Mathematical Problems in Molecular Evolution and Next Generation Sequencing

zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

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

genehmigte

Dissertation

von

Kassian Kobert

aus Bergisch-Gladbach

Tag der mündlichen Prüfung: 2. Mai 2016

1. Gutachter: Prof. Dr. Alexandros Stamatakis, HITS Heidelberg, KIT Karlsruhe
2. Gutachter: Prof. Dr. Tanja Stadler, ETH Zürich

Acknowledgements

First of all, I want to thank Prof. Dr. Alexandros Stamatakis for his outstanding performance as my PhD supervisor.
Furthermore, I thank the HITS Stiftung, as well as the Klaus Tschira Stiftung, and the Tschira family personally, for providing young researchers like me with financial support during their PhD studies. Additionally, I am thankful for the productive work environment that the Heidelberg Institute for Theoretical Studies (HITS) provided to me.
I thank all my current, and past, colleagues and collaborators of the Scientific Computing Group at the HITS, for making my time at the HITS so enjoyable. In particular, and in no certain order, I want to mention Andre Aberer, Lucas Czech, Diego Darriba, Tomáš Flouri, Fernando Izquierdo, Paschalia Kapli, Alexey Kozlov, Pavlos Pavlidis, and Jiajie Zhang for being a pleasure to work with.

To my family,
who means the world to me.

Zusammenfassung der Dissertation

Hauptaugenmerk der Dissertation ist die Entwicklung von neuen mathematischen Methoden für die genetische Stammbaumanalyse. Moderne Sequenzierungsmethoden liefern heutzutage sehr große Datenmengen, so dass eine effiziente Berechnung der so genannten Likelihoodfunktion unumgänglich ist. Dies gilt insbesondere für Analysen nach dem Maximum-Likelihood Verfahren, sowie für die Baysche Inferenz auf Stammbäumen.

Die Dissertation ist gegliedert in zwei Hauptteile.

Im ersten Teil untersuchen wir die Schwierigkeit von genetischen Stammbaumanalysen auf partitionierten Datensätzen. Als partitionierte Sequenzalignments bezeichnen wir all solche Alignments, bei denen wir annehmen, dass verschiedenen Regionen (Partitionen/Ansammlung von Seiten) verschiedene evolutionäre Modelle zugrunde liegen können. Dieser Teil ist in drei Kapitel gegliedert.

In Kapitel 4 der vorliegenden Dissertation, zeigen wir, dass die Wahl des besten evolutionären Modells für jede der individuellen Partitionen schwer (d.h. NP-Schwer) ist, wenn ein gemeinsamer evolutionärer Stammbaum angenommen wird. Um NP-Vollständigkeit zu zeigen reduzieren wir das wohlbekannte $3 - SAT$ Problem [79].

Dieses Kapitel rechtfertigt die Verwendung von Approximationsalgorithmen um dieses Problem zu lösen.

Im Zusammenhang der Dissertation, dient der rigorose, detaillierte NP-Schwere Beweis als Beispiel für die folgenden Kapitel. Dem Leser werden in den darauf folgenden Kapiteln weitere Probleme vorgestellt die ihrerseits selbst NP-Schwer oder NP-Vollständig sind.

Dieses Resultat wurde von uns in [82] publiziert.

Weiterhin zeigen wir in Kapitel 5, dass die Annahme von verschiedenen evolutionären Modellen für verschiedene Partitionen weitere Fragen im Bezug auf Berechnungen von Stammbäumen auf Hochleistungs-Parallelrechnern aufwerfen. Die Zeit zur Berechnung der so genannten Likelihoodfunktion für einen Stammbaum, gegeben eines evolutionären Modells und eines Sequenzalignments für einen einzelnen Processor/Rechner hängt unter anderem von zwei Faktoren ab. Erstens ist die Anzahl der Seiten, die zu berechnen sind entscheidend. Zweitens wird für jedes evolutionäre Modell eine von der Länge der Partition unabhängige Zeit zur Initialisierung benötigt. Da jede Partition von einem eigenen Modell abhängt, stellt sich

hier das Problem, die einzelnen Seiten der Partitionen möglichst kosten- bzw. zeitsparend auf die parallelen Rechner zu verteilen.

