption, that there exists only one transition rate between all states. The Kimura model [59] introduces two different rates (for state transitions between two biological classes of nucleobases). Both models have Q matrices that yield time-reversible Markov chains, that is, $\pi_i q_{ij} = \pi_j q_{ji}, \forall i, j \in \mathcal{S}, i \neq j$. While there is no biological foundation to assume time-reversibility, it is a desirable mathematical property and assumed for simplicity and computational efficiency. We can formulate a 6 parameter model comprising 6 rates $r_{i \leftrightarrow j}$ with $q_{ij} = q_{ji} = r_{i \leftrightarrow j}$. Thereby, we obtain different rates for each type of (reversible) transition. However, for such a Markov chain, the stationary distribution π_i would be strictly determined by Q. From a biological perspective it is important to allow for an arbitrary state composition in the equilibrium state (e.g., we want to assume that a gene has a preference for certain bases). Thus, the stationary frequencies π_i should be free parameters as well. This requirement lead to the development of the generalized time-reversible (GTR) model [115]. The Q matrix of the GTR model parameterizes a reversible rate matrix with stationary frequencies

$$Q = \begin{pmatrix} q_{AA} & r_{AC}\pi_C & r_{AG}\pi_G & r_{AT}\pi_T \\ r_{AC}\pi_A & q_{CC} & r_{CG}\pi_G & r_{CT}\pi_T \\ r_{AG}\pi_A & r_{CG}\pi_C & q_{GG} & r_{GT}\pi_T \\ r_{AT}\pi_A & r_{GT}\pi_C & r_{CG}\pi_G & q_{TT} \end{pmatrix}, \tag{2.4}$$

with $q_{ii} = -\sum_j q_{ij}$ and thus yields a time-reversible Markov chain. Since (i) $\sum_{i \in \mathcal{S}} \pi_i =$

1 and (ii) because we fix $r_{GT} = 1.0$ and represent remaining r_{ij} relative to r_{GT}, the GTR model has 8 free parameters. Constraint (ii) in fact is optional. However, if fulfilled and if we scale Q such that the average transition rate $\rho = 1$, then branch lengths are normalized such that a value of 1 indicates that we expect that 1 substitution has occurred on average per character. The average transition (resp., substitution) rate is defined as

$$\rho = -\sum_{i \in \mathcal{S}} \pi_i q_{ii} = \sum_{(i,j) \in \mathcal{S}^2 \wedge i \neq j} \pi_i \pi_j r_{ij}. \tag{2.5}$$

The GTR model is the most general time-reversible substitution model and is a generalization of various models mentioned previously (e.g., Jukes-Cantor). It is worth mentioning that there exists a total of 203 distinct time reversible models and that choosing a model with as few free parameters as required is desirable to avoid overparametrization [55]. It is common practice to test several typically employed models of DNA substitution on a MP tree [e.g., 90] under the likelihood criterion (described in the following). Then, for the comprehensive analysis, we use the model that optimizes a score based on the number of parameters in the model and the likelihood of the model. Nevertheless, for DNA data, alignments usually provide enough information for a meaningful application of the GTR model.

In a ctMC, we can derive the probabilities $p_{ij}(t)$ for a transition from state i to j after time t as $P(t) = \exp(Q \cdot t)$. For simple models, such as the Kimura model, there exist closed form derivations of $P(t)$. In case of the GTR model, an Eigendecomposition of Q is necessary for matrix exponentiation. That is, we need to determine an invertible matrix U and its inverse U^{-1}, such that $Q = U \Lambda U^{-1}$. Λ is a diagonal matrix containing the Eigenvalues λ_i of Q. This allows us to exponentiate Eigenvalues and we obtain

$$P(t) = U \cdot \begin{pmatrix} \exp(\lambda_1 \cdot t) & 0 & 0 & 0 \\ 0 & \exp(\lambda_2 \cdot t) & 0 & 0 \\ 0 & 0 & \exp(\lambda_3 \cdot t) & 0 \\ 0 & 0 & 0 & \exp(\lambda_4 \cdot t) \end{pmatrix} \cdot U^{-1}. \tag{2.6}$$

Notice that in **Fig. 2.2**, arrows depict various transition probabilities $p_{ij}(t_1)$ and $p_{ij}(t_2)$. For each row transition probabilities must sum up to 1.

2.6.2 Likelihood Computation

Given an alignment $\mathcal{A} = a_{ij}$ and a transition rate matrix Q (parameterized with the vectors $\vec{\pi}$ and $\vec{r}$), we can use the transition probabilities $P(v_a)$ and $P(v_b)$ to compute the likelihood that two sequences a and b evolved from any common ancestor into their observed state after time v_a and v_b. As mentioned previously, we can scale the transition rate matrix arbitrarily and thus employ a scaling factor, such that the expected number of substitutions per character is 1. If we assume independence among characters, that is, each character evolves independently from its adjacent characters, then we can compute the likelihood as the product of the per-site likelihoods. In reality, this assumption is frequently violated. For numerical reasons, likelihood values usually are represented on the logarithmic scale, thus we compute the sum of log-likelihood values over sites.

Consider taxa V and W in **Fig. 2.1** that have diverged from their common ancestor (node 1) in times v_1 and v_2. For each character $ij \in \mathcal{S}^2$ in their pairwise alignment $\mathcal{A}'$, we can compute the probability of observing these characters by assuming that the ancestral sequence has state $s \in \mathcal{S}$ with probability π_s and transitions into state i after time v_a and into state j after time v_b. Since we do not know the ancestral state, we sum over all four possibilities (see **Eq. 2.7**). The time-reversibility of the GTR model allows us to simplify the term to **Eq. 2.8**. In other words, the observation probability can be reduced to the stationary frequency of state i times the transition probability to state j (or vice versa):

$$\begin{aligned} f_{ij}(v_a, v_b) &= \sum_k \pi_k p_{ki}(v_a) p_{kj}(v_b); && (2.7)\\ f_{ij}(v_a + v_b) &= \pi_i p_{ij}(v_a + v_b) = \pi_j p_{ji}(v_a + v_b); && (2.8)\\ g(\mathcal{A} \mid v_a + v_b, \vec{\pi}, \vec{r}) &= \sum_{i,j} n_{ij} \log(\pi_i p_{ij}(v_a + v_b)). && (2.9) \end{aligned}$$

Finally, we obtain the logarithmic likelihood g of observing alignment $\mathcal{A}'$ given branch length $v_a + v_b$, the stationary frequencies $\vec{\pi}$ and reversible substitution rates $\vec{r}$ as the weighted sum of the observation probabilities of alignment patterns (see **Eq. 2.9**). The weight n_{ij} specifies, how often a pattern ij occurs in $\mathcal{A}'$.

We can now extend the likelihood criterion to non-trivial tree topologies (such as **Fig. 2.1**). Essentially, we assume a virtual root (star symbol in **Fig. 2.1**) where the sequence represented by a ctMC (in stationarity) enters the tree at an arbitrary point of time on v_3. As mentioned previously, we do not have to decompose v_3 into $v_3' + v_3''$ because of the time reversibility of the ctMC (as in **Eq. 2.8**). Instead, we only need to sum over the stationary frequencies π_{x_1} and the transition probability $p_{x_1x_2}(v_3)$ (as in **Eq. 2.9**). We then add the transition probabilities from the two inner nodes 1 and 2 to the outer nodes V, W, X, Y, Z and sum over all possible states of the inner (i.e., ancestral) nodes 1, 2 and 3. The summation over the sequence length from $1, \ldots, j$ (e.g., W_j is the j-th character of sequence W) yields

$$\begin{aligned} g(\mathcal{A} \mid \tau, \vec{v}, \vec{\pi}, \vec{r}) = \sum_{j=1}^{n} \Bigg(& \sum_{(x_1,x_2,x_3)\in\mathcal{S}^3} \pi_{x_1} \cdot p_{x_1x_2}(v_3) \\ \times\; & p_{x_1V_j}(v_2) \cdot p_{x_1W_j}(v_1) \\ \times\; & p_{x_2Z_j}(v_4) \cdot p_{x_2x_3}(v_5) \cdot p_{x_3X_j}(v_6) \cdot p_{x_3Y_j}(v_7) \Bigg). \end{aligned} \qquad (2.10)$$

In **Eq. 2.10**, $\vec{v} = \{v_1, \ldots, v_7\}$ denotes the vector of all branch lengths in τ. The summation over all states of the ancestors allows us to essentially consider any number and combination of substitutions that may have occurred along the evolution of the sequences. Evidently, a naïve computation of **Eq. 2.10** becomes computationally intractable for larger trees. For the tree depicted in **Fig. 2.1**, we can reorder/factorize summands in **Eq. 2.10** using a principle called *nesting rule*. In the phylogenetic likelihood context, this rule has been denoted as *pruning* algorithm [34] (not to be confused with two different applications of the term *prune* that follow later). Notice that, this pruning algorithm in essence is a dynamic programming algorithm. Thus, we can rewrite **Eq. 2.10** as

$$\begin{aligned} L_3(x_3) &= p_{x_3X_j}(v_6) \cdot p_{x_3Y_j}(v_7); && (2.11)\\ L_2(x_2) &= p_{x_2Z_j}(v_4) \cdot \sum_{x_3 \in \mathcal{S}} \left(p_{x_2x_3}(v_5) \cdot L_3(x_3)\right); && (2.12)\\ L_1(x_1) &= p_{x_1W_j}(v_1) \cdot p_{x_1W_j}(v_1); && (2.13)\\ g(\mathcal{A} \mid \tau, \vec{v}, \vec{\pi}, \vec{r}) &= \sum_{x_1 \in \mathcal{S}} \sum_{x_2 \in \mathcal{S}} \pi_{x_1} \cdot p_{x_1x_2} \cdot L_1(x_1) \cdot L_2(x_2). && (2.14) \end{aligned}$$

The three terms L_1, L_2 and L_3 represent the conditional probabilities of observing a state in one of the ancestral nodes. Thus, we also refer to L_i as a conditional probability vector (CPV) at node i. Because of the aforementioned properties of the GTR model, we can place the virtual root at an arbitrary branch in the tree and compute CPVs with respect to this root, whereas the virtual root can be placed anywhere along a branch (typically with distance 0 to one of the adjacent nodes). This concept has been termed as the *pulley principle* [34]. Assume that we subsequently want to evaluate the likelihood of $\Theta = \{\tau', \vec{v}', \vec{\pi}, \vec{r}\}$, that is, a parameter vector where the topology has been perturbed into τ' and/or some branch lengths $\vec{v}'$ have been modified. Then we can reuse all CPVs, that represent subtrees in which neither topology nor branch lengths have been modified. Thus, as a consequence of the pruning algorithm, we can substantially reduce the runtime costs of likelihood evaluation by re-using already computed CPVs. As a downside, CPVs typically dominate the memory requirements of likelihood computation (discussed in detail in **Sect.** 5.4).

With respect to runtime requirements of likelihood computation, we can distinguish three distinct classes of CPV computations, depending on the type of descendants a and b of an inner node: (i) a and b are outer nodes (see **Eq. 2.11** and **Eq. 2.13**), (ii) one descendant is internal, one is external (see **Eq. 2.12**) and (iii) both nodes are internal nodes (identical to the evaluation at the virtual root in **Eq. 2.14** but without $\vec{\pi}$). The computational effort for computing the CPV at a node directly follows from **Eq. 2.11**, **Eq. 2.12** and **Eq. 2.13**. For the mathematical representation of a base x at, for instance, W_j, we initialize a state frequency vector with $\delta_{x,i}$ for the i-th position (where δ is the Kronecker δ). Deletions and ambiguities (e.g., caused by sequencing errors) can be incorporated as well: here, multiple entries of the state frequency vector of an outer node can be initialized with 1 (resp., all entries are 1 in case of a missing character).

A Q matrix can be defined for different data types, such as AA or even binary data (i.e., presence or absence of a trait in a species). For AA data however, the Q matrix is typically not estimated empirically, because of the substantial amount of free parameters

(189 for $\vec{r}$ and 19 for $\vec{\pi}$). Instead, fixed-rate matrices are used (e.g., [68]) and stationary frequencies are either also provided by the model or estimated as free parameters.

Finally, this framework allows incorporating data from distinct data types and also allows assuming different model parameters for partitions of the alignment. Thus, $\mathcal{A}$ can be composed of $\{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n\}$ (e.g., various genes), where $\mathcal{A}_i$ is called a partition. Each partition can have distinct model parameters, for instance $\vec{\pi}_1, \vec{\pi}_2, \ldots, \vec{\pi}_n$. The combined log-likelihood is the sum of the log-likelihoods over all partitions. We call a parameter *linked* with respect to a set of partitions p, if all partitions in p share this parameter. Typically, τ and $\vec{v}$ are linked across all partitions (although StarBeast for instance models a species tree and several gene trees separately [48]).

2.6.3 Variable Rates Among Characters

The model introduced in this Section so far represents a notable advancement over parsimony (see **Sect.** 2.5) or distance-based methods (see **Sect.** 2.4) in terms of accurately modeling evolution. An important extension to this model is the consideration that some characters may evolve faster than others. Some characters may be highly conserved (e.g., character 3 in **Fig. 2.1**), since a mutation would disrupt — for instance — the function of a vital protein, while mutations at other characters are neutral since they do not affect a functional part of the genome (e.g., character 1 evolves at a fast rate).

One of the most popular ways to account for this heterogeneity of the evolutionary rate among characters is modeled via a discretized $\Gamma(\alpha, \beta)$ distribution [128]. The Γ distribution has the probability density function (PDF)

$$f(x; \alpha, \beta) = \frac{\beta^{\alpha} x^{\alpha-1} e^{-x\beta}}{\gamma(\alpha)}, \tag{2.15}$$

where $\gamma(\alpha)$ is the gamma function.

Since we are only interested in the rates relative to each other, we set $\beta := \alpha$ and thus constrain the Γ distribution to a mean of 1 (since the expectation value for $X \sim \Gamma(\alpha, \beta)$ is $\mathbb{E}[X] = \frac{\alpha}{\beta}$). For large values of α, we obtain distributions similar to a normal distribution with low variance (since $Var[X] = \frac{\alpha}{\beta^2}$), in other words, we assume that all characters evolve at a similar rate. The variance of the Γ distribution increases for small values of α (see **Fig. 2.3**). Here, (specifically, if $\alpha < 1$) the model fits well to datasets where the majority of the characters evolve slowly and the minority evolves quickly (i.e., we have a high degree of rate heterogeneity).

We typically discretize the Γ distribution into four classes each with a different range (c_a, c_b). The probability of each class must be the same, that is for any two classes c and c', we have $\int_{c_a}^{c_b} f(x; \alpha, \beta) \partial x = \int_{c'_a}^{c'_b} f(x; \alpha, \beta) \partial x$. Finally, one rate ρ_i is determined for each class (either median or mean of the class) that represents the rate class.

We incorporate rates ρ_i during computation of the transition probabilities from the transition rate matrix

$$P_{\rho_i}(t) = \exp(Q \cdot t \cdot \rho_i). \tag{2.16}$$

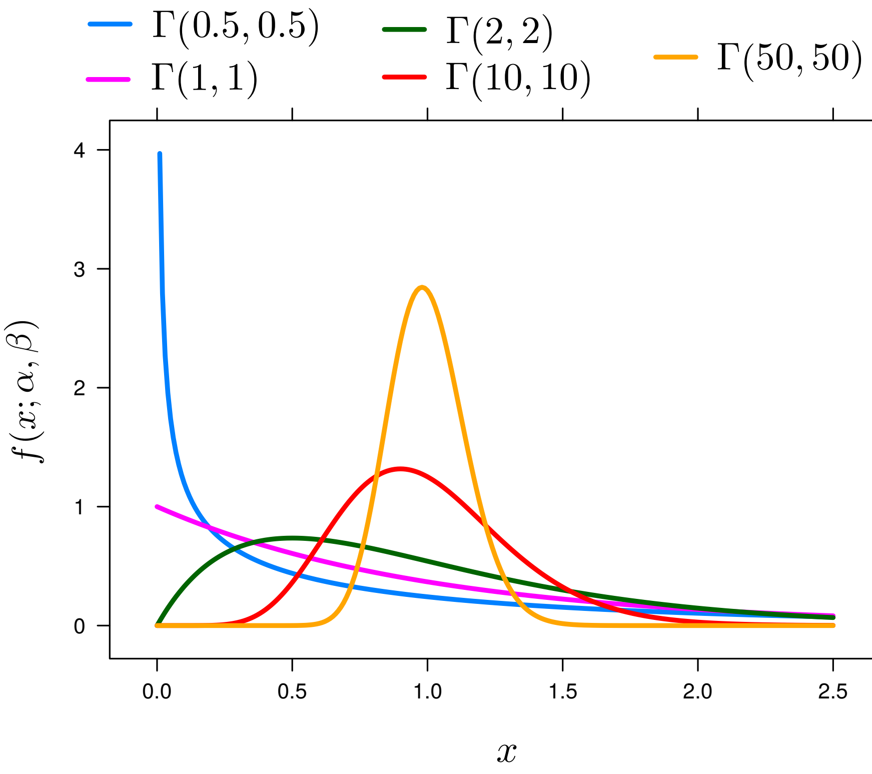


Figure 2.3: Illustration of typical single-parameter Γ distributions.

In other words, for the discretized Γ model, we assume 4 different models of sequence evolution (with slower or faster rates) and ultimately average the likelihood over the 4 rate classes. For an appropriate α, this approximate integration of a Γ distribution allows barely evolved characters to achieve high likelihood values for smaller ρ_i, while highly diverged characters have a high likelihood for larger ρ_i. The number of free parameters of the GTR+Γ model increases by 1 compared to the default GTR model (namely, the α parameter). However, computational requirements as well as memory requirements for storing CPVs increase by a factor of four.

2.6.4 Maximum Likelihood Analysis

The phylogenetic likelihood function (PLF) $g(\mathcal{A} \mid \Theta)$ provides us with a means to compare different parameter vectors $\Theta = \{\tau, \vec{v}, \vec{r}, \vec{\pi}, \alpha\}$. In the maximum likelihood (ML) framework, we try to find an estimate of the tree and the remaining parameters, that maximizes the PLF. While the problem of determining *the* ML tree is $\mathcal{NP}$-hard [22], various tools exist that use search heuristics to deliver point-estimates of the PLF landscape. Phylip [31] and Paup* [114] pioneered phylogenetic ML inference. PhyML [46] uses inexpensive to compute parsimony scores to eliminate topologies without the need to evaluate their likelihood. Ipqnni [122] generates a set of candidate trees and optimizes trees based on quartet information in the candidate tree set. Garli [133] and specifically RAxML [108] are high-performance computing (HPC) tools that implement an evolutionary optimization algorithm (Garli) and a multi-phase greedy hill-climbing algorithm (RAxML).

Typically, ML analyses start from a randomly created tree or a MP tree and subsequently optimize parameter values. With respect to the topology parameter τ, the topology is refined iteratively (using operations similar to those described in **Sect.** 3.3.1). A generic algorithm for optimization of continuous single parameters is *Brent's method* [18]. Brent's method typically is used iteratively for the optimization of one parameter at a time. An exception is the branch length parameter vector $\vec{v}$. Since the partial first and second derivatives of the PLF with respect to $v_i \in \vec{v}$ can be computed easily, more efficient optimization is possible [36] (for details, see **Sect.** 4.2).

The ML framework allows us to obtain an locally optimal estimate on the multidimensional likelihood surface. Typically, the search for an optimal tree is complemented by a non-parametric *bootstrap analysis* [33]. The underlying idea is that by permuting the data, we obtain confidence-like support values for assessing the robustness of the inferred phylogenetic relationships. This however is only one of several proposed interpretations of the method's result and the discussion did not result in a broadly accepted consensus [105]. For an alignment with n characters, we can create an alignment replicate (see **Fig. 2.1**) by drawing n characters from our original alignment with replacement. We typically create between 100 and 1000 alignment replicates and infer a ML tree for each of the replicates. Thereby, we obtain *bootstrap trees.* If there is conflicting signal in the alignment, then a proportion of bootstrap trees will differ from the best-known ML tree inferred on the original data. For downstream analysis of bootstrap trees, see **Sect.** 6.1. Notice that, the use of the phylogenetic bootstrap is not limited to ML, but we can also employ it in different phylogenetic frameworks, such as MP or NJ.

3 Elements of Bayesian Inference in Phylogenetics

The content of this Chapter is predominantly of introductory nature and its content is **not** an original contribution, unless stated otherwise. Parts of this Chapter have been derived from two peer-reviewed publications:

1. AJ **Aberer**, K Kobert, and A Stamatakis. "ExaBayes: Massively Parallel Bayesian Tree Inference for the Whole-Genome Era". In: *Molecular biology and evolution* 31.10 (2014), pp. 2553–2556
2. AJ **Aberer**, A Stamatakis, and F Ronquist. "An Efficient Independence Sampler for Updating Branches in Bayesian Markov chain Monte Carlo Sampling of Phylogenetic Trees." In: *Systematic biology* 65.1 (2016), pp. 161–176

Contributions: All modification of Bayesian proposals with respect to the implementation in MrBayes (e.g., the differing implementation of stochastic proposals on topologies). The partial Dirichlet proposal for AA-GTR matrices is an original contribution (see **Sect.** 3.2.5). Furthermore, the derivation of the Hastings-ratio for the node slider (see **Sect.** 3.2.3) is an original contribution.

This Chapter introduces a framework for Bayesian inference (BI) of phylogenetic trees under the models described in **Sect.** 2.6. We first cover the basics of BI. Subsequently, we discuss the essential proposals that allow us to sample proportional to the multi-dimensional posterior. We differentiate between continuous and discrete parameters. The entire framework is implemented in ExaBayes [1]. Many essential proposals are not novel and have already been implemented in MrBayes for instance. We explicitly describe how the set of proposals implemented in ExaBayes is different from traditional proposals and provide a rationale. Finally, we describe the variant of Metropolis-coupling that is implemented in ExaBayes.

3.1 Methodology

3.1.1 Bayesian Inference

BI is very similar to ML inference in that we can use the same ctMC model of sequence evolution as the underlying model for phylogenetic inference (see **Sect.** 2.6). However, it differs in one essential aspect: the interpretation of probability. The classic (i.e., frequentist) interpretation of the probability $\Pr[X = x] = p$ of a dichotomic event is that if we repeat an experiment n times, the outcome x will be observed $p \cdot n$ times, while the complementary outcome $\bar{x}$ will be observed $(1-p) \cdot n$ times (given that n is large enough). In other words, probability is defined by the relative frequency of events. Contrary to that, in Bayesian statistics the same probability is a measure for the *degree of belief.* For the above example, this means that, the degree to which we believe in the occurrence of an event x is p. Conversely, our uncertainty whether the event occurs in one instance of the experiment is $1-p$.

In other words, frequentists rely on objectively measurable frequencies, while Bayesian statisticians subjectify probabilities. The subjective nature of Bayesian belief manifests itself in the *Bayes' theorem*, that underlies Bayesian statistics:

$$\underbrace{\Pr[\Theta \mid \mathcal{A}]}_{\text{posterior probability}} = \frac{\overbrace{\Pr[\Theta]}^{\text{prior probability}} \cdot \overbrace{\Pr[\mathcal{A} \mid \Theta]}^{\text{likelihood}}}{\underbrace{\Pr[\mathcal{A}]}_{\text{data probability}}} \tag{3.1}$$

$$= \frac{\Pr[\Theta] \cdot \Pr[\mathcal{A} \mid \Theta]}{\int \Pr[\Theta] \cdot \Pr[\mathcal{A} \mid \Theta] \, \partial\Theta}. \tag{3.2}$$

The theorem directly follows from the law of total probability. Applied to the phylogenetic context with a parameter vector $\Theta = \{\tau, \vec{v}, \vec{\pi}, \vec{r}\}$ and an alignment $\mathcal{A}$, it means that we can determine the posterior probability (PP) of the model Θ given the data $\mathcal{A}$ as the product of the *prior* belief in the model Θ and the likelihood of $\mathcal{A}$ given Θ, divided by the probability of the data $\mathcal{A}$. For instance, we can choose the prior distribution to favor a simple model over a parameter-rich model for explaining the data. We then assess our prior belief in the light of the data (i.e., using its likelihood) and obtain an updated posterior belief for the model hypothesis. The probability of the data $\Pr[\mathcal{A}]$ is often hard or expensive to calculate. In simple cases, we can obtain $\Pr[\mathcal{A}]$ by integrating out Θ (see **Eq. 3.2**).

In the phylogenetic setting, we can use the Bayesian framework to obtain the joint posterior probability of all parameters Θ given the alignment $\mathcal{A}$. Naturally, we are particularly interested in the topology parameter τ. If we know the joint probability distribution $\Pr[\Theta \mid \mathcal{A}]$, we can determine the probability distribution of τ by integrating out any remaining parameters (referred to as *nuisance* parameters):

$$\Pr[\tau \mid \mathcal{A}] = \iiiint \Pr[\tau, \vec{v}, \vec{\pi}, \vec{r}, \alpha \mid \mathcal{A}] \, \partial\vec{r} \, \partial\vec{v} \, \partial\vec{\pi} \, \partial\alpha. \tag{3.3}$$

By doing so, we now know the probability of a tree τ with respect to any choice of all other parameters (rates in the GTR matrix, the branch lengths, or the α shape parameter of the Γ model of rate heterogeneity). Alternatively, if we integrate out all parameters except for α and branch lengths $\vec{v}$, we obtain their joint probability distribution of α and $\vec{v}$ and can examine relationships of the joint distribution.

3.1.2 Metropolis-Hastings Algorithm

Eq. 3.3 describes a high-dimensional integral. The number of free parameters grows with the number of taxa. Furthermore, in practice datasets are often also *partitioned*, that is, distinct genes are concatenated into an alignment matrix and we allow to have distinct (partially or fully independent) model parameters for each partition (e.g., $\alpha_1, \alpha_2, \ldots$). Eventually, only the topology might be shared across all partitions. Via partitioning we increase the realism of the model as we allow for distinct evolutionary scenarios (e.g., substitution rates) for distinct genes. However, when analyzing large-scale datasets, we can easily accumulate hundreds or even thousands of free parameters.

For computing the joint probability $\Pr[\Theta \mid \mathcal{A}]$ (i.e., the posterior probability), we can use a numeric approximation algorithm, the Markov chain Monte Carlo (MCMC) algorithm [76]. The MCMC method simulates a random walk of a Markov chain on the PP landscape. The Markov chain in MCMC differs in two ways from the ctMC that is used as a probabilistic model of sequence evolution. Firstly, we simulate a discrete-time Markov chain, that is, time is discretized into generations. Secondly, instead of a single discrete state (i.e., the nucleotide state), in each generation, the Markov chain assumes values for each random variable in Θ (with τ being the only discrete state and the remaining parameters all being continuous). The stationary distribution of the Markov chain approximates the distribution of interest: $\Pr[\Theta \mid \mathcal{A}]$.

Algorithm 1 Metropolis-Hastings algorithm

```
Input: alignment $\mathcal{A}$, weighted set of proposal functions $\Psi$, initial parameters $\Theta_0$,
Input: chain length $n$, uniform random numbers $r_1, \ldots, r_n \in [0, 1)$
Output: chain $c = \{\Theta_0, \ldots, \Theta_n\}$ sampling proportionally to $\Pr[\Theta \mid \mathcal{A}]$
 1: function METROPOLIS-HASTINGS( $\mathcal{A}$, $\Psi$, $\Theta_0$,$n$)
 2:     for $i \in \{1..n\}$ do
 3:         $q_j \leftarrow$ drawn proportionally to its weight $\omega_j$ in $\Psi$
 4:         $\Theta^* \leftarrow$ parameter update drawn proportionally from $q_i(\Theta^* \mid \Theta)$
 5:         $a = \xi(\Theta^* \mid \Theta_{i-1}) = \min\left(1, \frac{f(\Theta^*) \cdot f(\mathcal{A} \mid \Theta^*) \cdot q(\Theta_{i-1} \mid \Theta^*)}{f(\Theta_{i-1}) \cdot f(\mathcal{A} \mid \Theta_{i-1}) \cdot q(\Theta^* \mid \Theta_{i-1})}\right)$
 6:         if $r_i \leq a$ then
 7:             $\Theta_i \leftarrow \Theta^*$
 8:         else
 9:             $\Theta_i \leftarrow \Theta_{i-1}$
10:         end if
11:     end for
12:     return $c = \{\Theta_0, \ldots, \Theta_n\}$
13: end function
```

In the following, we describe the general phylogenetic MCMC algorithm (see **Alg.** 1 for the pseudo code version). We start the random walk of the Markov chain in an arbitrarily chosen initial configuration Θ_0. The MCMC algorithm employs a weighted set of proposals functions Ψ (also referred to as *proposal mixture*). Each proposal is associated with one or more parameters in Θ and provides a means to update the respective parameter values. In other words, proposals and their associated densities describe the "steps" of the random walk that is performed on the posterior probability landscape $\Pr[\Theta \mid \mathcal{A}]$ as well as the probability of performing the respective step. Each proposal $q_j \in \Psi$ has a weight ω_j, since we expect the number of generations that is necessary to reliably integrate over a parameter to vary. Typically, a large fraction of generations is required to integrate over the branch length parameter $\vec{v}$ and the topology parameter τ.

We simulate a generation of a chain c by first drawing a proposal q_j at random from Ψ according to its weight ω_j. Subsequently, we draw an update Θ^* from the density of the proposal. In the original version of the MCMC algorithm by Metropolis *et al.*, proposal densities were constrained to be symmetrical. In other words, we propose according to a density $q_j(\Theta)$. The Metropolis-Hastings (MH) algorithm [47] represents a powerful extension (detailed in **Alg.** 1), where we allow for asymmetrical proposal densities $q_j(\Theta^* \mid \Theta)$. This means that, we can propose an update Θ^* based on the current state Θ. As a consequence, the MH algorithm allows us to propose updates that are close to the current value of Θ. Thus, the chain is less likely to leave a region of high PP. Since we want to sample proportionally to $\Pr[\Theta \mid \mathcal{A}]$, we accept the updated parameter vector Θ^* proportionally to the change in posterior density. Notice that, we calculate the ratio of posterior densities of Θ^* and Θ as the prior ratio and likelihood ratio of Θ^* and Θ. Thus, using the MH algorithm we do not have to calculate the problematic probability of the data $\Pr[\mathcal{A}]$, since it cancels out in the posterior density ratio.

In the MH extension of the MCMC algorithm, we have to consider another factor when computing the acceptance probability $\xi(\Theta^* \mid \Theta)$ of a proposal q_j: by using an asymmetric proposal density $q_j(\Theta^* \mid \Theta)$, we bias the Markov chain towards certain values of Θ. In other words, if we frequently propose specific values of Θ and accept proportionally to the posterior density ratio, our samples of the PP are biased according to $q_j(\Theta^* \mid \Theta)$. To correct for this bias, we have to include the proposal density ratio $\frac{q_j(\Theta \mid \Theta^*)}{q_j(\Theta^* \mid \Theta)}$ to accept or reject an update. This term is also referred to as Hastings-ratio and consists of two components: in the denominator we have the density $q_j(\Theta^* \mid \Theta)$ of the update at hand and the numerator is the density $q_j(\Theta \mid \Theta^*)$ of the reverse update that will restore the original state Θ. The update and reverse update are sometimes also called forward and backward move. Including the Hastings-ratio, the probability of accepting an update Θ^* in the current state Θ is

$$\xi(\Theta^* \mid \Theta) = \min\left(1,\ \underbrace{\frac{f(\Theta^*)}{f(\Theta)}}_{\text{prior ratio}} \times \underbrace{\frac{f(\mathcal{A} \mid \Theta^*)}{f(\mathcal{A} \mid \Theta)}}_{\text{likelihood ratio}} \times \underbrace{\frac{q(\Theta \mid \Theta^*)}{q(\Theta^* \mid \Theta)}}_{\text{Hastings-ratio}}\right). \tag{3.4}$$

We use a random number $r_i \in [0,1)$ and accept the update Θ^*, if $r_i \leq \xi(\Theta^* \mid \Theta)$. Otherwise, the Markov chain remains in its current state Θ.

For adequately designed proposal functions Ψ, the Markov chain will rapidly leave the initial state Θ_0 and sample from regions of high PP. **Fig. 3.1** depicts the unnormalized posterior density trace plot of three Markov chains (i.e., not normalized by probability of the data, see **Eq. 3.1**). After 5,000 generations, all three chains sample parameter values with very similar logarithmic posterior density. Thus, it is likely (yet never guaranteed), that all chains have reached their *stationary phase*, that is, they sample parameter values that are proportional to the absolute PP $\Pr[\Theta \mid \mathcal{A}]$. The time (expressed as a number of generations) that is necessary to reach this stationary phase is called *burn-in*. Chain states prior to the burn-in are typically discarded from the sample. In **Fig. 3.1** chain c_1 (blue) and c_2 (pink) employ the default proposal mixture as of ExaBayes version 1.4.1. While c_1 uses a MP tree as starting tree, c_2 starts with a random tree. Although the initial topology c_2 is substantially less likely than the initial topology of c_1, the burn-in only takes about 2,000 generations for either chain. In contrast to the diverse proposal set of c_1 and c_2, chain c_3 (green) only applies a single topological proposal (the stochastic nearest neighbor interchange (stNNI), see **Sect.** 3.3.1) that moves particularly slowly through the tree topology space. Thus, the burn-in takes considerably longer. We call a proposal q_j more *modest* than another proposal q_k, if q_j does not change parameters as drastically as q_k. In the opposite case, we call q_k *bolder* than q_j. Typically (but not necessarily), the *acceptance rate* of bolder proposals is lower than the acceptance rate of more modest proposals.

The detailed Subfigure in **Fig. 3.1** starting at generation 5,000 is a trace plot that is typical for the random walk of a Markov chain on the landscape of $\Pr[\Theta \mid \mathcal{A}]$. It directly follows from the MH algorithm, that subsequent chain states are highly correlated. Thus, we do not loose important information by *thinning* the chain, that is, we only store each n-th sample. In practice, we always thin the chain. However, thinning is foremost a pragmatic means to reduce disk space requirements for the samples.

Note that, the trace plot does not provide a means to identify whether the chains correctly sample the posterior density proportionally (e.g., all there chains may have ended up in the same local optimum). Furthermore, we can not tell from the trace plot, how many generations will be necessary until we have an accurate approximation of $\Pr[\Theta \mid \mathcal{A}]$. Inefficient proposals will result in a high *rejection rate* and it will therefore take a large number of generations until the states of the chain accurately approximate the posterior. If a chain c_a approximates the posterior faster than another chain c_b, we say that c_a has better *mixing* than c_b. Alternatively, we can expect that the *time to convergence* is longer for c_b than for c_a.

Finally, BI inherits desirable properties from ML, such as statistical consistency. The MH algorithm guarantees that the simulated Markov chain converges towards the target distribution $\Pr[\Theta \mid \mathcal{A}]$ as the number of generations approaches infinity. Thus, under the correct model and given appropriate priors, BI will reconstruct the true topology as the amount of data in $\mathcal{A}$ approaches infinity.

3.1.3 Priors

For the phylogenetic models implemented in ExaBayes, priors on parameter values play a minor, yet not negligible role. Even for comparably small input alignments, the likelihood ratio between the proposed values Θ^* and the current values Θ has a substantially stronger impact on the acceptance probability $\xi(\Theta^* \mid \Theta)$ than the respective

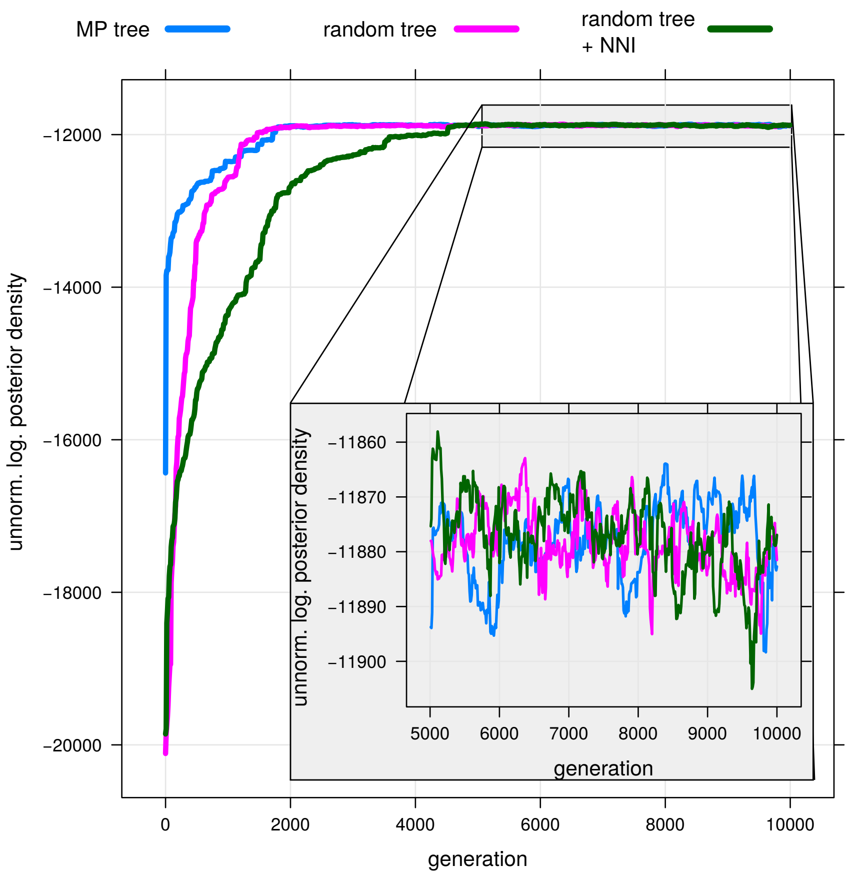


Figure 3.1: Logarithmic unnormalized posterior density trace of three chains that were simulated using ExaBayes version 1.4.1. For the first chain (blue), we employed the default proposal mixture and started in a MP tree. In contrast, the second chain (pink) uses a random starting tree. The third chain (green) uses a random starting tree and only employs a single particularly modest proposal for integration of the tree topology parameter. The inner Subfigure depicts the random walk of the Markov chains in their potentially stationary phase.

prior ratio. The exponential distribution with the density function $f(x;\lambda) = \lambda \exp(-\lambda x)$ is a particularly popular prior for univariate parameters (such as $\vec{v}$ or α). The density function favors small values (i.e., high rate heterogeneity in case of α) and we only have to choose one parameter λ, that reflects both the mean and standard deviation (both λ^{-1}). As a default in ExaBayes, we choose $\lambda := 10$ resulting in an expectation that branch lengths are longer (i.e., 0.1 expected substitution per bp) than observed in many biological scenarios. However, an exponential prior is primarily problematic, if λ is too large [61], since we then strongly favor short branch lengths over longer ones. For particularly small values of λ, an exponential prior converges against a uniform prior and thus comes as close as possible to being *non-informative.*

Notice, that even uniform priors can not be considered as non-informative, since an upper and lower bound are needed for a uniform distribution. Typically, we want to avoid using strongly-informative priors, except when there exist good reasons for such a strong prior belief: a reasonable application of strongly informative priors are priors on the ages of internal nodes under models that use a molecular clock. In these cases, an appropriate prior allows to incorporate carbon-dating evidence into the phylogenetic analysis. Without such carbon dating evidence, practitioners of BI typically want an exponential prior to reflect their prior belief in short branch lengths. Similar to MP (see **Sect.** 2.5), we can thereby incorporate the assumption that a tree with few substitutions is generally preferable.

For strongly interdependent parameters for which individual values need to sum to 1, we typically employ a Dirichlet prior. In the phylogenetic context, a Dirichlet prior is used for the stationary frequencies $\vec{\pi}$ or the normalized substitution rates $\vec{r}$. For instance for $\vec{\pi}$, the density of the Dirichlet prior is defined as

$$f(\pi_A, \pi_C, \pi_G; \lambda_A, \lambda_C, \lambda_G, \lambda_T) = \frac{1}{B(\vec{\lambda})} \cdot \prod_{i \in \mathcal{S}} \pi_i^{\lambda_i - 1}, \tag{3.5}$$

where $B(\vec{\lambda})$ is the Beta function parametrized with $\vec{\lambda}$. If $\lambda_i = 1, \forall i \in \mathcal{S}$, we obtain the equivalent to a multivariate uniform prior, which is the typical setting for phylogenetic BI. Typically, no prior information with respect to the stationary frequencies or reversible substitution rates is available (although fixed-rate AA substitution models can optionally include empirical frequencies).

For topologies, ExaBayes only allows for a uniform prior on τ (apart from fixing τ to a certain topology). Consider that the prior belief that all topologies are equally probable does not imply that we believe that all bipartitions have equal probability. In this case, our prior belief depends on the number of taxa in the partitions of a bipartition $B = (b \mid \bar{b})$.

Assume, we want to compute the probability of a bipartition that partitions the set of 10 taxa in a tree into a partition of size 2 and a partition of size 8. The number of unrooted phylogenetic trees that have a specific bipartition is the product of the number of rooted trees that can be constructed from one partition (i.e., only one possible rooted tree for the partition with 2 taxa) times the number of rooted trees that can be constructed from the complementary partition (i.e., there are 135,135 rooted trees with 8 taxa). A bipartition with $|b| = 3$ and $\overline{|b|} = 7$ on the other hand only occurs in $3 \times 10,395$ trees. Thus, a uniform prior on bipartitions would conflict with a uniform prior on trees [89]. This example further illustrates that a uniform distribution is not necessarily a straight-forward non-informative choice for a prior.

3.1.4 Tuning Proposals

Many proposals on continuous parameters are parametrized themselves. Assume that as a proposal density $q(v^* \,|\, v)$ for updating branch lengths, we employ a normal distribution $\mathcal{N}(v, \sigma^2)$ that proposes a new branch length v^* around the current branch length v with variance σ^2. Here, σ^2 is an adjustable/tunable proposal parameter. As we increase the value of σ^2, our proposal becomes bolder. However, drastic parameter updates usually induce a stronger change in the posterior density and thus, the acceptance rate is likely to decrease (i.e., the chain moves more slowly through parameter space).

By modifying these proposal parameters such as σ^2, we can adjust how *bold* or *modest* the proposed values Θ^* will be. For proposals that draw updated values from a multivariate normal distribution, it was shown that there exists an optimal acceptance rate $\xi(\Theta^* \mid \Theta) = 0.234$ [94]. Because of the involved nature of the PLF, no attempt has been conducted to analytically determine an optimal acceptance rate for phylogenetic BI. Yet, for practical reasons we can strive for a target acceptance rate of 25% and thereby reduce the chance that proposals are either extremely modest or extremely bold for the dataset being analyzed. For the example described above (i.e., a proposal using the normal distribution that is parametrized with σ^2), we can determine a new proposal parameter σ^{2*} given the previous parameter σ^2 as [after 93]

$$\sigma^{2*} = \exp(\log(\sigma^2) \pm (i+1)^{-0.5}). \tag{3.6}$$

In **Eq. 3.6**, i counts the number of times, the parameter has already been tuned. We typically estimate the current acceptance rate $\hat{\xi}$ as the number of times the proposal has been accepted during the 100 previous attempts. Whether we have to increase or decrease σ^2 depends on whether $\hat{\xi}$ is larger or smaller than the target acceptance rate and the direction that increases boldness/modesty of the proposal. Notice that, for $i \to \infty$, adaptations to σ^2 become negligible, that is, $\sigma^{2*} \to \sigma^2$. This means that asymptotically, the tuning mechanism does not influence the proportional sampling of the posterior by the Markov chain. However, tuning proposals improves the mixing of the chain by dynamically adjusting the boldness of the proposals.

Many of the proposals that will be introduced throughout this Chapter have tunable parameters. For the sliding window (see **Sect.** 3.2.1), we can adapt the window size δ. For the branch length (see **Sect.** 3.2.2) or tree length multipliers (see **Sect.** 3.2.4), the λ parameter that specifies the size of the sliding window on the logarithmic scale can be tuned. Finally, for a Dirichlet proposal (see **Sect.** 3.2.5), we can tune the scaling factor α.

3.1.5 Assessing Convergence

Apart from criticism about the subjectivity in BI, another major problem with respect to BI is the difficulty to determine whether a chain has converged against the posterior we want to estimate. In practice, we assume that, if multiple chains that started at independent locations in the parameter space (i.e., at $\Theta_0, \Theta'_0, \Theta''_0, \ldots$) converge against the same stationary distribution, we can be increasingly confident about the convergence (i.e., correctness) of the results as the number of independent chains we run increases. However, in practice there is no way to rule out that all chains have been attracted by the same local minimum and thus provide an incorrect or incomplete sample of $\Pr[\Theta \,|\, \mathcal{A}]$. ExaBayes implements the following three methods for assessing convergence.

Effective Sample Size

An important convergence criterion for continuous parameters is the effective sample size (ESS) of all states extracted from the chain. Since proposals for new states of the Markov chain usually depend on the previous state of the chain, it is expected that parameter values (e.g., branch lengths $\vec{v}$) for subsequent generations are highly correlated. A particularly high correlation can be expected, if the proposal in question is frequently rejected (i.e., the parameter value remains the same) or if the parameter updates are chosen in a too narrow region around the original value (e.g., see **Sect.** 3.2.1). For highly correlated samples, the absolute number n of parameter samples therefore does not indicate whether the underlying distribution has been sufficiently sampled. In other words, we can not conclude that a chain approximates the target distribution well, just because the chain has been running for a large number of generations.

The ESS measures the correlation in a set of samples $\Theta = \theta_1, \theta_2, \ldots \theta_n$ with mean μ. It is based upon the autocorrelation coefficient r_k that correlates a sequence with itself for a given lag k (see **Eq. 3.7**). In other words, for a lag k an element θ_i is matched by element θ_{i+k}. The auto-correlation at lag k is determined as

$$r_k = \frac{\sum_{i=1}^{n-k} (\theta_i - \mu)(\theta_{i+k} - \mu)}{\sum_{i=1}^{n} (\theta_i - \mu)^2}. \tag{3.7}$$

Summing over all possible lags, we obtain the ESS as [see 127, p.161]

$$\mathrm{ESS}_\Theta = \frac{n}{1 + 2 \cdot \sum_{i=1}^{\infty} r_i}. \tag{3.8}$$

In essence, for a marginal distribution the corresponding ESS represents the number of uncorrelated samples, if those were drawn identically and independently from the underlying posterior distribution. In practice, the sum of autocorrelation coefficients is typically not computed beyond a fixed number of elements (e.g., 2,000 in ExaBayes or MrBayes), since autocorrelation rapidly decreases for increasing lags.

Potential Scale Reduction Factor

ESS values provide no inherent guarantee that the posterior has been sampled correctly. High ESS values are necessary for reliable estimates of the posterior, but they are not sufficient: the ESS indicates that the available number of samples corresponds to a certain number of uncorrelated samples, but not whether these samples approximate the posterior. To satisfy the latter requirement, the potential scale reduction factor (PSRF) [41] was introduced. Its underlying rationale is to start several chains at random values that have a higher variance than the target posterior. Subsequently, it computes two distinct estimators of the variance of the samples and uses their ratio to decide upon the convergence of the samples of all chains to the target posterior:

$$W = \frac{1}{m}\sum_{i=1}^{m}\left[\frac{1}{n-1}\sum_{j=1}^{n}(x_{ij}-\mu_i)^2\right], \tag{3.9}$$

$$B = \frac{n}{m-1}\sum_{i=1}^{m}(\mu_i-\mu)^2, \tag{3.10}$$

$$\hat{\sigma}^2 = \frac{n-1}{n}W + \frac{1}{n}B. \tag{3.11}$$

The PSRF is based upon the *within-chain variation* W and the *between-chain variation* $\frac{B}{n}$ for m chains with n samples each (where x_{ij} is the i-th sample of the j-th chain). The within-chain variation W is defined as the average of the variances of the samples x_{ij} in a chain (see **Eq. 3.9**) with respect to mean μ_i of samples x_{ij} in a chain i. We compute the between-chain variation $\frac{B}{n}$ as the variance of the sample averages μ_i of chain i around the global average μ (computed from all samples x_{ij}, see **Eq. 3.10**). The between-chain variation $\frac{B}{n}$ is multiplied by n to account for the chain length. In case all chains approximated the overall mean μ correctly, $\frac{B}{n}$ approaches 0. The term $\hat{\sigma}^2$ (see **Eq. 3.11**) is designed to be an estimator of the variance σ^2 of the target posterior. By design, $\hat{\sigma}^2$ overestimates the true σ^2, when chains have not sufficiently converged. The second estimator for σ^2 is W, which is highly likely to underestimate the σ^2, if chains are stuck in local maxima. The PSRF is then defined as the ratio of $\hat{\sigma}^2$ and W. If all chains have sufficiently converged (i.e., they do not just sample a local optimum close to their initial state), the PSRF assumes values close to one (typically < 1.1 is deemed sufficient). For the calculation of this statistic, it is important to discard the burn-in, since otherwise the within-chain variance will be inflated.

Standard Deviation of Split Frequencies

A measure analogous to the ESS for the discrete topology parameter τ does not exist. As an alternative, we can assess whether independent chains that have started at different random or parsimony trees yield the same distribution for τ. However, given the large number of trees that may be contained in the high PP region of τ (often exceeding 100 – 1,000 trees), estimating the marginal PP of distinct topologies can be prohibitively time consuming in practice. The reason is that, often the total collection of sampled trees in the high PP region is only marginally smaller than the number of unique sampled topologies. If the PP is low for inner branches of the tree, then the high number of unique tree topologies in the sample is the combinatorial result of competing inner branches. Specifically, for trees comprising a high number of taxa with a high degree of uncertainty there are cases, where no single tree is sampled more than once (e.g., see **Sect.** 4.1.1). If split frequencies are similar for various independent chains, we can nonetheless assume that we have accurately sampled the tree parameter τ. A popular measure for this, the average standard deviation of split frequencies (ASDSF) [65] is defined as

$$\text{ASDSF}(\mathcal{B}) = \frac{1}{m}\sum_{b\in\mathcal{B}}\sqrt{\frac{1}{n}\cdot\sum_{i\in\{1..n\}}(f_i(b)-\mu_b)}, \tag{3.12}$$

$$\text{MSDSF}(\mathcal{B}) = \max_{b\in\mathcal{B}}\left\{\sqrt{\frac{1}{n}\cdot\sum_{i\in\{1..n\}}(f_i(b)-\mu_b)}\right\}, \tag{3.13}$$

for n bipartitions $b \in \mathcal{B}$, where $\mathcal{B}$ is the set of all bipartitions that occur in any of the trees that have been sampled in any of the m independent chains $c_1 \dots c_m$. Here, $f_i(b)$ indicates the frequency of bipartition b in the i-th chain c_i and μ_b is the average frequency of b. The default convergence threshold for the ASDSF is $\leq 5\%$. However, it has been argued that this threshold should be reduced to at least 1% [125]. Typically, the initial 25% of samples are discarded as burn-in. Bipartitions that do not occur in more than 10% of samples in any of the chains are excluded from ASDSF computation. Similarly to the PP of unique tree topologies, the accurate PP of a rare bipartition is hard to estimate. A more conservative measure is the maximum standard deviation of split frequencies (MSDSF) (see **Eq. 3.13**) that only considers the bipartition that exhibits the highest deviation in split frequencies among chains.

3.2 Proposals for Continuous Parameters

In this Section, we discuss proposals for the continuous parameters comprising the α shape parameter of the Γ model of rate heterogeneity, the substitution rates $\vec{r}$, stationary frequencies $\vec{\pi}$, and branch length parameters $\vec{v}$.

3.2.1 Sliding Window

One of the most basic proposals is the sliding window proposal. For a value x of parameter X and a given window size δ, we propose a new value $x^* := x + \delta \cdot (u - 0.5)$, where u is drawn from a uniform distribution in $[0, 1)$. ExaBayes implements a sliding window for proposing rates $\vec{r}$ and frequencies $\vec{\pi}$. Since for n frequencies, there are only $n - 1$ free parameters, we are forced to modify more than one value at a time, if we attempt to update a frequency using a sliding window. This also applies to the substitution rates $\vec{r}$, which at the level of ExaBayes are represented, such that their sum is one. This is because the representation relative to a reference rate (as used by the phylogenetic likelihood library (PLL) and the underlying model, see **Sect.** 2.6.1) is impractical for Bayesian proposals, specifically the Dirichlet proposal (see **Sect.** 3.2.5), respectively Dirichlet priors (see **Sect.** 3.1.3).

In ExaBayes, for a sliding window proposal on rates or frequencies, we choose two parameter values at random and increase one of them by some value drawn from the sliding window, while the second value is decreased accordingly. Since the move is symmetric, its Hastings-ratio is 1. Thus, apart from its simplicity, one of the major advantages of the sliding window is the fact, that the acceptance probability is not penalized by a Hastings-ratio.

For numerical reasons, the range of parameter values is further constrained by the PLL [39], which is used for likelihood computations in ExaBayes. Furthermore, any single component of $\vec{\pi}$ or $\vec{r}$ must not become negative. If the value drawn from a sliding window exceeds the valid range, we project the value back into the valid range. Assume we draw $x^* = x + 0.5\delta$ and there exists an upper bound b_{upper} with $x < b_{\text{upper}} < x^*$. Then, we instead propose a reflected $x^* = b_{\text{upper}} - (x + 0.5\delta - b_{\text{upper}})$. Note that, the reflection does not affect the Hastings-ratio: while there is a higher probability of drawing values close to b_{upper}, the probability of drawing x from a sliding window around x^* in the reverse move increases by the same degree. Thus, the Hastings-ratio also is 1 for this boundary case. Finally, δ is a tunable parameter (see **Sect.** 3.1.4).

3.2.2 Multiplier

Motivation

The range of values for a univariate component (e.g., branch lengths $\vec{v}$ or α parameter of rate heterogeneity), for which we observe high PP can often stretch across more than one order of magnitude. Furthermore, it is considered good practice in BI to start the analysis in overdispersed states (i.e., parameters assume values drawn from a random distribution that has a higher variance than the PP distribution). In both cases, a sliding window has severe drawbacks. Depending on the window size, an excessive number of steps may be necessary to change the component value by an order of magnitude (or even worse, by more than one order). If window sizes are chosen too large and we have a high PP region close to a parameter value of 0, then a considerable number of proposals may be necessary until a value for this narrow interval is proposed.

Derivation

In such cases, a multiplier represents an advantageous alternative: we propose a new value x^* as $x^* = m \cdot x$, where x is the current state of the component and $m = h(u)$ is the multiplier generated by some function h from a random variable u that is drawn from a uniform distribution in $[0, 1)$. The following derivation of the proposal and its Hastings-ratio represents a more detailed version of the derivation published in a study on efficiency of topological proposals [65].

We require $h(u)$ to be $\in (a^{-1}, a)$ to guarantee the reversibility of the proposal (otherwise the chain would not be positively recurrent any more). In other words, the reversibility ensures that we do not have proposals with a Hastings-ratio of 0. We also postulate as a desirable property of $h(u)$, that the probability of generating a multiplier that increases the component value should be equal to the probability of decreasing the component value. That is: $\int_1^a h(u)\ \partial u = \int_{a^{-1}}^1 h(u)\ \partial u$. This is because there usually exists no *a priori* reason to bias the proposal in either direction. In other words, we want to draw a multiplier from a sliding window on the logarithmic scale in the range (a^{-1}, a). To satisfy the aforementioned requirements, we choose the density of the multiplier as $g(m) = \frac{1}{2\ln(a)\cdot m}$.

Inverse transform sampling allows us to determine a function $h(u)$, that transforms uniformly distributed random numbers u into random numbers that are distributed according to $g(m)$, if the cumulative distribution function (CDF) has a closed form. Thus, we first have to determine the inverse function of the CDF. The generalized antiderivative of $g(m)$ is $G(m) = \frac{\ln(m)}{2\cdot\ln(a)} + C$. If we constrain $G(m)$ to the interval (a^{-1}, a), we obtain the CDF $G(m) = \frac{\ln m}{2\ln(a)} + 0.5$. We can now set $G(m) := u$ and thus effectively ask for which m we obtain a CDF value of u. As postulated earlier, the median of the CDF is 1.0 (resp., $G(1) = 0.5$). If we solve for m, we obtain $h(u)$ for generating multipliers distributed according to $g(u) \in (a^{-1}, a)$:

$$m = h(u) = \exp(\underbrace{2 \cdot \ln(a)}_{\kappa} \cdot (u - 0.5)). \tag{3.14}$$

Typically, the parameter $\kappa := 2 \cdot \ln(a)$ is either provided by the user or tuned via a tuning mechanism (see **Sect.** 3.1.4).

Hastings Ratio

Since the density of multipliers $g(m)$ monotonically decreases, so does the proposal density $q(x^* \mid x) = h(u) \cdot x$ according to which we draw our proposed value x^*. In other words, each multiplier < 1 (that shrinks x) has a higher density than each multiplier > 1 that enlarges x. For reverting a move with a multiplier m, we have to draw the inverse multiplier m^{-1}. Thus, it is intuitively clear, that we need to account for this disequilibrium (i.e., that either the forward or the backward move has a higher density than the associated reverse move) via a Hastings-ratio $\neq 1.0$ in a multiplier move.

Compared to the sliding window proposal, the Hastings-ratio of a multiplier is not trivial to determine. A powerful approach for calculating Hastings-ratios was developed for reversible jump Markov chain Monte Carlo (rjMCMC) [45]. The rjMCMC method allows for trans-dimensional moves, that is, moving between models with different numbers of parameters (one example of such a proposal would be to link or unlink partitions for a parameter, e.g., α). This method, referred to as Green's method [45], considers the vectors of component values $\vec{x}^*$ that have been proposed according to a deterministic scheme (such as the multiplied value x^* in our case) using a vector of random numbers $\vec{r}$ that follow a distribution $g(\vec{r})$. Inversely, values $\vec{x}$ are the initial component values (prior to the proposal) that can be reached from state $\vec{x}^*$ using random numbers $\vec{r}^*$ that are distributed according to $g^*(\vec{r}^*)$. According to Green's method for rjMCMC, a generalized formula for the Hastings-ratio (including rjMCMC) is

$$\frac{q(\vec{x} \mid \vec{x}^*)}{q(\vec{x}^* \mid \vec{x})} = \frac{j_M(\vec{x}^*)}{j_M(\vec{x})} \cdot \frac{g^*(\vec{r}^*)}{g(\vec{r})} \cdot \left| \frac{\partial(\vec{x}^*, \vec{r}^*)}{\partial(\vec{x}, \vec{r})} \right|, \tag{3.15}$$

where j_M is a function that specifies the probability of choosing a specific move (forward $\vec{x}^*$ or backward $\vec{x}$), which in case of the multiplier can be ignored (since multiplier m and reverse multiplier m^{-1} are fully determined by the proposal mechanism). As mentioned, g^* is the proposal density for proposing the backward move. However, in contrast to rjMCMC, there is no need to differentiate between forward and backward proposals in our move. Thus, we can set $g^* := g$. Finally, $\left| \frac{\partial(\vec{x}^*, \vec{r}^*)}{\partial(\vec{x}, \vec{r})} \right|$ is the absolute value of the determinant of the Jacobian matrix for transforming vector $\{\vec{x}, \vec{r}\}$ to $\{\vec{x}^*, \vec{r}^*\}$. Each entry of the Jacobian matrix contains the partial derivative of an element of the column vector $\{\vec{x}^*, \vec{r}^*\}$ with respect to an element of the row vector $\{\vec{x}, \vec{r}\}$ and accounts for interactions between component values and random numbers used in the transformation.

Thus, for a multiplier $m = h(u)$ on a single parameter component x (such as a single branch length), the proposed state is $x^* = m \cdot x$. For the reverse move, we need to multiply with $m^* = m^{-1}$. Thus, we can substitute **Eq. 3.15**:

$$\begin{aligned}
\frac{q(x \mid x^*)}{q(x^* \mid x)} &= \frac{g(m^*)}{g(m)} \cdot \begin{vmatrix} \frac{\partial x^*}{\partial x} & \frac{\partial m^*}{\partial x} \\ \frac{\partial x^*}{\partial m} & \frac{\partial m^*}{\partial m} \end{vmatrix} && (3.16) \\
&= \frac{\kappa^{-1} \cdot m}{\kappa^{-1} \cdot m^{-1}} \cdot \begin{vmatrix} m & 0 \\ x & -m^{-2} \end{vmatrix} && (3.17) \\
&= m^2 \cdot |-m^{-1} - 0| && (3.18) \\
&= m. && (3.19)
\end{aligned}$$

To summarize, the Hastings-ratio for the proposed value $x^* := m \cdot x$, given the current state x is simply the multiplier m, that is $\frac{q(x \mid x^*)}{q(x^* \mid x)} = m$.

Application

Typically, classical statistical quantities that are used to characterize a distribution are the mean and the standard deviation. However, for an optimal choice of κ (i.e., the window size on the logarithmic scale), the only relevant statistic for characterizing an underlying marginal posterior is its coefficient of variation (CV). The CV is a scale-invariant statistic that indicates how strongly a distribution varies around its mean value μ. It is defined as $\frac{\sigma}{\mu}$ (where σ is the standard deviation). As an example, consider a marginal branch length posterior f_{v_1} (with $\mu = 0.0001$, CV $= 1.0$), a marginal branch length posterior f_{v_2} (with $\mu = 1.0$, CV $= 1.0$) and f_{v_3} (with $\mu = 0.0001$, CV $= 0.001$). Assuming, for simplicity, that f_{v_1} and f_{v_2} are similar (e.g., both unimodal distributions), a branch length multiplier with tuned κ^* can integrate over both distributions f_{v_1} and f_{v_2} with similar efficiency, although both distributions have a mean that is at opposite ends of the spectrum of typically observed branch lengths. However, such a proposal will integrate over f_{v_3} in a particularly inefficient manner. A branch length multiplier based on the respective κ^*, proposes values that are too bold for f_{v_3}, which results in an excessive rejection rate.

In ExaBayes, the multiplier proposal is used for integrating over branch lengths $\vec{v}$ and the α shape parameter of the Γ distribution that models among-character rate heterogeneity. For the branch length parameter, the multiplier represents a particularly valuable proposal, since with an increasing number of taxa it becomes more likely that, in the same tree we have some particularly long branches (e.g., in the range $[0.1, 1.0]$), while other branches are particularly short (e.g., in the range $[10^{-5}, 10^{-3}]$). Efficient integration over such a diverse set of branch lengths using sliding windows would require several branch-specific sliding windows (i.e., one proposal instance for proposing branch lengths for each individual bipartition). As we will see in **Sect.** 4.1.3, the range of CVs of branch lengths in several empirical datasets is comparably limited. Thus, a single branch length multiplier is sufficient to yield sufficient sampling efficiency and mixing when applied to all branches in the tree.

3.2.3 Node Slider

In MrBayes a second move is specifically designed for integrating over branch lengths, the node slider. The node slider can be interpreted as a less bold version of the Local move [67] (introduced in BAMBE [102]). The Local move scales three adjacent branch lengths (v_{ax}, v_{xy}, v_{yc}) with a multiplier m, where $v_{ax} = (a, x)$, $v_{xy} = (x, y)$ and $v_{yc} = (y, c)$ and where a, x, y and c are nodes in a tree τ.

Subsequently, we choose either subtree x or y and move the subtree using a uniform slider along the path from a to c. All locations are allowed along $a \leftrightarrow c$. Assuming subtree x was chosen for sliding, this means that

1. x assumes a new position along a and y, such that $v^*_{ax} + v^*_{xy} = m \cdot (v_{ax} + v_{xy})$
2. or subtree x is reinserted in the adjacent branch (y, c) that is beyond $a \leftrightarrow y$. which results in new branch lengths v^*_{xc}, v^*_{yx} and v^*_{ay} with $v^*_{xc} + v^*_{yx} = m \cdot v_{yc}$ and $v^*_{ay} = m \cdot (v_{ax} + v_{xy})$.

Thus, in case 2) the Local move also performs a nearest neighbor interchange (NNI) move (see **Sect.** 3.3.1). We may say that the Local move tries to bridge the gap between discrete topological changes and continuous branch length updates. Up until MrBayes v.3.2 (excluding), the Local proposal was the only topological proposal available except for the extending tree-bisection and reconnection (eTBR) proposal (see **Sect.** 3.3.1). Since the Local proposal is comparably complex, its Hastings-ratio was initially incorrectly determined as m^2 and later corrected to m^3 by two separate proofs [53, 66].

The node slider operates on two adjacent branches v_{ab} and v_{bc} (for an illustration, see **Fig. 3.2**), respectively a path from node a over b to node c. Similarly to Local, the node slider first scales these two adjacent branches and then slides the intermediate subtree rooted at node b in either direction using a sliding window. Thus, we first

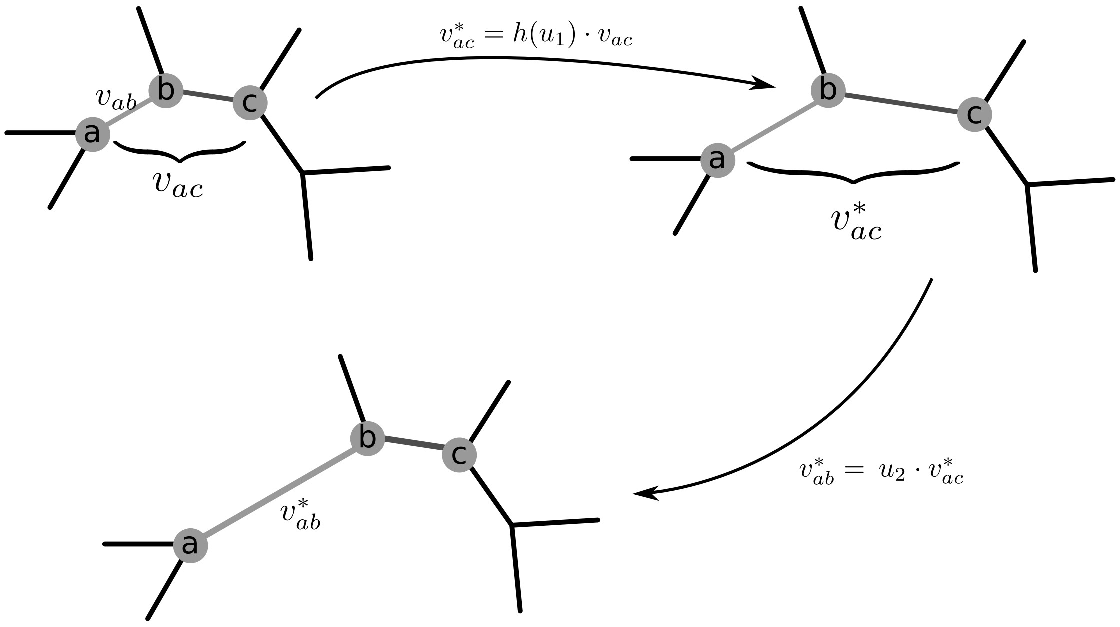

$v_{ac}^* = h(u_1) \cdot v_{ac}$
v_{ab}
a
b
c
v_{ac}
v_{ac}^*
$v_{ab}^* = u_2 \cdot v_{ac}^*$
v_{ab}^*

Figure 3.2: Illustration of the node slider move. The sum of two adjacent branches v_{ac} is first scaled via a multiplier $h(u_1)$. Subsequently, we propose a new ratio u_2 according to which v_{ab}^* splits up v_{ac}^*.

scale the joint branch path v_{ac} by a multiplier $m = h(u_1)$ for which we need a single uniformly distributed random variable u_1 (see **Eq. 3.20**). Thereby the length of the path v_{ac} is transformed into v_{ac}^*. Notice, that in contrast to the LOCAL proposal, no topological change can be induced. Next, we draw a second uniform random variable u_2 that determines, whereto the subtree rooted at b will be slid along the path v_{ac}^*. This yields the proposed branch length v_{ab}^* (see **Eq. 3.21**). Branch v_{bc}^* needs not be considered explicitly, since it depends on v_{ac}^* and v_{ab}^*. In contrast to the derivation of the Hastings-ratio for the single parameter multiplier, we do not consider the distribution of the multiplier for the terms $g(\vec{r}^*)$ or $g(\vec{r})$. Instead, we directly treat $h(u)$ as part of equations that yield updated branch lengths as a function of two uniform variables u_1 and u_2. Thus, for the Jacobian, we can consider the transformation of $\{\vec{x}, \vec{r}\} = \{v_{ac}, v_{ab}, u_1, u_2\}$ into $\{\vec{x}^*, \vec{r}^*\} = \{v_{ac}^*, v_{ab}^*, u_1^*, u_2^*\}$.

For reversing a node slider move, we have to consider the steps in inverted order. We need to draw the value from the uniformly distributed random variable u_2^* that restores the original ratio between v_{ab} and v_{ac} (see **Eq. 3.23**). After that, we scale v_{ac}^* back to v_{ac} using an inverse multiplier m^{-1}. As derived in **Sect.** 3.2.2, $h(1-u)$ yields the inverse multiplier of $h(u)$, thereby we obtain u_1^* (see **Eq. 3.22**).

$$\begin{aligned}
v_{ac}^* &= v_{ac} \cdot h(u_1), && (3.20)\\
v_{ab}^* &= u_2 \cdot v_{ac} \cdot h(u_1), && (3.21)\\
u_1^* &= 1 - u_1, && (3.22)\\
u_2^* &= \frac{v_{ab}}{v_{ac}}. && (3.23)
\end{aligned}$$

Since the node slider move is entirely deterministic, we do not have a choice between multiple alternatives. Thus, the term j_M in **Eq. 3.15** cancels out. The second ratio (i.e., $g(\vec{r})$) need not be considered, since u_1 and u_2 are uniformly distributed. Thus, the Hastings-ratio in this case, is simply the absolute value of the determinant of the partial derivatives in the Jacobian matrix. The partial derivatives of $\{\vec{x}^*, \vec{r}^*\} = \{v_{ac}^*, v_{ab}^*, u_1^*, u_2^*\}$ with respect to $\{\vec{x}, \vec{r}\} = \{v_{ac}, v_{ab}, u_1, u_2\}$ yield the following Jacobian matrix

$$|J| = \begin{vmatrix}
\frac{\partial v_{ac}^*}{\partial v_{ac}} & \frac{\partial v_{ab}^*}{\partial v_{ac}} & \frac{\partial u_1^*}{\partial v_{ac}} & \frac{\partial u_2^*}{\partial v_{ac}} \\
\frac{\partial v_{ac}^*}{\partial v_{ab}} & \frac{\partial v_{ab}^*}{\partial v_{ab}} & \frac{\partial u_1^*}{\partial v_{ab}} & \frac{\partial u_2^*}{\partial v_{ab}} \\
\frac{\partial v_{ac}^*}{\partial u_1} & \frac{\partial v_{ab}^*}{\partial u_1} & \frac{\partial u_1^*}{\partial u_1} & \frac{\partial u_2^*}{\partial u_1} \\
\frac{\partial v_{ac}^*}{\partial u_2} & \frac{\partial v_{ab}^*}{\partial u_2} & \frac{\partial u_1^*}{\partial u_2} & \frac{\partial u_2^*}{\partial u_2}
\end{vmatrix}. \qquad (3.24)$$

The partial derivatives of formulas **Eq. 3.20**, **Eq. 3.21**, **Eq. 3.22** and **Eq. 3.23** are straight-forward and allow us to determine the absolute value of the determinant of the Jacobian matrix of **Eq. 3.24**. For the terms $\frac{\partial v_{ac}^*}{\partial u_1}$ and $\frac{\partial v_{ab}^*}{\partial u_1}$, we have to calculate the derivative of **Eq. 3.14**.

$$|J| = \begin{vmatrix} h(u_1) & u_2 \cdot h(u_1) & 0 & -\frac{v_{ab}}{v_{ac}^2} \\ 0 & 0 & 0 & \frac{1}{v_{ac}} \\ \lambda v_{ac} h(u_1) & \lambda u_2 v_{ac} h(u_1) & -1 & 0 \\ 0 & v_{ac} \cdot h(u_1) & 0 & 0 \end{vmatrix} \tag{3.25}$$

$$= \left| 0 - \left(\frac{1}{v_{ac}} \cdot (-1) \cdot v_{ac} \cdot h(u_1) \cdot h(u_1) \right) \right| \tag{3.26}$$

$$= h(u_1)^2. \tag{3.27}$$

As we see in **Eq. 3.25**, only one component of the determinant of J is non-zero. We conclude that, the Hastings-ratio of the node slider move is m^2, where $m = h(u_1)$ is the initial multiplier. As mentioned previously, re-parametrization does not affect the outcome of Green's method, thus we could have considered v_{ab} / v_{bc} instead of v_{ab} / v_{ac}. Alternatively, we could have considered the vector $\{\vec{x}, \vec{r}\} = \{v_{ac}, v_{ab}, h(u_1), u_2\}$ with an altered $g(\vec{r})$ and a partial derivation with respect to $m = h(u_1)$ instead of u_1.

The parameter κ of the initial multiplier m is tunable. The sliding window however (as in the LOCAL move) is parameter-free. Thus, according to the authors of MRBAYES, tuning of κ does not necessarily improve the performance of the node slider, since the acceptance ratio may be obfuscated by extreme values of the sliding window u_2. Thus, EXABAYES implements a node-slider without tuning. As an alternative to a sliding window, a two-parameter distribution such as a Γ or a log-normal distribution could be used for determining a reinsertion point. These distributions *could* be parameterized, such that their expectation value equals $\frac{v_{ab}}{v_{ac}}$. This would leave room for another tunable parameter that determines the variance (and thus boldness) of the respective distribution. However, tuning multi-parameter proposals is substantially more challenging than tuning single-parameter proposals. For multi-dimensional optimization a variation of the popular simplex algorithm [83] could be employed. A possible optimality criterion would be the distance to the target acceptance rate. However, acceptance rates for the same set of proposal parameters can be different depending on whether the chain is still in the burn-in phase or has already reached stationarity.

3.2.4 Tree Length Multiplier

Another proposal that is specific to branch lengths $\vec{v}$ is the tree length multiplier. As the name suggests, we scale all values in the branch length vector $\vec{v}$ by a multiplier m

that is determined as described in **Sect.** 3.2.2. The tree length multiplier has been proposed after observing that, specifically in partitioned datasets, BI yields branch lengths that exceed the ML estimates by more than one order of magnitude [19, 75]. In many instances, branch lengths from different chains did not converge to a common posterior distribution with respect to the PSRF (see **Sect.** 3.1.5), for instance.

For n branch lengths $\vec{v} = \{v_1, v_2, \ldots, v_n\}$, we can infer the Hastings-ratio analogously to the Hastings-ratio of the branch length multiplier (see **Sect.** 3.2.2). Every proposed branch length is determined as $v_i^* = h(u) \cdot v_i$. By definition, deriving v_i^* with respect to any branch length other than v_i yields 0. When we derive with respect to v_i, we obtain the multiplier $m = h(u)$. As in **Eq. 3.22**, for the reversal, we have $u^* = 1 - u$ and thus only the diagonal of the Jacobian is non-zero. Thus, the Hastings-ratio of a tree length multiplier for a tree with n branch lengths is

$$|J| = \left|\frac{\partial(\vec{x}^*, \vec{r}^*)}{\partial(\vec{x}, \vec{r})}\right| = \begin{array}{c|ccccc|} & \partial v_1^* & \partial v_2^* & \ldots & \partial v_n^* & \partial u^* \\ \partial v_1 & m & 0 & \ldots & 0 & 0 \\ \partial v_2 & 0 & m & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \partial v_n & 0 & 0 & \ldots & m & 0 \\ \partial u & \lambda h(u) v_1 & \lambda h(u) v_2 & \ldots & \lambda h(u) v_n & 1 \end{array} = m^n. \tag{3.28}$$

ExaBayes inherits the internal representation of branch lengths from the PLL. Since branch lengths are represented relative to the mean substitution rate, a proposal on $\vec{\pi}$ or $\vec{r}$ (resp., proposing a new fixed-rate AA matrix) also affects absolute branch lengths $\vec{v}$. Thus, slider and Dirichlet proposals on $\vec{\pi}$ and $\vec{r}$ are tree length proposals as well and thus the factor m^n is added to the Hastings-ratio of these proposals.

3.2.5 Dirichlet Proposals

As mentioned in **Sect.** 3.1.3, using a Dirichlet prior represents a common choice for multivariate distributions such as the posterior distributions that we obtain for parameters $\vec{r}$ and $\vec{\pi}$. Analogously, we can draw multiple values at random from a Dirichlet distribution. Assume a parameter vector $\vec{x} = \{x_1, \ldots, x_n\}$, such as for instance $\vec{\pi}$ or $\vec{r}$. As a proposal we can draw new values $\vec{x}^* = \{x_1^*, \ldots, x_n^*\}$ from a Dirichlet distribution that is based on the previous values $\vec{x}$. In other words, we draw $\vec{x}^*$ from a Dirichlet distribution that has density $f(x_1^*, \ldots, x_n^*; \alpha x_1, \ldots, \alpha x_n)$, where α is a tunable parameter of this proposal. We can draw the i-th value of $\vec{x^*}$, by first drawing from a distribution with density $\Gamma(\alpha x_i, 1)$ and then normalizing all values $\vec{x}^*$, such that $\sum_{i \in 1,\ldots,n} x_i^* = 1$. For large values of α, the proposal $\vec{x}^*$ is closer to the original values $\vec{x}$ (see **Fig. 3.3**). A natural advantage of the Dirichlet proposal are the simultaneous updates of all values $\vec{x^*}$. The Hastings-ratio is determined by the Dirichlet densities (see **Eq. 3.5**) of the proposal and reverse proposal

$$\frac{q(\vec{x} \mid \vec{x}^*)}{q(\vec{x}^* \mid \vec{x})} = \frac{f(x_1, \ldots, x_n; \alpha x_1^* \ldots \alpha x_n^*)}{f(x_1^*, \ldots, x_n^*; \alpha x_1, \ldots, \alpha x_n)}. \tag{3.29}$$

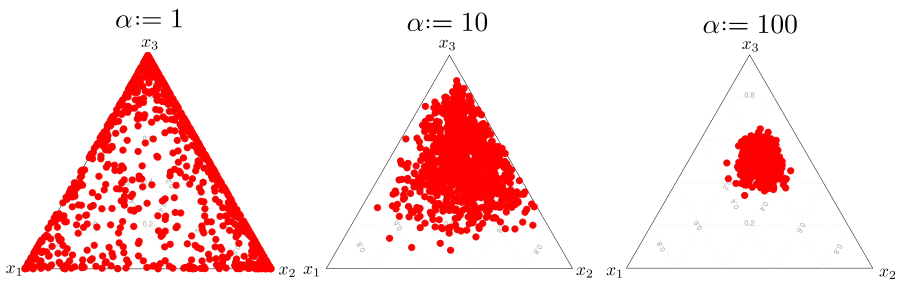


Figure 3.3: Ternary plot of 1,000 proposed values $\vec{x}^*$ drawn from a Dirichlet distribution that is based on original values $\vec{x} = \{0.2, 0.3, 0.5\}$ and has been scaled with an increasing value of α.

Dirichlet Proposals in ExaBayes

As discussed in **Sect.** 2.6.1, for likelihood computations, we have to represent substitution rates relative to one rate (which assumes a value of 1.0). In case of DNA data, this reference rate is typically defined to be r_{GT}. Furthermore, internally (in the numerical implementation of ExaBayes), the branch lengths are represented relative to the mean substitution rate (i.e., they depend only on $\vec{\pi}$ and $\vec{r}$ in the aforementioned representation). As a consequence, updating $\vec{\pi}$ and $\vec{r}$ also modifies the overall branch lengths (which needs to be considered in the Hastings-ratio and prior ratio of the affected proposals). Thus, proposals integrating over state frequencies or substitution rates effectively also integrate over the tree length. Using the proposal mixture as applied in MrBayes, we can observe that ExaBayes samples tree lengths much more efficiently than MrBayes. However $\vec{r}$ is typically not sampled sufficiently. Thus, in ExaBayes the weight ω_{TL} of the tree length multiplier proposal (see **Sect.** 3.2.4) is reduced by 50% and the weight ω_{sw} for the sliding window, respectively ω_{dir} of the Dirichlet proposals on substitution rates are doubled. While this allows for good overall sampling efficiency, the reference rate (in its relative form) is still sampled less efficiently in the worst case. This is because updates of the reference rate heavily affect the mean substitution rate and in turn lead to higher absolute values in the Hastings-ratio.

Modification for AA

We found that, even with tuning of proposal parameters, proposals usually applied to DNA GTR matrices perform sub-optimally on AA GTR matrices. In a AA GTR matrix, there are 189 substitution rates $\vec{r}$ to be estimated. Thus, a tuned Dirichlet proposal that modifies all values in one step, will only apply modest updates that do not change values substantially. On the other hand, a sliding window proposal that updates one or two rates can perform a bolder update. This however, only affects a small part of the matrix. As a compromise between the two alternatives, ExaBayes implements a Dirichlet proposal that modifies all substitution rates associated with a specific AA (e.g., $r_{A\cdot}$). This means, we only update 19 values at a time. The Hastings-ratio of this proposal directly follows from the Dirichlet densities of the affected rates.

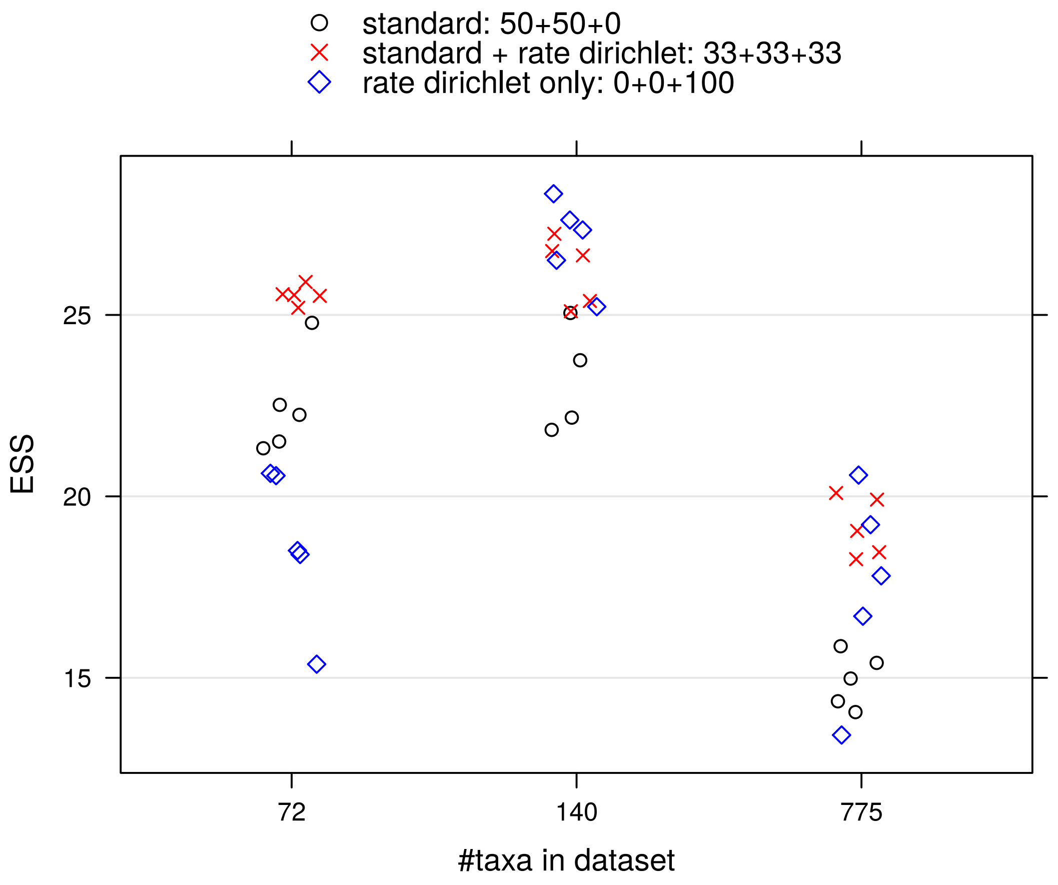


Figure 3.4: Sampling efficiency of AA GTR proposals: ESS averaged over all AA substitution rates for three different proposal mixtures. For each strategy the weight of proposals spent on (i) sliding window proposals, (ii) Dirichlet proposal and (iii) rate-specific Dirichlet proposals is given.

For three AA datasets (for details, see **Tab. 5.1**), we ran chains for 1,000,000 generations using three different proposal mixtures to evaluate this *rate-specific Dirichlet* proposal (5 chains per strategy). In ExaBayes, the relative proportion of updates on the substitution rate parameter amounts to 4.44%. In other words, the sum of weights ω_i of proposals that update the substitution rate parameter is 0.0444. Out of these 4.44%, the default proposal mixture (strategy 1) applies 50% sliding window and 50% Dirichlet proposals (also used as the default in MrBayes). For strategy 2, we added the rate-specific Dirichlet proposal and applied all three proposals (uniform slider, Dirichlet and rate-specific Dirichlet) with equal weight. In strategy 3, only the rate-specific Dirichlet proposal is applied.

As shown in **Fig. 3.4**, exclusive usage of the rate-specific Dirichlet proposal leads to increased sampling efficiency on 2 out of 3 datasets. However, it performs worse than the standard strategy on the 72-taxon dataset. Strategy 2, which uses the rate-specific Dirichlet proposal in addition to the default proposals leads to increased sampling efficiency on all three datasets. Thus, ExaBayes uses strategy 2 for integrating over the $\vec{r}$ parameter vector on AA GTR matrices.

3.3 Topological Proposals

Aside from uniform sampling across the protein model space (where we propose among the standard fixed-rate AA GTR models), the topology represents the only discrete parameter. In most cases, it is the primary parameter of interest. Proposals on topologies can be divided into two classes: (i) random perturbations of the topology and (ii) guided proposals. Guided proposals have to compute a guidance function that allows them to make an informed decision about the topology to propose.

3.3.1 Stochastic Topology Proposals

Stochastic Nearest Neighbor Interchange

A particularly simple and modest proposal is the stochastic nearest neighbor interchange (stNNI) [first discussed in 65]. Let b_{AB} denote the branch between nodes A and B as well as the subtree rooted at A (with B as sole descendant). For the stNNI, we first randomly draw an inner branch b_{EF} in τ, where E is an inner node with descendant subtrees A and B and F is an inner node with descendant subtrees C and D. We can create two alternative topologies around branch b_{EF}. We obtain the alternative topologies τ_1 and τ_2 by modifying the original branches $\mathcal{E}_\tau$ as follows

$$\begin{aligned}
\tau_1 &= \mathrm{NNI}(A, D, \tau) = \Big\{\mathcal{V}, \left(\mathcal{E}_\tau \setminus \{b_{AE}, b_{DF}\}\right) \cup \{b_{AF}, b_{DE}\}\Big\}, \\
\tau_2 &= \mathrm{NNI}(A, C, \tau) = \Big\{\mathcal{V}, \left(\mathcal{E}_\tau \setminus \{b_{AE}, b_{CF}\}\right) \cup \{b_{AF}, b_{CE}\}\Big\}.
\end{aligned}$$

Thus, we effectively obtain an interchange between subtrees A and either subtree C or subtree D (see **Fig. 3.5** for an equivalent representation). If the NNI operator is applied repeatedly to τ, we can transform τ into any tree τ'. Since there are three alternative topologies for each inner branch, the initial version of the stNNI proposal [65] proposed one of the three NNI alternatives uniformly and thus left τ unchanged in $\frac{1}{3}$ of all cases. Subsequently, the stNNI (as well as all stochastic proposals discussed in this Section) have been "metropolized". That means, we constrain the proposed topology τ^* to $\tau^* \neq \tau$. The Hastings-ratio of this proposal is 1. Note that, branch lengths previously associated with the root of a subtree (e.g., v_{AE}) remain attached to the subtree. Thus, for instance v_{AE} becomes v_{AF}.

Extending Subtree-Pruning and Regrafting

The extending subtree-pruning and regrafting (eSPR) [first discussed in 65] implements a specific subtree-pruning and regrafting (SPR) move: we pick an inner branch b_{BD} at random, where B is an inner node with descendants C and A. Then, we pick any branch $b_{EF} \neq b_{BD}$ (either node E or node F may be external nodes). We obtain τ' from τ after performing an SPR move as

$$\tau' = \text{SPR}(b_{BC}, b_{EF}, \tau) = \Big\{\mathcal{V}, \big(\mathcal{E}_\tau \setminus \{b_{AB}, b_{BD}, b_{EF}\}\big) \cup \{b_{AD}, b_{BE}, b_{BF}\}\Big\}. \tag{3.30}$$

Note that, if $b_{EF} = b_{DE}$ (i.e., b_{EF} is adjacent to b_{BD}), then $\text{SPR}(b_{BC}, b_{DE}, \tau) = \text{NNI}(C, E, \tau)$. In other words, if the pruned subtree only traverses one subtree, then there exists an equivalent NNI and therefore also an equivalent stNNI move. For reattaching the pruned subtree, there exist $2 \cdot n' - 3$ possibilities, where n' is the number of external nodes in the remaining tree, after pruning (see **Fig. 3.5** for illustration). A completely random SPR proposal was found to be too bold [65] (i.e., acceptance probability is low). Instead the eSPR proposal prunes a subtree and then randomly descends into the subtrees adjacent to the pruning position (here A and D). At each descent point, we draw a random number r from a uniform distribution $[0, 1)$ and compare it to a stopping probability p_s. If $r < p_s$, then the traversal stops and the current branch is chosen as reattachment location b_{EF}. However, if $r \geq p_s$ the traversal continues and the subtree descends randomly in either subtree of the current branch (but not the branch it previously visited – except external branches are encountered in ExaBayes as explained later). Thus, the number of branches that is traversed by the pruned subtree before it is inserted follows a geometric distribution. If in MrBayes the subtree traverses an external branch, the traversal is stopped immediately without the test against p_s and the external branch is chosen as reattachment location. It is easy to see that, the probability of doing such a constrained move (terminating in an external branch) is not the same as the probability of the reverse move. Since here we have one test less in the forward move compared to the reverse move, the Hastings-ratio is p_s. The Hastings-ratio is p_s^{-1} if the backward move is constrained (i.e., terminates at an external branch), while the forward move is unconstrained. If none of the moves is constrained, the Hastings-ratio is 1.0.

In contrast to MrBayes, we allow the subtree to continue its traversal in case an external branch is encountered in ExaBayes. In other words, at outer branches we also apply the test against the stopping probability. If the traversal continues, then we decide at random whether the subtree either traverses the yet unvisited neighboring subtree of the current external branch or whether it traverses back to the branch from which it arrived. Since the number of traversals (i.e., number of possible paths) for the forward SPR move are the same as for the reverse SPR move, the Hastings-ratio of the eSPR implemented in ExaBayes is always 1. A Hastings-ratio of 1 is advantageous in this case, since there is no biological motivation to bias topological rearrangements towards reattachment of subtrees at external branches.

In an eSPR, the previously existing branch length v_{BD} is mapped to b_{Bk}, where k is the last node that has been traversed by the subtree (either E or F for the notation given above). Apart from that, all branch lengths remain unchanged at their associated subtrees (e.g., b_{AD} inherits branch length v_{AB}).

Extending Tree-Bisection and Reconnection

The eTBR move is the boldest among the three stochastic topological proposals. A tree-bisection and reconnection (TBR) operation is equivalent to performing two SPR operations using the same initial branch b_{AB}. Thus, in the eTBR, we first determine an initial inner branch b_{AB} and perform an eSPR move where the subtree b_{AB} is pruned, traverses adjacent branches and is reinserted according to the stopping criterion p_s into some branch b_{CD}. Then, we prune the subtree b_{BA}, perform the same procedure and reinsert into a branch b_{EF}. Note that, for sufficiently small p_s, the eTBR can essentially re-connect any two branches from the disconnected components of the graph that originated from the tree after bisection. In other words, the number of TBR moves for each inner branch is in $\mathcal{O}(n^2)$, where n is the number of taxa (see **Fig. 3.5** for an illustration). We can formally define the TBR move as:

$$\mathrm{TBR}(b_{AB}, b_{CD}, b_{EF}) = \mathrm{SPR}\left(b_{BA}, b_{EF}, \mathrm{SPR}(b_{AB}, b_{CD}, \tau)\right). \tag{3.31}$$

The Hastings-ratio of the eTBR directly follows from **Eq. 3.31**. Thus, in ExaBayes, the eTBR has a Hastings-ratio of 1 as well. Furthermore, when an initial inner branch b_{AB} is chosen, that only has outer nodes as descendants of either A or B, the eTBR automatically degenerates into a eSPR or stNNI move. The probability of an eTBR to degenerate is huge, since by definition $> 50\%$ of all nodes are outer nodes. Up until MrBayes version 3.1, the eTBR was the main topological proposal (complemented by the Local move, see **Sect.** 3.2.3). Since, in practice, the acceptance rate of the eTBR consistently is about 50% of the acceptance probability of the eSPR proposal, we can assume, given the above argument about $> 50\%$ being outer nodes, that the eTBR proposal rarely performs an actual non-SPR move. Thus, the eTBR proposal is not used by default in ExaBayes as of version 1.4.

3.3.2 Guided SPR Proposals

The stochastic proposals discussed in **Sect.** 3.3.1 typically favor local rearrangements and thus are likely to propose alternative topologies that are close (w.r.t. the RF-distance) to the original topology. Yet, acceptance rates of topological proposals are

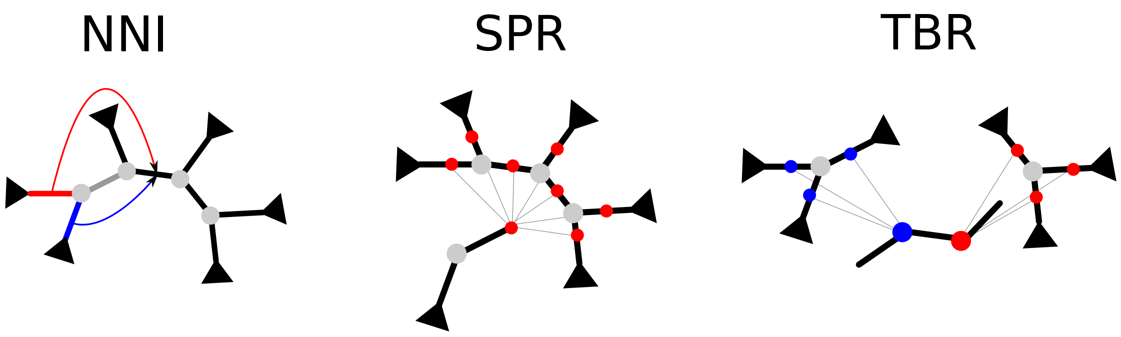


Figure 3.5: Three elementary operators for proposing topologies. In the NNI (*left,*) there are 2 alternatives for generating a new tree: either the red or the blue subtree interchanges with its neighbor. In case of the SPR (*middle*), a pruned subtree B can be inserted into 6 out of 7 positions to yield a tree topology that is distinct from the original tree. For the TBR (*right*), the tree is bisected between two nodes C and D. There are $3 \times 3 - 1$ possibilities to reconnect the tree that result in a new topology.

still notoriously low, since a large proportion of proposals has a negative impact on the likelihood ratio (given the immense number of trees). MrBayes, version 3.2, introduced a SPR proposal that uses parsimony scores in order to derive a proposal density for τ [first described in 65]. For the parsimony-guided subtree-pruning and regrafting (parsSPR) move, we determine a random subtree b_{AB} to be pruned and reinsert it into all possible reattachment positions and calculate the parsimony score of this topology. Assume, the alignment $\mathcal{A}$ is partitioned into $P_1, P_2, \ldots, P_k$, where partition P_i has $s_i = |\mathcal{S}_i|$ states (4 for DNA, 20 for AA and 2 for morphological data). Let $\text{Pars}(P_i, \tau)$ denote the parsimony score for partition P_i and tree τ. Then in MrBayes, we derive a score

$$\text{SP}(\tau \mid \mathcal{A}) = -\sum_{i=1}^{k} \text{Pars}(P_i, \tau) \cdot w \cdot \log\left(s_i^{-1} \left(1 - \exp(\frac{-s_i}{s_i - 1} \cdot v_{\text{typ}}) \right) \right), \qquad (3.32)$$

where v_{typ} represents a typical branch length (of 0.05), where the logarithmic factor allows to combine the parsimony score across partitions with different data types and w is a heating factor that determines how strongly differences in parsimony score affect the score SP. Let $\vec{\tau}$ be all trees that originate from regrafting the pruned subtree. We then obtain the proposal density as

$$\begin{aligned} S(\tau^* \mid \mathcal{A}) &= \exp\left(\min_{\tau' \in \vec{\tau}} \left\{ \text{SP}(\tau' \mid \mathcal{A}) - \text{SP}(\tau^* \mid \mathcal{A}) \right\} \right), \\ q(\tau^* \mid \tau) &= \frac{S(\tau^* \mid \mathcal{A})}{\sum\limits_{\tau' \in \vec{\tau}} S(\tau' \mid \mathcal{A})}. \end{aligned}$$

The Hastings-ratio of the parsSPR directly follows from $q(\tau^* \mid \tau)$ and $q(\tau \mid \tau^*)$. Finally, MrBayes implements a random reweighting of character weights similar to the parsimony ratchet (not implemented in ExaBayes).

Several alternative guidance functions for proposing topologies in an informed manner are possible. For instance, the posterior-guided subtree-pruning and regrafting (ppSPR) proposal [52] directly uses the PP for deriving a proposal density (i.e., $S(\tau^* \mid \mathcal{A})$ is replaced by the posterior density of τ^* given $\mathcal{A}$ while all remaining model parameters are kept fixed). This means that a substantial number of PLF evaluations is necessary for merely proposing a single novel tree topology. After the burn-in, obtaining a large RF-distance between current topology τ and proposed topology τ^* via a topological proposal becomes unlikely. Thus, we typically only consider reattachment positions that are within a radius of r of the pruning position in the guidance function of the ppSPR. The proposal parameter r is either provided by the user or set to $\lceil \frac{\log(n)}{log(2)} \rceil$ by default, where n is the number of taxa. In ExaBayes such a radius parameter is offered for both the ppSPR as well as the parsSPR moves. The guided SPR proposals, always and often substantially outperform the acceptance ratio of the eSPR (for details, see **Sect.** 4.3.3).

3.4 Metropolis-Coupled Markov Chain Monte Carlo

While the MH algorithm guarantees that the stationary distribution simulated via a chain will approximate the posterior distribution after an infinite number of generations, the practical challenge of MCMC is to maximize mixing efficiency (i.e., to minimize the time to apparent convergence). The mixing efficiency depends on prior choice, signal in the data, proposal design, and the initial values of parameters of the Markov chain. A generalized approach that potentially improves the mixing of a chain is Metropolis-coupled MCMC (MC^3) [42].

In MC^3, we simulate n chains that sample from *heated* posterior distributions $\Pr[\Theta|\mathcal{A}]^{\beta_i}$, where the posterior of the i-th chain is heated by a chain-specific factor β_i with $\beta_i \leq 1$. Note that, $\beta_1 := 1$, thus chain 1 samples from an unmodified posterior and is referred to as the *cold* chain. Phylogenetic BI software packages (such as MrBayes or ExaBayes) typically use a static heat ladder in which β_i is given by **Eq. 3.33** and $\delta > 0$ determines by which factor the heat increases among two chains with adjacent heat increments β_i and β_{i+1}. Dynamic tuning of β_i can potentially further increase mixing efficiency [11]. If we modify **Eq. 3.4** to **Eq. 3.34**, we ensure that the i-th chain samples from posterior $\Pr[\Theta \mid \mathcal{A}]^{\beta_i}$. The central purpose of MC^3 is to allow two chains i and j to swap their states Θ_i and Θ_j with probability $\alpha_{\text{swap}}(i,j)$ as described in **Eq. 3.35**. In simple terms, a colder chain i always swaps with a hotter chain j, if the hotter chain is in a region of higher PP. Otherwise, we swap proportionally to $\alpha_{\text{swap}}(i,j)$ and the swapping probability increases for similar β_i and β_j, respectively similar unnormalized posterior densities $f(\Theta_i \mid \mathcal{A})$ and $f(\Theta_j \mid \mathcal{A})$. Thus, after burn-in, a swap most likely occurs between two adjacent chains with an adjacent heat factor β_i and β_{i+1}. This scheme allows the cold chain to leave local optima of regionally high PP.

$$\beta_i = \frac{1}{1+\delta(i-1)}, \tag{3.33}$$

$$\xi_i(\Theta^* \mid \Theta) = \min\left(1, \left(\frac{f(\Theta^*)\cdot f(\mathcal{A}\mid\Theta^*)}{f(\Theta)\cdot f(\mathcal{A}\mid\Theta)}\right)^{\beta_i} \times \frac{q(\Theta\mid\Theta^*)}{q(\Theta^*\mid\Theta)}\right), \tag{3.34}$$

$$\xi_{\text{swap}}(i,j) = \min\left(1, \frac{f(\Theta_i)^{\beta_j}\cdot f(\Theta_j)^{\beta_j}}{f(\Theta_j)^{\beta_j}\cdot f(\Theta_i)^{\beta_i}}\right). \tag{3.35}$$

The integration of a swap proposal into the MCMC framework is straight-forward: for instance, in MrBayes, we allow each chain to proceed by k generations and then propose l swaps between two chains that are randomly chosen for each swap. In ExaBayes, instead of a fixed number of swap attempts, that takes place after a fixed number of generations, we draw the number l of swap attempts from a binomial distribution: for a user-specified expected number of swaps per generation μ and a number of coupled chains c, we draw the number l of swaps we want to try from a binomial distribution

$$l := \begin{cases} \text{Bin}(c, \frac{\mu}{c}), & \text{if } \mu < 1; \\ \text{Bin}(2c\mu, \frac{1}{2c}), & \text{else.} \end{cases} \tag{3.36}$$

This scheme allows for a higher randomness when swap attempts are proposed. This random procedure increases the chance that there is a consecutive number of generations in which no swap attempt is scheduled (this will be exploited later for the parallelization of MC^3, see **Sect.** 5.2.5). Parameter μ allows the user to specify an expected number of swaps.

In practice, a study showed that MC^3 improves convergence, when the posterior density exhibits multiple distinct peaks [125]. BI in phylogenetics has been severely criticized, since we can construct a class of phylogenetic problems, where a single chain takes an exponential number of generations to converge between mixtures of trees [78] that have a high RF-distance among each other. Here, the application of MC^3 resolved the concerns raised by the introduction of this problem instance [96].

3.5 Summary

This Chapter offered a detailed introduction to and description of BI in phylogenetics. For integrating over continuous parameters, we described sliders, Dirichlet proposals, multipliers, the node slider as well as a tree length multiplier (whereas the latter two are only applicable to branch lengths). By applying Green's method, we demonstrated how the Hastings-ratios of the multiplier, the node slider, and the tree length proposal can be derived. To the best of our knowledge, this represents the first correct formal derivation of the Hastings-ratio of the node slider (since up until MrBayes 3.2 the Hastings-correction of the node slider was implemented incorrectly). We also introduced a rate-specific Dirichlet proposal for the integration of substitution rates in the high-dimensional AA GTR matrix. We determined that, for integrating AA substitution rates, a mixture of slider, Dirichlet, and partial Dirichlet proposal performs best. Furthermore, ExaBayes-specific modifications have been discussed that are a consequence of the internal branch length representation in the PLL. We explained simple stochastic proposals for updating the topology parameter that are employed in ExaBayes and showed the transitivity of these proposals. We explained a variation of the eSPR and eTBR as implemented in ExaBayes that have a simple Hastings-ratio of 1.0, which is advantageous for the acceptance probability of the proposal. Next, we discussed the variants of guided topological proposals that are implemented in ExaBayes. To the best of our knowledge, the ppSPR is the first production-level posterior guided proposal for trees without molecular clock that is employed *by default* in a phylogenetic BI software. Finally, we introduced a variation of the MC^3 algorithm that is implemented in ExaBayes and can yield higher parallel efficiency.

4 Advanced Proposals On Branch Lengths and Topology

This Chapter exclusively features original work by Andre Aberer, if not explicitly stated otherwise. This Chapter is based on the following publication:

- AJ **Aberer**, A Stamatakis, and F Ronquist. "An Efficient Independence Sampler for Updating Branches in Bayesian Markov chain Monte Carlo Sampling of Phylogenetic Trees." In: *Systematic biology* 65.1 (2016), pp. 161–176

Contributions: All contributions by Andre Aberer, except for the empirical version of **Eq. 4.2** (derived by Fredrik Ronquist).

The central aspect of this Chapter is the development of a novel approach that allows proposing branch lengths from scratch with a particularly high probability of acceptance. The developed proposal has the atypical, yet highly desirable quality that it yields a branch length update without reliance on the current branch length value. Thus, the proposal is a so-called *independence sampler*[120]. Throughout this Section we refer to the *branch length posterior* for brevity what is technically the conditional posterior probability of a focal branch length given that all remaining parameters are kept fix.

Initially, in **Sect.** 4.1 we characterize posterior distributions of branch lengths empirically on real data. Subsequently, we assess to which extent parametric distributions can be fitted to the observed posteriors. To achieve this, we define a measure that allows to estimate the expected acceptance probability of a proposal that is based on a parametric distribution. In **Sect.** 4.2, we adapt an existing optimization method for optimizing branch length posteriors and derive proposals with the desired properties. Finally, in **Sect.** 4.3, we combine the novel branch length proposal with several traditional topological proposals (that have been established in **Sect.** 3.3). We initially assess how branch length posteriors change when the topology is modified. Based upon this, we construct and evaluate hybrid proposals for proposing branch lengths and topology simultaneously.

4.1 Examination of Branch Length Posteriors

4.1.1 Data and Reference Runs

To analyze branch length posteriors and topological proposals, we compiled a representative set of 15 publicly available empirical datasets that have been previously used for assessing properties of topological proposals [65] or for assessing the convergence of bootstrap support values [86] (see **Tab. 4.1** for an overview of basic statistics extracted from the MSAs). The number of taxa in the datasets ranges between 24 and 500. While some of the datasets are concatenated super-matrices, for the sake of simplicity, we carried out all analyses on unpartitioned alignments (i.e., all parameters are linked across all partitions). Except for **dat-140** which consists of AA data, all datasets are DNA alignments. With 404 and 500 taxa, **dat-404** and **dat-500** induce large phylogenies that are hard to resolve. The "character to taxon" ratio is highest in **dat-125** (with close to 30,000 characters), **dat-24** and **dat-404** (both with more than 10,000 characters) and particularly low for **dat-354**.

We require robust reference runs for comparing the result of chains against the assumed true posterior probabilities of splits. Thus, we ran 10 independent chains for 10^7 generations using ExaBayes (version 1.3, see **Tab. 4.1** for the number of trees in the 50% credible set and the final ASDSF of all 10 chains). With an ASDSF of 1.25%, dataset **dat-27** does not satisfy a stringent convergence threshold of 1%. Since for the reference estimate of the posterior, we combine the samples from all 10 chains, we can assume that we obtain a sufficiently accurate sample of the posterior simply because we have 10× more samples than if we only considered samples from a single chain.

Despite the long chain length, split frequencies did not converge for datasets **dat-150** (ASDSF= 5.98%), **dat-404** (ASDSF = 10.58%) and **dat-500** (ASDSF = 3.653%, we obtained similar results with MrBayes). From bootstrapping under the ML criterion, we know that these datasets likely contain rogue taxa (see **Sect.** 6.4.2). Similarly, we observed in BI, that for a subset of taxa there are several possible locations in the tree with similar likelihoods. We nonetheless include these problematic datasets for the following experiments to avoid a bias towards overly simple datasets. We eventually obtained an acceptable degree of convergence for these datasets by only using ppSPRs and parsSPRs as topological proposals and using the branch length proposal presented in this Chapter. Dataset **dat-404** still has an ASDSF > 1%, despite the fact that, we only considered 8 chains, since 2 chains became trapped in regions of low PP.

4.1.2 Expected Acceptance Probability

Assume, we want to assess the capability of a proposal density function $q(x^* \mid x)$ to approximate a marginal posterior density $\varphi(x) = \int f(x, \Theta \mid \mathcal{A}) \; \partial\Theta$, where x is the parameter of interest, f is the PP and Θ is a vector of parameters excluding x. According to the MH algorithm, the acceptance probability is given by **Eq. 3.4** as $\xi(x^* \mid x)$.

We can determine the expected acceptance probability (EAP) $\hat{\xi}(q \mid \varphi)$ of a proposal density by averaging over all acceptance probabilities $\xi(x^* \mid x)$ weighted by the posterior

id	*#taxa*	*#characters*	*type*	*# unique patterns*	*gaps [%]*	*fit(b)*	*fit(c)*	*asdsf [%]*	*nucl. div.*	*#trees [50%]*	*#trees [99%]*
dat-24	24	14190	DNA	4600	0.26	3.43	-0.496	0.574	0.144	1	3
dat-27	27	1949	DNA	934	20.05	1.08	-0.473	1.25	0.206	120	20,037
dat-36	36	1812	DNA	1020	0.02	1.99	-0.490	0.108	0.231	7	1,889
dat-41	41	1137	DNA	768	10.79	1.51	-0.490	0.732	0.213	481	28,766
dat-43	43	1660	DNA	954	11.03	1.49	-0.488	0.447	0.197	45	8,112
dat-50	50	1133	DNA	489	9.33	1.56	-0.460	0.414	0.109	62,435	16,0440
dat-59	59	1824	DNA	1037	0.00	2.72	-0.491	0.190	0.213	2,016	50,074
dat-64	64	1008	DNA	406	21.21	1.69	-0.473	0.347	0.280	89,750	187,755
dat-71	71	1082	DNA	445	35.98	1.59	-0.462	0.355	0.125	100,005	198,010
dat-125	125	29149	DNA	19436	32.72	2.02	-0.499	0.0534	0.357	1	9
dat-140	140	1104	AA	1041	0.60	1.25	-0.494	0.547	0.456	245	20,396
dat-150	150	1269	DNA	1130	4.77	1.61	-0.486	0.99	0.220	100,005	198,010
dat-354	354	460	DNA	348	14.71	0.95	-0.380	0.899	0.0871	100,005	198,010
dat-404	404	13158	DNA	7429	78.92	1.54	-0.486	2.50	0.299	87,204	172,664
dat-500	500	1398	DNA	1193	2.48	1.35	-0.475	0.829	0.131	100,005	198,010

Table 4.1: Datasets used for analyses along with characterizing statistics. Labels *fit(b)* and *fit(c)* refer to fit parameters of the regression model fitting the negative value of the second derivative of the log-posterior to the standard deviation; *asdsf* denotes the ASDSF of split frequencies among 10 reference runs (8 in case of **dat-404**); *nucl. div.* denotes the nucleotide diversity in the alignment and *#trees[X%]* denotes the number of trees in the *X*-% credible tree set for reference runs.

density $\varphi(x)$ and the proposal density $q(x^* \mid x)$:

$$\hat{\xi}(q \mid \varphi) = \iint \varphi(x) \cdot q(x^* \mid x) \cdot \xi(x^* \mid x) \, \partial x^* \, \partial x. \tag{4.1}$$

In our case, we want to assess the densities of given distributions. This means that our proposals do not depend on the previous state x and we can simplify $q(x^* \mid x) = q(x^*)$. A branch length is a continuous random variable for which a closed form of the integral does not exist. Instead, we have to obtain an estimate of the posterior density $\hat{\varphi}(x)$ that is discretized into an arbitrarily chosen number of n bins. Consequently, we have to discretize the proposal density $q(x^*)$ as well and thus can reformulate **Eq. 4.1** as

$$\hat{\xi}(q \mid \hat{\varphi}) = \sum_{i=1}^{n} \sum_{j=1}^{n} \hat{\varphi}(x_i) \cdot q(x_j) \cdot \xi(x_i \mid x_j). \tag{4.2}$$

Naturally, the ability of $\hat{\xi}(q \mid \hat{\varphi})$ to accurately predict the acceptance probability of a proposal density $q(x)$ with respect to $\varphi(x)$ depends on the number of samples used for estimating $\hat{\varphi}(x)$ and the number of bins n for the discretization of $\hat{\varphi}(x)$.

We can assess the accuracy of **Eq. 4.2** by applying MCMC with a known posterior and proposal density and subsequently measuring the observed acceptance probability (OAP). Values determined via calculation of $\hat{\xi}(q \mid \hat{\varphi})$ are within the expected deviation. For instance, consider two Γ distributions $\Gamma(15,4)$ and $\Gamma(8,2)$ (see **Fig. 4.1a**) and two normal distributions $\mathcal{N}(6,2)$ and $\mathcal{N}(8,2)$ (see **Fig. 4.1b**). For the two Γ distributions, the EAP measure predicts an acceptance ratio of 78.17% (with 10^6 samples and 50 bins), whereas empirical verification via MCMC yields an acceptance probability of 78.04% (10^6 generations). The same experimental setup for the two normal distributions with equal variance but shifted mean yields an EAP of 48.17% and a verified empirical acceptance probability of 47.91%. Typically, the EAP slightly over-estimates the true acceptance probability, a fact that most likely is owed to the binning.

As an alternative, we can construct the *continuous overlap* of two density functions as the integral of the minimum of either function divided by the integral of the maximum of either function. However, despite being intuitive, this continuous overlap of proposal density and posterior distribution can not be used as a reliable proxy for estimating the acceptance probability. Instead, the acceptance probability of a proposal decreases superlinearly with the overlap of distributions. We define the *histogram overlap* of two sample populations of equal size as the sum of the minimum counts of either population in every bin divided by the total number of samples in a population. Thus, the histogram overlap of samples drawn from the two Γ distributions (**Fig. 4.1a**) is 83.42% (with an acceptance ratio of $\sim$ 78%) and for the two normal distributions (**Fig. 4.1b**) we obtain a histogram overlap of 61.70% (the acceptance ratio in this case is $\sim$ 48%). This illustrates that if we want to use a proposal density $q(x)$ to propose values for the posterior φ, then $q(x)$ has to be as similar to φ as possible in order to be effective.

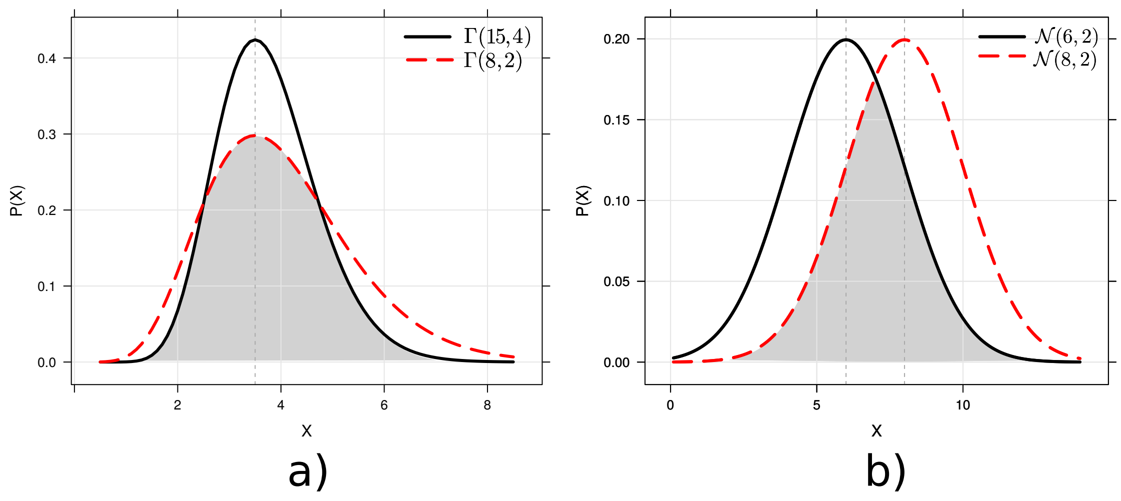


Figure 4.1: Approximation of posterior distributions using a proposal density using two Γ distributions in *a)* and two normal distributions in *b)*. Gray area indicates overlap (respectively the histogram overlap of empirical data).

4.1.3 Estimation of Posteriors

For obtaining a representative set of empirical branch length posteriors, we ran a chain on each of the datasets listed in **Tab. 4.1**. Depending on the dataset, we set the chain length to a number of generations between 50,000 and 100,000 to assure that the chain had reached a state that is typical for the stationary distribution. After this initial phase, we saved the current parameter state of the chain. We then used this state as a starting point for new chains that sample the conditional posterior of a single branch length (one branch length per chain at a time). For these secondary chains, the only proposal employed was the branch length multiplier (see **Sect.** 3.2.2). We extracted a sample from the chain every 10 generations and continued sampling a branch until we achieved a highly accurate ESS of at least 10,000. We specified an exponential prior with parameter $\lambda = 10$ for this and all following experiments within this Section. We occasionally verified that results did not change under a uniform prior. For runs on DNA alignments, we used the GTR+Γ model [115, 128] and for AA datasets, we integrated over all fixed-rate models available in ExaBayes.

Fig. 4.3 displays 4 typical branch length posteriors. For comparably short branch lengths (see **Fig. 4.3a**, where mean $\mu = 0.00173$), we observe an exponential-like distribution, while branch length posteriors with a comparably high average as in **Fig. 4.3d** (μ=1.06) resemble a bell-shaped distribution. Distributions between these two extremes usually have a positive/right skew (i.e., according to the non-parametric definition the mean of the distribution is greater than its median). Associated log-posterior density curves usually have a short left tail and a long right tail (see **Fig. 4.2**). In case of bell-shaped distributions, this does not affect the effectively sampled posterior distribution. This is because the probability of excessively long branches becomes too small for being sampled. In case of **Fig. 4.3a**, the logarithmic posterior density is linear, which from

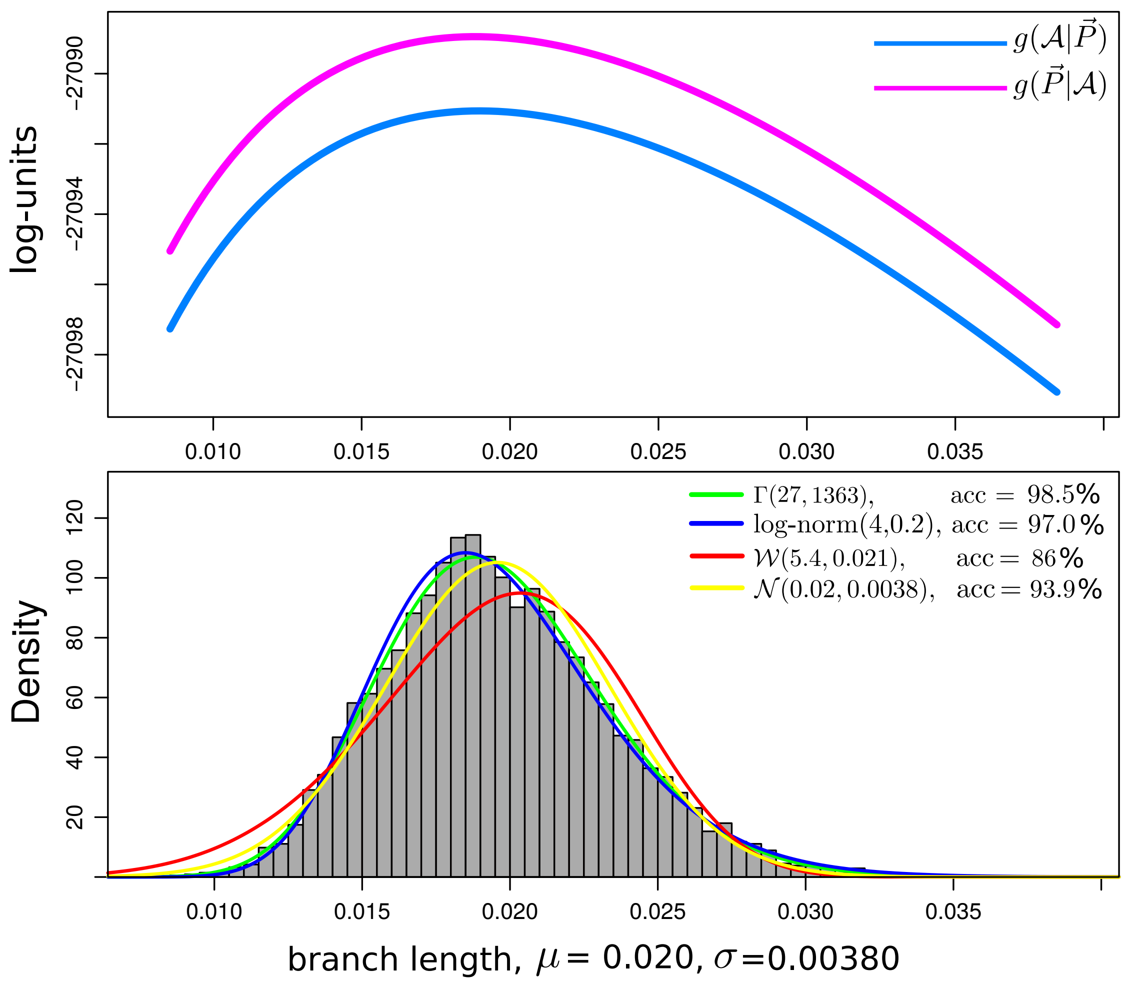


Figure 4.2: *Top:* Logarithmic posterior density $g(\Theta \,|\, \mathcal{A})$ and logarithmic likelihood $g(\mathcal{A} \,|\, \Theta)$ for one branch length (x-axis). *Bottom:* Observed posterior $\hat{\varphi}$ (as histogram) along with ML fits of 4 different distributions and their respective EAP.

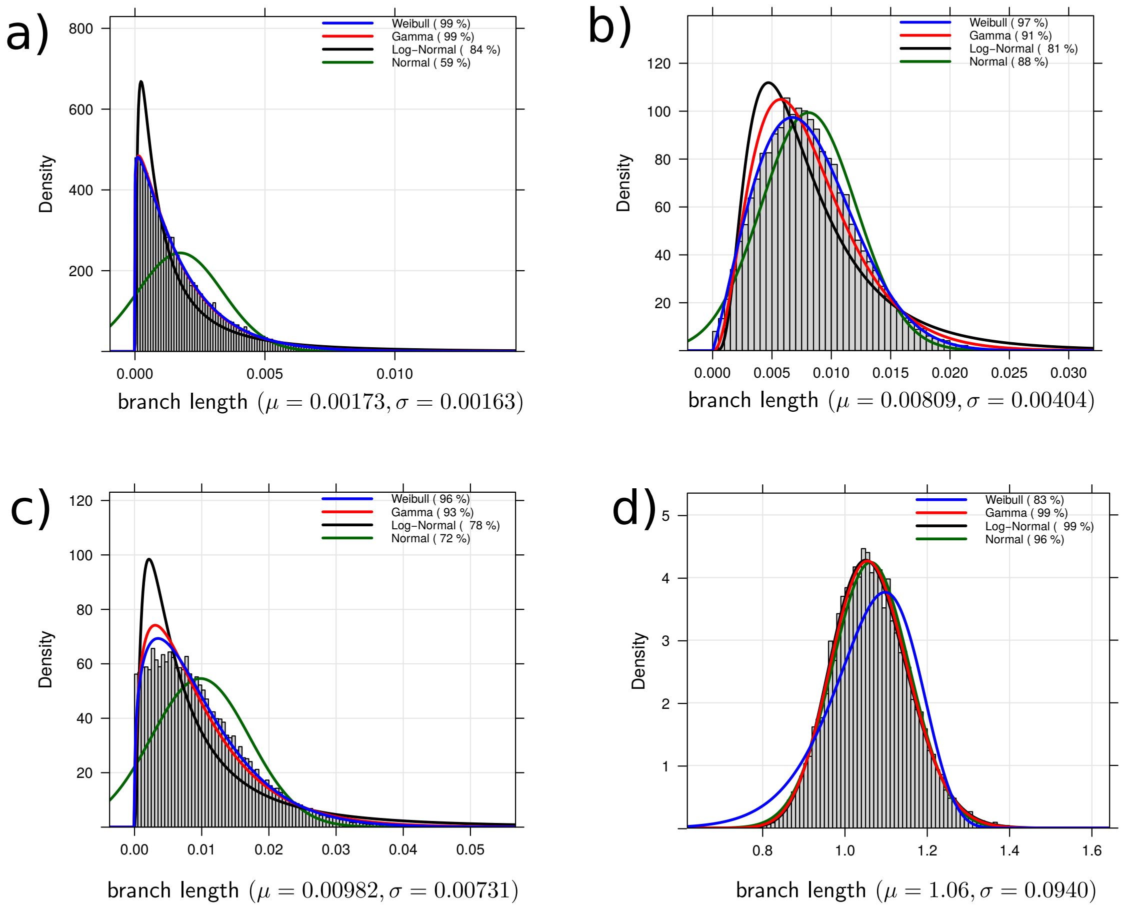


Figure 4.3: Typical branch length posteriors with mean μ and standard deviation σ and ML fits for 4 different probability densities under examination. Percentages in brackets indicate the EAP for proposing on the given posterior using the respective density function. Posteriors are given for a) an outer branch length from dataset **dat-27**, b) an inner branch of **dat-59**, c) an inner branch length of **dat-500** and d) an outer branch length of **dat-354**.

a phylogenetic point of view indicates that the most probable event is that no substitutions have occurred between two nodes in a tree. The sampling accuracy for branch lengths close to 0 is limited by the numerical stability of the underlying software. Thus, the PLL prohibits values that become too small relative to the mean substitution rate. On most datasets, this corresponds to minimum values in the range of $\approx 10^{-6} - 10^{-7}$.

While exponential-like distributions only occurred for distributions with the shortest mean within a dataset, we noticed that the transition from an exponential to a bell shape is dataset-specific. This transition is well-characterized by the skewness, defined as $\frac{\mu_3}{\sigma^3}$, where μ_3 is the third standardized moment and σ is the standard deviation. Across all datasets, the skewness is $\in [0.0187, 2.16]$. Thus, the skewness was positive for all 4,019 branches under examination. We repeated the experiment for selected datasets under a uniform prior for branch lengths and successfully verified that the exclusively positive skew is not a mere consequence of the positively skewed exponential prior.

While **dat-24**, **dat-36**, **dat-125** and **dat-140** almost exclusively exhibited weakly skewed distributions, we observed a particular abundance of strongly skewed distributions in **dat-71** and **dat-354**. Considering the number of trees in the 50% or 90% credible set (see **Tab. 4.1**), this suggests that a high number of strongly skewed (i.e., exponential-like) branch length posteriors is characteristic for a high degree of uncertainty in the tree topology (as represented by large numbers of trees in the credible set). In other words, if the most probable event is no substitution on a branch, we have weak to non-existing evidence for the phylogenetic relationship represented by the branch and thus many likely outcomes.

Finally, we observed a linear relationship between skewness and CV (defined as CV $= \frac{\sigma}{\mu}$) of the posteriors (see **Fig. 4.4**). In other words, weakly skewed distributions deviate less from the mean than highly skewed distributions.

4.1.4 Appropriateness of Fitted Distributions

For finding a distribution that is suitable as a proposal density for a guided branch length proposal, we fitted various distributions to the observed branch length posteriors $\hat{\varphi}$. For continuous distributions in the interval $[0, \infty)$ that exhibit a positive skew, the *Weibull* distribution $\mathcal{W}(\lambda, k)$, the *log-normal* distribution log-norm(μ, σ) and the gamma distribution $\Gamma(\alpha, \beta)$ are common choices. All three distributions depend on 2 parameters. The exponential distribution Exp(λ) is a special case of the Γ distribution (with $\beta := \lambda$) and a special case of the Weibull distribution (with $k := \lambda$). All of them are special cases of the generalized Γ distribution, which however was not considered because of its dependence on three parameters. Because of the bell shape of many of the distributions, we also considered the normal distribution $\mathcal{N}(\mu, \sigma)$ as a candidate for a proposal function.

We employed the R packages MASS [121] and FITDISTR [26] for obtaining ML estimates of parameters fitted to branch length posteriors. While closed formulas exist for the normal and log-normal distribution, parameters are numerically optimized for the Γ and Weibull distribution. We used the EAP to determine the appropriateness of a fitted distribution as a proposal density function, if applied to the observed branch length posterior $\hat{\varphi}$. **Fig. 4.3** depicts the four fitted distributions for 4 representative branch length posteriors.

For exponential-like distributions (see **Fig. 4.3a**), we get a close to optimal EAP for the Γ and Weibull distributions that essentially degenerate to an identical, almost exponential distribution ($\alpha := 1.08$ for the example in **Fig. 4.3a**). As expected, the log-normal distribution achieves a much lower EAP of 84% and a proposal based on a fitted normal distribution would only be accepted in 59% of all cases. In contrast, for bell-shaped posteriors with a high mean (see **Fig. 4.3d**), the Γ and log-normal distribution perfectly match the observed posterior, whereas the normal distribution only performs 3% worse than Γ or the log-normal distribution. Here, the Weibull distribution exhibits a poor fit and would only be accepted in 83% of all cases. **Fig. 4.3b** displays a posterior for which the Weibull distribution outperforms any competing distribution with $\approx 97\%$ EAP compared to – for instance — 91% for a Γ distribution. Also, for the distribution

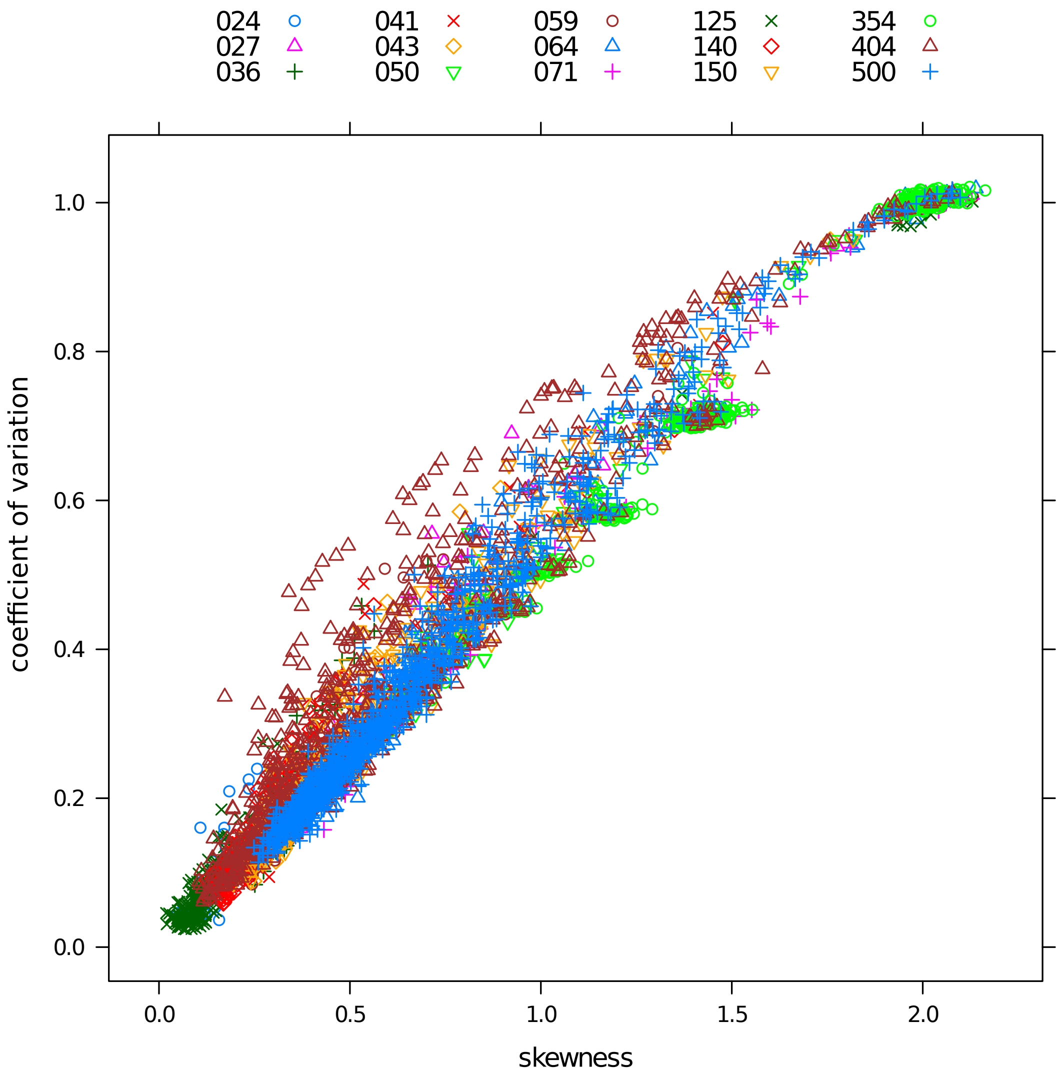


Figure 4.4: Skewness and CV of all observed branch length posteriors sampled from any of the 15 datasets.

depicted in **Fig. 4.3c**, the Weibull distribution yields a better proposal density than the Γ distribution by a small margin of 3%. It is notable, that the branch of **Fig. 4.3c** is an inner branch of the hard-to-resolve dataset **dat-500**. Despite the substantial number of underlying samples (ESS $> 10,000$), none of the distributions under examination represent an optimal match and the region around the mode of the distribution is not well-shaped.

Fig. 4.5 depicts the EAP for the distributions fitted to branch length posteriors across all 15 datasets against the type of distribution. We here use the CV for characterizing the distribution. Given the linear relationship we observed between skewness and CV, this means that, exponential-like posteriors generally have a CV close to 1, whereas bell-shaped posteriors have a CV close to 0. **Fig. 4.5** confirms our observations from **Fig. 4.3**: the normal distribution generally exhibits a good fit for posteriors with small CVs, but becomes inefficient (with a EAP of less than 70%) for a CV close to 1. The same holds for the log-normal distribution, which however still fits better to most posteriors with intermediate skewness, respectively CV. For the vast majority of posteriors, the Γ distribution has close to optimal ($> 95\%$) EAP values. However, in the CV range between 0.5 and 1.0 (i.e., strongly skewed or exponential-like), there exists a group of outliers for which the Γ distribution merely achieves $85\% - 95\%$ EAP. In contrast, the Weibull distribution shows optimal performance for strongly skewed branch length posteriors. In particular, it has a higher EAP for the aforementioned outlier group. For bell-shaped distributions, however, the performance of the Weibull distribution consistently decreases to $80\% - 85\%$ EAP.

Across all posteriors under examination, the Γ distribution achieves an average of 97.4% EAP. If we assume that branch length posteriors are in fact Γ distributions, we can explain the linear relationship between skewness and CV, since the Γ distribution has a skewness of $\frac{2}{\sqrt{\alpha}}$ and CV of $\frac{1}{\sqrt{\alpha}}$.

4.2 Newton-Raphson-guided Branch Length Proposals

4.2.1 Branch Length Optimization

The Newton-Raphson (NR) procedure is an iterative optimization algorithm, that allows determining the optimal branch length v_{opt} with respect to the likelihood for a given branch in a given tree [36]. An update v_{k+1} that is closer to the optimum branch length than the current value v_k, is defined as

$$v_{k+1} = v_k - \frac{g'(\mathcal{A} \mid v_k, \Theta)}{g''(\mathcal{A} \mid v_k, \Theta)}, \tag{4.3}$$

where $g'(\mathcal{A} \,|\, v_k, \Theta)$ is the first and $g''(\mathcal{A} \,|\, v_k, \Theta)$ is the second derivative of the logarithmic PLF with respect to the branch length v_k. The PLL that is employed by ExaBayes, offers an efficient implementation of the NR procedure. In most instances, no more than 3-5 iterations are necessary to determine the optimum v_{opt}. For instance, for the optimization of 3,000 randomly drawn branches of dataset **dat-150**, in only 17.1 % of all cases more than 5 iterations were necessary. Because of precomputations, iterations

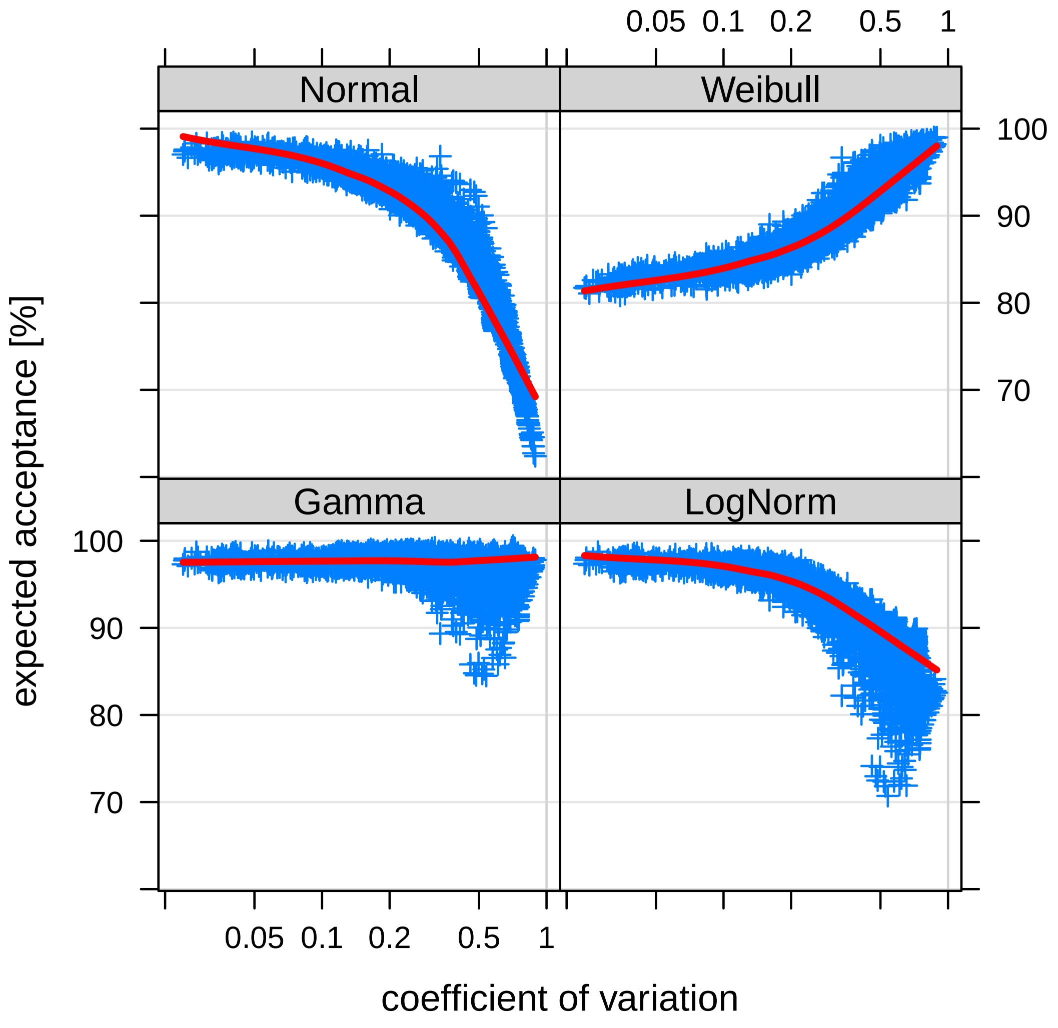


Figure 4.5: CVs of sampled branch length posterior distributions (x-axis) of all datasets versus the EAP (y-axis) for ML fits of parameters for a Γ, a normal, a log-normal and a Weibull distribution. Red line represents a LOESS [23] fit.

after the first one can be computed more rapidly. The procedure can easily be modified to optimize the logarithmic posterior density $g(v_k, \Theta \,|\, \mathcal{A})$ instead of the logarithmic PLF $g(\mathcal{A} \,|\, v_k, \Theta)$ by adding the first (resp., second) derivative of the logarithmic prior density to the respective derivatives of the PLF. Evidently, for a uniform prior no modifications are necessary. For an exponential prior on branch lengths parametrized by λ, we have to add λ to $g'(\mathcal{A} \mid v_k, \Theta)$ (no modification of the second derivative is necessary, since the second derivative of the exponential prior is 0). Recall that in ExaBayes, branch lengths are represented relative to the mean substitution rate ρ, that is, the relevant internal representation is $a = -\frac{v}{\rho}$ and we derive the PLF with respect to a. Thus, the derivative of the logarithmic prior density $f(a; \lambda)$ with respect to a is

$$\begin{aligned} f(v;\lambda) &= \lambda \cdot \exp(-\lambda \cdot v), \\ f(a;\lambda) &= \lambda \cdot \exp(\lambda \cdot a\rho), \\ \ln f(a;\lambda) &= \ln(\lambda) + a \cdot \lambda \cdot \rho, \\ \frac{\partial \ln f(a;\lambda)}{\partial a} &= \lambda\rho. \end{aligned}$$

4.2.2 Newton-Raphson on Branch Length Posteriors

Previous results (see **Fig. 4.5**) suggest that either the Γ or the Weibull distribution can be employed to design a branch length proposal that samples the branch length parameter with close to optimal efficiency (i.e., we obtain a high ESS). While the posterior itself is not known *a priori*, we have to estimate two parameters in order to determine a proposal kernel that is a good approximation of the posterior. Here, we can employ a NR optimization of the posterior density for obtaining the mode of the branch length posterior (see **Sect.** 4.2.1). As a by-product of the NR method (and only in case of convergence to the optimal branch length v_{opt}), we obtain the value of the second derivative of the log-posterior density $g''(v_{\text{subopt}})$ where v_{subopt} is the branch length of the iteration before convergence is achieved. Typically, the difference between $g''(v_{\text{subopt}})$ and $g''(v_{\text{opt}})$ is negligible, thus we assume that $g''(v_{\text{subopt}})$ is very close to $g''(v_{\text{opt}})$. We can expect that the second derivative at the optimum yields information about the variation in the posterior distribution.

We determined $g''(v_{\text{opt}})$ for all branch length posteriors sampled in **Sect.** 4.1.3. **Fig. 4.6** indicates that, there exists a power-law relationship between the standard deviation of a branch length posterior and the negative value of $g''(v_{\text{opt}})$. For 11.1% of all branches, the NR method did not converge to the optimum. This exclusively affects posteriors with a CV close to 1. In other words, since for exponential distributions (which have a CV of 1) the mode is 0 and the PLL does not allow evaluation of values below a threshold for numerical reasons, the NR method fails by design. However, in these cases we can assume that the posterior can be approximated by an exponential distribution.

For all branches, where NR successfully determines v_{opt} and $g''(v_{opt})$, we fitted a power function $h(x) = b \cdot x^c$ with $g''(v_{opt})$ as response variable $h(x)$ and the standard deviation of the posterior $g(v)$ as explanatory variable x. We solved this non-linear regression model for parameters b and c using numerical optimization (and with $b, c \in \mathbb{R}$ as model

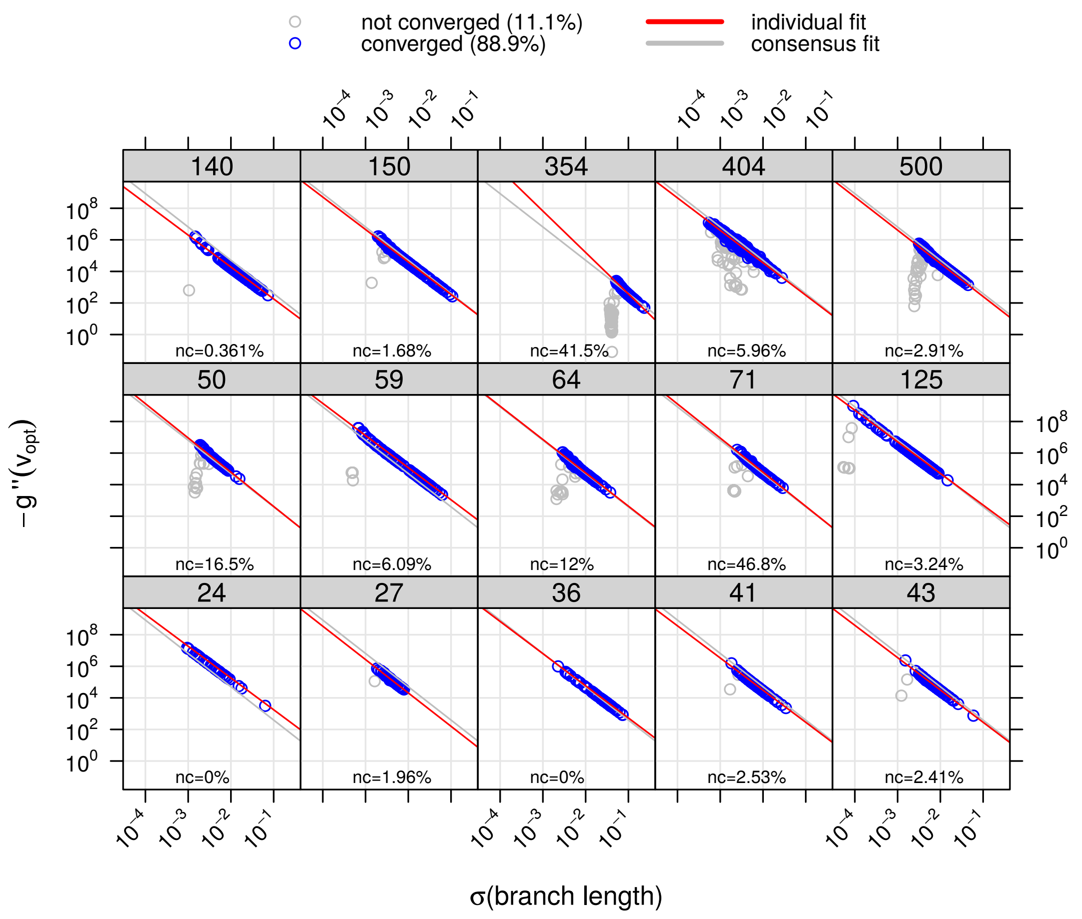


Figure 4.6: Relationship between negative second derivative of the log posterior at the optimized branch length $g''(v_{opt})$ in case of convergence and the standard deviation of the posterior ($\sigma_{\text{branch length}}$). Blue circles indicate that branch length optimization has converged, gray circles indicate failed branch length optimization because of an exponential posterior. The red line represents a dataset-specific (individual) fitted power function h(x), the gray line represents an equal-weight global (consensus) fit. For each dataset *nc* yields the proportion of branches with exponential posterior.

parameters). An extension with an additive factor $a \in \mathbb{R}$ to $h(x) = b \cdot x^c + a$ did not improve the model fit. **Tab. 4.1** displays the fitted parameters for each dataset as well as a consensus fit ($b = 1.61$, $c = -0.473$) across branches from all datasets. For the consensus fit, we randomly chose an equal number of branches from each dataset in order to avoid that datasets with large numbers of taxa have a stronger impact on the consensus than datasets with smaller numbers of taxa.

We observe that, specifically the exponential parameter b (which translates into the slope of the doubly logarithmic **Fig. 4.6**), is similar across all datasets (i.e., b is in $[-0.463, -0.499]$), except for dataset **dat-354**. Dataset **dat-354** furthermore stands out as having a particularly low nucleotide diversity (see **Tab. 4.1**), which appears to be the cause for its high proportion of exponential-like branch length posteriors, respectively posteriors for which NR does not converge. However, other datasets (e.g., **dat-50**) with similar characteristics did not show atypical parameter values for a and b. Thus, we assume that the comparably narrow range of σ values (i.e., the standard deviations) as well as the location of the shortest values of σ observed among the branch lengths posteriors, in this case leads to sub-optimal regression performance.

4.2.3 Proposal Design

Given the optimal branch length v_{opt} and the second derivative of the log posterior density $g''(v_{opt})$, we can now formulate a system of equations that allows approximating the posterior via some two-parameter distribution. For the Γ distribution, we can determine α and β as follows:

$$\begin{aligned} \text{mode:} \quad & \tfrac{\alpha-1}{\beta} &&= v_{opt}, \\ \text{standard deviation:} \quad & \tfrac{\alpha}{\sqrt{\beta}} &&= b \cdot (-g''(v_{opt}))^c, \end{aligned}$$

where b and c can either be fitted or we can use the aforementioned consensus parameters. The solution of these equations for the Γ distribution is straight-forward. For the Weibull distribution, it requires numerical optimization (i.e., Brent's method [18]). In case of the Γ distribution, we obtain

$$\begin{aligned} \beta &= \frac{v_{opt} + \sqrt{v_{opt}^2 + 4 \cdot \hat{\sigma}^2}}{2 \cdot \hat{\sigma}^2}, \qquad \text{where } \hat{\sigma} = b \cdot \left(g''(v_{opt})\right)^c, \\ \alpha &= v_{opt} \cdot \beta + 1. \end{aligned}$$

Thus, given a current branch length v_k, our NR-based proposal involves (i) a branch length optimization, (ii) determining the proposal density function q (here using a Γ distribution), (iii) drawing a random number v_k^* from q and (iv) calculating the Hastings-ratio of $\frac{q(v_k)}{q(v_k^*)}$. We can use the same density $q(x)$ for calculating the forward and the backward probability, since the optimum v_{opt} does not change (i.e., it does not depend on the current branch length value).

In cases, where NR optimization fails, the branch length posterior predominantly is an exponential distribution (i.e., in $\sim 80\%$ of all instances). This means that, the log-posterior density g is linear with a slope of $\mathrm{nr}_{d1} = g'(v)$ for arbitrary branch length

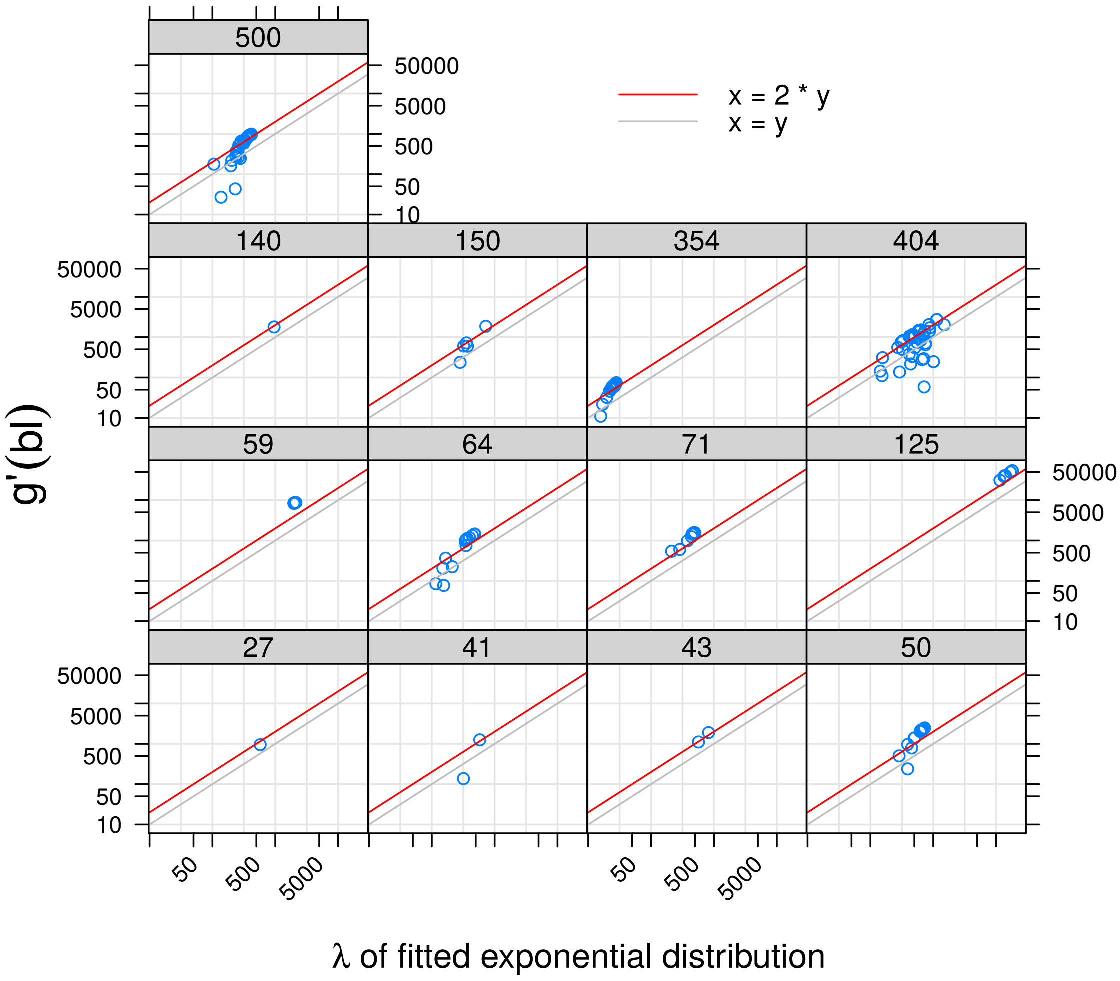


Figure 4.7: On x-axis: λ-parameter of exponential distributions fitted to non-converged exponential-like branch length posteriors. On y-axis: value of the first derivative of the log-posterior density $g'(x)$.

values v. The first derivative of the logarithmic density of the exponential distribution is $-\lambda$. Thus, we can use $q := \exp(-d \cdot \mathrm{nr}_{d1})$ as a proposal kernel for branch lengths, where the NR method does not converge. While we can expect $d := 1$ to yield good results for purely exponential posteriors, there exists a problematic intermediate group of Γ-like posteriors with comparably low α (and with $\alpha > 1$) for which NR does not converge.

Tab. 4.1 suggests that the exponent c (representing the slope in **Fig. 4.6**) generalizes well across various datasets, however that parameter b (the intercept in **Fig. 4.6**) is dataset-specific (most likely coupled to the mean substitution rate ρ). Thus, we can create a tuning scheme for the multiplicative factor b in order to increase the acceptance probability of the proposal. As discussed in **Sect.** 3.1.4, for typical MH proposals we aim for a target acceptance probability of $\sim 25\%$ and tune according to the OAP within the previous n (typically 100) generations. For the NR-based distribution proposal at hand, our target acceptance probability is 100%. If the proposal density approximates the branch length posterior well, then proposals can not be too modest (which is the motivation for the target rate of 25% otherwise). In contrast to the tuning of conventional proposals, we have to keep track of the previous acceptance probability and change the direction of tuning (i.e., whether to increase or decrease parameter b), once we observe a lower acceptance probability compared to the previous tuning interval. Thus, we can tune parameter b in case of NR convergence and parameter d for the proposal kernel in case of non-convergence (this implies that internally we keep track of the observed acceptance ratios separately for updates based on converged and non-converged optimizations).

4.2.4 Observed Acceptance Probabilities of NR-based Proposals

We implemented three flavors of NR-based Γ proposals: (i) using *individual* (i.e., dataset-specific) estimates for b (Γ_{indi}), (ii) using the *consensus* over all datasets for b (Γ_{cons}) and (iii) a scheme that tunes parameter b in case of convergence and parameter d in case of non-convergence as described in **Sect.** 4.2.3 (Γ_{tuned}). Given the occasionally superior performance of the Weibull distribution, we implemented a NR-based Weibull proposal ($\mathcal{W}_{\text{tuned}}$) for comparison that employs the same tuning scheme as used for the Γ distribution. For each dataset we ran 10 chains for 300,000 generations using (i) one of the 4 branch length proposals (3 flavors of Γ and 1 Weibull version), (ii) a tree length multiplier (see **Sect.** 3.2.4), and (iii) proposals on $\vec{\pi}$, $\vec{r}$ and the α value of the Γ model of rate heterogeneity (fixed topology, proposals weighted in a mixture of 20:1:1).

Fig. 4.8 depicts the OAPs for the 4 proposals under examination. While for 10 datasets Γ_{cons} consistently achieves an OAP $>$ 80%, the OAP is as low as $\approx$ 65% for datasets like **dat-354**. On many datasets, the OAP of Γ_{indi} exceeds the OAP of Γ_{cons} by 10%, such that more than 90% of proposals are accepted for 6 datasets. Naturally, since per-dataset parameters are not known *a priori*, Γ_{indi} is impractical for BI and only serves as an indicator of performance loss by using consensus parameters. Surprisingly, for **dat-50** and **dat-36**, Γ_{indi} performs worse than Γ_{cons}, although according to **Fig. 4.6** consensus parameters are close to the individually fitted per-dataset parameters. For **dat-125**, OAPs in chains vary strongly for Γ_{cons}. Additional experiments showed that OAPs further decreased and many chains (also for other datasets) explore sub-optimal regions of the posterior landscape, if we do not employ a tree length multiplier and proposals on GTR parameters. Since BI on **dat-125** is straight-forward otherwise (given the small number of trees in the 50% and 100% credible set), we assume that tree length (which is also modified by GTR proposals in ExaBayes) impacts the performance of NR-based distribution proposals.

Finally the tuned proposal Γ_{tuned} consistently achieves high OAPs: for 10 datasets, the average is above 90% and for none of the chains, the OAP is below 85%. While for some datasets such as **dat-43** or **dat-64**, the tuned proposal does not outperform Γ_{indi}, it yields an additional 20% of accepted proposals for **dat-71** compared to Γ_{cons} (that represents the only design alternative for production runs). As expected, the tuned Weibull proposal $\mathcal{W}_{\text{tuned}}$ consistently performs worse than (or at most as good

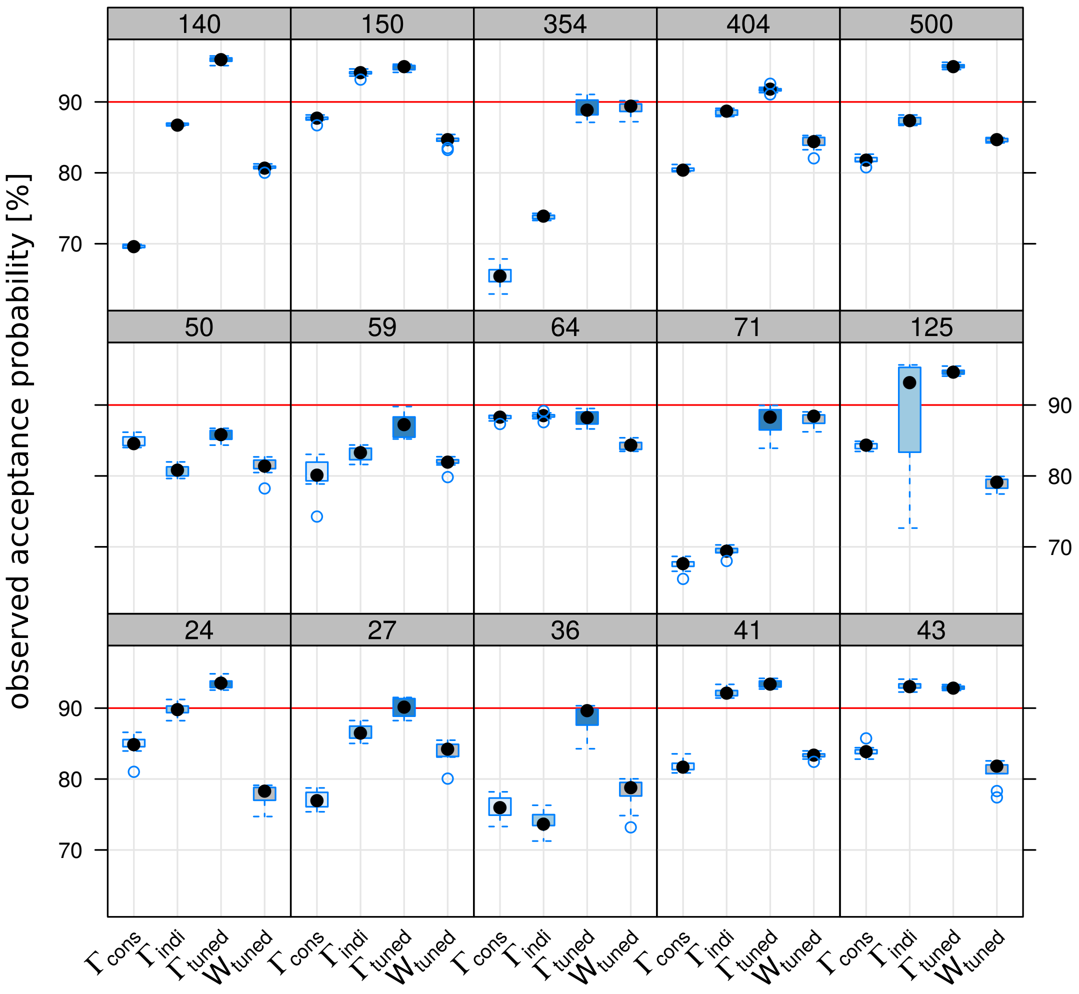


Figure 4.8: OAPs for a NR-based Γ proposal using consensus parameters (Γ_{cons}), for a Γ distribution proposal using parameters estimated on the individual dataset only (Γ_{indi}), for a Γ distribution proposal with auto-tuned parameters (Γ_{tuned}) and a Weibull distribution proposal using auto-tuned parameters. Each panel represents one of the 15 datasets.

as) its tuned Γ counterpart (e.g., **dat-140** by more than 10%). Datasets with a high number of exponential posteriors (**dat-71** and **dat-354**) indicate the importance of tuning parameter d of the exponential proposal kernel: here $\mathcal{W}_{\text{tuned}}$ as well as Γ_{tuned} achieve comparable OAPs.

4.2.5 Sampling efficiency

In conclusion, compared to a standard branch length multiplier proposal, we can expect a $3-4\times$ higher OAP for a tuned NR-based Γ proposal. Another striking advantage is that branch length updates v^* do not depend on the current branch length value. Thus, we expect a low auto-correlation among states in a chain. In the following, we evaluate the sampling efficiency of Γ_{tuned} compared to two popular proposals for branch lengths: the branch length multiplier (see **Sect.** 3.2.2) and the node slider (see **Sect.** 3.2.3).

For each dataset with n taxa, we executed 1 chain and chose chain length and sampling frequency as a function of n (total number of generations: $20,000 \cdot (2n-3)$, sample extraction every n generations), because substantial thinning can obfuscate low OAPs and high auto-correlation among states in the chain. We used a typical tree extracted from the previously discussed reference runs, kept all parameters fixed (using identical values for frequencies, substitution rates and $\alpha := 1$ for the Γ distribution of rate heterogeneity) and using proposals for integration over branch lengths. We compare Γ_{tuned} to a tuned branch length multiplier (i.e., we tune the uniform window size on the logarithmic scale, see **Sect.** 3.2.2) and an untuned node slider (as described in **Sect.** 3.2.3).

Fig. 4.9a depicts ESS values for all branch lengths on a respective fixed tree for all 15 datasets depending on its CV. Apart from its relation to skewness, it is important to verify that proposals depending on a previous state, have sufficient sampling efficiency for high and low CV values of the underlying conditional posterior (e.g., a multiplier tuned for CV ≈ 1 will have a low OAP and thus low ESS for posteriors with CV $\approx \frac{1}{32}$). As expected, Γ_{tuned} overall samples branch lengths substantially more efficiently than the multiplier or node slider proposal. On average (see **Fig. 4.9b**), Γ_{tuned} achieves an ESS that is between $2\times$ (**dat-354** and **dat-500**) and more than $8\times$ (**dat-140**) higher than the sampling efficiency of a multiplier proposal. For 11 datasets, the performance gain is in a range between $3-4\times$. At the same time, Γ_{tuned} requires only an additional $10-20\%$ of runtime (see **Fig. 4.9c**). For all proposals, the runtime dominating factor appears to be the evaluation of CPVs. That is, the runtime requirement for evaluating the likelihood at a randomly proposed branch in the tree. The cost for NR in case of Γ_{tuned} is moderate, specifically given its at least quadratic rate of convergence [see 44]. In comparison, the node slider is faster than the multiplier, specifically on datasets with a small number of taxa and/or characters. Since the node slider operates on two branch lengths, it is more likely that CPVs of outer branches of the tree are involved which are cheaper to compute. Our observations regarding runtime suggest that there is a huge potential for runtime optimization, if branches for proposals are drawn such that the number of CPVs that we need to evaluate is minimal.

In general, **Fig. 4.9a** corroborates our expectation that Γ_{tuned} mostly performs suboptimal for branch posteriors with a CV between 0.5 and 1, where NR does not converge

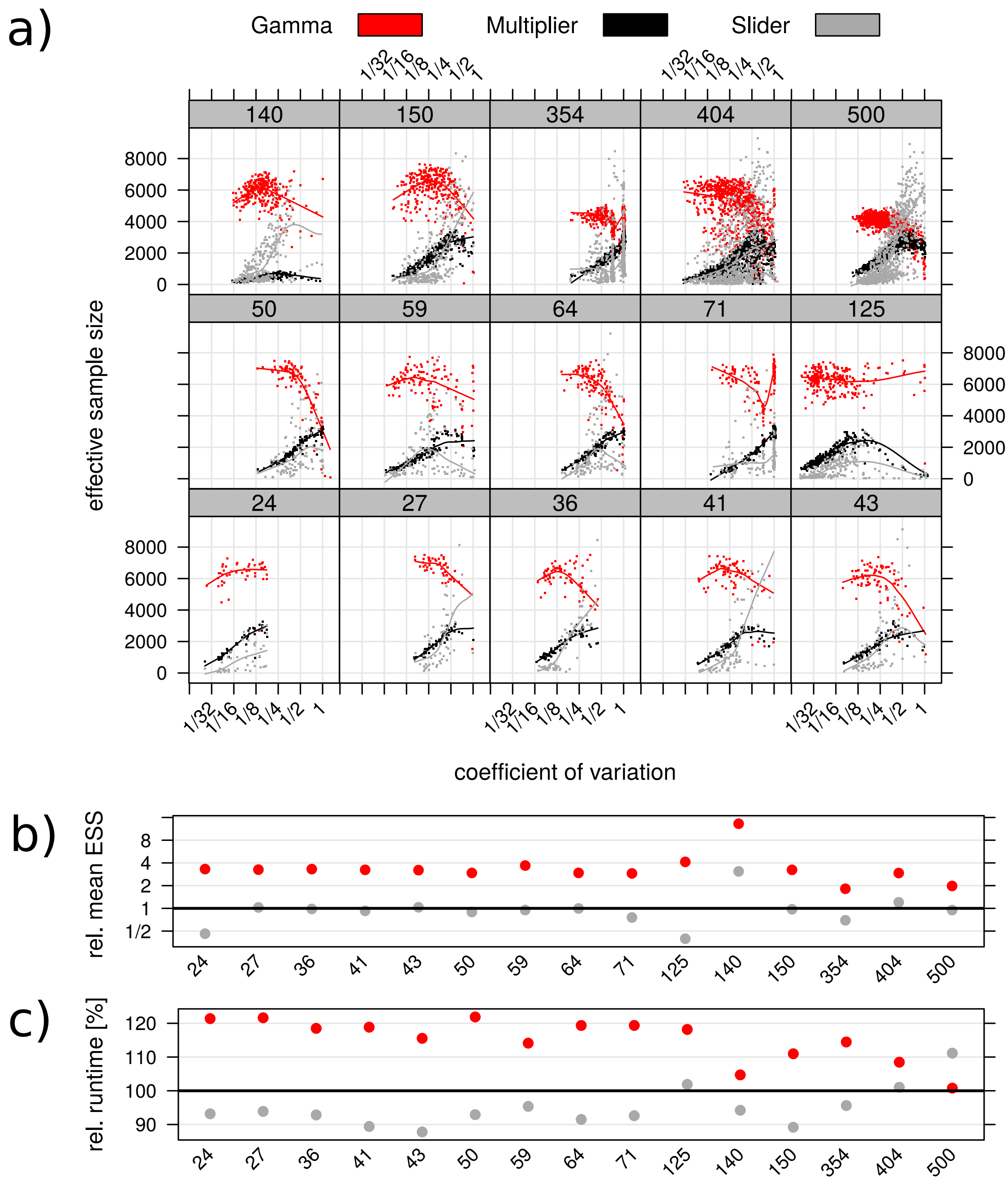


Figure 4.9: Comparison of sampling efficiency and runtime efficiency between the NR-based Γ proposal (red), the branch length multiplier (black) and the node slider (gray). Panel **a)** depicts the ESS versus the CV for posteriors of each branch length in a dataset. Panel **b)** depicts the average ESS for datasets relative to the average ESS obtained by the multiplier proposal. Panel **c)** depicts the mean runtime of proposals relative to the mean runtime of the branch length proposal.

on exponential posteriors. If the number of exponential posteriors is low, the efficiency of Γ_{tuned} may be lower in this experiment than in practice, since in this case the proposal has not been tuned often enough. We observe a negative correlation between sampling efficiency of the two traditional proposals and Γ_{tuned}. That is, in contrast to Γ_{tuned}, the two traditional proposals perform particularly well on exponential posteriors and comparably bad on bell-shaped posteriors. This can be explained by considering the Hastings-ratio of these proposals that depends on the value that the branch length (resp., sum of branch lengths) is multiplied with. For instance, in case of strongly skewed distributions doubling a branch length mostly yields a negative log-posterior ratio, which however is compensated by a positive logarithmic Hastings-ratio. The opposite holds for halving the branch length (except for values on the left side of the optimum of non-exponential posteriors with high skewness). Thus, the Hastings-ratio is also responsible for sub-optimal efficiency with almost symmetric posteriors (i.e., low skewness and low CV). Proposals on posteriors with a high CV are more likely to be accepted (in case of a multiplier proposal) than proposals for posteriors with low CV. Thus, we can expect that high-CV posteriors have a strong impact on tuning and thus yield larger multipliers that further decrease the performance of the multiplier on low-CV posteriors. For the node slider (with proposal density ratio m^2, where m is the multiplier), the effect of the Hastings-ratio on the sampling behavior is even more pronounced.

4.3 Topology Hybrid Proposals

We extend our general analysis of branch length posteriors to the change of the posteriors of branch lengths in topological proposals. As established, branch lengths represent our expectation in the number of substitutions that have occurred along a given bipartition of the tree defined by that branch. Since the set of bipartitions changes in a topology proposal that transforms tree τ into τ^*, there is no inherently *correct* way to transfer branch lengths from τ to τ^* for new bipartitions. Consider a stNNI move (equivalent to an eSPR that stops the traversal after a single step, see **Fig. 4.10a**), where subtree S is removed from adjacent nodes A and B. Here, the position of the associated branch length v_{SB} can remain unchanged. Since the bipartition of v_{SB} is the only bipartition to change for this stNNI, this is the mapping that preserves most branch lengths at its associated bipartitions. **Fig. 4.10b** depicts a bolder move. We refer to this class of proposals as a SPR-3 move, since they change three bipartitions, respectively three branches are traversed by the moving subtree S. The typical approach for mapping branch lengths (e.g., in MrBayes) is to leave branch lengths unchanged except for v_{SB} which becomes the branch length connecting the subtree S and the last node that has been traversed by S (v_{DS} in **Fig. 4.10b**).

If branch lengths are ignored, for each of the three stochastic operators – stNNI, eSPR and eTBR – a bolder proposal (with stNNI being less bold than eSPR and eSPR being less bold than eTBR) is the transitive closure of less bold operators. For instance, each eTBR can be expressed as 2 eSPRs and an eSPR can be expressed as n stNNIs. With the branch length mapping of MrBayes, we loose this transitive property. In other

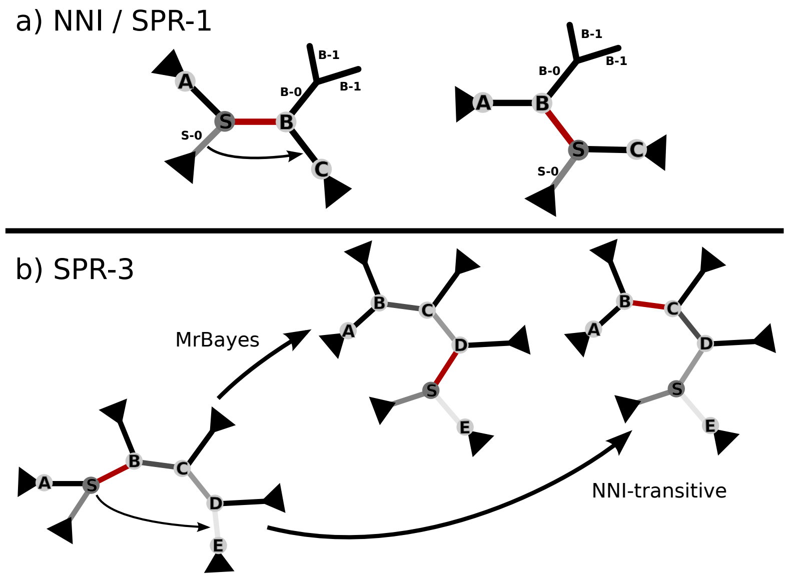


Figure 4.10: a) Branch length mapping for a NNI (resp., SPR-1) move. **b)** Two alternative branch length mappings for a SPR-3 move (i.e., a SPR move that is equivalent to three NNI moves). Left-hand version is typically applied in MrBayes/ ExaBayes, right-hand version represents an alternative NNI-transitive version.

words, if branch lengths are altered, one eSPR can not be expressed by multiple stNNI moves. We can however formulate an alternative NNI-transitive mapping for the eSPR that yields an identical result as three stNNIs (in the example shown in **Fig. 4.10b**) and effectively shifts branch lengths to the adjacent node in direction of the wandering subtree S. For the given example, we thus map (v_{SB}, v_{BC}, v_{CD}) to $(v^*_{BC}, v^*_{CD}, v^*_{DS})$.

4.3.1 Impact of Topological Moves on Branch Lengths

For an assessment of how branch length posteriors change when we apply a single topological move at a time, we ran a chain using default settings for each dataset for 20,000 generations to achieve a state typical for the convergent phase of the Markov chain and then enumerated all possible moves for a SPR-n move (where $n \in [1, 4]$), respectively limited the number to 200 randomly chosen SPR-n moves. For each distinct move, we integrated over branch lengths before and after applying the proposed move. However, because of the computational cost, aside from the central branch lengths (where the respective bipartition vanishes or appears, e.g., v_{SB} and v_{DE}), we limited the set of integrated branches to those that surround the central branches. More specifically, we ignore branches that are connected to the central branches by a path that is longer than 5 branches. In the following, we otherwise identify (resp., categorize) branches by their function in *or* their relationship to the central branches. As depicted in **Fig. 4.10**, we name subtrees according to the following scheme: subtree S is removed from subtrees A and B (where A is the subtree that keeps its bipartition and B is the first bipartition that is lost because of the traversing subtree). Subsequent subtrees (if distinguishable) are numbered alphabetically. Branches that are contained in these subtrees are numbered by their depth (e.g., B-0 is the root of subtree B and B-1 is a branch that we obtain by descending one branch from B-0). Thus, for instance for B-1 there exist up to 2 branches (resp., up to 4 branches for B-2) that are indistinguishable with respect to their relationship to the root of subtree B. For integration, we applied a branch length multiplier on randomly chosen branches in the branch length subset as defined above and stopped the chain when the ESS for each posterior was > 200.

As expected, a topological change has a strong impact on the inner branches of a move (*inner-default*, resp. *inner-alt* in **Fig. 4.11**). The topological change affects branches at the root of involved subtrees to a lower degree (class *close*) and has nearly no effect on branches that are further away from the central bipartitions (class *distant*). Particularly for bolder moves, we observe that, if we apply the MrBayes mapping (*inner-default*) branch length posteriors change substantially less on average than if we apply the NNI-transitive mapping. For the MrBayes mapping a clear trend to shorter inner branch lengths is observed. However, the NNI-transitive mapping leads to branch length changes that are as drastic as the worst case in *inner-default* (i.e., v_{SB}, the branch that is removed and remapped) and that vary in both directions. Among branches at the root of involved subtrees, there is a general tendency towards longer average branch lengths after the move has been applied, specifically for the moving subtree (S-0). **Fig. 4.11** only depicts branches in moves that do not decrease the log-likelihood by more than 100 units. If all branches are considered, some branch lengths are shortened by a factor of 1,024 after applying the move. As we increase the likelihood threshold (focusing on *better* moves), we gradually obtain less extreme ratios.

If we consider all moves (regardless of impact on likelihood), there is no correlation among inner branches (mapped with *inner-default*) and branches in the class *close* ($r = -0.0532$). For branches in the moves depicted in **Fig. 4.11**, there is a weak correlation ($r = -0.232$) and if we only focus on moves with a comparably high likelihood ratio of at least -10 log units, we obtain a moderate negative correlation of $r = -0.403$. In other words, moves that are accepted have a strong tendency to shorten inner branches and at the same time tend to extend adjacent branches of involved subtrees. We assume that, this is because after burn-in, and before applying a move, the topology is in a comparably parsimonious state (i.e., the topology pairs up taxa, s.t. few substitutions explain the observed alignment). Moves applied to this state are more likely to yield a worse model for the alignment. That is, we have to assume that more substitutions took place between the common ancestor and the respective subtrees and thus branches adjacent to the inner branches are elongated, while inner branches become shorter in comparison.

4.3.2 Construction of Hybrid Proposals

The comparison of branch length posteriors in the context of topological moves suggests that the acceptance ratio of topological moves may suffer from branch lengths that are sub-optimal for the newly proposed topology. To account for that, MrBayes typically applies a weak branch length multiplier (multiplies by a factor of $\in [0.95, 1.05]$) to the remapped branch length v_{SB}. While the proposed value is based upon the previous value (and is likely to propose a less optimal value), a central advantage of the NR-based distribution proposal is the possibility to propose branches *de novo*. That is, we can directly propose a branch length that most likely is close to the optimum. Ideally, for proposing more than one branch length simultaneously with an update of the topology, we want to propose according to the joint optimum of all proposed branches. This means that we need to repeatedly optimize all branch lengths to be proposed until they jointly converge. We consider a branch as converged, when the optimum changes by a factor of less than 0.01 compared to the previous iteration [see 133, for more involved convergence criteria].

Fig. 4.11 suggests that for the NNI, it is sufficient to either only propose the remapped branch (v_{SB}) or (as a bolder alternative) also propose lengths of branches A-S, B-0, B-C and S-0. For SPR-n moves (with $n > 1$), we can construct increasingly bolder moves by consecutively increasing the set of branches $\mathbb{S}$ to be proposed in the following order: S-B, last branch traversed by subtree S (i.e., C-D in SPR-3), S-0, A-S, first non-traversed branch (i.e., D-E in SPR-3), continuing with subtree branches (e.g., D-0) and inner branches (e.g., B-C) in the direction from the reattachment location toward the pruning location. However, since reversibility is a must for proposals, we have to choose the set $\mathbb{S}$ such that the original state can be restored (i.e., such that we obtain a non-zero Hastings-ratio). For instance, if for a SPR-3 move, we propose a new length for branch C-D, we have to propose B-C in the forward move as well in order to obtain a reversible proposal. This constraint hinders the construction of hybrid proposals that are tailored for changes in the posterior density after applying a move. In other words, we have to propose additional branch lengths to make the hybrid move reversible.

Thus, we examine 4 increasingly bold classes of hybrid proposals by adapting (i) only the remapped branch S-B (abbrev. *switch*), (ii) all inner branches (abbrev. *inner*, e.g., B-C, C-D, D-S in SPR-3), (iii) all inner as well as the branches adjacent to the path of branches traversed by S (A-S and D-E in SPR-3, abbrev. *inner**) and (iv) all branches from (i) - (iii) as well as the root of all traversed subtrees (abbrev. *close*, S-0, B-0, C-0 and D-0 in SPR-3). Since after determining the joint optimum of all branches $v_i^* \in \mathbb{S}$, each branch is proposed from a separate distribution $\Gamma_{v_i^*}$, the Hastings-ratio can be computed as the ratio

$$\text{Hasting's ratio of } \mathbb{S} = \frac{\prod_{v_i \in \mathbb{S}} \Gamma_{v_i}(v_i)}{\prod_{v_i^* \in \mathbb{S}} \Gamma_{v_i^*}(v_i^*)}. \tag{4.4}$$

This implies, that we also have to determine the joint branch length optimum for the reverse move.

We implemented hybrid proposals for all 4 proposed sets $\mathbb{S}$ for the eSPR and parsSPR (see **Sect.** 3.3.2) moves. In case of the stNNI proposal, only the *switch* and *close* case are relevant. Since the eTBR is equivalent to two SPR moves, we can easily construct analogous hybrid proposals by combining the branches to be proposed for both SPR moves and by also including the bisected branch in set $\mathbb{S}$. Furthermore, we adapted the 4 alternatives for a ppSPR (see **Sect.** 3.3.2). For instance, for a hybrid proposal

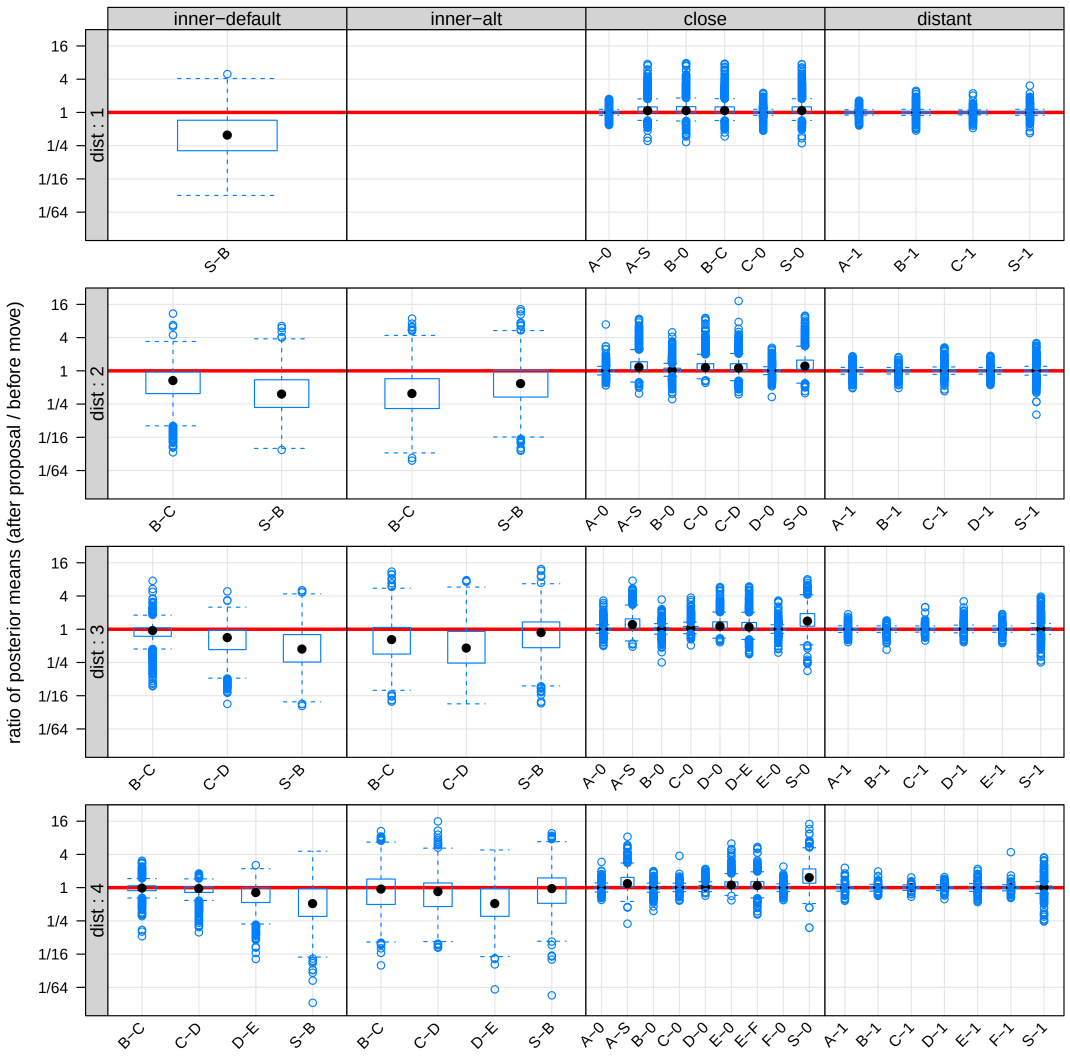


Figure 4.11: Ratios of branch length posterior means before and after application of a SPR-n move (here, indicated as *dist-n*). Only moves that do not decrease the likelihood by more than −100 log units are depicted. Values above red line indicate that branch length means are longer after the topological change. The x-axis enumerates branch names before a topological move has been applied. For branches with vanishing bipartitions, *inner-default* displays ratios according to the MrBayes-mapping and *inner-alt* displays ratios according to the alternative NNI-transitive mapping. The classes *close* and *distant* display branch categories by their depth in the respective subtrees (more distant branches not shown).

of *switch* and ppSPR, we determine the PP at each reattachment location in a radius around the pruning point. Whenever the subtree is reattached, we optimize all branch lengths in set $\mathbb{S}$, before evaluating the PP of this particular proposal. This then allows proposing a SPR move based upon the PP with optimized branch lengths. However, since we can not propose optimized branches in conjunction with the SPR move, we then propose all branch lengths in set $\mathbb{S}$ using the NR-based Γ proposal. Thus, depending on radius and set $\mathbb{S}$ these posterior-guided hybrid proposals rapidly become computationally prohibitive.

4.3.3 Evaluation of Hybrid Proposals

In the following, we assess the convergence efficiency of the aforementioned 5 topological proposals either (i) in their plain form, (ii) as a hybrid proposal in combination with a branch length multiplier (only modes *switch* and *inner*) or (iii) as hybrid proposals in combination with a NR-based Γ proposal (all 4 modes). We ran 8 independent chains starting from a random tree and employing default proposals for all other model parameters with the exception of topological and branch length proposals. The proposal under examination (i.e., cases i-iii) was the only topological proposal allowed and a NR-based Γ proposal replaced the branch length multiplier and node slider. After 1,000,000 generations, we calculated the ASDSF between the sampled trees and the reference tree set created for the respective dataset (see **Tab. 4.1** and **Sect.** 4.1.1).

Overall, we do not observe that the Γ hybrid proposals consistently yield a better level of convergence than traditional proposals that do not propose branch lengths simultaneously (see **Fig. 4.12** for datasets discussed here, results for remaining datasets are less conclusive). On **dat-27**, multiplier hybrid proposals converge on average as fast as the respective plain proposals (except for the eSPR). However, the Γ hybrid proposals consistently achieve a higher level of convergence than the plain proposal for this dataset. Similarly, Γ hybrid proposals achieve an overall better level of convergence on datasets **dat-36** and **dat-125**. Both datasets have particularly small 50% and 90% credible tree sets. We also observe advantageous convergence behavior of Γ hybrid proposals on **dat-150** for the parsSPR and ppSPR (the three remaining classes of topological moves did not converge).

In contrast to **dat-36** and **dat-125**, for dataset **dat-150** no tree was sampled twice in the reference runs. This suggests a flat posterior landscape that only allows the two guided proposals to reach the region of high PP. While any kind of additional branch length proposal (also the multiplier in case of the parsSPR) improves convergence behavior, there still is a noticeable proportion of chains employing hybrid proposals, that end up in local optima. We assume that, while Γ hybrid proposals may decrease burn-in time, they do not necessarily traverse a rough topological posterior landscape more efficiently.

For dataset **dat-354**, any set of branch lengths proposed via the NR-based Γ proposal clearly decreases convergence. In contrast, adding a multiplier decreases convergence to a lesser degree. We assume that, the NR-based Γ proposal performs poorly in this case because of a large proportion of branches for which the NR method does not con-

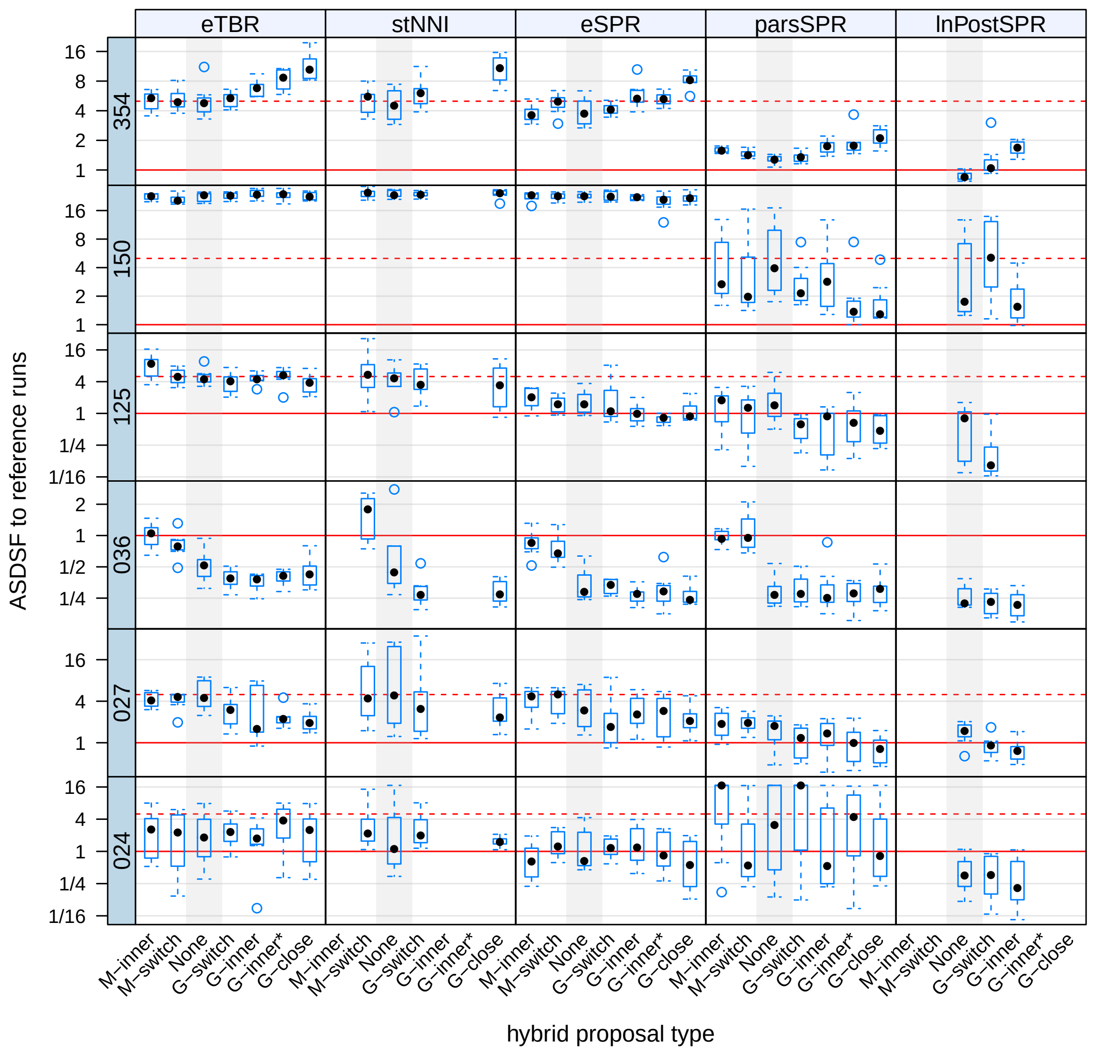


Figure 4.12: Performance of 4 classes of hybrid proposals employing a Γ-proposal (*G-switch, G-inner, G-inner** and *G-close*) compared to hybrid proposals employing a multiplier (*M-inner* and *M-switch*) and also contrasted by default proposals (*None*) for 6 datasets. Each box depicts the ASDSF between 8 independent chains and the concatenated set of reference trees. Dashed red line represents a lax convergence threshold (5%), red solid line a strict threshold for convergence (1%).

verge (see **Fig. 4.6**). For all chains, we obtain an OAP for the NR-based Γ proposal ranging between 89% and 90% (identical to results from **Fig. 4.8**). We expect that the convergence disadvantage of Γ hybrid proposals increases with the number of proposed branches, because more factors are added to its Hastings-ratio (see **Eq. 4.4**). Thus, it is likely that the Γ distribution poorly approximates a specific kind of branch length posteriors that might be important for accepting hybrid moves. A higher number of proposed branches increases the probability that a Γ distribution poorly approximates a branch with ensuing consequences for the acceptance probability of the move.

For datasets **dat-125** and **dat-24** that have particularly few trees in their credible sets, Γ-ppSPR hybrid proposals achieve particularly rapid and accurate convergence to the topology of the reference runs (ASDSF $\approx$ 0.0625). In contrast, while multiplier hybrid proposals perform comparably on most datasets, the addition of the multiplier clearly decreases convergence performance for datasets **dat-43**, **dat-59** and **dat-64**.

4.4 Summary

The results presented in this Chapter suggest that in the absence of a molecular clock, a Γ distribution accurately describes the posterior distribution of branch lengths (with some exceptions, where a Weibull distribution yields a more precise description). We determined that the skewness and CV (both closely entangled for a Γ distribution) are important characteristics that essentially divide branch length posteriors into two classes of distributions: bell-shaped distributions that represent strong phylogenetic signal and exponential-like distributions for which the most likely outcome is the absence of substitutions (thus weak phylogenetic signal). We observe a relationship between the number of trees in the credible set and the presence of exponential-like distributions. However, phylogenies of datasets with a huge number of trees in the credible set are not necessarily hard to resolve.

While a huge number of branch length posteriors was visually inspected, no clearly bimodal distribution was encountered. However, for a small proportion of branch lengths, we observed that the mode was flattened out and inaccurately matched by a Γ distribution.

We determined rules for a highly accurate approximation of branch length posteriors based on the optimized branch length posterior density and its first and second derivative. We derived a branch length proposal that is highly efficient in terms of acceptance ratio and allows us to propose branches *de novo* without depending on the previous branch length. While slightly slower, this proposal yields substantially higher ESS values for sampled branch lengths (although the advantage over the node slider and branch length multiplier decreases in the presence of a tree length multiplier). Interestingly, this proposal often has a sub-optimal OAP of 50% - 70% during burn-in. Furthermore, it often performs sub-optimally if a chain is stuck in a local optimum. The results suggest that the efficiency of commonly employed branch length proposals (branch length multiplier and node slider) is biased by their respective Hastings-ratio. This is, they are more efficient on exponential-like distributions. In contrast, the novel NR-based proposal performs sub-optimally, when the NR method does not converge for highly skewed posteriors.

In this Chapter, we examined how branch length posteriors change under topological moves. The results validate that the branch length mapping currently employed in MrBayes/ExaBayes is the best choice among the available alternatives. Particularly, the initial branches that are traversed by a pruned subtree in a move do not change drastically. Furthermore, we observed a negative correlation between the impacts on branch lengths after applying a move with high acceptance probability: using the mapping of MrBayes, the traversed inner branches become shorter while adjacent branches (i.e., branches peripheral to the central branches) are extended to a minor degree.

Based on our observations, we formulated a strategy for proposing branch lengths simultaneously with a SPR-n move. The insight, which branch lengths are strongly affected by a topological move, can furthermore be useful for the development of ML search strategies. Ultimately, the resulting hybrid proposals sporadically improved convergence behavior. However, they did not consistently and substantially outperform existing proposals to justify the additional runtime required for computing the expensive proposal kernels. Moreover, we found that multiplier hybrid proposals are not as detrimental to convergence as previous experiments suggested. Finally, the lacking performance improvement that comes with the application of Γ hybrid proposals strongly suggests that we can reject ill-fitting branch lengths as the major reason for slow topological convergence. It is likely that the notoriously difficult task of achieving and improving topological convergence requires a new class of radical topological proposals. For instance, proposals that decompose and reconstruct a connected part of the topology may be able to switch between trees of high PP and can thus be considered more powerful than SPR moves. The capability to propose branch lengths *de novo* (as established with the NR-based Γ proposal), could prove essential for the design of such radical proposals on topology.

In ExaBayes v.1.4, we employ the NR-based Γ proposal to replace the multiplier and node slider proposals on branches. Given the efficiency of the NR-based Γ proposal, we decrease the default proportion of branch length proposals and increase the frequency of topological proposals. A higher proportion of topological proposals reduces the time to topological convergence. In production runs using the new default settings and starting with a random tree, we noticed an increased probability for the chain being attracted by a local minimum. According to our experience, the NR-based Γ proposal also performs well for unlinked (i.e., per-partition) branch lengths. To compensate for the primary weakness of the Γ proposal (reduced efficiency when integrating over exponential-like branch length posteriors), we also added a modified branch length multiplier to the proposal mixture. All hybrid proposals (employing multipliers or NR-based branch length proposals) are available in ExaBayes v.1.4, but disabled by default.

5 Massively Parallel Bayesian Inference

Parts of the content of this Chapter are derived from the following peer-reviewed publications:

1. AJ **Aberer**, K Kobert, and A Stamatakis. "ExaBayes: Massively Parallel Bayesian Tree Inference for the Whole-Genome Era". In: *Molecular biology and evolution* 31.10 (2014), pp. 2553–2556
2. K Kobert, T Flouri, AJ **Aberer**, and A Stamatakis. "The Divisible Load Balance Problem and Its Application to Phylogenetic Inference". In: *Algorithms in Bioinformatics*. Springer Berlin Heidelberg, 2014, pp. 204–216

Contributions: Works within this Chapter are original contributions by Andre Aberer, if not explicitly stated otherwise. The algorithm outlined in **Sect.** 5.3.2 was developed by Kassian Kobert and Tomáš Flouri. Implementation and evaluation in ExaBayes and ExaML were performed by Andre Aberer. Several design decisions in the early development of the sequential version of ExaBayes emerged from discussions with Kassian Kobert. Large-scale alignments used within this Chapter were created by Kassian Kobert.

In this Chapter, we focus on the implementation of phylogenetic BI, specifically with respect to parallelization and technical aspects that coincide with parallization. After an initial overview in **Sect.** 5.1.1 over the general design, we evaluate the sequential performance of ExaBayes in comparison to MrBayes (see **Sect.** 5.1.2). We then discuss the parallelization scheme of ExaBayes in **Sect.** 5.2. Subsequently, we discuss various techniques to improve load balance in ExaBayes (see **Sect.** 5.3) and continue with techniques for reducing the memory requirements of ExaBayes (see **Sect.** 5.4). Finally, we conduct BI on a full-length simulated genome alignment using ExaBayes and discuss the results.

5.1 Sequential ExaBayes

5.1.1 Sequential Software Design

In the following, we describe the overall program design of ExaBayes version 1.4, not considering the parallelization that is covered in more detail in **Sect.** 5.2. Furthermore, we focus on the part of the program that does BI and do not go into detail for any of the helper programs (e.g., for consensus tree computation). For an accompanying unified modeling language (UML) class diagram, see **Fig. 5.1**.

Top-Level View

The specification of the analyses to be conducted is passed to ExaBayes via a configuration file. Thus, the configuration file specifies *what* kind of MCMC analysis is conducted (e.g., parameter setup, enabling and weighting of proposals), while command line arguments indicate *how* the analysis is executed (e.g., the degree of parallelization employed). ExaBayes employs the Nexus class library (NCL) [70] for parsing configuration files in `Nexus` format [73].

At the root of the ExaBayes class hierarchy is the `SampleMaster`, that sets up and orchestrates the analysis, for instance, with respect to input/output (I/O). Thus, transitively, `SampleMaster` has ownership of any object instance except for `ParallelSetup` (see **Sect.** 5.2) that needs to be initialized statically at the very beginning of program execution (in order to set up the parallel environment). `SampleMaster` initializes a number of independent Markov chains according to the information in the configuration file. Each of the independent chains may be coupled with several heated chains in order to execute MC^3 simulations (see **Sect.** 3.4). An independent run (resp. chain) is represented by `CoupledChains`. In the instantiation of this class, ExaBayes constructs as many chains as requested for MC^3, but at least one. Thus, class `Chain` implements the MCMC algorithm and `CoupledChains` is responsible for MC^3, if specified.

As established in **Sect.** 3.1.1, in BI we want to sample across a multivariate parameter space. Parameters such as the α shape parameter of the Γ distribution of rate heterogeneity or the stationary frequencies $\vec{\pi}$ are implemented as realizations of the abstract class `AbstractParameter`. Realizations of `AbstractParameter` are not aware of the current state of the Markov chain. Therefore, concrete parameter values such as branch lengths are stored in class `TreeAln`. Realizations of `AbstractParameter` must instead implement methods for communicating (resp., applying) concrete parameter values to class `TreeAln`, respectively extracting parameter values from `TreeAln`. All parameters are owned by a `Chain` and each parameter requires a prior that is a realization of the abstract class `AbstractPrior`. Priors must yield the logarithmic prior probability (`getLogProb`) or provide certain short-cuts to update the prior for a change in tree length for instance after we proposed new parameters in the GTR matrix (`accountForMeanSubstChange`). In case of an exponential prior on branch lengths, these short-cuts ensure, that we do not have to compute the prior probability for each value, but instead we can perform a single update depending on the tree length. For instance, if an exponential prior with parameter $\lambda := 10$ is required for branch lengths, then we create an instance of `ExponentialPrior` and set it as a prior in the relevant `BranchLengthParameter` instance.

Random Numbers

`Randomness` provides an interface to the pseudo random number generator (PRNG) library `Random-123` [98] that is used in ExaBayes for generating the random numbers

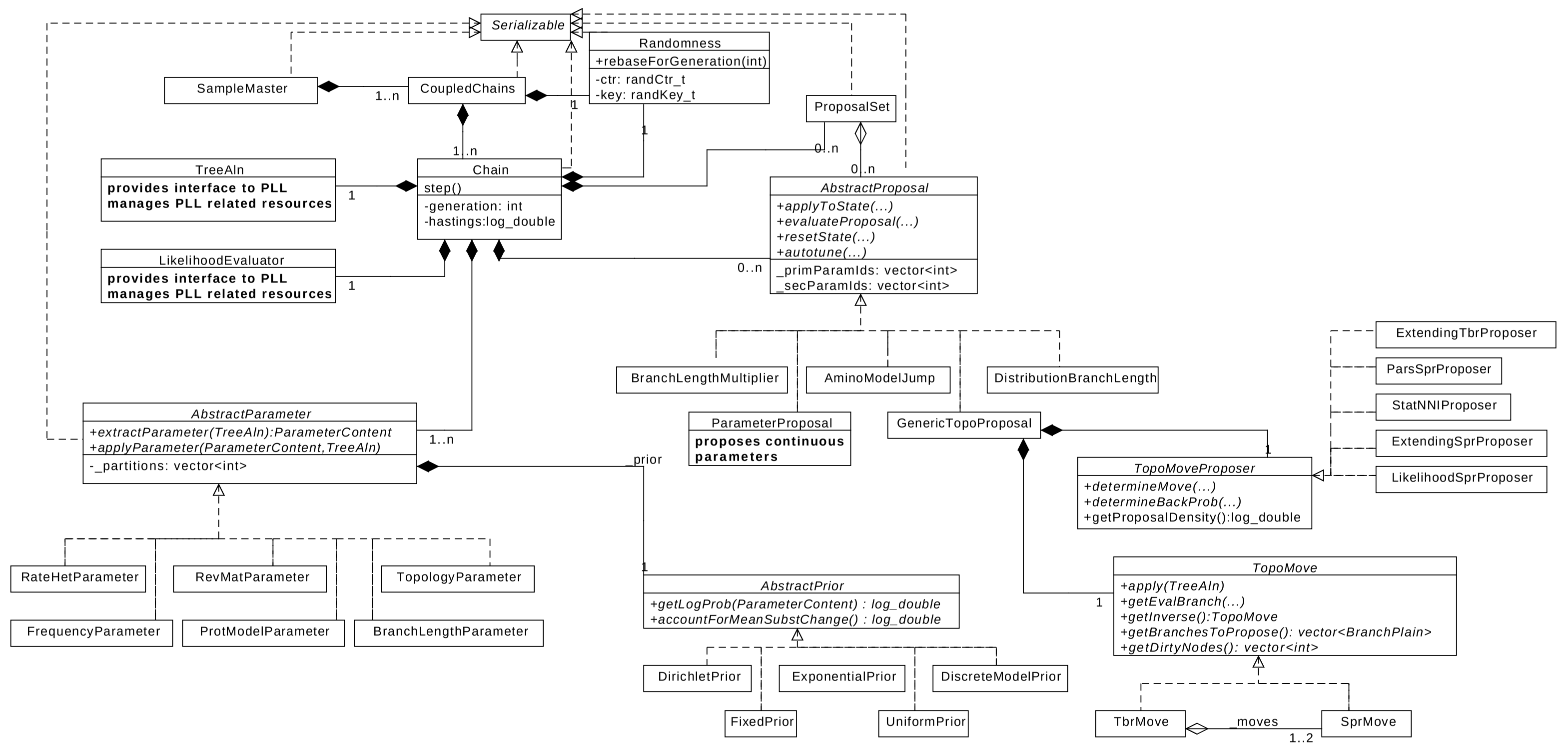


Figure 5.1: Simplified top-level view of the object tree that implements BI in the sequential version of ExaBayes. Several classes and class methods are omitted for clarity. Bold font outlines class responsibilities.

that are needed to simulate Markov chains. `Random-123` is a counter-based PRNG, that is particularly suited for parallel computing. Essentially, it maps a key k (equivalent to a *seed* number) and a counter value c to a pseudo random number r. `Random-123` provides several PRNG algorithms that pass rigorous tests for correlation or structure within the sequence of random numbers (e.g., they pass the TestU01 suite [69]). A general advantage of counter-based PRNGs (although not required for ExaBayes) is that they avoid the so-called *birthday problem*: for different key values k_1 and k_2, we are guaranteed to obtain distinct random sequences. In contrast, the popular 19,937-based Mersenne Twister (MT19937) does not offer this guarantee, when initialized with distinct seed values. Thus, with the MT19937 for an increasing number of PRNG instances (e.g., in case every process instantiates a PRNG), we increase the probability that the random sequences of two PRNGs overlap and we obtain compromised random numbers. In ExaBayes, we compose counter c (typically 2×32-bit integer) from the current generation value (using 48 bits, that is, we can execute up to $2^{48} - 1 = 2.8 \cdot 10^{14}$ generations) and use a 16-bit integer for counting, how many random numbers have been consumed for computing the proposal of the current generation. Thus, this scheme allows us to set the PRNG to the adequate state for each generation. Specifically, we can *leap-frog* into the next generation in order to determine for instance the next branch that will be chosen for a branch-length proposal. Based on this knowledge, we can choose an optimal virtual root for likelihood evaluations in the current generation. This has been implemented in ExaBayes. However, no substantial gain in runtime performance has been observed.

In ExaBayes, `Chain` owns a chain-specific PRNG that provides random numbers for proposals. Furthermore, `CoupledChains` owns a PRNG for proposing swaps among coupled chains, when MC^3 is enabled.

Interface to the PLL

`TreeAln` and `LikelihoodEvaluator` represent the primary interface to the PLL [39], that is used for the efficient evaluation of the PLF. `TreeAln` holds an instance of `pllInstance` and `partitionList`, two `struct`s that are defined in the PLL. Thus, ExaBayes effectively uses the PLL's topology representation. `TreeAln` manages all resources required for likelihood evaluations, such as for instance, arrays for multiple sets of branch lengths $\vec{v}_1, \ldots, \vec{v}_n$. In order to reduce code complexity on the side of the PLL, ExaBayes-specific modifications to the PLL are reduced to a minimum. In those cases where code modification of the PLL is unavoidable for employment in ExaBayes, the respective functions are implemented as part of the ExaBayes code instead. One such exception is that ExaBayes allows for a dynamically chosen number of distinct branch length parameters (i.e., the maximum number of distinct branch lengths is not chosen at compile-time). The associated resource arrays are managed within `TreeAln`. To fulfill this requirement, direct modification of the PLL was necessary.

The only PLL-resource not managed by `TreeAln`, but by class `LikelihoodEvaluator` are the CPVs that are used for likelihood evaluations. `LikelihoodEvaluator` provides an interface for invalidating CPVs. As established in **Sect.** 2.6.2, in many instances it is sufficient to update only a small set of CPVs (e.g., in the best case, we only have to update $n + 1$ CPVs for an SPR-n move). `LikelihoodEvaluator` offers an `imprint` method that effectively creates a snapshot of all current CPVs. For likelihood evaluations that are requested after an `imprint` call was issued, CPVs are swapped with reserve arrays if there is the need to recompute CPVs under changed parameters. After an `imprint`, each CPV is swapped at most once. That means that, if a proposal needs to conduct several evaluations at an inner node (e.g., for computing a ppSPR, see **Sect.** 3.3.2), the CPVs will be reused after the initial swap. During the `imprint` call, we also create a backup of the orientations of the CPVs (i.e., the information on which subtree is represented by the CPV). For an illustration see **Fig. 2.1**, where orientations of CPVs are represented by red arrows. In case a proposal is rejected, `LikelihoodEvaluator` can therefore restore all CPVs and their respective orientations to the state before evaluating the proposal.

However, the PLL does not allow for keeping track of per-partition orientations of CPVs, a feature that is needed in some configurations by ExaBayes (e.g., when no proposal sets are used, see **Sect.** 5.3.1). Therefore `LikelihoodEvaluator` can also store per-partition orientations and has to prepare the `pllInstance` for partition-wise computation of CPVs.

Proposals

Class `AbstractProposal` provides an abstraction for the various proposals that are implemented in ExaBayes. At start-up, proposals are instantiated based upon

- instantiated parameters (e.g., for an AA partition, a `FrequencyParameter` that models stationary frequencies is only instantiated if explicitly specified by the user),
- available priors (i.e., no proposals are instantiated for a parameter with a prior that enforces fixed values),
- proposal configuration provided by the user (i.e., users may disable specific proposals by setting their weight to 0).

All proposals must implement methods for (i) proposing a new state Θ^* of the parameter vector, (ii) for resetting the parameter vector into its previous state Θ and (iii) a method that describes the most efficient way to evaluate the likelihood of the proposed change. Proposals can have tunable parameters (e.g., the δ of a sliding window, see **Sect.** 3.2.1) that are different for each chain. Thus, proposals are chain-specific and are owned by a chain. Each proposal holds ids of primary parameters of interest (owned by the chain, typically only one) as well as secondary parameters that may need to be changed by the proposal as well (e.g., branch lengths when the tree length has changed as a consequence of a frequency proposal).

`ParameterProposal` is a generic proposal that uses its associated parameter to set parameter values (e.g., of frequencies $\vec{\pi}$) that are proposed by a template parameter class (either a sliding window or a Dirichlet proposer). All proposals on topology are implemented using an *abstract factory pattern.* The abstract factory `TopoMoveProposer` produces a topological move `TopoMove` as its abstract product. `TopoMove` can either be a NNI, a SPR or a TBR move (see **Sect.** 3.3). Furthermore, `TopoMoveProposer` provides the probabilities of the forward and backward move. This is non-trivial in cases of guided hybrid proposals, such as a ppSPR that optimizes branches at each location. As derived in **Sect.** 3.3, two branches suffice to represent a SPR. Furthermore, the NNI is a special case of a SPR and a TBR is the concatenation of two SPR moves. Thus, it is sufficient to implement all functionality required by `TopoMove` once in `SprMove`. For instance, `SprMove` provides a method that yields its inverse (i.e., a complementary SPR move that reverts the tree to its original state). Furthermore, `SprMove` has a method that determines branches (depending on configuration, see **Sect.** 4.3) that need to be optimized for hybrid moves. Both methods also apply to NNI moves (since these are implemented as a SPR-1 move). `TbrMove` employs these methods of `SprMove` to effectively execute two subsequent SPRs.

Typically, realizations of `AbstractParameter` do not own any parameter-specific data and tunable parameters are stored in realizations of `AbstractProposal`. An exception is `BranchLengthParameter`. Here, tunable parameters (i.e., parameters b and d) of the NR-based Γ proposal (see **Sect.** 4.2.3) are stored in `BranchLengthParameter`. Thus, hybrid proposals can propose branch lengths *de novo* by employing the tuned parameters of the NR-based Γ proposal.

Serialization and Checkpointing

ExaBayes implements checkpointing, that is, the current state of all Markov chains, proposals and PRNGs can be written to file at given intervals. In case the inference is aborted (e.g., by wall-time restrictions of cluster resources), a *roll-back* is possible. If ExaBayes is invoked with checkpointing enabled, an aborted analysis can be continued. For a restart from a checkpoint file, ExaBayes sets up `SampleMaster` as if invoked anew and then overwrites all state-specific fields with data from the checkpoint file.

ExaBayes comes with a strong guarantee of reproducibility, that is, if restarted from a checkpointing file, ExaBayes produces the exact same outcome. To achieve this goal, all classes relevant to the state of the MCMC process have to be serializable (i.e., inherit from the abstract class `Serializable`).

5.1.2 Sequential Performance

ExaBayes implements a set of proposals that is similar enough to the proposals implemented in MrBayes, such that a fair runtime comparison between the two tools is feasible. To achieve this, we have to set the radius parameter of the parsSPR to a number that covers the entire tree. The sequential runtime performance is vastly dominated by likelihood evaluations (typically about 95%). Among the remaining factors that affect the runtime, the parsimony score computation deserves mentioning. In ExaBayes parsimony computations account for a particularly low fraction of total runtime (typically 1-2%), since ExaBayes integrates the highly efficient parsimony implementation of the PLL [originally published in 6].

Thus, given the comparability of the proposal mechanisms, the runtime assessment largely boils down to a comparison of the underlying likelihood implementations in a Bayesian framework. For ExaBayes as well as MrBayes a number of different PLF implementations are available. The native likelihood implementation in MrBayes uses streaming SIMD extensions (SSE) for vectorization. Furthermore, the native version employs single-precision floating point arithmetics. While conducting numerical operations on single-precision floating point numbers is faster and requires 50% less memory than under double-precision, using single-precision comes at the cost of reduced numerical accuracy of likelihood computations. The runtime costs of double-precision versus single-precision are not straight-forward to estimate: for single-precision, more operations that ensure numerical stability (so-called scaling operations) are necessary compared to double-precision. Particularly in the context of ML tree searches, the reduced numerical range can lead to problems with numerical stability [16]. Here, searches for trees with more than 2,000 taxa can become infeasible. While we will not analyze the numerical stability of BI programs under single-precision arithmetics, we did not encounter any issues pertaining to numerical stability with MrBayes. Note that, alternatively, MrBayes can also be configured to employ either single- (non-vectorized) or double-precision (vectorized) floating point numbers when using the Beagle [12] library for likelihood calculations, instead of its native implementation.

ExaBayes employs the highly optimized likelihood implementations (all double-precision) provided by the PLL. Apart from a SSE-optimized version, ExaBayes also comes with a version of the PLL that is optimized for central processing units (CPUs) offering advanced vector extension (AVX) intrinsics. Thus, ExaBayes makes full use of modern vector processing units that are available in recent x86 CPUs. Furthermore, the PLL offers a likelihood implementation that omits the computation of conditional likelihoods for subtrees that entirely consist of undetermined or gap characters (while still yielding the correct likelihood) using subtree equality vectors (SEVs) [56]. This extension (which can be combined with either the SSE or AVX version) is particularly useful for large phylogenomic datasets consisting of several concatenated genes that usually contain a substantial proportion of missing data (e.g., due to the unavailability of orthologs). For instance, the proportion of undetermined/missing data in the largest super-matrix in a recent study [27] is 78.68%. For comparison, one of the largest matrices [57] that has been analyzed to date (with 51 taxa and 322,150,876 bps) has 15.3 % missing data, which translates into $2.52 \cdot 10^9$ unknown/deletion (i.e., `-` and `N`) elements in the alignment matrix.

We executed a single chain using the different likelihood implementations in ExaBayes (version 1.2.1) and MrBayes (version 3.2.2.) on various empirical single-partition small- to medium-sized datasets (see **Tab. 4.1**) that were used for benchmarking proposal efficiency [64] and for evaluating the so-called bootstopping criterion [86]. In **Fig. 5.2**, we compare runtimes (averaged over 3 independent runs) relative to the non-SEV SSE version of ExaBayes (referred to as *reference case* or *exa-sse-nosev*). Compared to *exa-sse-nosev*, the SEV versions in ExaBayes perform similarly to the reference case. However, SEV-versions are substantially faster for datasets with many taxa and a high proportion of missing data. Specifically, on the 994-taxon and 1,481-

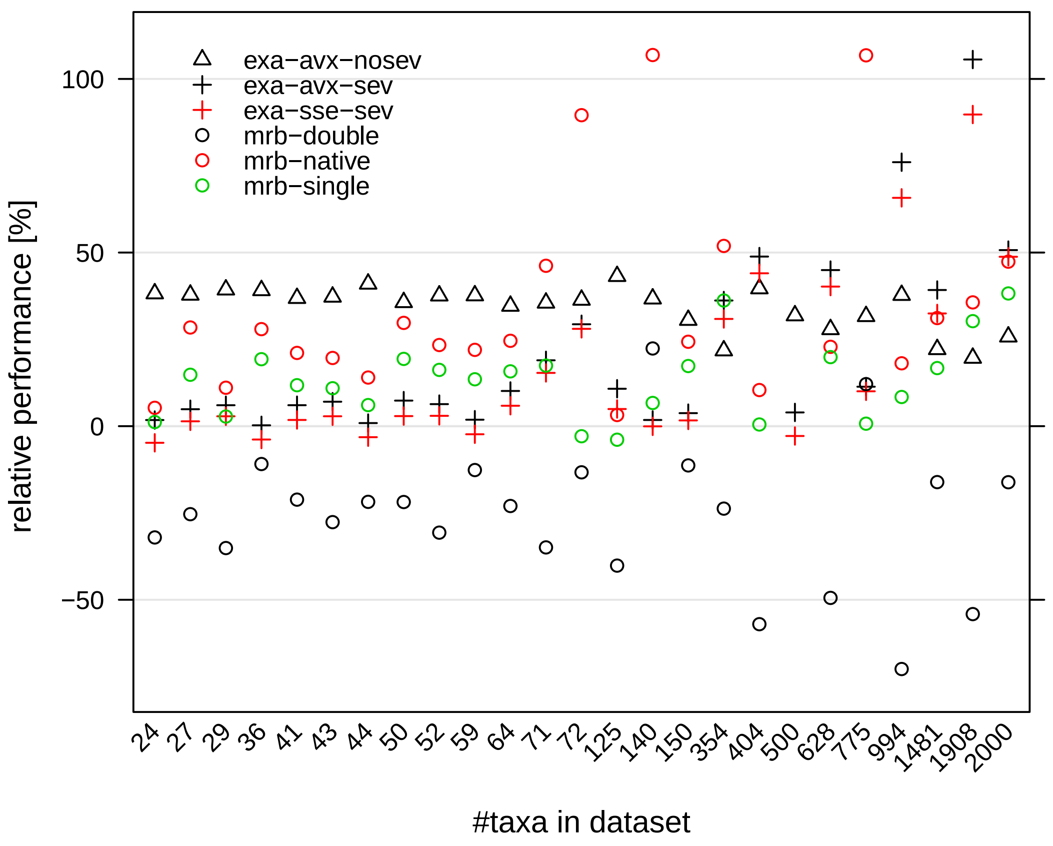


Figure 5.2: Sequential runtime performance comparison between ExaBayes and MrBayes employing various implementations of the PLF. Runtimes are relative to the runtime of ExaBayes using the non-SEV (*nosev*) SSE implementation. Runtimes for MrBayes using either its native (*mrb-native*) implementation or a single- (*mrb-single*) or double-precision (*mrb-double*) implementation provided by Beagle.

taxa	characters	proportion of gaps [%]	unique site patterns	data type
24	14,190	0.26%	4,600	DNA
27	1,949	20.42%	934	DNA
29	2,520	30.57%	1246	DNA
36	1,812	0.02%	1,020	DNA
41	1,137	10.79%	768	DNA
43	1,660	11.03%	954	DNA
44	5,582	8.60%	2,788	DNA
49	1,149	3.29%	628	DNA
50	1,133	9.33%	489	DNA
52	2,157	25.19%	867	DNA
59	1,824	0.03%	1,037	DNA
64	1,008	21.21%	406	DNA
71	1,082	36.28%	445	DNA
72	32,883	52.09%	30,274	AA
125	29,149	32.72%	19,436	DNA
140	1,104	0.60%	1,041	AA
150	1,269	4.77%	1,130	DNA
354	460	14.71%	348	DNA
404	1,3158	78.92%	7,429	DNA
628	1,228	36.44%	1,033	DNA
775	4,519	19.35%	3,838	AA
994	5,533	71.39%	3,363	DNA
1,481	1,241	26.58%	1,241	DNA
1,908	1,424	58.38%	1,209	DNA
2,000	1,251	12.98%	1,251	DNA

Table 5.1: Real-world datasets for runtime comparisons listing number of taxa, number of characters, proportion of gaps in percent, number of unique site patterns and data type.

taxon datasets, the SEV implementations noticeably outperform all alternatives. For the latter case, the AVX version outperforms the reference case by more than a factor of 2.

As expected, the AVX version of the likelihood function is consistently faster than the SSE version (between 19.6% and 43.2%). With the exception of two AA datasets, the double-precision implementation of BEAGLE is consistently and noticeably slower than its EXABAYES counterpart. For three DNA datasets (with 994, 404 and 1,908 taxa), *exa-sse-nosev* is more than two times faster than its BEAGLE equivalent. The single-precision version of BEAGLE in most cases is negligibly faster than the reference case, but never outperforms neither of the AVX implementations in EXABAYES. On every dataset, the native likelihood implementation in MRBAYES is faster than its BEAGLE counterparts. While the native likelihood implementation of MRBAYES is consistently faster than *exa-sse-nosev*, it generally is slower than either the AVX version of either the SEV or the non-SEV likelihood implementation in EXABAYES, except for 5 datasets. Three of these are AA datasets, where the native MRBAYES likelihood implementation outperforms any alternative noticeably (more than two-fold in 2 cases).

5.2 Parallelization

5.2.1 Types of Parallelism

At implementation level, two popular approaches to implement parallelism are *processes* and *threads*. We refer to either of these as parallel entities (PEs), if there is no need to distinguish between them.

The central difference between processes and threads is that threads share a common memory space, while distinct processes do not. This allows for highly efficient memory-based communication with thread-level parallelism, however also poses challenges to implementers: if thread t_a wants to read memory that is provided by thread t_b, then we must make sure that t_a does not read sections from memory that have not been written by t_b yet. Constructs that allow to order multi-threaded read-write operations comprise *barriers* (threads have to wait until all threads have reached the barrier) or *semaphores* (only m of n threads can execute a part of the code in parallel while remaining threads have to wait until threads have left the protected code area). Another disadvantage of shared-memory space are memory allocation dependencies: if memory is managed via a central lock (which is the case in the default memory allocator), then simultaneous memory allocations by several threads can yield a bottleneck. In EXABAYES, we employ the portable interface for creation and management of threads that has been introduced to the standard template library (STL) in the `C++11`-standard.

The Message Passing Interface (MPI) [79] is an interface specification that allows multiple processes to communicate via messages. We distinguish between point-to-point communication among two processes and collective communication. An example for the latter is the `MPI_Broadcast` operation, where one process (referred to as *root*) broadcasts a data array to all processes in a pre-defined process group (referred to as *communicator*). MPI operations can be blocking or non-blocking. In the former case, affected processes

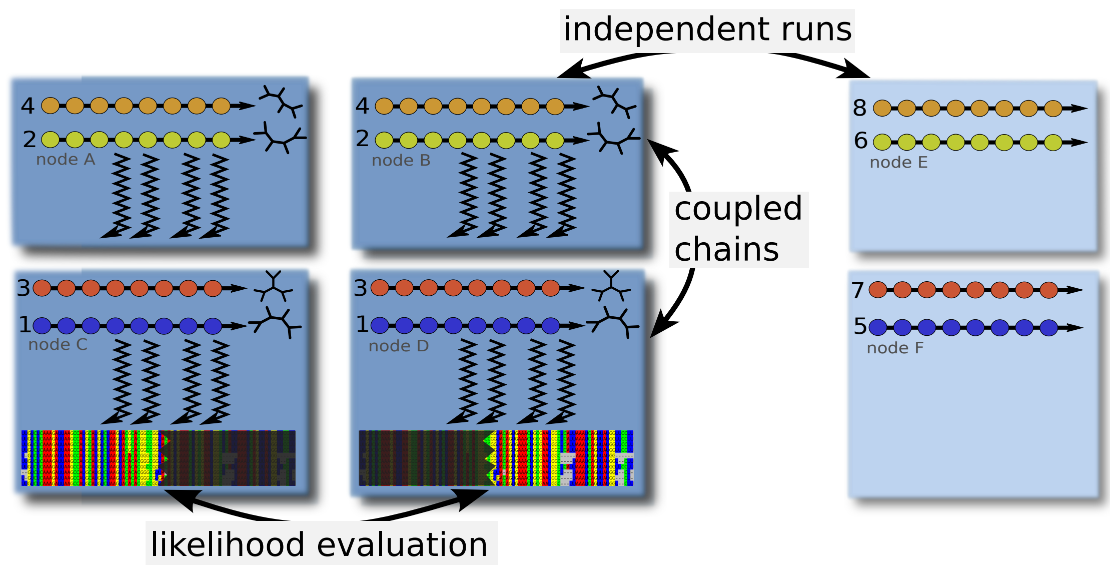


Figure 5.3: The three levels of parallelism implemented in ExaBayes. *Run-level parallelism:* PEs on computing nodes A-D execute one independent run; nodes E,F and other nodes (not shown) execute another independent run. *Chain-level:* nodes A and B simulate the chains with heat ids 2 and 4, while nodes C and D execute chains with heat ids 1 and 3. *Data-level:* All 4 PEs on one node evaluate the PLF and parsimony score for a part of alignment $\mathcal{A}$.

wait for the completion of a call (i.e., they possibly have to wait for their peer processes to send or receive data). In case of non-blocking communication, the sending as well as the receiving process initiate a communication step. Communication is then executed in the background (e.g., using a threaded implementation). Processes can check whether the communication has succeeded and either release memory resources needed for the send operation or process the received data. Notice that, collective non-blocking operations have not been introduced to the MPI standard until version 3.0 [80]. It is debatable, whether non-blocking collective operations would allow for more efficient communication: potentially, communication overhead is hidden within the MPI implementation, whereas the more complex implementation in ExaBayes (that is described later) is tailored for its use case and therefore more efficient. Thus, ExaBayes does not employ these operations, since in ExaBayes, we aimed for MPI-2 compatibility.

5.2.2 Parallelism in Phylogenetic BI

Phylogenetic BI is suited for parallelization at three distinct program layers (see **Fig. 5.3**). Several analyses can be started independently from each other. Such independent runs are mostly *embarrassingly parallel*, that is, there exist almost no dependencies among PEs. PEs only have to communicate in order to write checkpoints, to print status information and in order to check for topological convergence. We refer to this parallelism as *run-level* parallelism.

Chain-level parallelism specifies the concurrent execution of Metropolis-coupled chains (see **Sect.** 3.4). In other words, a group of PEs simulates one Markov chain out of several coupled chains. Naturally, there is a strong dependency among PE groups, since in order to execute a swap, a PE group has to receive the posterior value from its peer group working on the chain to be swapped with.

Finally, *data-level* parallelism means that several PEs contribute to the simulation of a chain. The runtime-intensive kernel to be parallelized is mainly the evaluation of the PLF and to a lesser extent the parsimony score evaluation (e.g., required by the parsSPR proposal). Here the dependencies among PEs are even more fine-grained: for instance, if we evaluate several possible reattachment positions for a ppSPR, a PE has to wait for all peer PEs that simulate the same chain in order to obtain the posterior density at the given attachment location.

We thus formulate our requirement to the parallelism in ExaBayes as follows: we want ExaBayes to be able to execute n independent analyses, where each analysis comprises m coupled chains. ExaBayes shall use an arbitrary number of PEs p and execute n' analyses in parallel, while for each analysis m' coupled chains are executed in parallel. This implies that we do not require p to be divisible by either m' or n'.

If p is a multiple of m and n, then we can use a Cartesian grid to model the assignment of runs and coupled chains to PE groups. For a three-tier parallelization, this means that we use a Cartesian cube, assigning three coordinates (x_1, x_2, x_3) to each PE group. Thus, all runs i with $(i \mod n) \equiv a$ are executed by PEs with $x_1 = a$ (where "mod" is the modulo operator). Furthermore, a chain j in run i with $(i \mod n) \equiv a$ and $(j \mod m) \equiv b$ is only computed by PEs with $(x_1 = a) \wedge (x_2 = b)$. Typically, the modulo operator is used to derive (x_1, x_2, x_3) from the r-th PE (i.e., its rank is r). However, this results in an assignment scheme, that does not favor locality: for example for $n' := 4$, rank 0 and rank 192 would be assigned $x_1 := 0$ and thus would contribute to the same independent runs, although it is likely that these PEs are executed on different computing nodes. Instead, we require an assignment that minimizes the rank-differences among these PEs that will communicate frequently.

Assume communicator C comprises several PEs with ranks $\{r_j\} = \{0, 1, \ldots, \text{size}(C) - 1\}$, where size$(C)$ denotes $\left|\{r_j\}\right|$. We want to split C into k new sub-communicators C'_i such that they obey the previously postulated locality property. The size of the new sub-communicators C'_i is determined as

$$\text{size}(C'_i) = \begin{cases} \lceil \frac{\text{size}(C)}{k} \rceil, & \text{if } i < k'; \\ \lfloor \frac{\text{size}(C)}{k} \rfloor, & \text{else;} \end{cases}$$

where $k' = (\text{size}(C) \mod k)$ is the number of PEs that obtain one additional PE. Given size(C'_i) and k', we can now compute the assignment of rank r_j to communicator id i. When we split a communicator, id i is referred to as *color*. We can determine the new rank r'_j of a PE with current rank r_j within the new sub-communicator C'_i as

$$r'_j = \begin{cases} \frac{r_j}{\text{size}(C'_i)}, & \text{if } r_j < k \cdot \left\lceil \frac{\text{size}(C)}{k} \right\rceil; \\ k' + \left(r_j - k' \cdot \left\lceil \frac{\text{size}(C)}{k} \right\rceil \right) \cdot \left(\left\lfloor \frac{k}{\text{size}(C)} \right\rfloor \right)^{-1}, & \text{else.} \end{cases}$$

r_j	$\to$ (x_1, r'_j)	$\to$ (x_1, x_2, x_3)
0	$(0,0)$	$(0,0,0)$
1	$(0,1)$	$(0,0,1)$
2	$(0,2)$	$(0,1,0)$
3	$(0,3)$	$(0,1,1)$
4	$(0,4)$	$(0,2,0)$
5	$(1,0)$	$(1,0,0)$
6	$(1,1)$	$(1,0,1)$
7	$(1,2)$	$(1,1,0)$
8	$(1,3)$	$(1,1,1)$
9	$(1,4)$	$(1,2,0)$
10	$(2,0)$	$(2,0,0)$
11	$(2,1)$	$(2,0,1)$
12	$(2,2)$	$(2,1,0)$
13	$(2,3)$	$(2,2,0)$

Table 5.2: Example of recursive splitting of a communicator C with 14 ranks (named r_j) for the parallel execution of 3 independent runs with 3 coupled chains (also executed in parallel). We first split into 3 communicators C'_{x_1} (for three distinct independent chains), then the ranks r'_j of the new sub-communicators are used to split each of the 3 sub-communicators into 3 sub-sub-communicators (each for a coupled chain in an independent run). Thus, for instance, $(2,2,0)$ is the 0-th (and only) PE that computes all chains l of runs k with $(l \mod 3 \equiv 2) \wedge (k \mod 3 \equiv 2)$.

The given splitting scheme ensures that remainder ranks (in case of division with remainder) are first assigned to sub-communicators with ids $i = 0$, then $i = 1$ and so on.

Using the scheme described above, we can split the initial communicator (in case we solely use MPI this is `MPI_COMM_WORLD`) with ranks r_j into sub-communicators C'_{x_1}. In other words, we map r_j to (x_1, r'_j), where x_1 is the color of the rank and r'_j is the rank within the new sub-communicator. We repeat the split step and map ranks r'_j to (x_2, r''_j). Thus, recursive splitting of the initial communicator yields a tuple (x_1, x_2, x_3) with $x_3 = r''_j$ that specifies that a PE is the x_3-th PE that computes all chains assigned to x_2 in all independent runs that are assigned to x_1.

Consider **Tab. 5.2** for an example: here we split a communicator with 14 ranks into 3 PE groups that are each further sub-divided into 3 PE sub-groups. Notice that, multiples of 9 PEs would be optimal for the given example. Yet, our two-step splitting scheme only yields a slight imbalance, since some chains are computed by 2 PEs, while other chains are only computed by one PE.

5.2.3 Implementation in ExaBayes

Decentralized Parallelism

ExaBayes features a MPI and thread-based hybrid parallelization, that is, for instantiating n PEs, we can start k MPI processes (possibly on compute nodes that do not share memory) and each process subsequently starts l processes, such that $n = k \cdot l$. In order to retain maximum flexibility of the threaded implementation, we allow threads to behave like processes, that is, the shared memory among processes is reduced to message queues for exchanging information to peer threads. In other words, the globally decentralized parallelization scheme also extends to thread parallelism. We call the parallelization in ExaBayes *decentralized*, since every PE essentially conducts the simulation of all chains and all runs assigned to it independently and there exists no central instance that orchestrates the inference.

A popular counterpart to decentralized parallelism is the master-worker approach: here a single PE is declared the master and activates busily waiting worker PEs. For instance, in the thread parallelizations of RAxML [108] or RAxML-Light [111] this scheme is used efficiently to evaluate the PLF or calculate the derivatives of the PLF for the NR optimization of the branch lengths. However, in the MPI version of RAxML-Light, this master-worker scheme induced a substantial communication overhead with a noticeable negative impact on parallel efficiency. Thus, for large-scale phylogenetic ML inferences, ExaML was introduced that implements a decentralized parallelization scheme [110].

See **Fig. 5.4** for a UML class diagram representing the hybrid parallelization. `ParallelSetup` is at the root of the object tree and essentially is a singleton in that it is instantiated only once and passed as reference where needed. `ParallelSetup` holds three communicators. A global communicator allows communication with all PEs that have been started. Furthermore, there is a run-specific communicator for communication with all PEs that share the same x_1 (resp. compute the same runs) and a chain-specific communicator for the communication with all PEs with common x_1 and common x_2. The latter communicator is used wherever multiple PEs evaluate the PLF or parsimony score simultaneously, or when derivatives of the PLF are computed concurrently for branch length optimization.

Hybrid Parallelism

`IncompleteMesh` implements the mapping of the rank of the global communicator r_j to the Cartesian coordinates (x_1, x_2, x_3) (see **Sect.** 5.2.2). `ThreadResource` allows a process to initialize threads. When a thread is started, an instance of `ParallelSetup` is created by the initial process and passed to all threads that are started subsequently. This means that this instance and its members are the sole regions of the program where memory areas are shared among threads. Thus, the initial process only sets up `ParallelSetup`, initializes MPI (a static method in `ParallelSetup`) and initializes the unique log file. Afterwards, threads are started and behave like MPI processes. This also applies to the splitting of communicators: in the abstraction `Communicator`, each

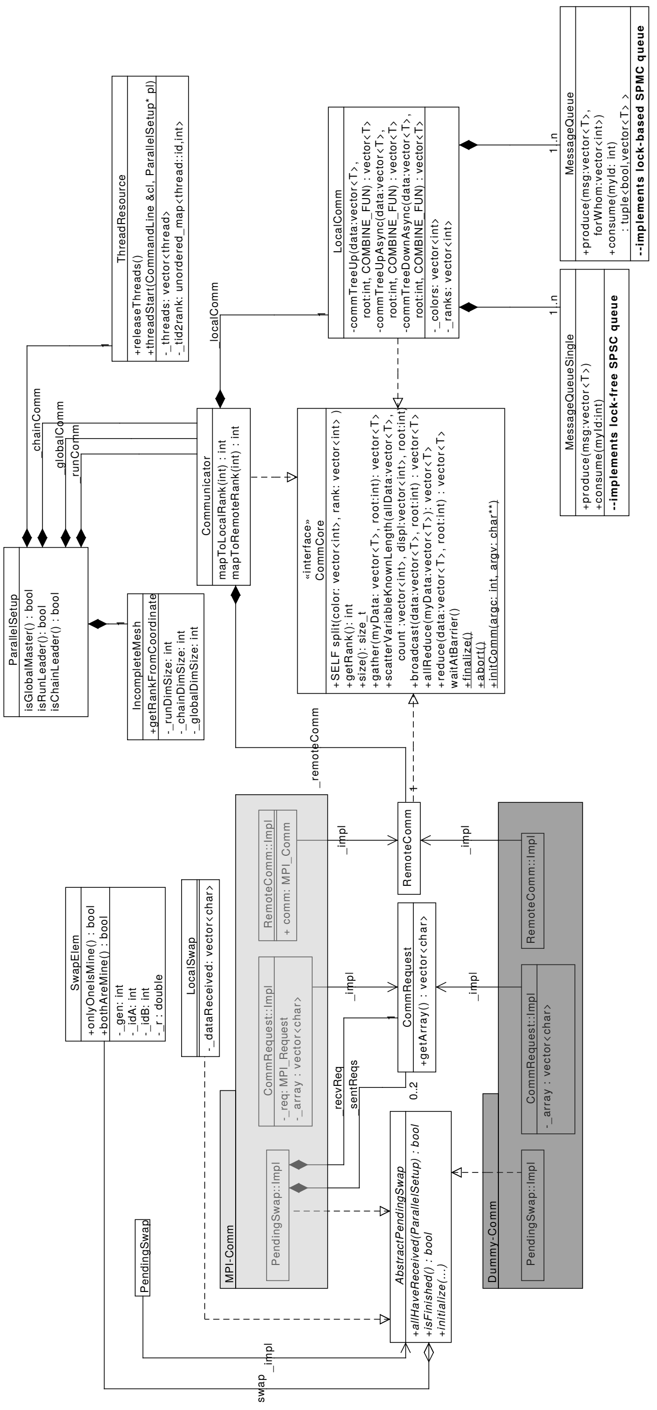


Figure 5.4: Class diagram illustrating the parallelization scheme as of ExaBayes, v. 1.3.

thread is a distinct PE with rank r_j that is determined from the MPI rank and its thread-specific rank stored in `ThreadResource`.

In ExaBayes, the `C` interface to the MPI implementation is wrapped in `RemoteComm` (notice that a `C++` interface specification has been deprecated as of version 2.2 of the MPI standard). `LocalComm` on the other hand represents a thread-level communicator. All three communicator classes have to implement a common interface that provides essential parallel operations such as the previously mentioned *broadcast* operation. The methods in the respective interface `CommCore` are template methods, that is, the operations can be performed on various types (e.g., `int` or `double`) and template arguments are then mapped to wrapper classes from which the respective MPI constant (e.g., `MPI_INT` or `MPI_DOUBLE`) can be retrieved. While this increases type safety, the natural choice of implementing `CommCore` as an abstract class is prohibited, since template methods can not be declared `virtual`. In other words, combining run-time with compile-time polymorphism would be required here, but is not available in `C++`.

`Communicator` C has an instance of `RemoteComm` R and an instance of `LocalComm` named L. Thus, we can compose operations on all PEs in a `Communicator` C using operations on R and L. For example consider the `Allreduce` operation, which for ExaBayes is one of the most important operations: `Allreduce` is a blocking collective call that reduces a vector with respect to an arithmetic operation such that the result is available at every PE in the communicator. The `Allreduce` operation is essential for the data-level parallelism: several PEs that evaluate the parsimony score or likelihood of a tree all obtain the same value and are able to continue the simulation based upon the result that has been computed in parallel. `Communicator` implements its `Allreduce` by (i) reducing the vector in its `LocalComm`, (ii) an `Allreduce` in `RemoteComm` (only one thread of `LocalComm` participates in this call) and (iii) broadcasts the result to all members of the `LocalComm`.

Notice that there exists an important constraint to the hybrid parallelism implemented in ExaBayes: all threads started by a process must *either* be part of the same communicator C *or* if threads started by a process belong to several communicators C_k, then no threads started by any further process can be part of any of the communicators C_k. This restriction has few consequences for productive use of ExaBayes, since the thread-level parallelism mostly is implemented as an alternative to MPI parallelism (for systems, where no MPI implementation is available). Thus, the instances of either `LocalComm` or `RemoteComm` typically are trivial (i.e., there is only one process or processes do not start threads).

Modular Compilation

The ExaBayes software package includes a purely threaded binary executable and several helper programs (e.g., for computing consensus trees). These post-processing tools employ the same code basis as the ExaBayes executable. Thus, it is desirable to be able to compile generic ExaBayes code into a static library that can then be linked with application-specific code into the respective executable. Application-specific code of – for instance – the `consense` program (for computing consensus trees) only consists of command line interpretation and library calls to the ExaBayes-library (e.g., a class `ConsensusTree` is instantiated). In ExaBayes, essentially all code that needs MPI support is encapsulated by a few classes of which `RemoteComm`, `CommRequest` and `AbstractPendingSwap` are represented in **Fig. 5.4** (for more information about the functionality of the classes, see **Sect.** 5.2.5). ExaBayes uses the pointer-to-implementation (PImpl) idiom to specify an interface that hides the encapsulated MPI functionality. We provide a MPI-capable implementation in the compilation module `MPI-Comm` (that has been compiled using the MPI compiler) as well as a dummy module `Dummy-Comm` (see **Fig. 5.4**). The latter implements the sequential equivalent of the requested operations. For instance, the dummy `Allreduce` simply returns the input array. Thus, we can create MPI-capable executables by statically linking the MPI compilation unit (and by using MPI linker flags) and obtain an MPI-less executable by statically linking the dummy module.

In the PImpl idiom, a wrapper class essentially only forwards all calls to its implementation (which in our case can either be provided by the dummy module or by the MPI module). This approach comes with a slight performance disadvantage that is caused by the indirection. Also, methods provided by the implementation can not be inlined, since they are part of different compilation units.

Implementation of Thread Parallelism

`LocalComm` implements thread parallelism in ExaBayes and emulates communication among processes (as e.g., `MPI_COMM`). A simple pattern for the communication among two threads is a message queue. A message queue is a *thread-safe* data structure, that is, methods that extract the state or modify the state of an instance must work consistently under concurrent access. In other words, we guarantee that a modification on the data structure is propagated to all threads and threads do not operate on an outdated version of the data.

Essentially, message queues are queue data structures (i.e., messages are processed in the order that they have entered the structure). A producer can add messages into the queue, consumers extract information. Message queues are characterized by the number of producers and consumers. ExaBayes employs two different kind of messages queues: single producer single consumer (SPSC) queues and single producer multiple consumer (SPMC) queues. In the simplest case, we implement parallelized access to data structures by protecting critical operations via locks (e.g., `std::mutex` in `C++11`). If concurrent access to data structures does not employ locks, we call the data structure *lock-free*. Lock-free queues guarantee system-wide progress. That is, according to the strict definition of system-wide progress [49], we are guaranteed that for an infinite number of times, the operation on the data structure finishes after a finite number of steps. In other words, there exists at least one thread that makes progress. *Wait-freedom* has the even stronger guarantee, that every thread makes progress (w.r.t. the access call) within a finite number of steps [49]. As we guarantee stronger properties progress, the implementation complexity increases.

As a thread-to-thread communication pipeline, ExaBayes implements a lock-free

SPSC queue, since for the SPSC case, a lock-free implementation is straight-forward [43]. Class `MessageQueueSingle` implements such a lock-free SPSC queue. For n threads `LocalComm` needs to instantiate $\frac{1}{2}(n^2 - n)$ pipelines (i.e., instances of `MessageQueue-Single`). Lock-freedom can yield advantageous performance for fine-grain parallel tasks (such as the data-level parallelism in BI). *Lock-freedom* of SPMC queues is substantially more complex. Thus, ExaBayes implements a blocking SPMC queue (i.e., a lock is employed) in `MessageQueue`. A `LocalComm` first instantiates several SPMC queues. ExaBayes then uses these queues to emulate asynchronous communication similar to the request-based system in MPI (e.g. `MPI_Request`, for details see **Sect.** 5.2.5).

ExaBayes uses the point-to-point pipelines provided by the lock-free SPSC queues to implement collective operations. For instance, for a straight-forward emulation of a `MPI_Broadcast`, the root thread can post a message to the queue of each participating peer thread. In this case, we would require $(n - 1)$ sequential communication steps for n participating threads. To improve efficiency, `LocalComm` implements communication trees. This means that messages are propagated in a tree-like manner: the root thread posts a message into the queue of threads 1 and 2 using the pipeline from the root to thread 1, respectively from the root to thread 2. After receiving the data, thread 1 posts to threads 3 and 4, while, at the same time, thread 2 posts to threads 5 and 6. In the ideal case, a communication tree takes the equivalent of $\lceil \log_2(n) \rceil$ sequential communication steps. We call the concrete example a *downward communication tree*, since information flows from the root to external nodes.

Alternatively, messages are propagated from the external nodes towards the root via an *upward communication tree.* For instance, in a *gather* operation, each thread contributes a fraction of an array that is concatenated at the root. Here, the thread with rank 1 checks its pipelines for incoming messages from ranks 3 and 4. If both messages are received, the messages are concatenated and are posted into the pipeline from thread 1 to the root.

`LocalComm` provides generic upward and downward communication trees that take a function f as input. Function f specifies, how communicated messages are transformed. Thus, a broadcast for instance is implemented as a downward communication tree (i.e., information is propagated from the root to peer threads) and f is a trivial function that returns the input data. A gather operation on the other hand is implemented as an upward tree and f concatenates messages. To reduce code complexity, the important `Allreduce` operation is implemented as a `Reduce` with a subsequent `Broadcast` operation, although more involved algorithms exist that are more runtime-efficient depending on hardware latency and message size [116].

5.2.4 Evaluation of MPI-based Data-Level Parallelism

In the following, we evaluate the MPI-based data-level parallelism implemented in ExaBayes. The correctness of the thread- and MPI-parallelizations implemented in ExaBayes can typically easily be verified given the reproducibility of ExaBayes runs. In most instances, ExaBayes simulates the exact same chain (or chains in case of MC^3) for a given random number seed independent of the number of processes or the parallel configuration (i.e., any assignments (x_1, x_2, x_3)) that is employed. However, data-level parallelism changes the order for adding up per-site log-likelihoods to the final likelihood of a tree. Thus, a different number of processes can sometimes induce altered floating point rounding errors that can lead to divergences (i.e., while this does not violate the correctness of the result, we will not simulate the exact same chain). For sufficient load (e.g., every PE is assigned ~300 characters of an alignment), we did not observe performance differences between the MPI and thread parallelism. For small alignments and many PEs however, the thread implementation performs worse than MPI. In contrast to the internal MPI implementation, the thread implementation in `LocalComm` is not optimized for non-uniform memory access (NUMA) architectures. Potentially, performance under a master-worker scheme for threads could achieve higher efficiency, however this conflicts with the decentralized parallelization approach used in ExaBayes. Thus, in this Section and **Sect.** 5.2.5, we will not evaluate the efficiency of thread parallelism at data-level or chain-level.

For evaluation, we sub-sampled a simulated whole-genome alignment (see **Sect.** 5.5) to obtain a dataset D^{5e5} with 500,000 bp, a dataset D^{1e6} with 1,000,000 bp and a dataset D^{5e6} with 5,000,000 bp. All datasets comprise 200 taxa. The memory requirements of BI for these datasets are roughly 24 giga byte (GB), 48 GB and 240 GB, respectively (see **Sect.** 5.4 for the calculation of memory requirements). We measured the strong scaling capabilities of ExaBayes, that is we increased the number of processors while the problem instance (i.e., alignment size) was kept fixed. We quantify the strong scaling behavior of ExaBayes using *parallel speedup* and *parallel efficiency* metrics. Parallel speedup is defined as the ratio of the execution time of the fastest sequential implementation (fT_1) and the execution time with n processes (T_n). Parallel efficiency measures whether speedup increases proportionally to the number of processes employed (it is thus defined as the speedup divided by the number of processes).

We examined the scaling behavior of ExaBayes on AMD Magny-Cours Opteron nodes. A node contains 4 multi-core CPUs with 12 cores each and is equipped with 128 GB random access memory (RAM). We ran a chain using an increasing number of processes for 3,000 generations on dataset D^{1e6} and using default MCMC parameters with the SSE version of ExaBayes. For this experiment, we focus on D^{1e6}, since this is the largest dataset of the available datasets that can be processed using the sequential version of ExaBayes to determine ft_1 as the fastest known reference (for DNA data and with double-precision, see **Sect.** 5.1.2). In **Fig. 5.5.a**, we observe a super-linear speedup for 2 and 4 processes compared to the sequential runtime. The peak parallel efficiency for D^{1e6} on the Magny-Cours nodes is 110.3%. As we increase the number of processes, the parallel efficiency decreases to 43.1% for 48 processes (i.e., full utilization of a single compute node). If we increase the number of processes to 96, 192, 288 and 384 (which corresponds to 2, 4, 6 or 8 entire compute nodes), the parallel efficiency recovers and reaches 58.4% in the best case (for 288 processes). On dataset D^{1e6}, ExaBayes achieves a maximum parallel speedup of 208.1× for 384 processes.

The runtimes for an increasing number of processes in **Fig. 5.5.a** indicate that parallel efficiency on data-level in phylogenetic BI is primarily memory-bound. The initial super-linear speedup occurs because of increased cache efficiency (i.e., a larger total of cache

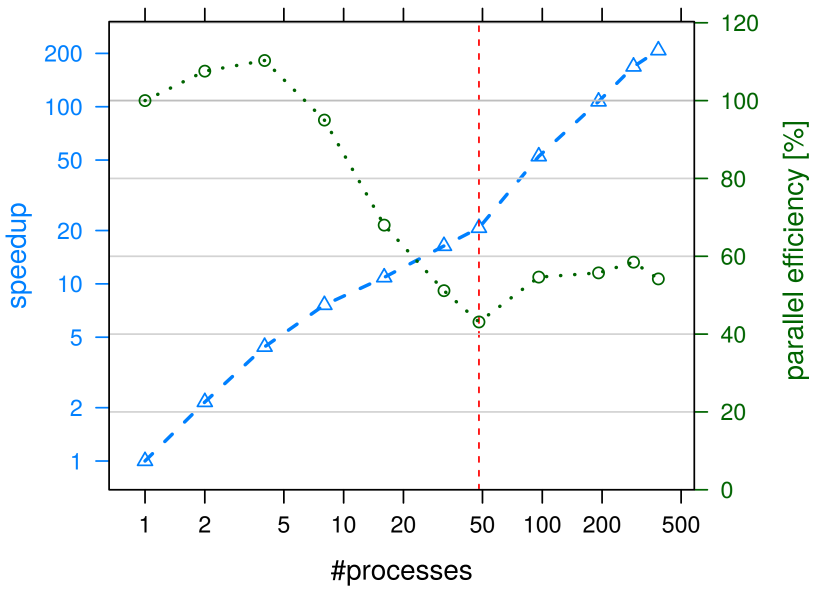


Figure 5.5.a: Parallel speedup and parallel efficiency for 3,000 generations of a chain on a dataset with 200 taxa and 1,000,000 bp. Starting from the sequential case (1 process), the number of processes on the x-axis increases in powers of 2 until a full node is utilized (48 processes, indicated by dashed red line). Afterwards, several computing nodes are employed and a total of 96, 192, 288 and 384 processes are used.

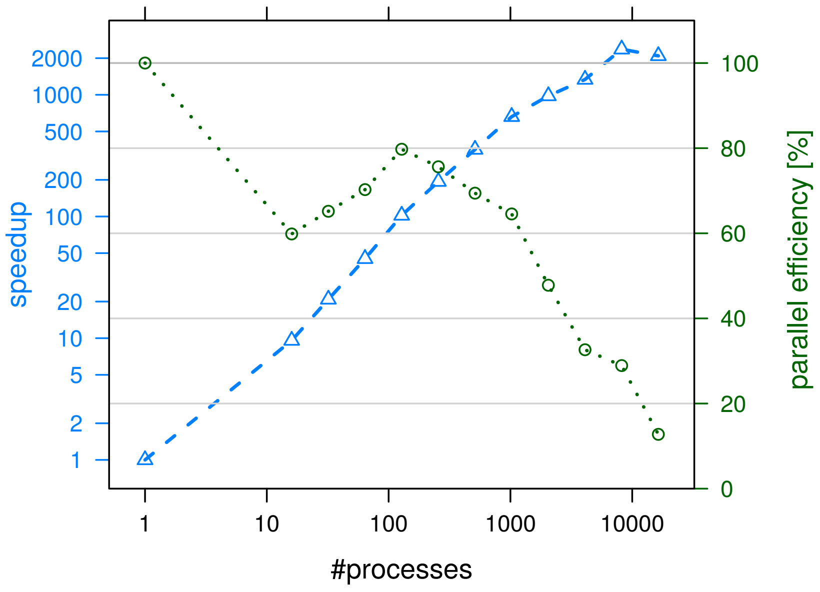


Figure 5.5.b: Parallel speedup and parallel efficiency for 100,000 generations of a chain on a dataset with 200 taxa and 500,000 bp. Aside from the sequential case, the number of processes are powers of 2 (starting at 16, ending at 16,384) on a logarithmic x-scale.

memory reduces the number of read/write accesses involving the RAM). As the number of processes that are executed on a single computing node increases, RAM bandwidth becomes the major bottleneck for competing CPU cores and cancels out any gains in cache efficiency. When we use several computing nodes, the total memory bandwidth increases and leads to the observed improvement in parallel efficiency. Eventually, parallel efficiency decreases for a second time when communication overhead kicks in. The inevitable limits to the scalability of any parallelized code are given by *Amdahl's law* [9]: Assuming that the execution time for parallelized regions of the code can be reduced almost arbitrarily (given enough processors), the execution time for sequential parts remains constant. Thus, sequential execution time leads to hard limits on the maximum speedup that can be achieved for a parallel code.

We considered a second computer architecture for the evaluation of data-level parallelism: computing nodes of the SuperMUC supercomputer that consist of 4 AVX-capable Intel Sandy Bridge processors with 4 cores each (i.e., 16 processes per node in total; 32 GB RAM per node, whereas 24 GB are typically available to users). For determining runtimes, we ran one chain for 100,000 generations using default settings on the SuperMUC supercomputer. Here, D^{5e5} is the largest alignment that can still be computed on a single SuperMUC computing node. For D^{5e6}, we instead report *scaling factors/efficiency* that indicate how well the code scales compared to reference cases that are already executed on several computing nodes. We measured parallel execution times for numbers of processes that are a power of two (from 16 to 16,384 for D^{5e5} and from 256 to 32,768 for D^{5e6}).

Fig. 5.5.b shows parallel speedup and efficiency for a single chain on D^{5e5} and for numbers of processors that are powers of 2 (from 16 to 16,384). We obtain a maximum parallel speedup of 2,368 with 8,192 processes. In absolute terms this means that the runtime could be reduced from 103,191 seconds (sec) ($\approx$ 1 day, 4 hours) in the sequential case to 43.57 sec with more than 8,000 processors. The runtime can not be decreased beyond this point, since with 16,384 processors, each processor works on less than 31 characters and thus communication overhead dominates runtime (resp., *Amdahl's law* applies). For **Fig. 5.5.b**, we did not examine single-node scaling behavior, but we expect that for 2 or 4 processes we would observe an initial increase in parallel efficiency before a parallel efficiency decrease for the full computing node (as in **Fig. 5.5.a**). Here, parallel efficiency peaks at 128 processors and decreases afterwards because of communication overhead. Thus, with 128 processors the runtime can be reduced to $\approx 1,010$ sec, while the waste of CPU cycles because of communication overhead is minimal.

Memory requirements of large alignments often exceed the available main memory on a single computing node. In these instances, it is not possible to determine the sequential runtime fT_1. Instead, we use a $T_{n'}$ (with $n' < n$) as reference and report *scaling factors* and *scaling efficiency* accordingly. For **Fig. 5.6**, we ran a chain on the 200-taxa alignment with 5,000,000 bp (one order of magnitude larger than the aforementioned dataset). The increased size of the alignment allows the code to scale up to more than 32,000 processes achieving a scaling factor of 42.5× compared to using 256 processes. For the smaller dataset (i.e., $5 \cdot 10^5$ bp), the scaling efficiency peaks at 1,024 processes. Since we can not determine the sequential runtime of the code, the initial rise in scaling

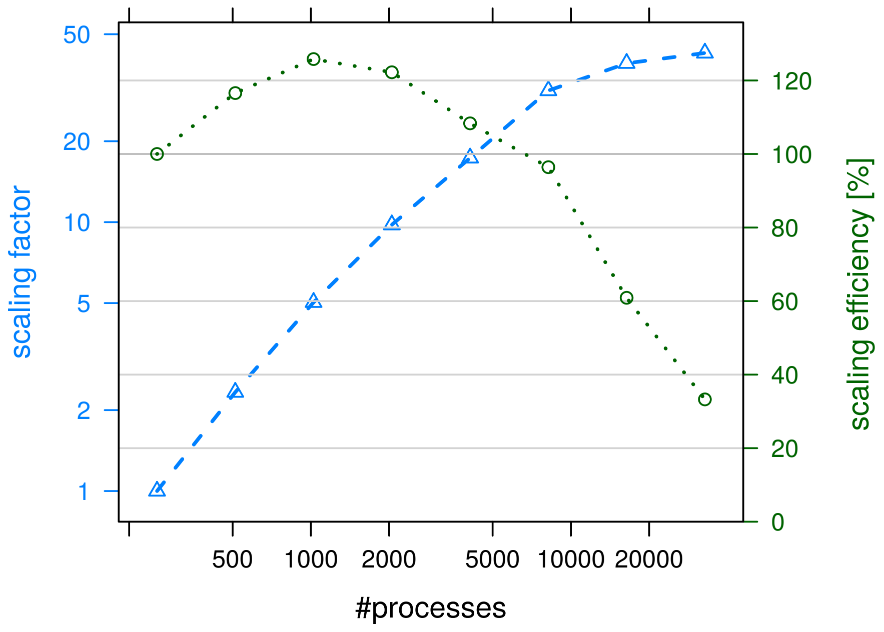


Figure 5.6: Scaling Efficiency (on log-scale) and scaling efficiency for 100,000 generations of a chain on a dataset with 200 taxa and 5,000,000 bp. The number of processes starts with 16 processes and doubles at every data point up until 32,768 processes.

efficiency (which corresponds to the second peak of parallel efficiency for multi-node runtimes) here leads to super-linear scaling.

5.2.5 Chain-Level Parallelism

Blocking Implementation

For parallelization of the MC^3 procedure (see **Sect.** 3.4), ExaBayes extends the chain-level parallelization introduced in MrBayes [7] for the case where several PEs jointly simulate a single chain: each PE instantiates an analysis-specific PRNG and can thus precompute all swap attempts at any generation number for the entire analysis. A swap attempt is sufficiently defined as a tuple $S = (c_i, c_j, r, g)$ (represented by class `SwapElem` in **Fig. 5.4**), where c_i and c_j are the ids of the involved chains i and j. Furthermore, r is a uniform number in $[0, 1)$ used for determining acceptance and g is the generation in which i and j attempt a swap. PEs compute generations of all chains assigned to them. Since swaps occur at random in ExaBayes, there is a higher chance, that there is a window, where several consecutive generations can be computed without the need to swap.

If a PE does not simulate any chain i or j of a swap S, then S can be ignored by the PE. We refer to *trivial* swaps, when i and j both are assigned to the respective PE. Thus, trivial swaps can be executed without any interaction with other PEs and are not different to the sequential MC^3 procedure. *Remote* swaps are swaps for which one local chain swaps with a chain j that is simulated by a different set of PEs. Here, the respective PE does not have any information about the state of j available. Remote swaps are problematic for parallel efficiency, since a set of PEs working on one chain has to wait for the set of PEs working on the other chain to complete. For a remote swap attempt in the original parallel algorithm, the two processes (simulating one of the chains involved in the swap) exchange the PP and their heat id value. If the swap is accepted (according to r), the identity (i.e., the heat id i used for calculation of heat β_i, see **Sect.** 3.4) of the remote chain is assumed. With the introduction of tuned proposals, chains have to swap the entire set of proposal states including observed acceptance probability counters, which drastically increases the amount of information to be communicated among processes.

In version 1.2.1, ExaBayes provides a blocking and a non-blocking implementation of MC^3. Note that, **Fig. 5.4** only depicts the non-blocking version of the MC^3 algorithm. The blocking algorithm was removed from subsequent versions of ExaBayes and thus no threaded implementation exists.

The blocking MPI implementation of chain-level parallelism employs `Intercomms` instead of default MPI communicators (both represented by `MPI_Comm`) which by default are intra-communicators. Given two distinct groups of processes with coordinates (x_1, x_2, x_3) that are either part of the same communicator C_k, respectively C_l. If processes in C_k and C_l contribute to the same run (i.e., share x_1) and m chains are executed in parallel, then we have to create an intercomm I_{lk} for communication between processes in C_k and processes in C_l. All processes in C_k with $x_2 = i$ create $(m - 1)$ intercomms with all process groups with $x_2 \neq i$. The intercomm needs a designated root in the remote communicator C_l, which is set to $(x_1, l, 0)$ in our case. Given an established intercomm I_{ij}, the implementation of blocking MC^3 is straight-forward: assume processes C_k simulate chain i and processes C_l simulate j, where $i < j$. Then processes in C_k first use their intercomm I_{ij} to broadcast a serialized version of the chain (i.e., heat id, PP and proposal parameters) to processes in C_l. After the blocking broadcast has succeeded, the C_k receive a broadcast from C_l containing the serialized version of chain j.

Non-blocking Implementation

As an alternative to the blocking implementation, ExaBayes introduces a non-blocking algorithm for MC^3 (using threads and/or processes). The central difference between both algorithms is illustrated in **Fig. 5.7**: in the blocking version and for a remote swap, PEs have to wait for PEs that simulate the remote chain. In the non-blocking algorithm on the other side, processes initiate a request for communication (i.e., instantiate a `CommRequest`) and can proceed to compute generations of other chains that are assigned to them (e.g., the blue and yellow chain that are assigned to process group 1). PEs can later check, whether their request has been fulfilled and can continue to simulate the potentially swapped chain, if information about the remote chain has been transmitted. Notice that, waiting times still may occur, if one of the chains runs out of work. In principle, this scheme can be extended to the run-level as well: in case PEs are waiting for remote chains to finish, these PEs could continue to simulate chains of other independent runs. However, when chains that form an independent run are not being simulated (i.e., suspended), CPVs are deallocated to reduce the overall memory footprint. Thus, this extension of the non-blocking algorithm would also substantially increase memory requirements.

For the MPI and threaded implementation, a PE creates an instance of `PendingSwap` (see **Fig. 5.4**). As explained in **Sect.** 5.1.1, we here employ the PImpl idiom and provide either a MPI based implementation or a dummy implementation for a trivial `RemoteComm`. Assume we are using MPI and that processes in sub-communicator C_k simulate chain i and processes in C_l simulate chain j, then for a swap $S = (i, j, r, g)$, each process with coordinates (x_1, i, x_3) has to exchange information with the process that has coordinates (x_1, j, x_3). In other words, each process executes the swap with the process that has the same rank (i.e., x_3 coordinate) in the remote process group. If $\text{size}(C_i) < \text{size}(C_j)$, then processes (x_1, i, x_3) also send chain information via a `CommRequest` to processes $(x_1, i, x_3 + \text{size}(C_i))$, if such a process exists. Processes that do not have a peer in the remote process group do not send chain information. Using this asynchronous communication scheme, processes can continue to simulate other chains that are assigned to them and are not blocked, in case a remote chain has progressed to the generation in which the swap occurs.

However, synchronization among processes simulating the same chain still is necessary. While some processes in a communicator C_i may have already received information from processes in C_j, it is essential for forthcoming operations (e.g., PLF evaluations) that all peers in C_j have received the respective information and all processes in C_j continue to evaluate the same consistent chain. For this, an `Allreduce` operation is necessary.

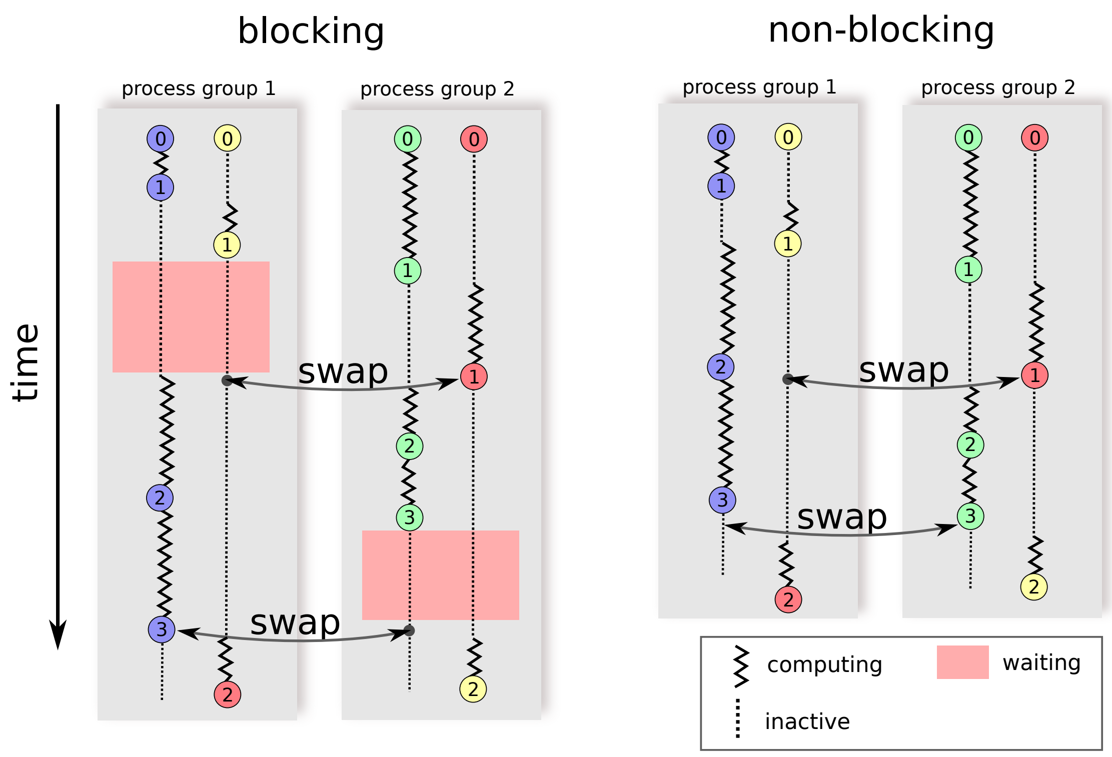


Figure 5.7: Comparison between the blocking (*left*) and the non-blocking (*right*) algorithm for MC^3. Two distinct groups of processes compute a total of 4 coupled chains (i.e., two chains are computed in parallel and 2 chains are assigned to each group). The color of a chain represents its heat id (e.g., blue chain is the non-heated chain), numbers inside circles indicate the generation of the respective chain. In this scenario, the yellow and the red chain successfully swap after generation 1 and the green and the blue chain attempt a swap after generation 3.

Thus, after computing as many generations as possible for assigned chains, all processes in a communicator enter a phase where they check which requests have been resolved for all peers in the same communicator and then execute the respective swaps. This step allows processes to make a synchronized decision on which chain to execute next.

`LocalSwap` is a realization of `AbstractPendingSwap` that emulates a similar behavior for the thread-based implementation. For the implementation of non-blocking communication, MPI employs tags. That is, processes send tagged messages and receiving processes check for messages by tag. Similarly, for the thread-level implementation, we can use the SPMC message queue (class `MessageQueue` in **Fig. 5.4**) to post messages with tags. Note that, the most efficient way to map integers $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is the Cantor pairing function

$$\text{pair}(a, b) = \frac{1}{2} \cdot (a + b) \cdot (a + b + 1) + b. \qquad (5.1)$$

For the MPI implementation, if processes with ranks r_i and r_j have to swap chains a and b, the respective processes can use the tags $\text{pair}(r_i, \text{pair}(a, b))$ and $\text{pair}(r_j, \text{pair}(a, b))$ for sending and receiving messages.

On the other hand, when a thread communicator C_k initiates a swap, the root of C_k (i.e., the thread with coordinates $(x_1, k, 0)$) posts a messages with tag $\text{pair}(i, j)$ into the run-specific message queue. The equivalent for checking, whether all processes have received the remote chain information is that all threads in C_k check for a message with tag $\text{pair}(j, i)$. When this message has been consumed by each thread in C_k, the swap has been executed and we can continue simulating the potentially swapped chain. Thus, for synchronization, the thread-implementation also needs an `Allreduce` over the threads in C_k in order to assure that all threads have received the remote chain information.

Notice that, our definition of a SPMC queue slightly deviates from how SPMC message queues are commonly defined. Typically, a SPMC means that many consumer threads can dequeue messages that have been produced by a single producer thread and it is not important how many messages are dequeued by which consumer. In our case, we need to make sure that each consumer has read the message exactly once, before dequeuing it.

Evaluation of MPI-based implementation

A straight-forward way of demonstrating scalability of chain-level parallelism is to analyze its *weak-scaling* properties (e.g. as performed in [7]). Weak scaling means that the amount of work per process is kept fix (i.e., each process is assigned one of the n coupled chains) and we examine the ratio of sequential runtime (employing a single chain) to the runtime of p processes working on p chains (where one swap is attempted once all chains have progressed by 1 generation). As illustrated in **Fig. 5.8.a** on a dataset with 150 taxa and 1,269 bp, the implementation in ExaBayes has a minor speed advantage over MrBayes, mostly because of the aforementioned modification to draw the number of swaps per generation at random. While linear scaling could be demonstrated for MrBayes [7], in our experiment neither MrBayes nor ExaBayes exhibit linear scaling. We assume this is because of the substantially higher communication overhead that is necessary for exchanging chains with tuned proposals. In **Fig. 5.8.a**, we also included runtimes for the case that 2 processes are assigned to each chain. For 2 processes per chain, the runtime is close to 2× as fast (however, scaling efficiency is lost because of the increased need for synchronization among processes).

A major short-coming of this weak scaling plot is, that the number of swap attempts is limited to an average of one per generation. The simple addition of heated chains to a set of coupled chains with a fixed heating scheme and a fixed number of swaps can be highly detrimental to overall performance [11] (aside from increasing runtime). For instance, with 32 coupled chains, 1 swap attempt per generation, and the default heating scheme (as also employed in MrBayes), a cold chain was only involved in 92 successful swap attempts during 50,000 generations. Thus, it becomes increasingly unlikely that any information from hotter chains is propagated to the cold chain.

Thus, in **Fig. 5.8.b**, we compare the scaling efficiency of 16 coupled chains running on 32 processes for increasing granularity of chain-level parallelism and for an increasing number of swap attempts per generation. As expected, scaling efficiency decreases as the number of swap attempts per generation is increased. We have chosen a small simulated dataset that comprises 200 taxa and 5,000 characters. Since this dataset is too small for 16 processes, we observe sub-optimal parallel efficiency at the left-hand side of **Fig. 5.8.b**, where only data-level parallelism is employed (i.e., all 16 processes evaluate a part of the alignment and no remote swaps occur). As more and more coupled chains are executed in parallel (to the point where each chain is executed in parallel by 2 processes at the right-hand side of **Fig. 5.8.b**), efficiency of data-level parallelism increases, since every process has a sufficient amount of PLF work. With increased chain-parallelism however, processes spend more time waiting for remote chains to complete.

This effect is alleviated by the non-blocking implementation. In the best case, the non-blocking version improves the parallel efficiency by 10.3%. In absolute terms this means that the non-blocking version reduces the runtime by 18.6%, a difference that easily accumulates to several hundred CPU hours for average phylogenomic datasets. The non-blocking implementation consistently outperforms the blocking version, even in extreme cases such as with 16 swap attempts per generations (i.e., each chain is expected to swap 2 times per generation).

5.3 Load Balancing

5.3.1 Partitioned Analyses in ExaBayes-1.2.1

Improvements

As mentioned in **Sect.** 2.6.2, alignments can be subdivided into partitions where distinct evolutionary models are assigned to each partition. When p PEs are employed for data-level parallelism, we usually employ a cyclic data distribution (CDD) scheme to assign every p-th character of a partition to PE i using an offset of i. PEs evaluate their portion of the data and communicate their partial result to peer PEs to determine the tree likelihood. Aside from the compute-intensive parallelized evaluation of the likelihood, preprocessing steps are necessary to determine in which order CPVs need to

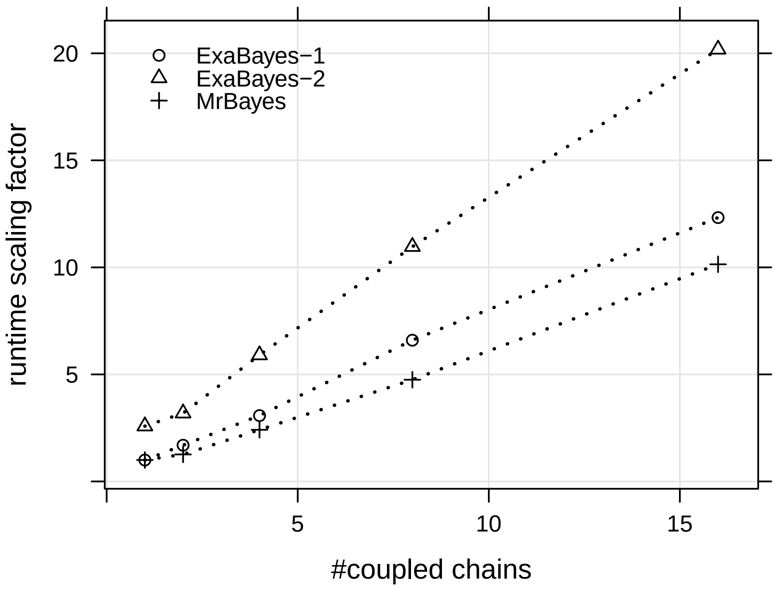


Figure 5.8.a: Comparison of weak scaling at chain-level between ExaBayes and MrBayes. Number of coupled chains (all executed in parallel) on the x-axis. The y-axis shows ratio of sequential runtime of a single chain of either ExaBayes or MrBayes and the runtime with n coupled chains. We employed 1 process per chain in *ExaBayes-1* and 2 processes per chain in *ExaBayes-2*.

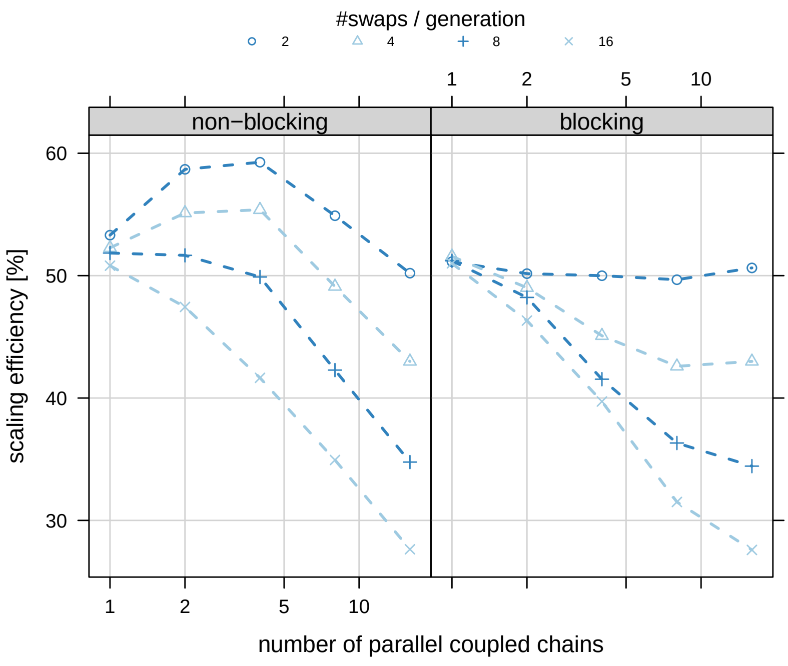


Figure 5.8.b: Scaling efficiency of parallel execution of 16 Metropolis-coupled chains using 32 processes. Scaling efficiency for non-blocking algorithm *left* and blocking algorithm *right* for an increasing number of swaps per generation and an increasing number of chains executed in parallel (x-axis).

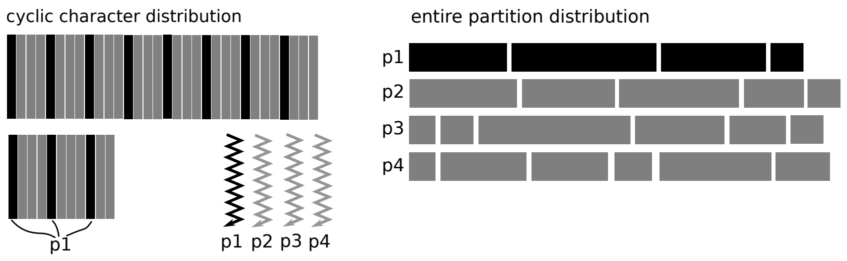


Figure 5.9: Distribution schemes in ExaBayes: in the CDD scheme (*left*) with two partitions every 4^{th} site is assigned to PE p_1, whereas in the MPS distribution scheme (*right*, illustrated with 21 partitions), entire partitions are assigned to PEs such that the number of sites per PE is similar (yet rarely identical).

be evaluated and need to exponentiate the transition rate matrix Q using the precomputed eigenvector/eigenvalue decomposition. Since each partition requires these steps, the sequential overhead can outweigh the distributed (resp., parallelized) likelihood evaluation. If we assign entire partitions to PEs using a multi-processor scheduling (MPS) algorithm [130], the sequential overhead is distributed equally among PEs (see **Fig. 5.9** for an illustration of data distribution).

If all partitions are unlinked for a parameter, BI on highly partitioned datasets (i.e., the alignment comprises > 10 partitions) only requires the evaluation of one partition per proposal, yet when executed in parallel an expensive communication step (i.e., `Allreduce`) is necessary after evaluation. In these cases, ExaBayes bundles all proposals of a specific type (e.g., proposals on per-partition substitution rates $\vec{r_i}$) into a *proposal set* (class `ProposalSet` in **Fig. 5.1**). If a proposal set is drawn for a generation, an update is proposed for each parameter in the proposal set (whereas a parameter can comprise a single partition only or is linked across several partitions). Since the likelihood ratio for each parameter is completely independent, we can evaluate the likelihood for the changed parameters on the entire alignment and then only require a single expensive `Allreduce` at the end. Proposals within a set are still tuned separately. Because of the independence of partitions, the usage of proposal sets does not have a negative impact on mixing efficiency. No modification of the Hastings-ratio is necessary, since this scheme essentially represents a mixture of strictly ordered proposals (i.e., proposals within a set) and random proposals (i.e., whether a proposal set or another proposal such as a topological proposal is applied is determined randomly). Both strategies are valid approaches and have been discussed in the original introduction of the MH algorithm [47].

As an alternative strategy, we could jointly propose new values for all parameters (e.g., $\vec{r_i}$) in a proposal set. However, with a number of parameters in the order of 100 or 1000, tuning of such a joint proposal (towards a target OAP of 25%, see **Sect.** 3.1.4) would result in extremely modest proposals along with inefficient sampling of the parameter space. Notice, that with several branch lengths parameters $\vec{v_i}$ (i.e., for the same branch b_i, we expect a different number of substitutions in different partitions), NR optimization as employed by the NR-based Γ proposal (as well as the hybrid proposals, see **Sect.** 4.2) is particularly communication intensive. Here, ExaBayes adapts the implementation in the PLL [see 112] that allows to optimize several parameters with only one communication per NR iteration for all parameters. The maximum number of `Allreduce` operations is thus determined by the branch length parameter that takes longest to converge.

Evaluation

For the evaluation of the aforementioned strategies, we sub-sampled D^{5e6} to obtain a 200 taxon alignment with 1,000 partitions of 1,000 bp. For various run settings, we ran a single chain employing 4 computing nodes (4 AMD Opteron processors with 12 cores) on a cluster using a total of 192 cores per run.

When proposal sets are used, a generation that evaluates the entire alignment is very expensive (w.r.t. to computational cost) compared to a generation in the original scheme, where only one partition is evaluated. Thus, we normalized runtime by the total number of proposals executed in one run. Notice that, while generations are not comparable among runs where proposal sets are or are not applied, the proportion of a proposal relative to the total number of proposals evaluated remains the same whether it is executed in a set or individually.

In **Fig. 5.10**, we examine normalized runtimes for run settings where we either use CDD or MPS data distribution combined with proposal sets enabled or disabled and relative to the case where none of the two novel techniques are enabled. The left hand side of **Fig. 5.10** shows runs for the more common case where branch lengths $\vec{v}_i$ are linked across all partitions. In the usual proposal mixture (employed in MrBayes and ExaBayes as of version 1.2.1), proposals on topology and branch lengths are assigned a high weight, since convergence is hard to achieve on these parameters. Thus, when using linked branch lengths, proposal sets are only applied in 6.67% of all generations. This explains, why in **Fig. 5.10**, the MPS distribution scheme improves runtime, although likelihood evaluations for single-partition proposals are executed sequentially. When both of our techniques are enabled, we achieve a speedup of 21.7 or in other words, we can compute more than 21$\times$ more proposals in the same time.

Combining the proposal sets with MPS distribution has an even higher impact on normalized runtime, when branch lengths are unlinked across partitions (i.e., each gene has distinct branch lengths). Here, the combination of both techniques achieves a relative speedup of 87.0$\times$. In this case, proposal sets are used for all but the topological parameters (which account for 44.4% of generations). In contrast to the case with linked branch lengths, ExaBayes even becomes slower when MPS distribution is enabled, but proposal sets are disabled. When repeating this experiment for more CPU-cores, we expect performance to further deteriorate.

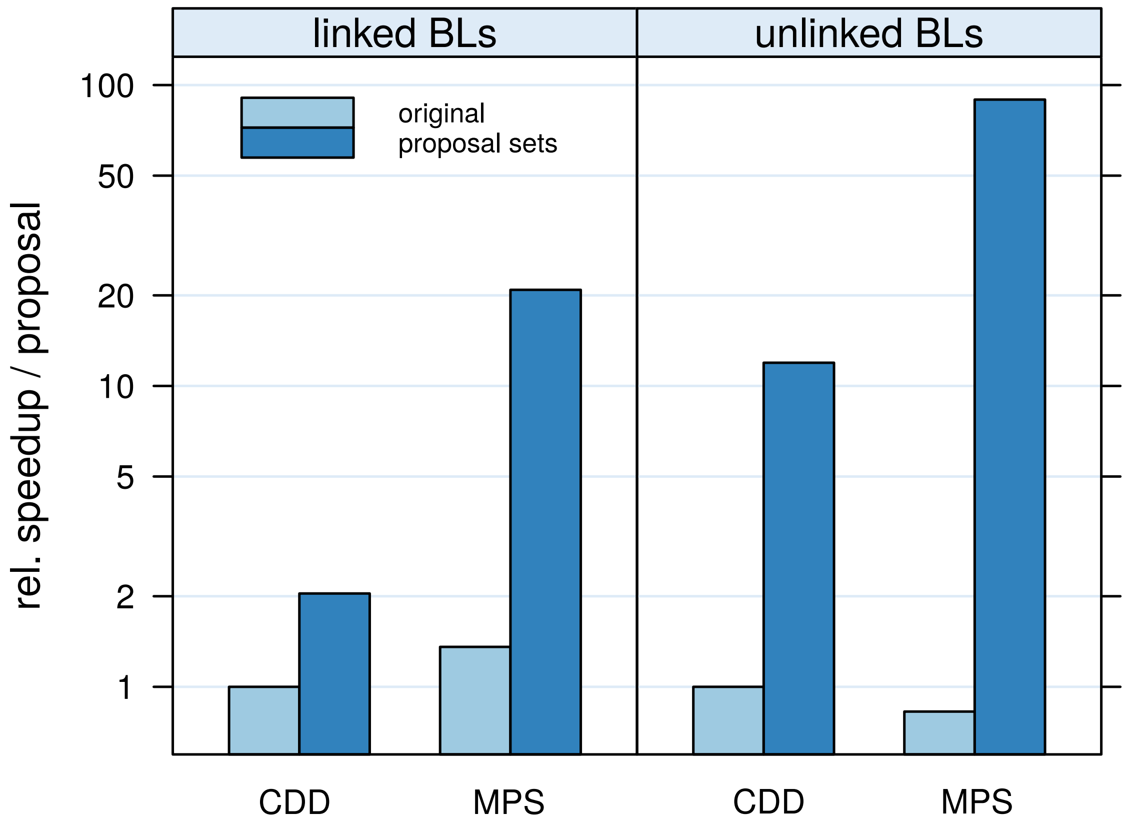


Figure 5.10: Performance on highly partitioned datasets: Speedup of per-proposal runtime for configurations relative to per-proposal runtime where neither the MPS algorithm nor proposal sets are employed. *Left:* branch lengths are linked across all partitions, *right:* branch lengths are unlinked across all partitions.

5.3.2 Improved Data-to-Processor Assignment

Algorithm

While in ExaBayes-1.2.1, the manual choice of the data distribution algorithm (i.e, MPS or CDD) is a minor inconvenience for users, pathological examples can be constructed where neither of the two data distribution algorithms achieves "good" performance. Intuitively, if we split up just some partitions among PEs, we will obtain a more balanced distribution scheme, while PEs will have to perform some additional precomputations (i.e., eigenvector/eigenvalue decompositions and exponentiation of the transition rate matrix Q). If we allow partitions to be only partially assigned to PEs, we obtain the following bicriterion problem for optimizing load balance:

1. distribute alignment patterns such that every PE has the "same" number of patterns (difference between minimum and maximum number of patterns assigned should be ≤ 1) and
2. assign partitions (partially) to PEs, such that the maximum number of partitions (partially) assigned to a PE is minimal.

An optimal solution for this problem is $\mathcal{NP}$-hard, yet a linearithmic-time approximation exists [60]. We refer to this algorithm as the divisible load balancing (DLB) algorithm. The approximation yields a solution such that the optimal solution assigns at most one partition less to the PE that has the most partitions assigned. In the following, we briefly describe the version of the algorithm implemented in ExaBayes and ExaML: First, we sort partitions in descending order of their number of patterns. Then, we assign entire partitions to PEs until no further partitions can be assigned without violating the first property of the criterion (close to equal number of patterns). After step 2, we divide processes into a queue A of PEs with the least number of partitions and a queue of PEs B that already have one partition more assigned to them than PEs in A. In the final step, we partially assign partitions to PEs and (i) try to fill up PEs that already have one partition more (i.e., we dequeue from queue B), (ii) then try to finalize a PE that has fewer partitions or (iii) only partially assign a partition to a PE that has less partitions.

Evaluation

We performed a runtime evaluation on a real-world alignment that is a subset of a large-scale supermatrix of insect sequences [77]. The alignment comprises 144 species and 38,400 AA characters. We used the alignment to create 10 distinct datasets with an increasing number of partitions. For each dataset, we determined partition lengths at random, while the number of partitions in a datasets was fixed to 24, 36, 48, 72, 96, 144, 192, 288, 384, and 768 respectively. For generating n partition lengths, we drew n random numbers $x_1, \ldots, x_n$ from an exponential distribution $\exp(1) + 0.1$. For a partition p, the value of $x_p / \sum_{i=1..n} x_i$ then specifies the proportion of characters that belong to partition p. A constant 0.1 is added to the random numbers in order to avoid

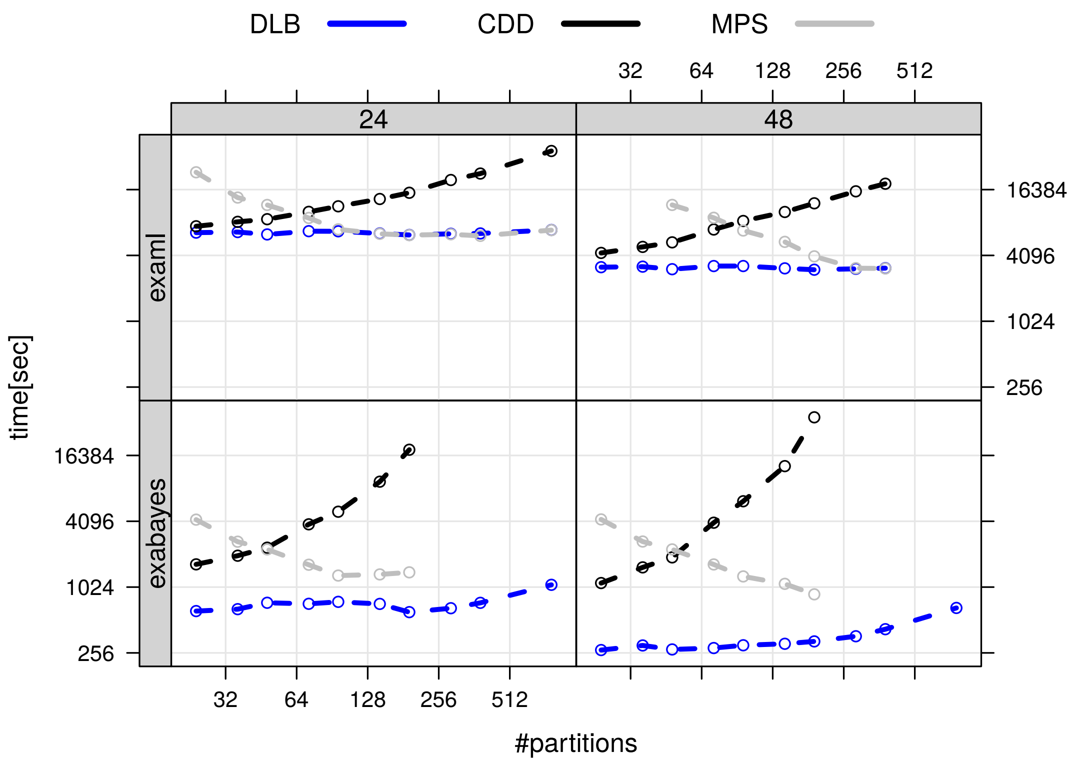


Figure 5.11: Runtime performance (y-axis) of three different algorithms for data distribution for increasing number of partitions. *Top:* Runtimes of ExaML using either 24 or 48 processes. *Bottom:* Runtimes of ExaBayes using either 24 or 48 processes.

that partition lengths become unrealistically small, since the exponential distribution strongly favors small values. Thus, partition lengths are distributed uniformly on the logarithmic scale.

The DLB algorithm is implemented in ExaBayes as of version 1.3 and in ExaML as of version 2.0. In ExaML we performed a typical tree search and in ExaBayes, we ran a chain under default MCMC parameters (as of version 1.2.1) for 10^4 generations. We executed both codes using either 24 or 48 processes on a cluster of Intel Sandy Bridge nodes (2×6 cores per node). Thus, a total of 2 nodes was needed for runs with 24 processes and 4 nodes were needed for runs with 48 processes (with accompanying higher inter-node communication costs). On the left-hand side of **Fig. 5.11**, runtimes for whole-partition distribution with less than 48 partitions are omitted, since here runtimes are identical to executing the run with 24 processes.

As depicted in **Fig. 5.11**, the new heuristic continuously executes at least as fast as the most favorable result of the two previous data distribution strategies with only one exception. Compared to CDD, the heuristic is 3.5× faster for 24 processes and up to 5.9× faster for 48 processes. Using the heuristic, ExaML needs up to 3.6× less time than with the MPS scheme for 24 processes and for 48 processes the runtime can be improved by a factor of up to 3.9×. For large numbers of partitions, the runtime of the MPS scheme converges against the runtime of the new heuristic. However, if the same run is executed with more processes (i.e., 48 instead of 24), this break-even point is substantially shifted towards a higher number of partitions.

The runtime results verify that CDD performs on acceptable levels for many processes and few partitions. The MPS method performs equally well as the DLB heuristic with few processes and many partitions. Both figures show, that there is a region where neither of the previous strategies perform acceptably compared to the new heuristic and that this performance gap widens with an increasing number of processes.

Finally, employing the DLB heuristic, ExaML executes twice as fast with 48 processes than with 24 processes and thus exhibits an optimum scaling factor of about 2.07 in all cases. For comparison, under CDD, scaling factors ranged from 1.24 to 1.75 and under MPS, scaling factors ranged from 1.00 (i.e., no parallel runtime improvement) to 2.04.

For ExaBayes, the comparison among the three data distribution algorithms is less straight-forward. This is because for runtimes of MPS and CDD,we employed ExaBayes version 1.2.1, whereas for the DLB algorithm, ExaBayes version 1.4.1 was used. Because of updates in the PLL and several substantial code modifications (including bug fixes), later versions of ExaBayes generally run faster (not considering the runtime gain achieved by DLB). For instance, along with the implementation of DLB, the I/O overhead at start-up could be reduced. Thus, each PE computes the assignment and subsequently only reads the portion of a preprocessed binary alignment that was assigned to it by DLB. Still, in ExaBayes, we observe a similar runtime behavior for the data distribution algorithms (see bottom of **Fig. 5.11**) as for ExaML. Analogously, there exists a range of partition schemes for which neither MPS nor CDD perform optimally. Specifically, the runtime penalty for CDD increases more rapidly in ExaBayes for a growing number of partitions than in ExaML. Even for the DLB data distribution, datasets with more partitions require more runtime. A likely cause for this is that ExaBayes has a constant proportion of AA matrix proposals, while in ExaML the initially specified matrix is kept fixed. For ExaBayes, we also obtain efficient relative speed-up values between 1.63 and 2.67 (comparing runs with 24 and 48 processes).

5.4 Memory Reduction Techniques

Memory Requirements

For the recursive evaluation of the likelihood of a tree, memory requirements are largely dominated by CPVs that represent the conditional probability of a subtree. For n species, $(n-2)$ arrays have to be kept in memory, such that after the first evaluation subsequent evaluations can be rapidly computed by reusing existing sub-tree CPVs, which have not been invalidated by the proposal. Specifically, for proposals on branch lengths or topologies, we can typically reuse the major part of CPVs. For an alignment with m characters, we require $r \cdot k \cdot m$ double-precision floating values, where r is the number of discrete categories of the Γ distribution (fixed to 4 in ExaBayes, see **Sect.** 2.6.3) and k is the number of states of the data type (2 for binary, 4 for DNA and 20 for AA data).

In phylogenetic BI, it is common to employ an additional set of CPVs as backup: thus, if a proposal is rejected, the CPVs for the previous state can be restored. When we apply MC^3 with c chains, we need one distinct set of arrays for each chain, since PEs evaluating the chains often only compute a few generations for all chains assigned to them. If p coupled chains are executed in parallel, an additional backup set of likelihood arrays is needed for each parallel coupled chain. If multiple independent runs are computed, we can switch between independent runs at reasonable intervals (e.g., after 500 generations) and do not have to keep CPVs in memory for efficient computation, and just recompute them instead. However, if we execute R independent runs in parallel, then the memory requirements increase accordingly by a factor of R. For a double-precision floating point value with 8 bytes, we can thus compute memory requirements as

$$\text{mem}(\mathcal{A}) = \overbrace{R}^{\text{par. runs}} \cdot (\underbrace{c}_{\#\text{chains}} + \overbrace{p}^{\text{par. chains}}) \cdot \underbrace{(n-2)}_{\text{inner nodes}} \cdot \overbrace{m}^{\#\text{patterns}} \cdot \underbrace{r}_{\#\text{cat in } \Gamma} \cdot \overbrace{k}^{\#\text{states}} \cdot 8 \text{ bytes.}$$

As a realistic example, consider a 200 taxon DNA alignment with 1,000 alignment patterns. If we execute 4 independent runs in parallel and use MC^3 with 4 chains (while 2 coupled chains are executed in parallel), the CPVs consume 608.256 mega byte (MB).

Memory Saving Techniques

ExaBayes offers two orthogonal memory saving strategies. Firstly, ExaBayes uses an alternative implementation of the PLF (provided by the PLL) that employs SEVs to omit the evaluation of characters in subtrees that entirely consist of undetermined or gap characters [56]. The SEV-technique reduces the length of several likelihood arrays proportional to the amount of missing data. Secondly, ExaBayes allows to trade memory for runtime by reducing the number of CPVs that are backed up between proposals. In case of rejection, CPVs for which no backup array exists are recomputed, thus increasing runtime. Given a number of taxa n, ExaBayes offers three settings:

1. Do not backup arrays that are computed from two external nodes in the tree. This saves between 2 (comb-like tree) and $(n-3)$ arrays (balanced binary tree).

2. Only backup arrays that are computed from two inner nodes in the tree. Saves between $\lceil \frac{n}{2} \rceil$ (balanced binary tree) and $(n-3)$ arrays (comb-like tree).

3. Recompute all arrays (saves $n-3$ arrays, regardless of tree topology).

Because of precomputations, CPVs are fastest to compute when both subtrees are external nodes (see **Sect.** 2.6.2 for details). In contrast, CPVs where both subtrees are inner nodes are most expensive to compute. Thus, in strategy (1), we only recompute CPVs that are fast to compute and strategy (3) is the only mode, where particularly expensive CPVs have to be recomputed.

Evaluation

Fig. 5.13 illustrates the combination of both techniques (i.e., SEVs and recomputation) to achieve an effective reduction in memory requirements of more than 60% for a dataset with 1,908 species and 1,424 DNA characters (proportion of missing data: 58.38%). For the illustration of a memory versus runtime trade-off (see **Fig. 5.12**), the sequential version of ExaBayes was used. We ran one chain for 10,000 generations for datasets in **Tab. 4.1**. Using strategy (1), we can reduce the memory requirements by more than 10% at the expense of about 5% additional runtime. For strategies (2) and (3), measurements for datasets are divided into two groups: for one group strategy (2) decreases memory requirements to 70%-80% (resp., 60%-70% for strategy (3)) of the original requirements while the induced additional runtime barely exceeds 30% for strategy (2) (resp., ≈45% for strategy (3)). The division is due to the fact that, many of the datasets are comparably small (i.e., single gene and few taxa see **Tab. 4.1**). Thus, they have small requirements to begin with and constant overhead (as illustrated in **Fig. 5.13**) induces a large part of the overall memory consumption. We conclude that, our recomputation strategies represent a flexible way to save up to 50% of memory at the expense of a slow-down of less than 1.5×.

5.5 Large-Scale Bayesian Inference

In the following, we describe BI from a whole genome DNA alignment that has been simulated using INDELible [38]. The alignment comprises 200 species and 100,000,000 characters and consists of 100 equally long partitions (in analogy to a chromosome partitioning scheme). Each partition was simulated to evolve according to a unique random GTR matrix along the same randomly chosen phylogenetic tree. The five free substitution rates were drawn uniformly form the interval $[0.1, 1.9]$ for each of the 100 GTR matrices. Stationary frequencies were drawn uniformly from a Dirichlet distribution $D(1, 1, 1, 1)$. All in all, for constructing the alignment, $1,000$ individual alignments with $100,000$ characters each, were simulated using INDELible. Of these 1,000 alignments, 10 always share the same underlying GTR matrix and stationary frequencies that have been used for simulation. The $1,000$ individual alignments were then concatenated. The tree used for simulation is depicted in **Fig. 5.14**. The phylogenetic tree has a clock-like shape (i.e., the sum of branch lengths from the biological root to each taxon is similar). Branch lengths in this tree range from $4.74482 \cdot 10^{-4}$ to $3.16679 \cdot 10^{-1}$.

For the analysis, 4 independent runs were started from MP starting trees and 2 independent runs from random starting trees. For the analysis, we linked branch lengths across all partitions for all runs, otherwise default parameters as of ExaBayes version 1.2.1 apply. The concatenated alignment file requires 21 GB disk space and the RAM requirements for CPVs exceed 4.61 TB. We executed analyses at the Leibniz Supercomputing Centre (LRZ) using the SuperMUC system. Thus, at the SuperMUC we need a minimum of 193 computing nodes (each with 16 CPU-cores and 24 GB of accessible main memory) to conduct a single run because of the immense underlying memory requirements. We used $4,096$ CPU cores for runs starting from MP starting

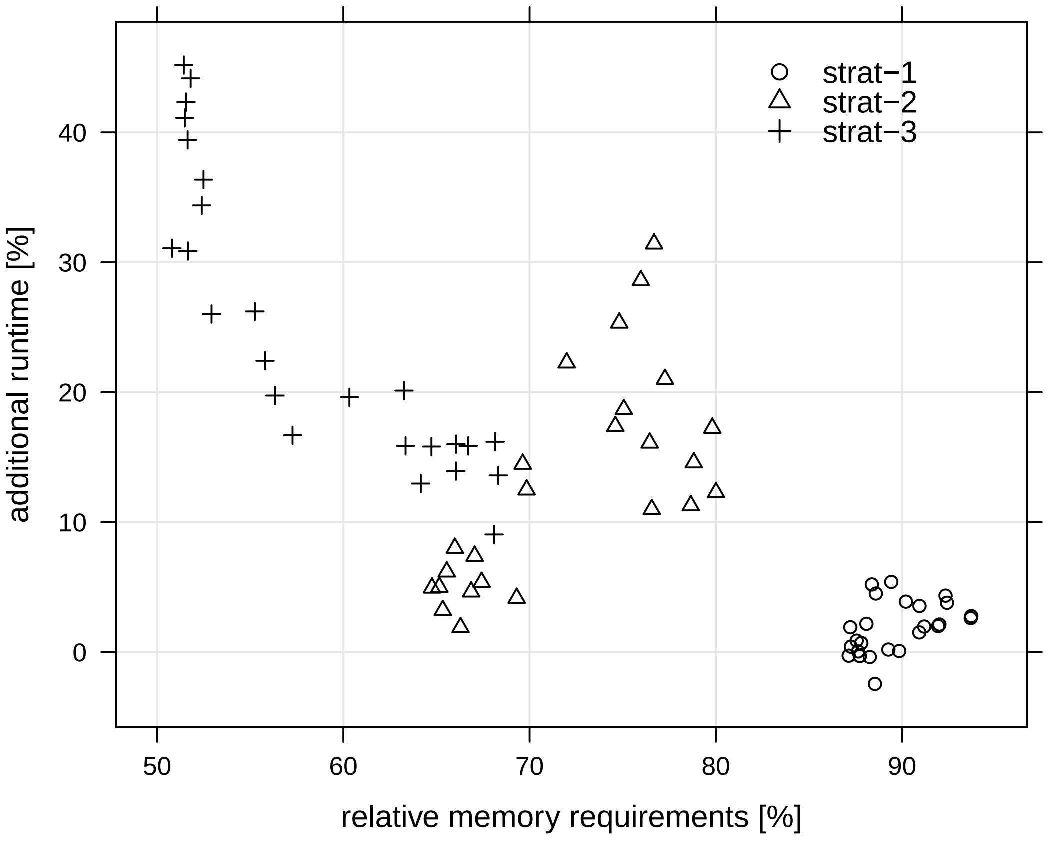


Figure 5.12: Memory/runtime trade-off for different recomputation strategies (not using the SEV technique). Memory consumption (x-axis) and additional runtime (y-axis) are presented relative to respective values for a run without recomputation.

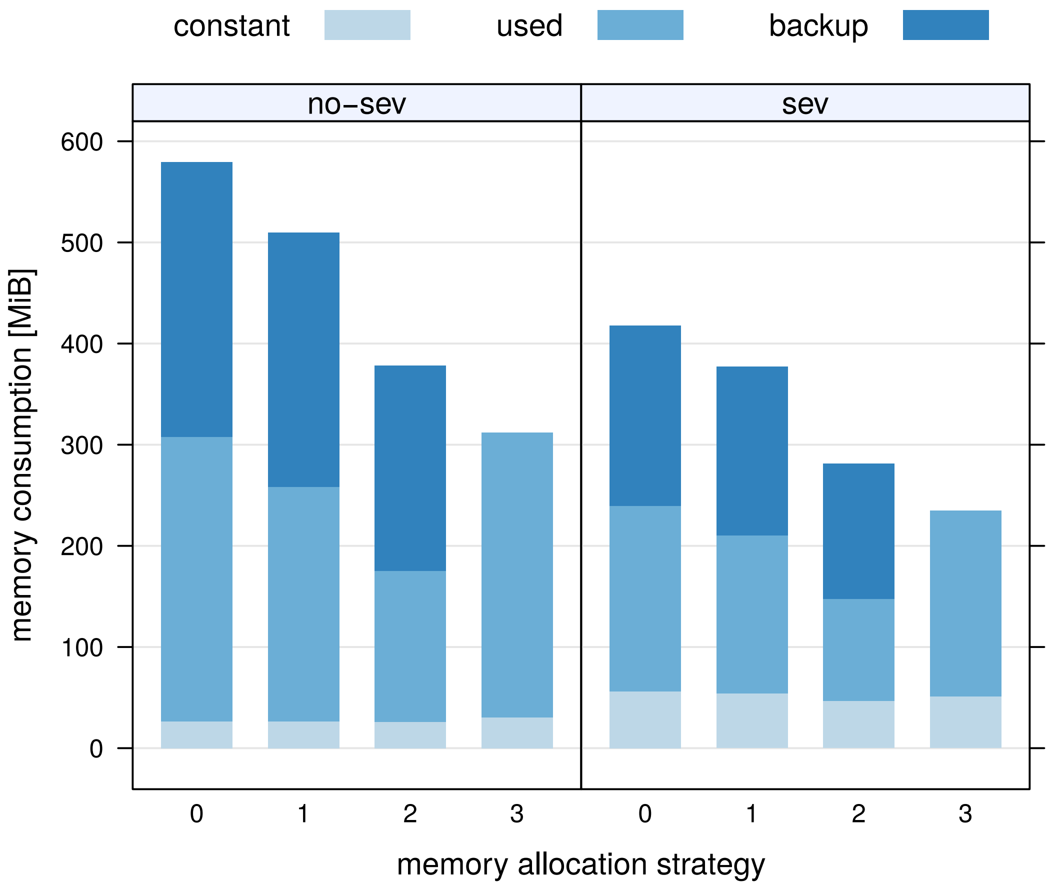


Figure 5.13: Absolute memory consumption for 1 chain running on a dataset with 1,908 taxa and 1,424 DNA characters (gap proportion: 58.38%) using the SEV-technique (*sev*) or standard implementation of the likelihood function (*no-sev*). Numbers on the x-axis correspond to the id of memory saving strategies explained in **Sect.** 5.4, whereas 0 indicates that no CPVs have been recomputed. Each measurement also indicates the proportion of memory *used* for likelihood computations, employed for *backup* arrays or *constant* memory overhead.

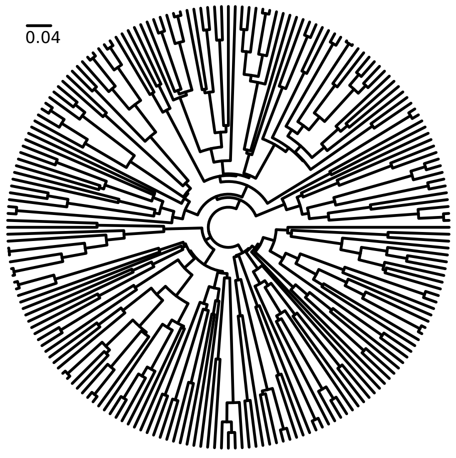


Figure 5.14: Topology of the phylogenetic tree (comprising 200 species in total) that was used for simulation of the whole-genome dataset.

trees and 8,192 cores for runs using randomized starting trees. Given the CPU budget limitations, we chose to run chains for 100,000 generations and extracted a sample from chains every 500 generations. Subsets of this alignment with 10,000 and 1,000,000 characters were run under the same conditions. Despite the low number of generations, clade posterior probabilities for chains that were started with a parsimony tree quickly converged against the simulated topology. The ASDSF of the two chains starting in a random tree is critically high at the end of the simulation, because of the substantially extended burn-in phase. Thus, while we did not achieve topological convergence, after $\approx$60,000 generations, the two chains started with random trees reached the same correct tree (that was also used for simulating the alignment). This indicates that convergence of the 4 other chains was not biased by using a MP starting tree. Furthermore, we observe that using a MP tree here instead of a random tree for initialization increases the burn-in length by a factor of 3.

In order to obtain good estimates of branch length distributions, we ran two additional chains for each dataset ($10^4, 10^6$ and 10^8 characters) using default parameters of ExaBayes version 1.4.1 (i.e., using the NR-based Γ proposal) and using an increased sampling frequency (extracting samples after increments of 100 generations). **Fig. 5.15** showcases distributions of branch length samples extracted for 4 distinct representative splits across the three datasets. As expected, with increasing amount of data, the branch length samples become more accurate. While for the datasets with 10^4 and 10^6 bp the true branch length typically is within the credible interval of the samples, we observe a consistent tendency on the 10^8 bp dataset to overestimate branch lengths. Thus, our experiment constitutes the first observation of the *long tree problem* [19] (i.e., the tendency of BI to overestimate branch lengths) on large-scale datasets. Here, the extent of the problem is not hidden within the credible interval. Compound Dirichlet priors on branch lengths have been proposed [91] as a solution. Using this prior, branch length estimates inferred via BI became closer to the ML estimate [129].

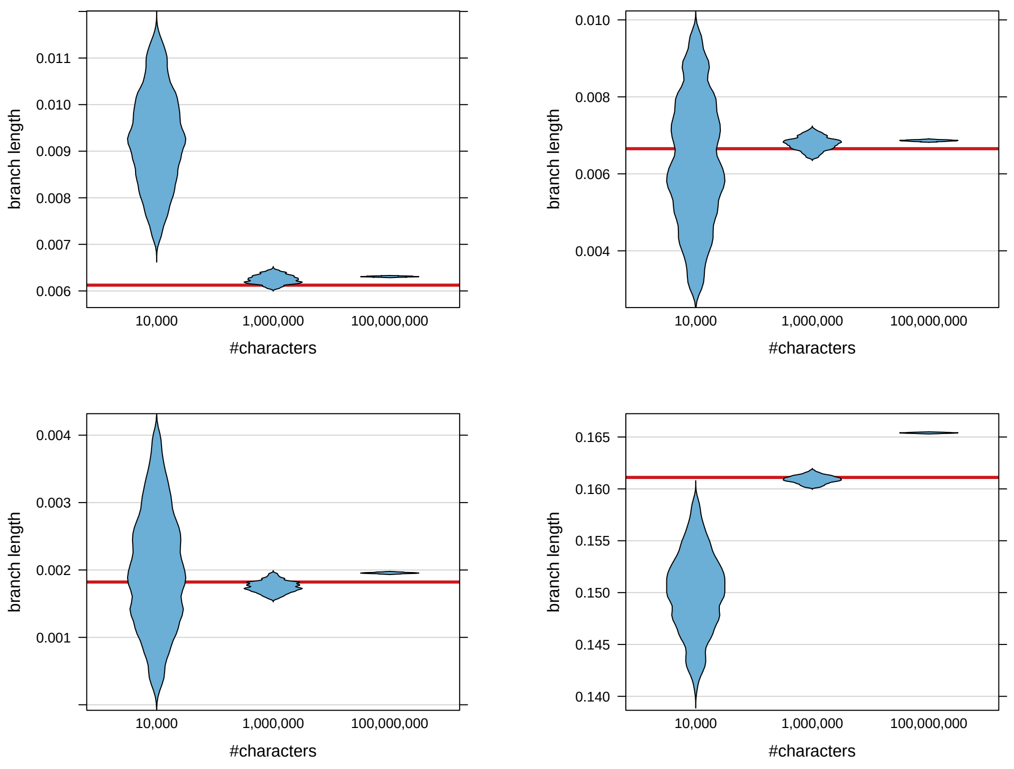


Figure 5.15: Distributions of branch length samples (y-axis) from chains run on the whole-genome alignment and two truncated versions. Each Figure corresponds to a distinct split in the tree used for simulation. Alignment size is increased in steps of 2 orders of magnitude (x-axis). Branch lengths in the tree used for simulation of the alignment in red.

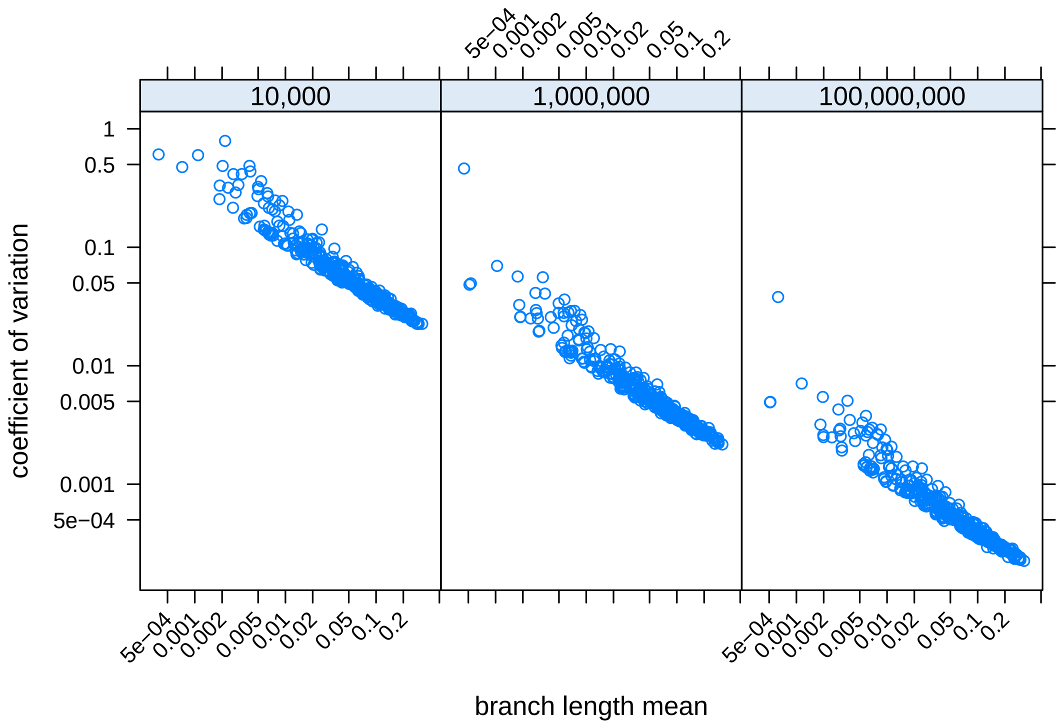


Figure 5.16: Mean and CV of branch length samples for the simulated whole genome dataset and its two truncated variants.

Although 10^4 characters can be considered an informative alignment, many branch length credible intervals span a wide range. For 10^4 characters, 4 branch length posteriors clearly have the shape of an exponential distribution (i.e., CV > 0.9) which indicates that the alignment lacks phylogenetic signal to support the split (see **Sect.** 4.1). An exponential-like branch length posterior (with a CV of 0.96) has a particularly large credible interval: 90% of the posterior probability are between branch lengths of $5.06 \cdot 10^{-5}$ and $2.17 \cdot 10^{-3}$. Such a high degree of branch length uncertainty does not allow for accurate statements about branch lengths. In contrast, on the alignment with 10^6 characters, the branch length posterior with the highest-CV has a 90% probability range between branch lengths $1.16 \cdot 10^{-5}$ and $2.18 \cdot 10^{-4}$. For 10^8 characters, the corresponding worst-case 90% probability range is in the much narrower interval $[1.77 \cdot 10^{-3}, 6.54 \cdot 10^{-3}]$.

Fig. 5.16 provides an overview of the CV relative to the sampling mean. For each split in the three datasets, we extracted the respective statistics after discarding a burn-in of 25% of samples. **Fig. 5.16** corroborates our observation on a global scale that more data can lead to more narrow branch length credible intervals. Furthermore, it shows that as we increase the amount of data in steps of two orders of magnitude, the overall CVs of branch length posteriors decrease in increments of one order of magnitude. In particular, with 10^8 characters, even specifically short branch length posteriors assume a narrow-ranged Γ-like distribution (i.e., their CV is orders of magnitude smaller than 1), which indicates that the alignment contains enough phylogenetic signal to resolve rapid ancient speciation events. If we consider the CV as an indicator for the degree of branch length uncertainty, then we can conclude that with 10^8 characters we have substantially more confidence in our estimate of the shortest branch lengths than we had in the longest branch lengths for posteriors that were inferred from the alignment with 10^4 characters.

Because of the clear signal provided by the large amount of data, no Metropolis-coupling is necessary. While ExaBayes does not constrain the size of the analysis to be conducted, we discuss the SuperMUC as an example of the limits imposed by modern super-computers on dataset size. Given the simulated alignment, a configuration that trades a maximum amount of runtime against memory and given the 9,216 computing nodes available at the SuperMUC (where usually only 2,048 nodes are available for a single job), Metropolis-coupling with 95 chains is theoretically feasible, if all coupled chains are executed in parallel. A dataset, that is merely one order of magnitude larger (i.e, 2,000 taxa or 1,000,000,000 characters), would allow for only 9 coupled chains if conducted on the entire SuperMUC computer.

5.6 Summary

In this Chapter, we initially discussed the sequential software design of ExaBayes for phylogenetic BI and subsequently described the design of the parallelization approach that allows ExaBayes to conduct phylogenetic BI on massively parallel machines. We evaluated the sequential runtime efficiency of ExaBayes by comparing it to analogous runs with MrBayes. On DNA datasets, the AVX-version of ExaBayes was consistently faster than MrBayes, while MrBayes (using its native PLF implementation) performed better than ExaBayes on AA data. Generally, for our test runs MrBayes is slower, if compiled with Beagle. We demonstrated, that the SEV approach provided by the PLL decreases runtimes for several instances.

In ExaBayes, phylogenetic BI is parallelized at run-level, chain-level and data-level. ExaBayes offers a constrained form of hybrid parallelization, that is, we can start processes on several computing nodes which in turn start threads for intra-node level parallelism. As a means to this end, threads that use two types of message queues emulate communication functionality that is provided by MPI at the processes level. Since we could not proof runtime advantages experimentally for the hybrid approach, the major advantage of this scheme is that it provides thread parallelism on systems where no MPI implementation is available. We analyzed the efficiency of data-level parallelism and could demonstrate a maximum parallel speedup of $> 2,300\times$ and validated that for appropriate datasets ExaBayes scales up to 32,768 processes. For the parallelization of coupled chains, we introduced a non-blocking algorithm for parallel MC^3. The non-blocking implementation increased the scaling efficiency of ExaBayes by up to 10% (which corresponds to an effective runtime improvement of 18.6%).

ExaBayes allows for parallel phylogenetic BI from highly partitioned datasets. We introduced proposal sets that propose new values for several partitions simultaneously without the need to modify the Hastings-ratio (which would deteriorate the acceptance rates of the respective proposals). We evaluated proposal sets in the context of two data distribution algorithms and observed a runtime improvement of 21.7× when branch lengths were linked across partitions and 87.0×, when branch lengths were unlinked across partitions. Finally, we showed that the implementation of the DLB algorithm in conjunction with proposal sets yields runtimes that are consistently at least as good as the optimal runtime under any of the two previously employed data distribution schemes.

We discussed the utilization of two orthogonal strategies to reduce the memory requirements of CPVs in ExaBayes. While using the likelihood evaluation functions that employ SEVs was straight-forward, we adapted a flexible recomputation approach for BI that had previously been used in ML inference. We observed that, we can often reduce memory requirements by more than 10% at the expense of less than 10% additional runtime. On large alignments and using maximum recomputation, we reduce the memory requirements by almost 50%, while the runtime increases by less than 1.5×.

Finally, we simulated a genome-sized alignment in order to conduct phylogenetic BI on the largest dataset in terms of memory footprint that has ever been analyzed using this method (comprising 10^8 alignment patterns). We compared the inference to results that were obtained from sub-sampled alignments comprising 10^4, respectively 10^6 patterns. While we could not correctly resolve the tree with the smallest alignment, the intermediate alignment yielded highly supported phylogenetic relationships for all except one inner branch. On the full genome-sized alignment, we recovered all branches of the tree that was used for simulation with 100% PP. Furthermore, we examined how the accuracy of Bayesian branch length estimates improves as the amount of data increases. We found that in our setting, increasing the input alignment by two orders of magnitude decreased phylogenetic uncertainty by roughly one order of magnitude.

6 Identification of Rogue Taxa in Tree Sets

This Chapter exclusively contains original contributions by Andre Aberer, if not explicitly stated otherwise. Parts of this Chapter are derived from the following peer-reviewed publication:

1. AJ **Aberer**, D Krompass, and A Stamatakis. "Pruning rogue taxa improves phylogenetic accuracy: an efficient algorithm and webservice". In: *Systematic biology* 62.1 (2013), pp. 162–166

Contributions: All contributions by Andre Aberer, except an initial draft of the web service that was provided by Denis Krompaß (see **Sect.** 6.5).

This Chapter describes an algorithm that allows to compute improved consensus trees from tree sets when the quality of consensus trees is deteriorated by the presence of rogue taxa (i.e., a form of noise that renders certain taxa unstable in a set of trees). The algorithm (called RogueNaRok algorithm) allows us to identify rogue taxa in tree sets and outperforms previous algorithms with respect to the quality of the results and runtimes. Thus, the RogueNaRok algorithm is particularly suited for large-scale datasets with respect to the number of trees and the number of taxa.

In **Sect.** 6.1, we define what consensus trees are and explain the rogue taxon problem. Subsequently, we briefly describe related algorithms (see **Sect.** 6.2). In **Sect.** 6.3, we at first give a detailed description of the general RogueNaRok algorithm. Then, we introduce an approximation parameter and a refinement that substantially reduce the computational burden of the implementation. In **Sect.** 6.4, we describe the experimental setup evaluating the runtime and the accuracy of rogue taxon analyses compared to related algorithms. Furthermore, we conduct extensive simulation experiments in order to verify that the removal of rogue taxa (as identified by RogueNaRok) yields tree sets that are closer to the correct trees that have been used for simulating the datasets. Finally, in **Sect.** 6.5, we describe the functionality of an associated webservice that implements an extended work-bench for rogue taxon analyses.

6.1 Introduction

6.1.1 Consensus Trees And Support

An important task in phylogenetic analysis is to assess to which degree inferred phylogenetic relationships are supported by the underlying data. In BI, PPs of splits are often referred to as "Bayesian support". As outlined in **Sect.** 2.6.4, a bootstrap analysis (that yields several bootstrap trees) is the preferred way of obtaining support in a ML or MP framework. Irrespective of the phylogenetic inference method used, support values are generally extracted from a set of trees $\mathcal{T} = \{T_1, \ldots, T_m\}$ (see **Fig. 6.1a**), where m is the total number of trees.

A consensus tree summarizes the information in a tree set $\mathcal{T}$ in a single tree. The consensus tree only contains those branches (i.e., bipartitions) that occur more often in $\mathcal{T}$ than a predefined threshold t. Inner branches (i.e., non-trivial bipartitions) are labeled by their frequency in $\mathcal{T}$. The degree of resolution (i.e., the number of inner branches) in a consensus tree is then determined by the chosen consensus threshold. In contrast to binary phylogenetic trees, we allow for nodes with a degree > 3 in consensus trees. Thus, for trees with n taxa, the number of branches in a consensus tree ranges between n and $(2 \cdot n - 3)$. In the worst case, there are no consensus bipartitions at all. In such a case the consensus tree is a trivial star tree (see **Fig. 6.1b**). As explained in **Sect.** 2.3, we can represent each inner branch b_i of a tree that divides (if removed) the set of taxa into the sets T and $\overline{T}$ (where $|T| > 1$ and $\overline{|T|} > 1$) as a non-trivial bipartition $B_i = (T \mid \overline{T})$. Just as we can represent a single tree as a set of bipartitions, a tree set $\mathcal{T}$ is equivalent to a bipartition profile $(\mathcal{B},\, \sigma)$. $\mathcal{B}$ is the set of all unique bipartitions that occur in $\mathcal{T}$ and $\sigma : \mathcal{B} \rightarrow 2^{\mathcal{T}}$ is a function that maps each bipartition B_i to the subset of trees in which it occurs.

The consensus tree for a given bipartition profile $(\mathcal{B}, \sigma)$ is then defined as $C_t(\mathcal{B}) = \{B \in \mathcal{B}, \text{s.t. } \big|\sigma(B)\big| > t\}$, where t is the frequency threshold. The threshold t ranges between $\frac{m}{2}$, which is referred to as the majority-rule consensus (MRC), and $m - 1$, which we call a strict consensus (SC) tree [reviewed in 20]. For a given threshold, we can construct a tree that contains the respective consensus bipartitions by iteratively adding inner nodes and additional branches to a star tree. Note that, for this algorithm we can not choose the threshold below $\frac{m}{2}$, since there is no guarantee that two bipartitions with a frequency $< \frac{m}{2}$ can occur in the same tree (i.e., we say they are not *compatible*). Two bipartitions $B_1 = (b_1 \mid \overline{b_1})$ and $B_2 = (b_2 \mid \overline{b_2})$ are called compatible, if

$$(b_1 \cap b_2 = \emptyset) \vee (b_1 \cap \overline{b_2} = \emptyset) \vee (\overline{b_1} \cap b_2 = \emptyset) \vee (\overline{b_1} \cap \overline{b_2} = \emptyset).$$

In order to nonetheless include bipartitions with frequency $\leq \frac{m}{2}$, we can greedily refine a MRC tree. We sort bipartitions in descending order of their frequency and then check for one bipartition at a time, if it is compatible with the current set of consensus bipartitions. If the bipartition is compatible, we add it to the set of consensus bipartitions and the next most frequent bipartition is tested for compatibility. The algorithm for the greedily refined majority-rule consensus (GMRC) tree is a heuristic. The search for a tree with maximum resolution (i.e., maximizing the number of consensus bipartitions) is $\mathcal{NP}$-hard [88].

Note that, specifically in the Bayesian context, the type of consensus tree described here can not be considered the mean (resp., median) of a posterior distribution of trees [13]. In other words, a consensus tree with respect to the tree parameter is not the equivalent of a median or mean of a continuous distribution. Instead, it represents a straight-forward summary statistic of bipartition frequencies.

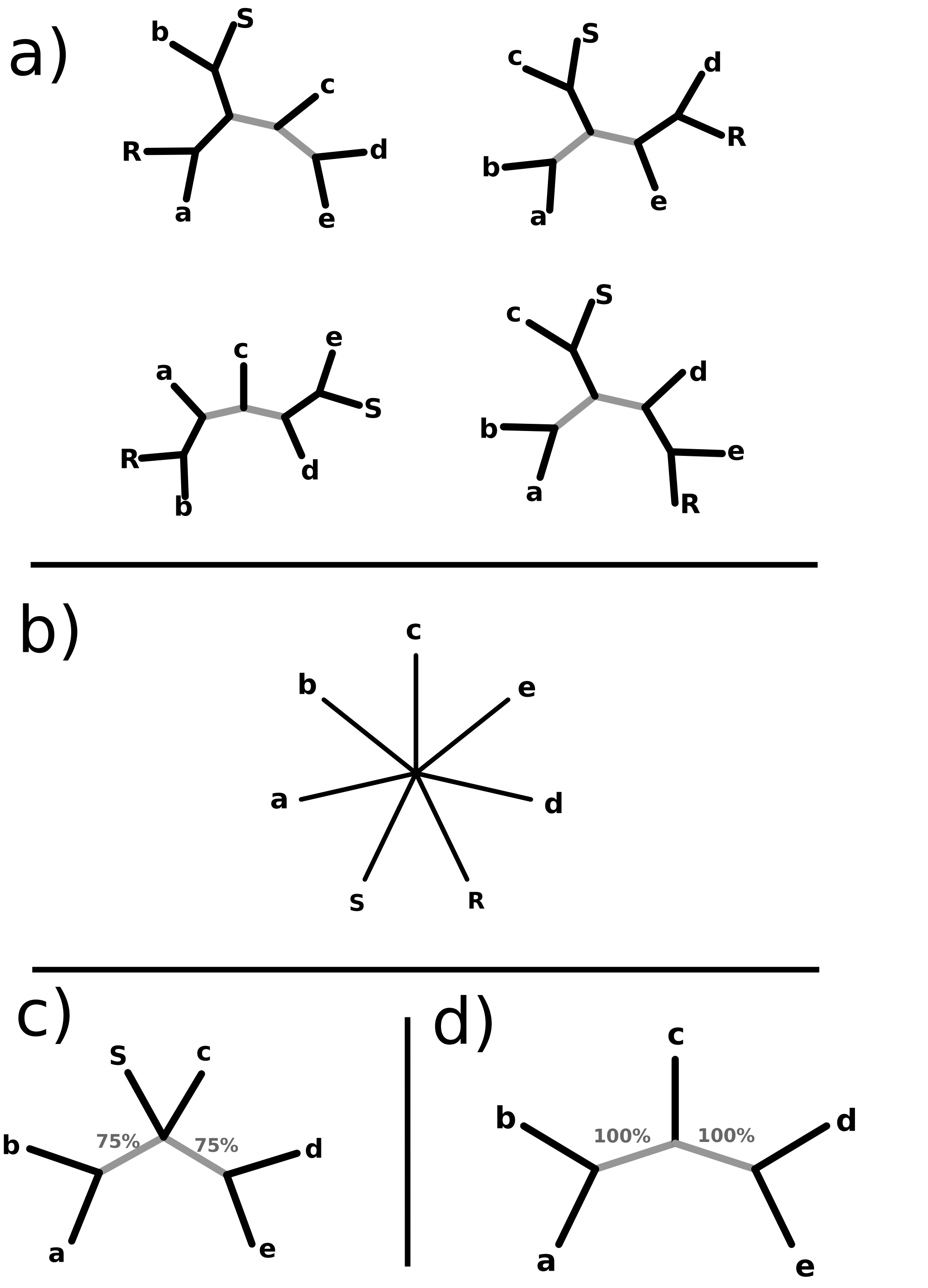


Figure 6.1: Example illustrating the impact of rogue taxa on the consensus of trees. Panel *a)* depicts a tree set comprising 4 unique trees with 7 taxa. Inner branches that can be recovered are highlighted in gray. A MRC or SC tree computed from this tree set is non-informative as shown in panel *b)*. Pruning of taxon *R* results in a more informative MRC tree (see panel *c)*). As depicted in *d)*, we obtain the most informative tree when we prune the set {S,R}.

As an alternative to computing a consensus tree, we can use the bipartition frequency information in $\mathcal{T}$ for assessing to which degree the branches in a single tree (e.g., a ML or MP tree) are supported by $\mathcal{T}$. In other words, we use the tree set to draw support onto a reference tree (e.g., the best-known ML or MP tree).

6.1.2 Problem Description: Rogue Taxa

The resolution (i.e., number of bipartitions) in a consensus tree and the branch support on the best-known tree can be substantially deteriorated by *rogue taxa* [the term rogue taxon was introduced in 126]. Rogue taxa assume varying and often contradictory positions in the tree set. The rogue phenomenon is usually attributed to ambiguous or insufficient phylogenetic signal [99]. Consider the trees in **Fig. 6.1a**: here taxa S and R appear as closest relatives to various other taxa in the tree. Thus, the MRC and SC are uninformative (see **Fig. 6.1b**). A GMRC tree (not shown) would consist of three bipartitions with 50% support, namely $(cS\,|\,abdeR)$, $(ab\,|\,cedRS)$ and $(abcS\,|\,edR)$, as well as a bipartition with 25% $(de\,|\,abcSR)$. While this GMRC is well-resolved, the presence of rogue taxa substantially reduces the overall confidence in evolutionary relationships (i.e., inner branches).

Determining the "correct" position of a rogue in a phylogenetic tree is tedious [101] and therefore rogues, once identified, are mostly simply excluded (pruned). If we prune for instance R from the example tree set, the MRC tree contains two bipartitions with 75% (see **Fig. 6.1c**), while the SC is still unresolved. If in addition to R, we also prune S, we obtain an even more informative tree (see **Fig. 6.1d**) for the MRC as well the SC tree. That is, in **Fig. 6.1d** two bipartitions are recovered that are present in all induced subtrees that do not contain R and S.

For evaluating the informativeness of a consensus tree with support values drawn onto inner branches, we need to consider three components: (i) the number of taxa n in the tree, (ii) the resolution of the consensus tree (i.e., the number of inner branches in the tree) and (iii) the support values for inner branches. Given a bipartition profile $(\mathcal{B}, \sigma)$ and a set of bipartitions $\mathcal{B}'$ (either consensus bipartitions or bipartitions extracted from a reference tree), we can compute the informativeness of $\mathcal{B}'$ as

$$\text{general\_RBIC}(\mathcal{B}', \sigma; \alpha, C_t) = \frac{\sum_{b \in \mathcal{B}'} \frac{|\sigma(b)|}{m} + \alpha \cdot n}{(n-3) + \alpha \cdot n}, \tag{6.1}$$

where m is the number of trees in $\mathcal{T}$ that induce $\mathcal{B}$ (i.e., that underlie the bipartition profile). We refer to **Eq. 6.1** as generalized relative bipartition information criterion (RBIC). The generalized RBIC incorporates the number of taxa on the one hand as well as the average support of all consensus bipartitions. Factor α is a weight factor (with $\alpha > 0$) that accounts for the influence of either criterion on the overall score. We can replace the average frequency with the total number of consensus bipartitions $|\mathcal{B}'|$. If we do so and set $\alpha := 1$, then we obtain the relative information content (RIC) [85]. If we prune taxa from trees, then the RIC will only increase, if we obtain an additional consensus bipartition for each taxon that is pruned. In the following, we will focus on the case, where $\alpha := 0$, that is, we want to optimize the accumulated support with respect to a consensus tree method C_t without considering the number of taxa that we have to prune. We refer to this optimality criterion simply as RBIC. In other words, the RBIC indicates what percentage of support a consensus tree (extracted from a bipartition profile $(\mathcal{B}', \sigma)$ under C_t) has, when compared to a fully resolved consensus tree with maximum support given no taxa have been pruned. Thus, the RBIC of a profile extracted from **Fig. 6.1a** increases from 0% (**Fig. 6.1b**) over 37.5% (when R is pruned in **Fig. 6.1c**) to 50% (when $\{R, S\}$ is pruned in **Fig. 6.1d**).

We call a set of taxa that we intend to prune from the trees underlying a bipartition profile a *drop set.* The task of criterion-based rogue taxon identification is to compute an optimal dropset d_{opt}, that maximizes the optimality criterion (e.g., RBIC or RIC) for the given set of input trees. A consensus tree that is optimal with respect to the above optimality criteria is called a maximum-information subtree consensus (MISC) [85].

Thus, the MISC is a pruned consensus tree that is more informative than the initial consensus tree. Also, the MISC constitutes a compromise between a conventional consensus tree and an alternative to the consensus known as the maximum agreement subtree (MAST) [10]. The MAST is the largest fully resolved tree that occurs in all trees of set $\mathcal{T}$. In practice, the MAST typically only comprises an impractical small number of taxa.

6.2 Related Algorithms

The rogue taxon problem was shown to be $\mathcal{NP}$-hard [25] and a super-polynomial integer linear programming (ILP) algorithm for an exact solution is available. In practice, stability measures based on triplet frequencies [119] or node distances [74] have been applied [117, 118, 29, 106] to identify rogues. Yet, it could be shown that two algorithms that strive to optimize either the RIC or the RBIC substantially outperform the aforementioned measures for rogue identification [3].

The bipartition merging algorithm (BMA) is an efficient method for approximating the expected increase of resolution in the consensus tree when rogues are pruned [85]. Initially, the algorithm extracts the bipartition profile from the tree set $\mathcal{T}$. Then, the BMA computes dropsets of taxa that induce a merger of two bipartitions (if pruned). For each possible pair of bipartitions that are not in the set of consensus bipartitions, we check whether merging the pair (by pruning the dropset) would give rise to a consensus bipartition. Thus, the BMA approximates the number of new consensus bipartitions that will appear, once a dropset is pruned. Subsequently, we compute the dropset that is optimal with respect to its optimality criterion (i.e., the RIC) and permanently remove this set from the set of taxa. Then, the algorithm updates the bipartition profile (i.e., extracts it anew) and starts another iteration. That is, it tries to determine a dropset that improves the RIC of the consensus tree that will be computed from the updated bipartition profile.

The following aspects have a deleterious effect on the accuracy and runtime performance of the BMA:

1. the algorithm is greedy (i.e., we possibly prune locally optimal dropsets in early iterations and thus globally fail to determine the optimal solution),
2. we only know approximately, how many new consensus bipartitions will emerge, after pruning a dropset (e.g., pruning the dropset may result in an existing consensus bipartition becoming trivial),
3. we repeatedly compute dropsets for pairs of non-consensus bipartitions in each iteration (although dropset information could be reused, if modified accordingly),
4. if there are several dropsets that improve the resolution of a consensus tree, we do not necessarily prune the set that yields the best improvement in support,
5. we also assess large dropsets that are unlikely to improve the number of consensus bipartitions (e.g., if we prune 30% of all taxa, there is a high chance, that this has a negative impact on existing consensus bipartitions, i.e., they will vanish).

The single-taxon algorithm (STA) [3] is an alternative to the BMA, that uses the RBIC instead of the RIC as an optimality criterion and thereby addresses issue (4). The STA determines the exact impact on the RBIC score, if we prune a single taxon from the tree set at a time. It thus overcomes issue (2) and to some degree issue (5). Naturally, the restriction to a dropset size of 1 prohibits the algorithm from solving hard instances of the rogue taxon problem, where several taxa have to be pruned simultaneously to improve the optimality score. Specifically for the MRC and GMRC, the STA performs well and also outperforms previously used methods, such as node distance and stability measures.

6.3 RogueNaRok algorithm

6.3.1 Algorithm Description

For our novel algorithm (called RogueNaRok), we use a graph-based formulation of the criterion-based rogue identification problem and design an algorithm, that can determine exactly how the bipartition profile changes (and therefore the resulting consensus tree), when a specific dropset is pruned from the underlying trees. Bipartitions that are stored in a bipartition profile can change in two ways when taxa are pruned:

1. a bipartition $B_i = (b_i \mid \overline{b_i})$ vanishes, because it degenerates into a trivial (non-informative) bipartition (i.e., $(|b_i| < 2) \vee (|\overline{b_i}| < 2)$),
2. a set of bipartitions $\{B_i, \ldots, B_j\}$ merges into a new bipartition B' with support $\sigma(B') = \sigma(B_i) \cup \ldots \cup \sigma(B_j)$.

Note that, when a consensus bipartition vanishes (i.e., the consensus tree has a branch less), this is detrimental for the optimality of the resulting consensus tree. Thus, case (2) can either increase or decrease the optimality score, depending on the respective support of B' and $B_i, \ldots, B_j$.

For each pair of bipartitions (B_i, B_j) there exist two dropsets $d = b_i \Delta b_j$ and $\overline{d} = \overline{b_i} \Delta b_j$ that induce a merging of the bipartition pair, where $a \Delta b$ is the symmetric set difference $(a \cup b) \setminus (a \cap b)$. Dropsets d and $\overline{d}$ are called *minimal* when the following property holds: If we remove any taxon from d or $\overline{d}$, then bipartitions B_i and B_j do not merge. Bipartitions $\mathcal{B}$ and the minimal dropsets d / $\overline{d}$ for each pair in $\mathcal{B}^2$ can be represented as a *merger graph* G of possible merging events (e.g., see **Fig. 6.2**). In G, each bipartition is represented by a node and every possible pair of nodes is connected by two directed edges (labeled by d and $\overline{d}$). The edge indicates that the respective bipartition pair merges, when d (resp. $\overline{d}$) is pruned.

Algorithm 2 maps all minimal dropsets to induced optimality change

Input: a set of bipartitions $\mathcal{B}$
Output: optimality change s_d for each minimal dropset d

```
1: function createCombineScoreDropsets($\mathcal{B}$)
2:     $D \leftarrow \{d \mid \exists (B_i, B_j) \in \mathcal{B}^2, (b_i \Delta b_j = d) \vee (b_i \Delta \overline{b_j} = d)\}$        ▷ I. compute dropsets
3:     for all $d \in D$ do
4:         $\varepsilon[d] \leftarrow \{(B_i, B_j) \in \mathcal{B}^2 \mid (b_i \Delta b_j = d) \vee (b_i \Delta \overline{b_j} = d)\}$
5:     end for
6:     for all $d \in D$ do                                          ▷ II. compute partial graphs
7:         $\varepsilon^*[d] \leftarrow \{(B_i, B_j) \in \mathcal{B}^2 \mid (B_i, B_j) \in \varepsilon[d'] \wedge d' \subseteq d\}$
8:         $\chi[d] \leftarrow$ connectedComponents($\varepsilon^*[d]$)
9:     end for
10:    for all $d \in D$ do                                          ▷ III. evaluate dropsets
11:        $s_d \leftarrow$ calculateOptimalityChange($d, \chi, \mathcal{B}$)
12:    end for
13: end function
```

Based on the merger graph G, we can now formulate **Alg.** 2 that iterates over all possible minimal dropsets and determines the optimality score of the resulting consensus tree induced by pruning each dropset (one at a time) from the underlying trees. The algorithm consists of three phases: In phase I, we determine all minimal dropsets and create a mapping ε, that maps each dropset d to edges of the merger graph G that are labeled by d. In phase II, for each dropset d, we detect all edges in G that are either labeled by d or a subset of dropset d to create ε^*. We have to extend ε to ε^*, since pruning a set of taxa S also induces a merger between all bipartitions that would merge, if merely a subset of S is pruned. Each non-singleton connected component (i.e., comprising at least 2 nodes) in the graph G_d induced by the node set $\mathcal{B}$ and the edge list $\varepsilon^*[d]$ describes a set of bipartitions $\{B_i, \ldots, B_j\}$ that will merge into a single novel bipartition B' if d is pruned. **Fig. 6.2** illustrates the merger graph for the example tree set in **Fig. 6.1a** (edges with drop sets larger than 3 are omitted) and highlights the connected components of the induced merger graph $G_{\{R,S\}}$. In $G_{\{R,S\}}$, we find two connected components, each of which can be merged into a new consensus bipartition.

Finally, in phase III, we calculate the optimality change s_d induced by removing each minimal dropset d (calculateOptimalityChange in **Alg.** 2). As mentioned, we

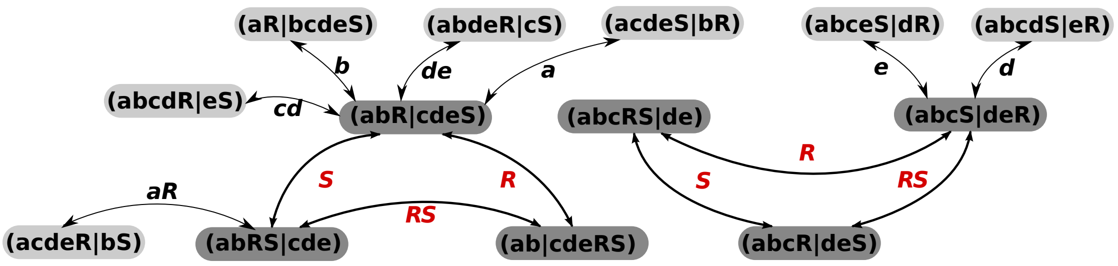


Figure 6.2: Merger graph for bipartitions of all trees depicted in **Fig. 6.1a** and for a maximum dropset size of 2. Two connected components are highlighted in dark gray. When $\{R, S\}$ is pruned, the components merge into two new consensus bipartitions.

use the RBIC as the optimality criterion for this step. Using resolution-based optimality (e.g., the RIC) instead is straight-forward. The bipartitions of each connected component $c = \{B_i, \ldots, B_j\} \in \chi[d]$ merge into a bipartition B', if d is pruned. Let

$$\delta(a,t) = \begin{cases} a, & \text{if } a > t; \\ 0, & \text{otherwise;} \end{cases} \tag{6.2}$$

where t is the consensus threshold. Then the optimality change s_d of d is

$$s_d = \sum_{c \in \chi[d]} \left(\overbrace{\delta(\left|\sigma(B')\right|, t)}^{s_{\text{gain}}(c)} - \overbrace{\sum_{B_k \in \{B_i,\ldots,B_j\}} \delta(\left|\sigma(B_k)\right|, t)}^{s_{\text{loss}}(c)} \right) - \overbrace{\sum_{B_l \in B^{cons'}} \left|\sigma(B_l)\right|}^{s_{\text{van}}(d)}, \tag{6.3}$$

where $s_{\text{gain}}(c)$ is the support attained by the new bipartition B' and $s_{\text{loss}}(c)$ the support lost by merging the bipartitions, respectively. The term $s_{\text{van}}(d)$ accounts for consensus bipartitions ($B^{cons'}$) that degenerate into trivial bipartitions and will not occur in any connected component.

createCombineScoreDropsets computes the exact optimality score change for each minimal dropset. We can use this algorithm to repeatedly determine the best dropset (i.e., with the most support recovered per taxon pruned) and then update the bipartition profile accordingly, until no further optimality score improvement can be achieved. The algorithm is of polynomial time complexity in the number of bipartitions $|\mathcal{B}|$, since all phases are polynomial (including the search for connected components [54]). Note that, this algorithm still represents a greedy approximation to the globally optimal MISC problem. This is because of the term s_{van} in the dropset evaluation. For the terms s_{loss} and s_{gain}, the merger graph implies that two dropsets d_1 and d_2 will either not influence each other's difference $s_{\text{gain}} - s_{\text{loss}}$, or, that there exists a dropset $d_1 \cup d_2$ that will be evaluated at some point. The term s_{van} however, needs to be evaluated for all possible combinations of minimal dropsets 2^D.

Phase III of the algorithm can be modified to also optimize the bootstrap support of bipartitions that are drawn onto a reference tree (e.g., the best-known ML tree). Note that, if bipartitions $\{B_i, \ldots, B_j\}$ merge into B', then B' forms part of the reference tree, if at least one bipartition in $\{B_i, \ldots, B_j\}$ occurs in the reference tree. Furthermore, we can also use the merger information in $\chi[d]$ for the optimization of support in a GMRC tree. This is computationally expensive (see [20]), since we have to compute a GMRC tree for each possible minimal dropset d from a bipartition profile that has been transformed according to $\chi[d]$.

6.3.2 Approximation via Dropset Size

There are $2 \cdot |\mathcal{B}|^2$ edges in the merger graph G. As explained previously, we have to search for all subsets of a dropset to create ε^*, such that we are able to determine the exact state of merger graph after pruning a dropset. This subsequent search for sub-dropsets has quadratic time requirements. Thus, the general algorithm described in **Sect.** 6.3.1 quickly becomes prohibitive on large real-world datasets. For example, consider that in the worst case 1,000 trees with 1,000 taxa can contain almost $1,000,000$ distinct bipartitions. Problematic real-world datasets of this size usually contain between 50,000 and 100,000 distinct bipartitions. Many edges in the merger graph are labeled by excessively large dropsets. For instance, if for two bipartitions B_i and B_j their dropset $d = b_i \Delta b_j$ is of size $|d| = 1$, then the inverse dropset $\overline{d}$ has size $\overline{|d|} = n-1$, where n is the number of taxa. This in fact means, that it is impossible for this dropset $\overline{d}$ to recover a bipartition.

Previous experiments on real datasets [3] showed that, the BMA prunes single-taxon dropsets in $\approx 90\%$ of its iterations. Thus, we approximate and parametrize **Alg.** 2 by only computing the exact optimality score change for minimal dropsets of sizes $\leq l$. To implement this we need to modify line 2 in **Alg.** 2 as follows:

$$D \leftarrow \{d \mid \exists (B_i, B_j) \in \mathcal{B}^2, \Big((b_i \Delta b_j = d) \vee (b_i \Delta \overline{b_j} = d) \Big) \wedge |d| \leq l\}. \tag{6.4}$$

If we only compute dropsets of size $l := 1$, at most two bipartitions will merge into a new bipartition at a time. This follows from the triangle inequality that holds for the lower-cardinality dropset of a set of bipartitions $\{B_i, \ldots, B_j\}$ (which define a metric in the space of bipartitions [85]). As a consequence, for $l := 1$, we can omit the expensive phase II of **Alg.** 2 in which we search for all subsets of a dropset. Thus,

$$l := 1 \quad \Rightarrow \quad \forall d \in D : \chi[d] = \varepsilon[d]. \tag{6.5}$$

The approximation via a drop set size parameter l offers the potential for a substantial optimization at implementation level. Firstly, we can represent a bipartition as a bit vector (e.g., an array of type `uint64_t`) and normalize bit vectors, such that the bit vector represents the smaller partition (i.e., we have always at most as many bits set as bits that are unset). After sorting, this normalization allows us to compute all dropsets of size l quickly, as shown in the following. Then, we create a hash table $h : \mathbb{N} \rightarrow 2^{\mathcal{B}}$ that maps the size of the smaller partition b_i of a bipartition B_i to the respective bipartitions with this property. Thus, the complementary partition $\overline{b_i}$ by default is at most as large as b_i. We can use the hash table h to substantially reduce the space of bipartition pairs that we have to compare in order to obtain all drop sets ε. If there exists a minimal dropset d (of size l) between bipartitions $B_i = (b_i \mid \overline{b_i})$ and $B_j = (b_j \mid \overline{b_j})$ with $|b_i| < \left|b_j\right|$, then $(|b_i| + d = \left|b_j\right|) \vee (|b_i| + d = \overline{\left|b_j\right|})$. Thus, for obtaining drop sets of size $\leq l$ for bipartition $B_i = (b_i \mid \overline{b_i})$, we only have to compare partition b_i against

$$\left\{ p \mid \big(B \in h[|b_i| + a]\big) \wedge \big(B = (p \mid \overline{p})\big) \wedge \big(|p| \leq |\overline{p}|\big) \wedge (a \leq l) \right\}$$

$$\bigcup \left\{ \overline{p} \mid \big(B \in h[|b_i| + a]\big) \wedge \big(B = (p \mid \overline{p})\big) \wedge \big(|p| \leq |\overline{p}| \wedge a \leq l\big) \wedge \left(\left\lceil \frac{|b_i| + |\overline{b_i}|}{2} \right\rceil \leq |b_i| + a \right) \right\}.$$

In simple terms for determining all dropsets of size l for B_i, it mostly suffices to compare the smaller partition b_i of B_i against the smaller partitions of any bipartition B_j, where the size of the smaller partition $\left|b_j\right|$ is $|b_i| + l$. The hash table provides rapid access to the bipartitions with the respective partition sizes. For bipartitions B_i that have a similar number of taxa in either partition, we have to compute dropset $b_i \Delta b_j$ as well as $b_i \Delta \overline{b_j}$ (see the second term in the above formula). In comparison, the BMA computes 2 dropsets from all pairs of bipartitions in $\mathcal{B}^2$.

6.3.3 Updating Instead of Recomputing the Merger Graph

Every iteration of **Alg.** 2 starts with the computation of minimal dropsets that define the edges in the merger graph. Thus, after selecting and removing a dropset d in the preceding iteration and transforming the bipartition profile as indicated by $\chi[d]$, we essentially need to recompute the edges of the merger graph from scratch. However, many of the recomputed edges will be identical to the edges of the merger graph in the previous iteration. Let $d = b_i \Delta b_j$: if both bipartitions B_i and B_j did not merge with any other bipartition in the previous iteration and no taxa of the respective bipartitions b_i and b_j have been pruned, then the edge $(B_i, B_j) \in \varepsilon[d]$ remains unchanged. This means that in phase I of **Alg.** 2, we only need to exhaustively compute all edges of the merger graph in the very first iteration. In subsequent iterations, we can simply update/modify the merger graph as needed. Also, if an edge has not changed between iterations, we do not need to recompute s_{gain} and s_{loss} for the respective merging event. While we still have to compute s_{van}, this modification nonetheless substantially accelerates phase III.

6.4 Evaluation

6.4.1 Runtime Improvements

We executed the RogueNaRok algorithm (RNR), the STA, and the BMA (implemented in RAxML) on collections of bootstrap trees from 25 real-world MSAs (see **Tab. 6.1**). All tree sets contain 1,000 trees. The number of taxa ranges between 24 and 7,764. Runtime measurements were performed on 48-core AMD Magny-Cours nodes and averaged over 4 runs. Missing data points represent runs with excessive execution times that were interrupted.

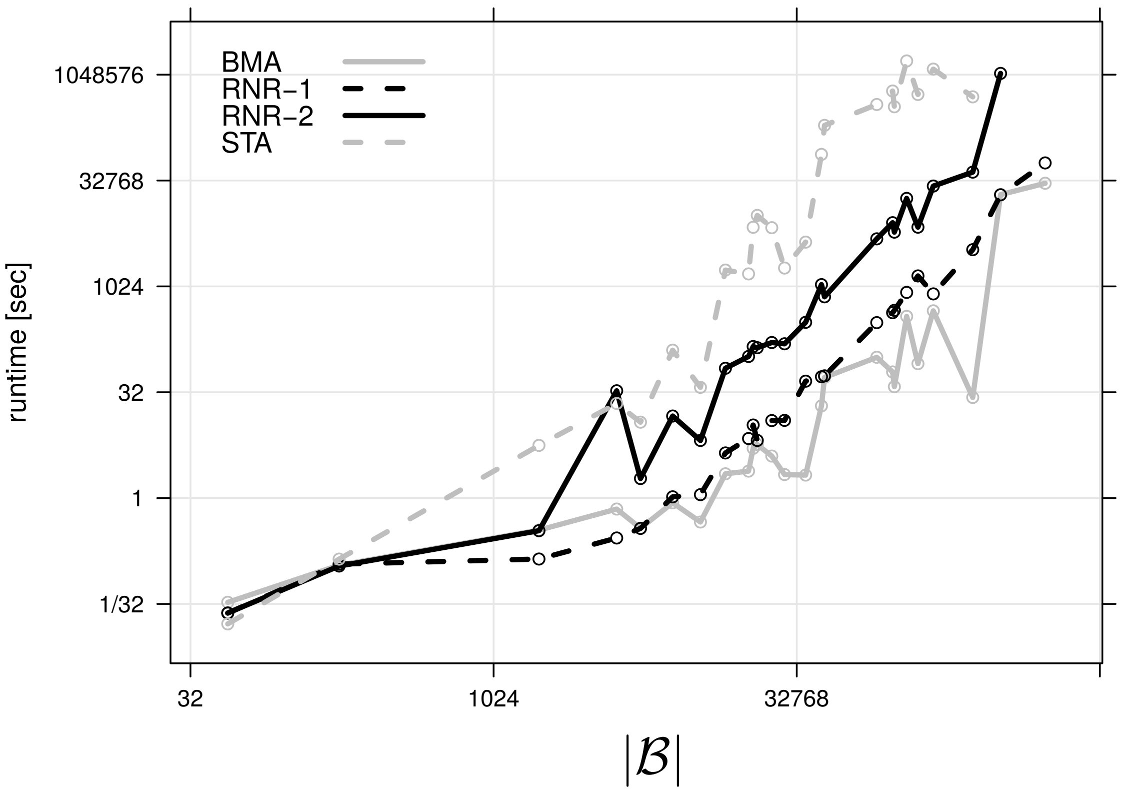


Figure 6.3: Runtimes for the STA, BMA and RNR algorithm with maximum dropset size $l := 1$ and $l := 2$. The x-axis refers to the initial number of bipartitions $|\mathcal{B}|$ for a bootstrap tree collection. Runtimes are depicted for MRC as consensus threshold (SC similar).

#taxa	#bip	#bp
24	49	14,190
125	175	29,149
141	5,491	7,036
143	10,897	1,535
148	1,720	4,101
150	4,183	1,269
218	7,931	2,294
350	28,506	4,451
354	36,274	460
404	18,885	13,158
500	14,497	1,398
628	24,630	1,228
714	19,907	1,241
885	245,118	623
994	20,869	5,533
1,288	43,474	1,200
1,481	130,976	1,241
1,512	100,049	1,577
1,604	81,808	1,276
1,908	98,053	1,424
2,000	156,133	1,251
2,308	45,022	1,224
2,554	115,159	1,232
6,718	336,864	1,122
7,764x	560,887	851

Table 6.1: Datasets that were examined for rogue taxa. The column *#bip* identifies the number of unique bipartitions in a bipartition profile extracted from 1,000 bootstrap trees.

Fig. 6.3 depicts the sequential execution times for the three algorithms. We executed the RNR algorithm for maximum dropset sizes of $l := 1$ and $l := 2$ (denoted as RNR-1 and RNR-2). For all, except the smallest datasets, RNR-1 is significantly faster than STA while yielding qualitatively identical results. Overall, we observe an average runtime improvement between two and three orders of magnitude. In the case with 2,308 taxa and 1,000 trees containing 45,022 distinct bipartitions, the RNR-1 algorithm is over 3,640 times faster than the STA. As indicated in **Sect.** 6.3, the largest fraction of the speed improvement in RogueNaRok can be attributed to successive merger graph updates (instead of full recomputations) between iterations. For instance, in the first iteration of the data set with 2,000 taxa, RNR-1 spends 137 sec in step 1 to compute the edges (and thereby the minimal dropsets) of the merger graph. In subsequent iterations, updating the merger graph takes between 0.05 and 10 sec (mean: 1.2 sec).

When choosing a larger value for l, the identification of edges induced by sub-dropsets for the quadratically growing number of possible dropsets starts dominating runtimes. Nevertheless, RNR-2 is —in most cases— still significantly faster than STA (see **Fig. 6.3**). At the same time more complex rogue taxon constellations are identified. While RNR-2 is considerably slower than the BMA, RNR-1 achieves runtimes that are comparable to the BMA. However, the RNR algorithm can typically identify at least 10 times more potential rogues than the BMA. In terms of runtime per identified rogue, RNR-1 is faster than the BMA in all but two cases.

6.4.2 Qualitative Improvement

Here, we evaluate how various input parameters of the RNR algorithm affect and improve the support in SC and MRC trees and compare this improvement to consensus trees that are produced after pruning rogues as suggested by the BMA. In general, comparisons to the BMA are difficult, since the BMA optimality criterion penalizes dropsets as a function of the number of taxa that are pruned. We thus adapted BMA (referred to as BMA-mod) to assess how a less conservative criterion for approximating support gain improves resolution. To achieve this, we changed the BMA scoring scheme for dropsets, such that it prunes dropsets with the highest per-taxon resolution improvement. Since the inherent approximation errors of the BMA-mod increase rapidly with the number of iterations, we also computed the exact overall support in the consensus tree after each BMA iteration (which substantially increases runtimes). In our analysis, we only consider that intermediate result of BMA which yields the highest overall support with respect to the exact evaluation based on the consensus trees for each iteration. A qualitative comparison of RNR with STA is not required, since RNR and STA can be modified such that RNR-1 and STA yield exactly identical results.

Fig. 6.4.a and **Fig. 6.4.b** depict the RBIC improvements obtained by the BMA, BMA-mod, and the RNR algorithm (with dropset sizes $l \in [1, 3]$). Overall, BMA-mod recovers substantially more support than the default BMA. For MRC trees, RNR-1 performs consistently and substantially better than the BMA and BMA-mod. While RNR-1 still performs better than BMA for SC trees (see **Fig. 6.4.b**), we have to chose $l > 1$ to outperform BMA-mod. This is in agreement with previous observations [3], that is, BMA is more accurate when a SC threshold is used. On the other hand, when using a MRC threshold, RNR may yield less optimal results for $l := 3$ compared to $l := 2$. Here, RNR-3 performs worse, because two dropsets of size 2 pruned in subsequent iterations may yield a higher overall per taxon RBIC improvement than a single larger dropset of size 3. If the larger dropset is optimal for an iteration, it will be pruned by RNR-3 (unlike RNR-2 which does not evaluate this dropset). Thereby, the possibility of achieving the same effect as pruning the two dropsets of size 2 can be lost. RNR-4 is capable of finding the optimal solution in such a scenario, however the general problem of local optima remains (e.g. RNR-4 may prune a dropset of size 4 instead of two dropsets of size 3). Finally, the inferior performance of RNR-3 suggests that for the MRC threshold, dropsets of size 3 (or larger) are rarely necessary or they do not noticeably increase the RBIC of a consensus tree.

6.4.3 Effect on Phylogenetic Accuracy

In this Section, we describe simulation experiments that were conducted for assessing to which extent pruning rogues can increase phylogenetic accuracy. Initially, we describe, how datasets were simulated. Then, we show that removing rogue taxa as suggested by RogueNaRok for two typical use cases yields trees that are topologically closer to the *true* (simulated) tree.

Simulation of Datasets

We used INDELible [38] to simulate DNA sequences for 100, 200, or 500 taxa with initial sequence lengths of 100, 500, or 1.000 bp (with resulting number of bp in the final alignment $\in [100, 31.161]$). Four different sets of GTR parameters were estimated from real-world datasets (see **Tab. 6.1**). For the insertion-deletion process, we set the power law shape parameter to 1.5, the maximum insertion/deletion length to 5 and the insertion/deletion rate to 10^{-3} and 10^{-4}. Up to 5 datasets were generated for each combination of these parameters. INDELible simulates sequences using a birth-death process and also outputs the underlying clock-like tree on which the sequences are based. Subsequently, the generated DNA sequences were aligned using Muscle [30]. For each alignment generated thereby, we conducted 20 tree searches and inferred a set of 200 bootstrap trees under the per-site rate (PSR) approximation [see 107] of the GTR+Γ model with RAxML [109]. We simulated and analyzed a total of 400 datasets. Thus, for each simulated dataset a RAxML-based ML tree, 200 bootstrap trees, and the tree used by INDELible to simulate the evolutionary process (referred to as *true tree* in the following) was created.

Reduced RF-distances of ML trees to the True Tree

Initially, we employed RogueNaRok to identify rogue taxa that have a negative impact on the bipartition support drawn onto the ML tree of a dataset. For each dataset, we determined the distance between the ML tree and the true tree before and after rogue taxon removal. Usually, the absolute RF-distance (see **Sect.** 2.3) is used for comparing

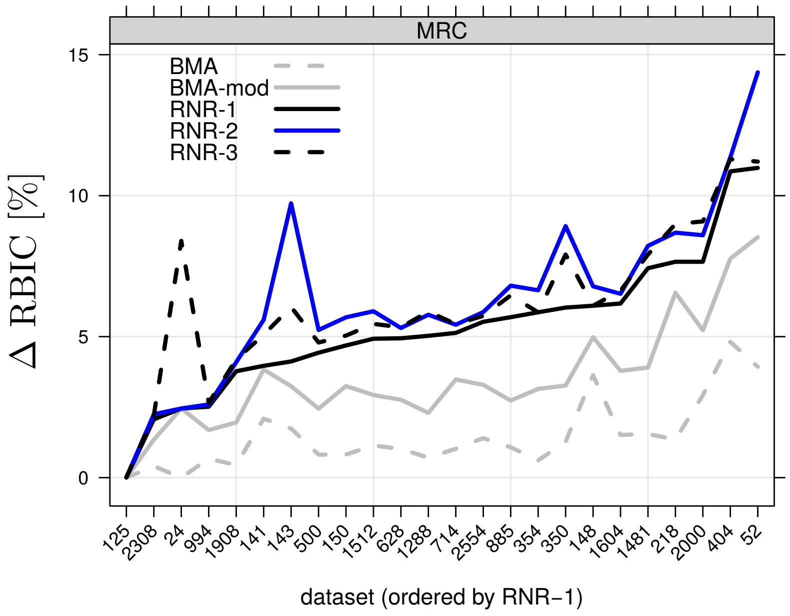


Figure 6.4.a: Support improvement (in %) for optimization with a MRC threshold. **RNR-l** depicts RNR runs with $l \in [1, 3]$, **BMA-mod** is a less conservative modification of the BMA.

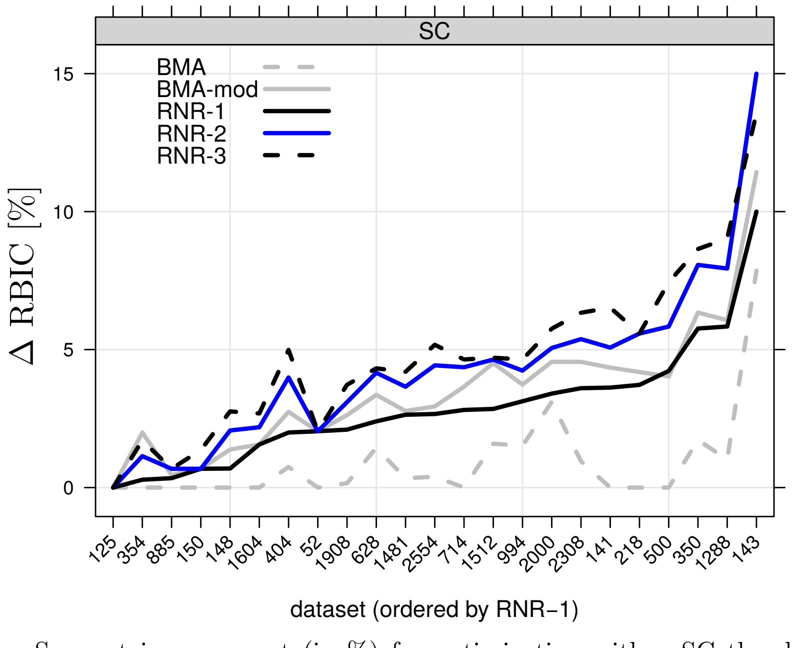


Figure 6.4.b: Support improvement (in %) for optimization with a SC threshold. **RNR-l** depicts RNR runs with $l \in [1, 3]$, **BMA-mod** is a less conservative modification of the BMA.

trees. In order to account for the loss of taxa after pruning, we have to use the relative RF-distance RF_{rel} here, which is normalized to values between 0.0 (minimum distance) and 1.0 (maximum distance). Thus, we can define the improvement of relative RF-distance between the ML tree and the true tree after rogue taxon removal as

$$\Delta\text{RF}_{\text{rel}} = \text{RF}_{\text{rel}}(\text{ML}, \text{true\_tree}) - \text{RF}_{\text{rel}}(\text{ML}_{\text{pru}}, \text{pruned\_true\_tree}), \tag{6.6}$$

where ML is the initial ML tree and ML_{pru} is the pruned ML tree.

By default, RogueNaRok uses bipartition support values for rogue taxon identification (i.e., the RBIC criterion). Therefore, we also considered the relative weighted Robinson-Foulds-distance (rWRF), a variant of the RF-distance that also takes into account the bipartition support values on the trees for computing the distance. For the rWRF-distance, we sum over the branch support values that are unique to either tree (assuming branches in the true tree have 100% support) and normalize again to relative values between 0.0 and 1.0. Thus, in analogy to **Eq. 6.6**, we computed the improvement of the relative rWRF-distance between the ML and true tree after rogue taxon removal.

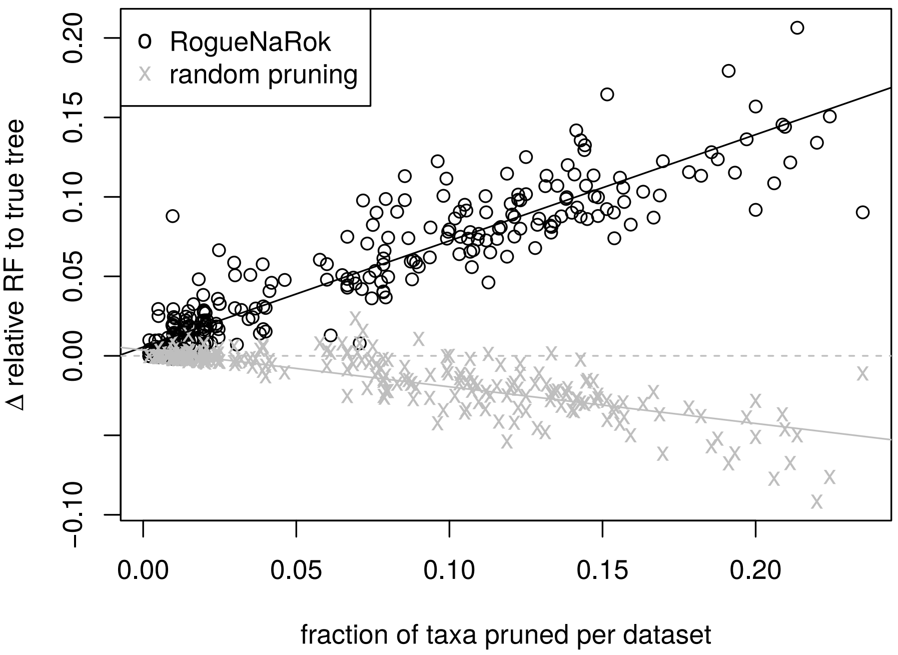
For each dataset, we also calculated the respective changes in RF-distances for the case that we do not prune taxa in an informed way as suggested by RogueNaRok, but prune a random set of taxa (that were not predicted by RogueNaRok) of equal size instead. These *random prunings* are used to obtain the baseline reference changes of the respective RF-distances for each simulated dataset.

Fig. 6.5 shows the improvement of both topological distance flavors (RF and rWRF) against the fraction of taxa pruned. We observe a clear trend: pruning rogue taxa as identified by RogueNaRok reduces both, the relative RF, as well as the rWRF distances of the ML tree to the true tree. The more rogue taxa a dataset contains (resp., the higher the overall instability of taxa in the bootstrap trees), the higher the potential for improving RF-distances by pruning rogue taxa becomes. Our simulations indicate that, the instability of taxa in bootstrap replicate trees reliably predicts taxa for which the position in the ML is incorrect. Thus, pruning rogue taxa as identified by the RogueNaRok algorithm improves phylogenetic accuracy.

The random prunings indicate that, the rWRF-distance and the relative RF-distance represent an appropriate measure for the task at hand. If only a small fraction of taxa is pruned, random prunings may yield ML trees that are closer to the true tree. However, this improvement is always less pronounced than for a set of rogue taxa as identified by RogueNaRok. For large sets of randomly pruned taxa, there is a clear trend for the resulting pruned ML tree to become increasingly distant to the true tree. Some improvements in RF-distances via random prunings are to be expected, since the rogue taxon set determined by RogueNaRok is only one of many possible sets of taxa that may increase bipartition support, if pruned.

Improved True Support in Consensus Trees

The primary and alternative use case for the RogueNaRok algorithm is to identify a set of rogue taxa that improves the overall bipartition support in a consensus tree,



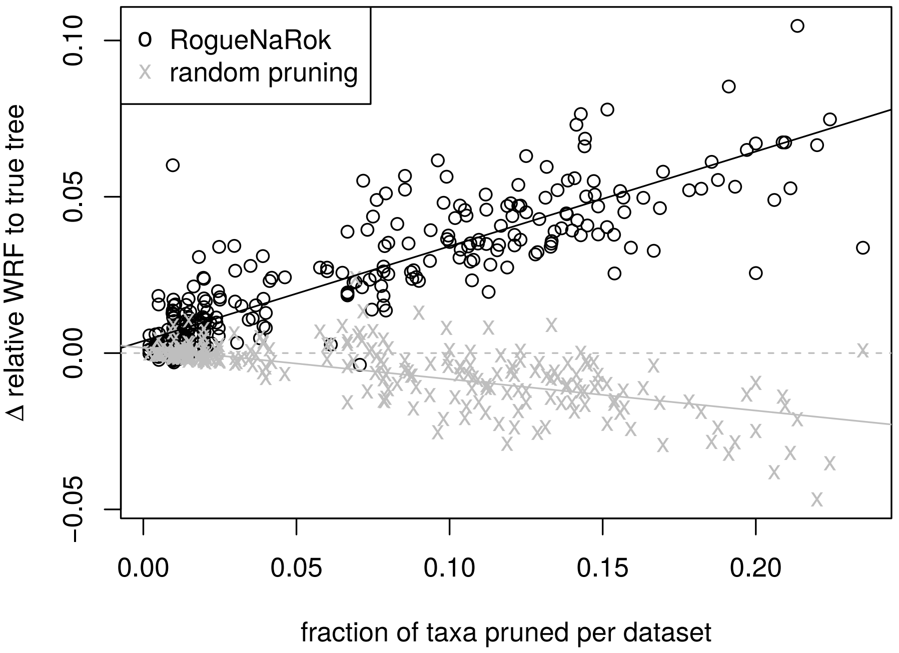


Figure 6.5: *Top:* Improvement of relative RF-distance of ML trees to the corresponding true tree after removal of rogue taxa as suggested by RogueNaRok (black) and after removal of a random set of taxa of equal size for each dataset (gray). *Bottom:* analog comparison for rWRF-distances.

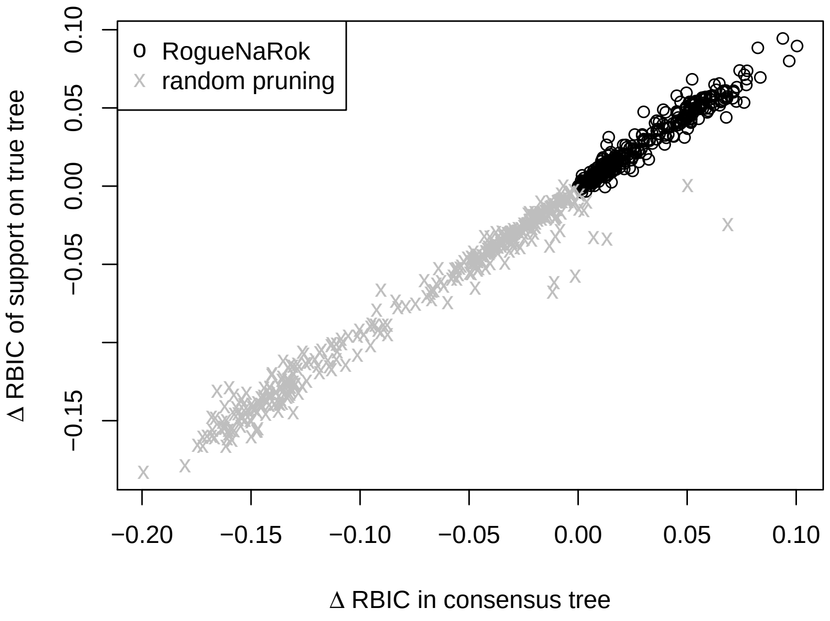


Figure 6.6: As pruning rogue taxa increases support in consensus trees, the overlap of consensus tree support with the true tree increases proportionally. Analogously, random prunings decrease support in consensus tree and the overlap with the true tree.

if pruned from the underlying bootstrap trees. In this experiment, we focus on rogue taxon identification for MRC trees.

A comparison of the RF-distance of a consensus trees to the corresponding true tree is problematic. Here, low RF-distances may be a consequence of a poorly resolved consensus tree (which is specifically the case before rogue taxon removal). Therefore, uninformative consensus trees appear favorable under this measure. Thus, for measuring the improvement of phylogenetic accuracy for consensus trees, we instead draw bootstrap support on the true tree before and after pruning rogue taxa. In both cases, we only consider branches with bipartition support $> 50\%$. Since the MRC tree does not contain bipartitions with support $\leq 50\%$, this approach describes the overlap/congruence between the MRC tree and the true tree. In other words, we obtain the true support of the consensus tree.

The measure for support in the consensus tree we use for this experiment is the RBIC (i.e., the default optimization criterion of RogueNaRok). We define it as the sum of support values in a tree (ranging between 0.0 and 1.0) normalized by the maximum possible support in a fully resolved tree with the initial number of taxa *before* pruning.

Fig. 6.6 shows that, the true support (as defined here) recovered in the consensus tree increases proportionally with the support that is added to the consensus tree after pruning rogue taxa. Inversely, pruning an equal number of random taxa decreases support values in the consensus and the true support recovered in the consensus tree also decreases proportionally. We conclude that, pruning rogue taxa as identified by RogueNaRok yields consensus trees that are closer to the true tree.

Typically, our algorithm identifies rogue taxa that have a strong negative impact on the overall support during early iterations, while late iterations prune rogue taxa that only slightly increase support. We may want to stop the algorithm, once the support gain in the reduced consensus tree drops below a specified threshold. We stopped the algorithm, as soon as the accumulated support improvement for an iteration was below 30%, 20%, and 10% respectively.

Fig. 6.6 depicts the true support improvement in the consensus tree divided by the consensus tree support improvement as reported by RogueNaRok. This ratio can be greater than 1.0, in cases where pruning rogue taxa decreased the RF-distance of the consensus tree to the true tree. We observe that, the aforementioned ratio of true support is not significantly influenced by any stopping threshold for the algorithm. This indicates that, even rogue taxa with weak negative impact on consensus tree support, increase (although to a lesser extent) the true support, if pruned from bootstrap trees.

6.5 Webservice

As described throughout **Sect.** 6.3, the RNR algorithm offers a variety of parameters. It is legitimate to examine several parameter settings in order to extract the desired (resp., the maximum amount of) information from a tree set. A practitioner can choose

- among various optimization criteria (i.e., resolution- versus support-based; with and without penalizing dropset sizes),
- between various consensus thresholds (MRC, SC or GMRC) or can try to maximize the support drawn onto a reference tree,
- different settings for the approximation parameter l (dropset size),
- to exclude sets of taxa from the rogue taxon analysis (e.g., taxa that are of central interest for a study will not be considered for pruning).

Specifically, the RBIC requires some user interaction, since under the RBIC RogueNaRok also identifies taxa as rogue that have a very modest impact on support values, if pruned. Thus, a user may potentially not want to prune all taxa that are identified as rogues, but only taxa from the initial iterations of the algorithm.

RogueNaRok is implemented in `C` (including a parallelized version) and is part of the RogueNaRok software package that also contains implementations of the MAST, alternative (and less efficient) rogue taxon algorithms (i.e., node distance and stability measures) as well as a tool for pruning taxa from tree sets. To improve usability, we also implemented a RogueNaRok webservice.

The RogueNaRok webservice is implemented using the `Ruby on Rails` web application framework. The framework provides an implementation for the Model-View-Controller pattern. Within this framework, we implemented the *model* consisting of so-called *active* objects (e.g., a `Search` object that represents a rogue taxon analysis) that are mapped to entries in an underlying `SQL` database. Furthermore, a *view* must be defined that specifies the web interface. Finally, *controller* classes implement the application logic that determines how model instances are updated and how views change for the user after, for instance, executing a rogue taxon analysis.

In a typical session, the workflow starts with users uploading their data (i.e., a tree set and optionally a reference tree). In the subsequent work-bench view, users can configure and launch rogue taxon analyses with all of the aforementioned settings (e.g., dropset size). For an analysis started by the user, a script is generated and queued in a batch-queuing system on an underlying compute cluster. When the job has completed, the model (i.e., active object) is annotated with the results of the rogue taxon analysis. As a consequence, the view is updated. In the beginning, the result view lists the taxa in the tree set. For each rogue taxon analysis, we add a column to this list that specifies the RBIC improvement, if the respective taxon is pruned (given that all taxa of previous iterations have been pruned). Users can sort the resulting table according to each column. Cells are color-coded in order to indicate how strongly a rogue taxon affects the consensus tree. Thus, users can easily identify which taxa appear as rogues for a given parameter setting or consensus threshold. Given the annotated taxon table, users can decide upon a set of taxa to prune from the tree set and visualize the resulting consensus tree (resp., reference tree annotated with support values). We use RogueNaRok and RAxML to compute a consensus tree and display it in a `Java` applet using the Archaeopteryx tree viewer [131]. Archaeopteryx displays all visualizations created so far as tabs in a separate window. Taxa that have been pruned from the tree set are colored red in all trees that were created in previous visualization invocations. Thus, users can locate the position of rogue taxa before pruning in less informative consensus trees that still contain rogue taxa.

There also exists a Ruby model class for sessions. Thus, users can close their current session and resume it at a later point in time with a completely restored work-bench view. This option is particularly practical for time consuming rogue taxon analysis.

6.6 Summary

In this Chapter, we described phylogenetic post-processing algorithms for building consensus trees or drawing support from a tree set onto a reference tree. In particular, we discussed the rogue taxon problem in which otherwise well-resolved evolutionary relationships among taxa are obfuscated by a comparably small set of taxa that assume various topological positions in this tree set. We briefly described related algorithms (i.e., the BMA and STA) for the identification of rogue taxa, before introducing the RogueNaRok algorithm.

Similar to the BMA, the RogueNaRok algorithm computes drop sets from non-consensus bipartitions that have been extracted from the tree set. An essential contribution of RogueNaRok is the usage of a merger graph that describes which bipartitions can be merged by pruning which minimal dropsets. Using the merger graph, the RogueNaRok algorithm can determine exactly (in contrast to the BMA), how a bipartition profile changes for a specific dropset. In the RogueNaRok algorithm, we subsequently evaluate these changes using a flexible optimization criterion. Two ideas improve performance substantially. Firstly, we only compute dropsets up to a certain size. The categorization of bipartitions according to their partition-sizes allows for an additional reduction of the otherwise quadratic number of comparisons that is necessary to compute all relevant dropsets. Secondly, we use the merger graph information to maintain (i.e., update) a bipartition profile instead of extracting it anew in each iteration.

For a diverse set of 24 real-world datasets, we found that RogueNaRok is up to $3,640\times$ faster than STA, if the dropset size is set to 1. Even with a dropset size of 2, RogueNaRok still outperforms the STA algorithm. If we account for the fact that RogueNaRok is substantially more sensitive (w.r.t. the number of rogues identified) than the BMA, then RogueNaRok is about one order of magnitude faster than the BMA per identified rogue taxon. We observed that, RogueNaRok (if invoked with dropset sizes of 2 or 3) consistently outperforms STA and a modified version of the BMA with respect to result quality. That is, after pruning taxa identified by RogueNaRok, we obtain more informative consensus trees, or reference trees with support values. In a large-scale simulation experiment, we demonstrated that pruning rogue taxa (as identified by RogueNaRok) yields consensus trees and reference trees that are closer to the "true" tree used for simulating the datasets. We thereby demonstrated that pruning rogue taxa can increase the phylogenetic accuracy of any phylogenetic inference method that calculates/yields a tree set.

Finally, we described a webservice for RogueNaRok. The webservice allows users to execute several rogue taxon searches and provides a convenient summary view of all conducted searches. Furthermore, users can interactively decide which taxa to prune and visualize the induced consensus trees / reference trees with support.

7 Conclusion and Outlook

With recent developments in phyloinformatics, evolutionary biology starts to shift its focus from an experiment-driven laboratory discipline towards an increasingly data-driven approach. Yet, the size of datasets for computational methods often imposes a hard limit on the feasibility of a study. In this thesis, we focused on large-scale datasets in the context of Bayesian phylogenetic inference. We developed two software packages (ExaBayes and RogueNaRok) which clear the way for future ambitious analytic projects. In each problem setting examined in this thesis, we strove to (i) improve the underlying method or algorithm, (ii) provide highly efficient implementations and (iii) address scalability issues such that problem instances of immense size become solvable given access to HPC resources.

ExaBayes is a self-contained production-level software package for BI in phylogenetics. With respect to functionality and usability, the scope of ExaBayes is limited compared to popular tools like Beast and MrBayes that have both been developed by small teams for almost one, respectively two decades by now. However, ExaBayes currently is the only Bayesian tool available that has specifically been designed and optimized for employment on large clusters and super-computers. We demonstrated its potential by inferring a tree from the largest input alignment to date (requiring more than 4.6 terabyte of RAM). In particular, we examined what can be expected of confidence intervals when we increase the size of the input alignment by several orders of magnitude.

ExaBayes has been developed in a comparably short amount of time, because of simultaneous development of ExaML and the PLL in the Exelixis lab. Specifically, the usage of the PLL will allow ExaBayes to incorporate future methodological and technological progress rapidly. For instance, efficient execution of ExaBayes on the Intel Xeon Phi co-processor [63] is currently only hindered by the insufficient support of the `C++11` standard in Intel's `C++` compiler. The hybrid parallelization in ExaBayes is encapsulated and can therefore easily be ported to similar Bayesian inference software that requires a hierarchical three-tier parallelization. In the course of this thesis, the need to infer phylogenetic trees from genome-sized datasets has evolved from being a predominantly academic exercise into a real analytical requirement [see 57, 77].

With the introduction of the ppSPR move and the development of the NR-based Γ proposal for branch lengths, ExaBayes has evolved from its initial state where it simply relied on modified MrBayes proposals. The unique property of the NR-based Γ proposal that samples branches *de novo* could not be used to full capacity in the hybrid proposals that were discussed here. However, given its efficiency, the NR-based Γ proposal has the potential of becoming one of the standard proposals for branch length integration in the future. Furthermore, this *de novo* proposal may play an essential role in the design of a more radical class of hybrid proposals. An alternative to elementary topological operations (i.e., NNI, SPR and TBR) is to erase a randomly chosen set of adjacent branches from the tree completely. The tree then is decomposed into n subtrees. The number of topologies that can be constructed from the n subtrees is equal to the number of different topologies with n taxa (see **Eq. 2.2**). We then can evaluate these topologies given a scoring function (as typically employed in guided proposals such as the ppSPR or the parsSPR) and propose a new topology proportionally to its score. This would allow to propose topologies that are substantially more diverse than those that could be proposed by using a single elementary operation. Most likely, no meaningful mapping of existing branch lengths to branches in the proposed topology can be determined. Thus, the simultaneous *de novo* proposal of branch lengths would become a necessity to such radical proposals. Interestingly, our analyses have shown that branch lengths in current topological proposals are not the limiting factor to achieving convergence. Thus, we suspect that the small diversity of topological proposals currently is the bottleneck that limits BI in phylogenetics.

The NR-based Γ proposal employs a highly accurate approximation of branch length posteriors. The quality of this approximation raises the question, whether the expensive numeric integration over branch lengths can be avoided altogether. While no closed form of the branch length posterior is available, a branch-length-free phylogenetic model is conceivable that essentially approximates the likelihood or PP for any choice of branch lengths. A particularly expensive way of implementing this idea would be a discretization of the approximate branch length posterior into a fixed number of representative branch lengths (similar to the discretization of the Γ model of among site rate-heterogeneity). However, more elegant solutions may exist. We expect that a branch-length-free model would alleviate one of the essential difficulties of phylogenetic inference, that is having to infer (or optimize) the branch lengths and topology at the same time (regardless of whether ML or BI is employed).

Finally, the RogueNaRok algorithm has been presented. It outperforms previous rogue taxon identification algorithms both in terms of runtime as well as in terms of result quality (i.e., complex rogue taxa scenarios can be identified). Given the runtime efficiency of RogueNaRok, users typically obtain the results of their analysis instantaneously for most common input datasets. This allows for a interactive and exploratory workflow when using the accompanying web service. An application of RogueNaRok that goes beyond the original motivation of the algorithm is the construction of constraint trees that allow for a more robust phylogenetic inference by reducing the search space. When poor phylogenetic signal in the alignment obstructs robust phylogenetic inference, RogueNaRok can be used to identify and prune rogue taxa from bootstrap trees or trees that have been sampled via BI. Subsequently, the rogue-free consensus tree can be used as a constraint tree in an attempt to infer more robust phylogenetic trees (using ML or BI) with rogue taxa included. Alternatively, rogues can be re-inserted into the phylogeny using the evolutionary placement algorithm (EPA) [15]. The EPA computes likelihood weights for each possible insertion branch of a taxon in a given topology. Such a procedure allows for making statements about the evolutionary history of rogue taxa with respect to a stable reference phylogeny.

List of Figures

List of Tables

List of Acronyms

AA	amino acid
ASDSF	average standard deviation of split frequencies
AVX	advanced vector extension
BI	Bayesian inference
BMA	bipartition merging algorithm
bp	base pair
CDD	cyclic data distribution
CDF	cumulative distribution function
CPU	central processing unit
CPV	conditional probability vector
ctMC	continuous-time Markov chain
CV	coefficient of variation
DLB	divisible load balancing
DNA	desoxyribonucleic acid
EAP	expected acceptance probability
EPA	evolutionary placement algorithm
eSPR	extending subtree-pruning and regrafting
ESS	effective sample size
eTBR	extending tree-bisection and reconnection
GB	giga byte
GMRC	greedily refined majority-rule consensus

GTR	generalized time-reversible
HPC	high-performance computing
I/O	input/output
ILP	integer linear programming
ILS	incomplete lineage sorting
LBA	long branch attraction
LRZ	Leibniz Supercomputing Centre
MAST	maximum agreement subtree
MB	mega byte
MC^3	Metropolis-coupled MCMC
MCMC	Markov chain Monte Carlo
MH	Metropolis-Hastings
MISC	maximum-information subtree consensus
ML	maximum likelihood
MP	maximum parsimony
MPI	Message Passing Interface
MPS	multi-processor scheduling
MRC	majority-rule consensus
MSA	multiple sequence alignment
MSDSF	maximum standard deviation of split frequencies
MT19937	19,937-based Mersenne Twister
NCL	Nexus class library
NJ	neighbor joining
NNI	nearest neighbor interchange
NR	Newton-Raphson
NUMA	non-uniform memory access

OAP	observed acceptance probability
parsSPR	parsimony-guided subtree-pruning and regrafting
PDF	probability density function
PE	parallel entity
PImpl	pointer-to-implementation
PLF	phylogenetic likelihood function
PLL	phylogenetic likelihood library
PP	posterior probability
ppSPR	posterior-guided subtree-pruning and regrafting
PRNG	pseudo random number generator
PSR	per-site rate
PSRF	potential scale reduction factor
RAM	random access memory
RBIC	relative bipartition information criterion
RF	Robinson-Foulds
RIC	relative information content
rjMCMC	reversible jump Markov chain Monte Carlo
RNA	ribonucleic acid
RNR	RogueNaRok algorithm
rWRF	relative weighted Robinson-Foulds-distance
SC	strict consensus
sec	second
SEV	subtree equality vector
SPMC	single producer multiple consumer
SPR	subtree-pruning and regrafting
SPSC	single producer single consumer

SSE	streaming SIMD extensions
STA	single-taxon algorithm
STL	standard template library
stNNI	stochastic nearest neighbor interchange
TBR	tree-bisection and reconnection
TOL	tree of life
UML	unified modeling language
UPGMA	unweighted pair group method with arithmetic mean

Bibliography

[1] AJ **Aberer**, K Kobert, and A Stamatakis. "ExaBayes: Massively Parallel Bayesian Tree Inference for the Whole-Genome Era". In: *Molecular biology and evolution* 31.10 (2014), pp. 2553–2556.

[2] AJ **Aberer**, D Krompass, and A Stamatakis. "Pruning rogue taxa improves phylogenetic accuracy: an efficient algorithm and webservice". In: *Systematic biology* 62.1 (2013), pp. 162–166.

[3] AJ **Aberer** and A Stamatakis. "A simple and accurate method for rogue taxon identification". In: *Bioinformatics and Biomedicine (BIBM), 2011 IEEE International Conference on.* IEEE. 2011, pp. 118–122.

[4] AJ **Aberer** and A Stamatakis. "Rapid forward-in-time simulation at the chromosome and genome level". In: *BMC bioinformatics* 14.1 (2013), p. 216.

[5] AJ **Aberer**, A Stamatakis, and F Ronquist. "An Efficient Independence Sampler for Updating Branches in Bayesian Markov chain Monte Carlo Sampling of Phylogenetic Trees." In: *Systematic biology* 65.1 (2016), pp. 161–176.

[6] N Alachiotis and A Stamatakis. "FPGA acceleration of the phylogenetic parsimony kernel?" In: *Field Programmable Logic and Applications (FPL), 2011 International Conference on.* IEEE. 2011, pp. 417–422.

[7] G Altekar, S Dwarkadas, JP Huelsenbeck, and F Ronquist. "Parallel Metropolis coupled Markov chain Monte Carlo for Bayesian phylogenetic inference." In: *Bioinformatics (Oxford, England)* 20.3 (Feb. 2004), pp. 407–15. ISSN: 1367-4803.

[8] SF Altschul, W Gish, W Miller, EW Myers, and DJ Lipman. "Basic local alignment search tool". In: *Journal of molecular biology* 215.3 (1990), pp. 403–410.

[9] GM Amdahl. "Validity of the single processor approach to achieving large scale computing capabilities". In: *Proceedings of the April 18-20, 1967, spring joint computer conference.* ACM. 1967, pp. 483–485.

[10] A Amir and D Keselman. "Maximum agreement subtree in a set of evolutionary trees: Metrics and efficient algorithms". In: *SIAM Journal on Computing* 26.6 (1997), pp. 1656–1669.

[11] YF Atchadé, GO Roberts, and JS Rosenthal. "Towards optimal scaling of metropolis-coupled Markov chain Monte Carlo". In: *Statistics and Computing* 21.4 (July 2010), pp. 555–568. ISSN: 0960-3174.

[12] DL Ayres, A Darling, DJ Zwickl, P Beerli, MT Holder, PO Lewis, JP Huelsenbeck, F Ronquist, DL Swofford, MP Cummings, A Rambaut, and Ma Suchard. "BEAGLE: an application programming interface and high-performance computing library for statistical phylogenetics." In: *Systematic biology* 61.1 (Jan. 2012), pp. 170–3. ISSN: 1076-836X.

[13] P Benner, M Bačák, and PY Bourguignon. "Point estimates in phylogenetic reconstructions". In: *Bioinformatics* 30.17 (2014), pp. i534–i540.

[14] DA Benson, I Karsch-Mizrachi, DJ Lipman, J Ostell, BA Rapp, and DL Wheeler. "GenBank". In: *Nucleic acids research* 28.1 (2000), pp. 15–18.

[15] SA Berger, D Krompass, and A Stamatakis. "Performance, accuracy, and web server for evolutionary placement of short sequence reads under maximum likelihood". In: *Systematic biology* 60.3 (2011), p. 291.

[16] SA Berger and A Stamatakis. "Accuracy and performance of single versus double precision arithmetics for maximum likelihood phylogeny reconstruction". In: *Parallel Processing and Applied Mathematics.* Springer, 2010, pp. 270–279.

[17] SA Berger and A Stamatakis. "Aligning short reads to reference alignments and trees". In: *Bioinformatics* 27.15 (2011), pp. 2068–2075.

[18] RP Brent. *Algorithms for minimization without derivatives.* Prentice-Hall, 1973.

[19] JM Brown, SM Hedtke, AR Lemmon, and EM Lemmon. "When trees grow too long: investigating the causes of highly inaccurate Bayesian branch-length estimates". In: *Systematic Biology* 59.2 (2010), pp. 145–161.

[20] D Bryant. "A classification of consensus methods for phylogenetics". In: *Bioconsensus: DIMACS Working Group Meetings on Bioconsensus: October 25-26, 2000 and October 2-5, 2001, DIMACS Center.* Amer Mathematical Society. 2003, p. 163. ISBN: 0821831976.

[21] D Bryant, J Tsang, P Kearney, and M Li. "Computing the quartet distance between evolutionary trees". In: *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms.* Society for Industrial and Applied Mathematics. 2000, pp. 285–286.

[22] B Chor and T Tuller. "Maximum likelihood of evolutionary trees: hardness and approximation". In: *Bioinformatics* 21.suppl 1 (2005), pp. i97–i106.

[23] WS Cleveland and SJ Devlin. "Locally weighted regression: an approach to regression analysis by local fitting". In: *Journal of the American Statistical Association* 83.403 (1988), pp. 596–610.

[24] D Darriba, AJ **Aberer**, T Flouri, TA Heath, F Izquierdo-Carrasco, and A Stamatakis. "Boosting the performance of Bayesian divergence time estimation with the Phylogenetic Likelihood Library". In: *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International.* IEEE. 2013, pp. 539–548.

[25] A Deepak, J Dong, and D Fernández-Baca. "Identifying rogue taxa through reduced consensus: NP-Hardness and exact algorithms". In: *Bioinformatics Research and Applications.* Springer, 2012, pp. 87–98.

[26] ML Delignette-Muller, R Pouillot, JB Denis, and C Dutang. *fitdistrplus: help to fit of a parametric distribution to non-censored or censored data.* R package version 1.0-2. 2014.

[27] E Dell'Ampio, K Meusemann, NU Szucsich, RS Peters, B Meyer, J Borner, M Petersen, AJ **Aberer**, A Stamatakis, MG Walzl, *et al.* "Decisive Data Sets in Phylogenomics: Lessons from Studies on the Phylogenetic Relationships of Primarily Wingless Insects". In: *Molecular biology and evolution* 31.1 (2014), pp. 239–249.

[28] WF Doolittle and E Bapteste. "Pattern pluralism and the Tree of Life hypothesis". In: *Proceedings of the National Academy of Sciences* 104.7 (2007), pp. 2043–2049.

[29] CW Dunn, A Hejnol, DQ Matus, K Pang, WE Browne, SA Smith, E Seaver, GW Rouse, M Obst, GD Edgecombe, *et al.* "Broad phylogenomic sampling improves resolution of the animal tree of life". In: *Nature* 452.7188 (2008), pp. 745–749.

[30] RC Edgar. "MUSCLE: multiple sequence alignment with high accuracy and high throughput". In: *Nucleic acids research* 32.5 (2004), pp. 1792–1797.

[31] J Felsenstein. *{PHYLIP}(Phylogeny Inference Package) version 3.6 a3.* 2002.

[32] J Felsenstein. "Cases in which parsimony or compatibility methods will be positively misleading". In: *Systematic Biology* 27.4 (1978), pp. 401–410.

[33] J Felsenstein. "Confidence limits on phylogenies: an approach using the bootstrap". In: *Evolution* (1985), pp. 783–791.

[34] J Felsenstein. "Evolutionary trees from DNA sequences: a maximum likelihood approach". In: *Journal of molecular evolution* 17.6 (1981), pp. 368–376.

[35] J Felsenstein. *Inferring phylogenies.* Vol. 2. Sinauer Associates Sunderland, 2004.

[36] J Felsenstein and GA Churchill. "A Hidden Markov Model approach to variation among sites in rate of evolution." In: *Molecular Biology and Evolution* 13.1 (1996), pp. 93–104.

[37] WM Fitch, E Margoliash, *et al.* "Construction of phylogenetic trees". In: *Science* 155.760 (1967), pp. 279–284.

[38] W Fletcher and Z Yang. "INDELible: a flexible simulator of biological sequence evolution." In: *Molecular biology and evolution* 26.8 (Aug. 2009), pp. 1879–88. ISSN: 1537-1719.

[39] T Flouri, F Izquierdo-Carrasco, D Darriba, A **Aberer**, LT Nguyen, B Minh, A Von Haeseler, and A Stamatakis. "The phylogenetic likelihood library". In: *Systematic biology* 64.2 (2015), pp. 356–362.

[40] T Flouri, K Kobert, SP Pissis, and A Stamatakis. "An optimal algorithm for computing all subtree repeats in trees". In: *Combinatorial algorithms.* Springer, 2013, pp. 269–282.

[41] A Gelman and DB Rubin. "Inference from iterative simulation using multiple sequences". In: *Statistical science* (1992), pp. 457–472.

[42] CJ Geyer. "Markov chain Monte Carlo maximum likelihood". In: *Computing Science and Statistics: Proceedings of the 23rd Symposium of the Interface.* Fairfax Station VA: Interface Foundation. Keramidas, E.M., 1991, pp. 156–163.

[43] J Giacomoni, T Moseley, and M Vachharajani. "FastForward for efficient pipeline parallelism: a cache-optimized concurrent lock-free queue". In: *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming.* ACM. 2008, pp. 43–52.

[44] PE Gill, W Murray, and MH Wright. "Practical optimization". In: *London: Academic Press* 1 (1981).

[45] PJ Green. "Trans-dimensional markov chain monte carlo". In: *Oxford Statistical Science Series* (2003), pp. 179–198.

[46] S Guindon, JF 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". In: *Systematic biology* 59.3 (2010), pp. 307–321.

[47] WK Hastings. "Monte Carlo sampling methods using Markov chains and their applications". In: *Biometrika* 57.1 (1970), pp. 97–109.

[48] J Heled and AJ Drummond. "Bayesian inference of species trees from multilocus data". In: *Molecular biology and evolution* 27.3 (2010), pp. 570–580.

[49] M Herlihy and N Shavit. "The art of multiprocessor programming". In: *PODC*. Vol. 6. 2006, pp. 1–2.

[50] DG Higgins and PM Sharp. "CLUSTAL: a package for performing multiple sequence alignment on a microcomputer". In: *Gene* 73.1 (1988), pp. 237–244.

[51] A Hobolth, JY Dutheil, J Hawks, MH Schierup, and T Mailund. "Incomplete lineage sorting patterns among human, chimpanzee, and orangutan suggest recent orangutan speciation and widespread selection". In: *Genome research* 21.3 (2011), pp. 349–356.

[52] S Höhna and AJ Drummond. "Guided Tree Topology Proposals for Bayesian Phylogenetic Inference". In: *Systematic Biology* 61.1 (2012), pp. 1–11.

[53] MT Holder, PO Lewis, DL Swofford, and B Larget. "Hastings ratio of the LOCAL proposal used in Bayesian phylogenetics". In: *Systematic biology* 54.6 (2005), pp. 961–965.

[54] J Hopcroft and R Tarjan. "Algorithm 447: Efficient algorithms for graph manipulation". In: *Communications of the ACM* 16.6 (1973), pp. 372–378.

[55] JP Huelsenbeck, B Larget, and ME Alfaro. "Bayesian phylogenetic model selection using reversible jump Markov chain Monte Carlo". In: *Molecular Biology and Evolution* 21.6 (2004), pp. 1123–1133.

[56] F Izquierdo-Carrasco, Sa Smith, and A Stamatakis. “Algorithms, data structures, and numerics for likelihood-based phylogenetic inference of huge trees.” In: *BMC bioinformatics* 12.1 (Jan. 2011), p. 470. ISSN: 1471-2105.

[57] ED Jarvis, S Mirarab, AJ **Aberer**, B Li, P Houde, C Li, SY Ho, BC Faircloth, B Nabholz, JT Howard, A Suh, CC Weber, RRd Fonseca, J Li, F Zhang, H Li, L Zhou, N Narula, L Liu, G Ganapathy, B Boussau, MS Bayzid, V Zavidovych, S Subramanian, T Gabaldón, S Capella-Gutiérrez, J Huerta-Cepas, B Rekepalli, K Munch, M Schierup, B Lindow, WC Warren, D Ray, RE Green, MW Bruford, X Zhan, A Dixon, S Li, N Li, Y Huang, EP Derryberry, MF Bertelsen, FH Sheldon, RT Brumfield, CV Mello, PV Lovell, M Wirthlin, MPC Schneider, F Prosdocimi, JA Samaniego, AMV Velazquez, A Alfaro-Núñez, PF Campos, B Petersen, T Sicheritz-Ponten, A Pas, T Bailey, P Scofield, M Bunce, DM Lambert, Q Zhou, P Perelman, AC Driskell, B Shapiro, Z Xiong, Y Zeng, S Liu, Z Li, B Liu, K Wu, J Xiao, X Yinqi, Q Zheng, Y Zhang, H Yang, J Wang, L Smeds, FE Rheindt, M Braun, J Fjeldsa, L Orlando, FK Barker, KA Jønsson, W Johnson, KP Koepfli, S O’Brien, D Haussler, OA Ryder, C Rahbek, E Willerslev, GR Graves, TC Glenn, J McCormack, D Burt, H Ellegren, P Alström, SV Edwards, A Stamatakis, DP Mindell, J Cracraft, EL Braun, T Warnow, W Jun, MTP Gilbert, and Ga Zhang. “Whole-genome analyses resolve early branches in the tree of life of modern birds”. In: *Science* 346.6215 (2014), pp. 1320–1331.

[58] TH Jukes and CR Cantor. “Evolution of protein molecules”. In: *Mammalian protein metabolism* 3 (1969), pp. 21–132.

[59] M Kimura. “A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences”. In: *Journal of molecular evolution* 16.2 (1980), pp. 111–120.

[60] K Kobert, T Flouri, AJ **Aberer**, and A Stamatakis. “The Divisible Load Balance Problem and Its Application to Phylogenetic Inference”. In: *Algorithms in Bioinformatics*. Springer Berlin Heidelberg, 2014, pp. 204–216.

[61] B Kolaczkowski and JW Thornton. “Effects of branch length uncertainty on Bayesian posterior probabilities for phylogenetic hypotheses”. In: *Molecular biology and evolution* 24.9 (2007), pp. 2108–2118.

[62] AM Kozlov, AJ **Aberer**, and A Stamatakis. “ExaML version 3: a tool for phylogenomic analyses on supercomputers”. In: *Bioinformatics (Oxford, England)* 31.15 (Aug. 2015), pp. 2577–2579.

[63] AM 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*. IEEE. 2014, pp. 518–527.

[64] C Lakner, P van der Mark, JP Huelsenbeck, B Larget, and F Ronquist. “Efficiency of Markov chain Monte Carlo tree proposals in Bayesian phylogenetics.” In: *Systematic biology* 57.1 (Feb. 2008), pp. 86–103. ISSN: 1063-5157.

[65] C Lakner, P Van Der Mark, JP Huelsenbeck, B Larget, and F Ronquist. "Efficiency of Markov chain Monte Carlo tree proposals in Bayesian phylogenetics". In: *Systematic biology* 57.1 (2008), pp. 86–103.

[66] B Larget. "Introduction to Markov chain Monte Carlo methods in molecular evolution". In: *Statistical methods in molecular evolution.* Springer, 2005, pp. 45–62.

[67] B Larget and DL Simon. "Markov chain Monte Carlo algorithms for the Bayesian analysis of phylogenetic trees". In: *Molecular Biology and Evolution* 16 (1999), pp. 750–759.

[68] SQ Le and O Gascuel. "An improved general amino acid replacement matrix". In: *Molecular biology and evolution* 25.7 (2008), pp. 1307–1320.

[69] P L'Ecuyer and R Simard. "TestU01: AC library for empirical testing of random number generators". In: *ACM Transactions on Mathematical Software (TOMS)* 33.4 (2007), p. 22.

[70] PO Lewis. "NCL: a C++ class library for interpreting data files in NEXUS format". In: *Bioinformatics* 19.17 (2003), pp. 2330–2331.

[71] A Löytynoja and N Goldman. "Phylogeny-aware gap placement prevents errors in sequence alignment and evolutionary analysis". In: *Science* 320.5883 (2008), pp. 1632–1635.

[72] M Lynch, R Bürger, D Butcher, and W Gabriel. "The mutational meltdown in asexual populations". In: *Journal of Heredity* 84.5 (1993), pp. 339–344.

[73] DR Maddison, DL Swofford, and WP Maddison. "NEXUS: an extensible file format for systematic information." In: *Systematic biology* 46.4 (Dec. 1997), pp. 590–621. ISSN: 1063-5157.

[74] WP Maddison and DR Maddison. *Mesquite: a modular system for evolutionary analysis. Version 3.01.* 2014. URL: `http://mesquiteproject.org`.

[75] DC Marshall. "Cryptic failure of partitioned Bayesian phylogenetic analyses: lost in the land of long trees". In: *Systematic Biology* 59.1 (2010), pp. 108–117.

[76] N Metropolis, AW Rosenbluth, MN Rosenbluth, AH Teller, and E Teller. "Equation of state calculations by fast computing machines". In: *The journal of chemical physics* 21.6 (1953), pp. 1087–1092.

[77] B Misof, S Liu, K Meusemann, RS Peters, A Donath, C Mayer, PB Frandsen, J Ware, T Flouri, RG Beutel, O Niehuis, M Petersen, F Izquierdo-Carrasco, T Wappler, J Rust, AJ **Aberer**, U Aspöck, H Aspöck, D Bartel, A Blanke, S Berger, A Böhm, TR Buckley, B Calcott, J Chen, F Friedrich, M Fukui, M Fujita, C Greve, P Grobe, S Gu, Y Huang, LS Jermiin, AY Kawahara, L Krogmann, M Kubiak, R Lanfear, H Letsch, Y Li, Z Li, J Li, H Lu, R Machida, Y Mashimo, P Kapli, DD McKenna, G Meng, Y Nakagaki, JLN Heredia, M Ott, Y Ou, G Pass, L Podsiadlowski, H Pohl, BMv Reumont, K Schütte, K Sekiya, S Shimizu, A Slipinski, A Stamatakis, W Song, X Su, NU Szucsich, M Tan, X Tan, M Tang, J

Tang, G Timelthaler, S Tomizuka, M Trautwein, X Tong, T Uchifune, MG Walzl, BM Wiegmann, J Wilbrandt, B Wipfler, TK Wong, Q Wu, G Wu, Y Xie, S Yang, Q Yang, DK Yeates, K Yoshizawa, Q Zhang, R Zhang, W Zhang, Y Zhang, J Zhao, C Zhou, L Zhou, T Ziesmann, S Zou, Y Li, X Xu, Y Zhang, H Yang, J Wang, J Wang, KM Kjer, and X Zhou. “Phylogenomics resolves the timing and pattern of insect evolution”. In: *Science* 346.6210 (2014), pp. 763–767.

[78] E Mossel and E Vigoda. “Phylogenetic MCMC algorithms are misleading on mixtures of trees”. In: *Science* 309.5744 (2005), pp. 2207–2209.

[79] MPI Forum. *MPI: A Message-Passing Interface Standard. Version 2.2.* available at: `http://www.mpi-forum.org` (last access: Oct. 2014). 2014.

[80] MPI Forum. *MPI: A Message-Passing Interface Standard. Version 3.0.* available at: `http://www.mpi-forum.org` (last access: Oct. 2014). 2014.

[81] HJ Muller. “The relation of recombination to mutational advance”. In: *Mutation Research/Fundamental and Molecular Mechanisms of Mutagenesis* 1.1 (1964), pp. 2–9.

[82] SB Needleman and CD Wunsch. “A general method applicable to the search for similarities in the amino acid sequence of two proteins”. In: *Journal of molecular biology* 48.3 (1970), pp. 443–453.

[83] JA Nelder and R Mead. “A simplex method for function minimization”. In: *The computer journal* 7.4 (1965), pp. 308–313.

[84] KC Nixon. “The parsimony ratchet, a new method for rapid parsimony analysis”. In: *Cladistics* 15.4 (1999), pp. 407–414.

[85] ND Pattengale, AJ **Aberer**, KM Swenson, A Stamatakis, and BM Moret. “Uncovering hidden phylogenetic consensus in large data sets”. In: *Computational Biology and Bioinformatics, IEEE/ACM Transactions on* 8.4 (2011), pp. 902–911.

[86] ND Pattengale, M Alipour, OR Bininda-Emonds, BM Moret, and A Stamatakis. “How many bootstrap replicates are necessary?” In: *Research in Computational Molecular Biology.* Springer Berlin Heidelberg, 2009, pp. 184–200.

[87] R Peters, K Meusemann, M Petersen, C Mayer, J Wilbrandt, T Ziesmann, A Donath, K Kjer, U Aspöck, H Aspöck, AJ **Aberer**, A Stamatakis, F Friedrich, F Hünefeld, O Niehuis, R Beutel, and B Misof. “The evolutionary history of holometabolous insects inferred from transcriptome-based phylogeny and comprehensive morphological data”. In: *BMC evolutionary biology* 14.1 (2014), p. 52.

[88] C Phillips and TJ Warnow. “The asymmetric median tree—a new model for building consensus trees”. In: *Combinatorial Pattern Matching.* Springer. 1996, pp. 234–252.

[89] KM Pickett and CP Randle. “Strange Bayes indeed: uniform topological priors imply non-uniform clade priors”. In: *Molecular phylogenetics and evolution* 34.1 (2005), pp. 203–211.

[90] D Posada and KA Crandall. “Modeltest: testing the model of DNA substitution.” In: *Bioinformatics* 14.9 (1998), pp. 817–818.

[91] B Rannala, T Zhu, and Z Yang. “Tail paradox, partial identifiability, and influential priors in Bayesian branch length inference”. In: *Molecular biology and evolution* 29.1 (2012), pp. 325–335.

[92] BD Redelings and MA Suchard. “Joint Bayesian estimation of alignment and phylogeny”. In: *Systematic Biology* 54.3 (2005), pp. 401–418.

[93] GO Roberts and JS Rosenthal. “Examples of adaptive MCMC”. In: *Journal of Computational and Graphical Statistics* 18.2 (2009), pp. 349–367.

[94] GO Roberts, A Gelman, WR Gilks, *et al.* “Weak convergence and optimal scaling of random walk Metropolis algorithms”. In: *The annals of applied probability* 7.1 (1997), pp. 110–120.

[95] DF Robinson and LR Foulds. “Comparison of phylogenetic trees”. In: *Math. Biosci.* 53 (1981), pp. 131–147. ISSN: 0025-5564.

[96] F Ronquist, B Larget, JP Huelsenbeck, JB Kadane, D Simon, and P van der Mark. “Comment on” Phylogenetic MCMC Algorithms Are Misleading on Mixtures of Trees””. In: *Science* 312.5772 (2006), pp. 367–367.

[97] N Saitou and M Nei. “The neighbor-joining method: a new method for reconstructing phylogenetic trees.” In: *Molecular biology and evolution* 4.4 (1987), pp. 406–425.

[98] JK Salmon, MA Moraes, RO Dror, and DE Shaw. “Parallel random numbers: as easy as 1, 2, 3”. In: *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*. IEEE. 2011, pp. 1–12.

[99] MJ Sanderson and HB Shaffer. “Troubleshooting molecular phylogenetic analyses”. In: *Annual Review of Ecology and Systematics* (2002), pp. 49–72.

[100] D Sankoff. “Minimal mutation trees of sequences”. In: *SIAM Journal on Applied Mathematics* 28.1 (1975), pp. 35–42.

[101] A Shafer and JC Hall. “Placing the mountain goat: a total evidence approach to testing alternative hypotheses”. In: *Molecular phylogenetics and evolution* 55.1 (2010), pp. 18–25.

[102] DL Simon and B Larget. “Bayesian analysis in molecular biology and evolution (BAMBE), version 2.03 beta”. In: *Department of Mathematics and Computer Science, Duquesne Univ., Pittsburgh, Pennsylvania* (2000).

[103] TF Smith and MS Waterman. “Identification of common molecular subsequences”. In: *Journal of molecular biology* 147.1 (1981), pp. 195–197.

[104] PH Sneath and RR Sokal. “Numerical taxonomy”. In: *Nature* 193.4818 (1962), pp. 855–860.

[105] PS Soltis, DE Soltis, *et al.* “Applying the bootstrap in phylogeny reconstruction”. In: *Statistical Science* 18.2 (2003), pp. 256–267.

[106] EA Sperling, KJ Peterson, and D Pisani. “Phylogenetic-signal dissection of nuclear housekeeping genes supports the paraphyly of sponges and the monophyly of Eumetazoa”. In: *Molecular biology and evolution* 26.10 (2009), pp. 2261–2274.

[107] A Stamatakis. “Phylogenetic models of rate heterogeneity: a high performance computing perspective”. In: *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International.* IEEE. 2006, pp. 8–16.

[108] A Stamatakis. “RAxML Version 8: A tool for Phylogenetic Analysis and Post-Analysis of Large Phylogenies”. In: *Bioinformatics* (2014).

[109] A Stamatakis. “RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models”. In: *Bioinformatics* 22.21 (2006), pp. 2688–2690.

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

[111] A Stamatakis, AJ **Aberer**, C Goll, SA Smith, SA Berger, and F Izquierdo-Carrasco. “RAxML-Light: a tool for computing terabyte phylogenies”. In: *Bioinformatics* 28.15 (2012), pp. 2064–2066.

[112] A Stamatakis and M Ott. “Load balance in the phylogenetic likelihood kernel”. In: *Parallel Processing, 2009. ICPP’09. International Conference on.* IEEE. 2009, pp. 348–355.

[113] M Steel and D Penny. “Origins of life: Common ancestry put to the test”. In: *Nature* 465.7295 (2010), pp. 168–169.

[114] DL Swofford. *PAUP*. Phylogenetic analysis using parsimony (* and other methods). Version 4.* 2003.

[115] S Tavaré. “Some probabilistic and statistical problems in the analysis of DNA sequences”. In: *Lect. Math. Life Sci* 17 (1986), pp. 57–86.

[116] R Thakur and WD Gropp. “Improving the performance of collective operations in MPICH”. In: *Recent Advances in Parallel Virtual Machine and Message Passing Interface.* Springer, 2003, pp. 257–267.

[117] RC Thomson and HB Shaffer. “Rapid progress on the vertebrate tree of life”. In: *BMC biology* 8.1 (2010), p. 19.

[118] RC Thomson and HB Shaffer. “Sparse Supermatrices for Phylogenetic Inference: Taxonomy, Alignment, Rogue Taxa, and the Phylogeny of Living Turtles”. In: *Systematic Biology* 59.1 (2010), pp. 42–58.

[119] JL Thorley and M Wilkinson. “Testing the phylogenetic stability of early tetrapods”. In: *Journal of Theoretical Biology* 200.3 (1999), pp. 343–344.

[120] L Tierney. “Markov chains for exploring posterior distributions”. In: *the Annals of Statistics* (1994), pp. 1701–1728.

[121] WN Venables and BD Ripley. *Modern Applied Statistics with S*. Fourth. ISBN 0-387-95457-0. New York: Springer, 2002. URL: `http://www.stats.ox.ac.uk/pub/MASS4`.

[122] LS Vinh and A von Haeseler. "IQPNNI: moving fast through tree space and stopping in time". In: *Molecular biology and evolution* 21.8 (2004), pp. 1565–1571.

[123] L Wang and T Jiang. "On the complexity of multiple sequence alignment". In: *Journal of computational biology* 1.4 (1994), pp. 337–348.

[124] JD Watson *et al.* "Molecular biology of the gene." In: *Molecular biology of the gene.* (1970).

[125] C Whidden and FA Matsen. "Quantifying MCMC Exploration of Phylogenetic Tree Space". In: *Systematic Biology* 64.3 (2015), pp. 472–491.

[126] M Wilkinson. "Majority-rule reduced consensus trees and their use in bootstrapping." In: *Mol Biol Evol* 13.3 (1996), pp. 437–444.

[127] Z Yang. *Computational molecular evolution.* Vol. 284. Oxford University Press Oxford, 2006.

[128] Z Yang. "Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites: approximate methods". In: *Journal of Molecular evolution* 39.3 (1994), pp. 306–314.

[129] C Zhang, B Rannala, and Z Yang. "Robustness of Compound Dirichlet Priors for Bayesian Inference of Branch Lengths". In: *Systematic Biology* 61.5 (2012), pp. 779–784.

[130] J Zhang and A Stamatakis. "The multi-processor scheduling problem in phylogenetics". In: *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International.* IEEE. 2012, pp. 691–698.

[131] CM Zmasek and SR Eddy. "ATV: display and manipulation of annotated phylogenetic trees". In: *Bioinformatics* 17.4 (2001), pp. 383–384.

[132] E Zuckerkandl and L Pauling. *Molecular disease, evolution and genetic heterogeneity.* 1962.

[133] D Zwickl. "Genetic algorithm approaches for the phylogenetic analysis of large biological sequence datasets under the maximum likelihood criterion". PhD thesis. 2006.