Auch hier zeigen wir, dass eine optimale Aufteilung NP-Schwer ist. Allerdings präsentieren wir einen Approximationsalgorithmus, der das Problem in polynomieller Zeit nahezu optimal zu lösen vermag. Für den Fall, dass $P \neq NP$, kann kein anderer polynomieller Algorithmus ein besseres Ergebnis für diese Fragestellung garantieren.

Berechnungen auf partitionierten Datensätzen belegen, dass dieser Algorithmus die Laufzeit, verglichen mit den bisher verwendeten Methoden um einen Faktor von bis zu 5.9 verkürzen kann.

Publiziert haben wir dieses Ergebnis bereits in [81].

Als drittes Resultat im Zusammenhang mit partitionierten Datensätzen haben wir in Kapitel 6 die so genannte Internode Certainty auf Teilbäumen untersucht. Das Bestreben der Internode Certainty ist es, ein Maß für die Konfidenz an inneren Knoten eines Stammbaumes darzustellen, welches nicht nur die absolute Anzahl von Beobachtungen widerspiegelt, sondern auch quantifiziert, wie sehr eine Bipartition im Konflikt mit anderen Beobachtungen steht. Hierfür wird Shannon‘s Definition der Entropie [123] herangezogen und als Maß berechnet. Bisherige Resultate, die nicht Teil der Dissertation sind, beschreiben die Internode Certainty auf Stammbäumen mit identischen Spezies [115, 116]. Wir generalisieren diese Resultate um die Berechnung auf Bäumen mit möglicherweise unterschiedlichen Spezies zu erlauben. Hierfür werden diverse mathematische Korrekturverfahren entwickelt und getestet. Publiziert wurden diese Ergebnisse in [83].

Im zweiten Teil der Dissertation beschäftigen wir uns mit wiederholenden Strukturen in der Topologie von (Stamm-) Bäumen und genetischen Sequenzen.

Auch dieser Teil ist in drei Kapitel gegliedert. Die ersten beiden Kapitel behandeln Baumstrukturen, während sich das letzte Kapitel genetischen Sequenzen widmet.

Die theoretischen Resultate zu wiederholenden Strukturen in Bäumen wurden bereits in [54, 51] veröffentlicht. In Kapitel 7 präsentieren wir den Algorithmus um alle identischen Teilbäume in einem gegebenen Baum identifizieren zu können. Dieser Algorithmus läuft in linearer Zeit und liefert bewiesenermaßen das korrekte Ergebnis. Der Algorithmus wurde konzipiert um Wiederholungen zu finden, egal ob Knoten beschriftet sind, oder nicht; oder die Reihenfolge der Knoten untereinander beliebig ist, oder nicht. Der

Algorithmus findet wiederholende Topologien sowohl auf gewurzelten als auch auf ungewurzelten Bäumen. Auch für eine Sammlung von gewurzelten Bäumen können auf ähnliche Weise alle sich wiederholenden Muster erkannt werden.

Letzteres machen wir uns für die Berechnung von Stammbäumen zu Nutzen.

Die Hauptlast bei der Suche nach dem besten Stammbaum (nach dem Maximum-Likelihood Kriterium) liegt bei der eigentlichen Berechnung der Likelihoodfunktion. Hier machen wir uns zu Nutzen, dass identische Bäume auch die gleiche Wahrscheinlichkeit zu dem Likelihood beitragen.

Da wir bei Maximum-Likelihood Analysen von unterschiedlichen evolutionären Zeiten (Kantenlängen) and verschiedenen Kanten ausgehen, sind Wiederholungen nur dann identisch, wenn sie an der gleichen Stelle im Baum auftreten. Wiederholungen sind also auf Ebene des Sequenzalignments zu finden, nicht in der eigentlichen Baumtopologie. Hierfür nehmen wir implizit einen, von der Topologie und Kantenlängen identischen, Baum pro Seite im Alignment an. Die Beschriftungen an den Blattknoten hängen somit von der entsprechenden Seite im Alignment ab.

In Kapitel 8 zeigen wir einen für genetische Stammbäume adaptierten bzw. optimierten Algorithmus der eben diese identischen Teilbäume identifiziert. Wir analysieren die Auswirkung auf die Laufzeit bei der Berechnung der Likelihoodfunktion, wenn identische Teilbäume nicht mehrfach berechnet werden müssen. Zusätzlich zur eigentlichen Laufzeitverbesserung braucht der vorgeschlagene Algorithmus weniger Speicherkapazitäten als herkömmliche Methoden. Dies wirkt sich besonders auf Analysen mit großen Datenmengen aus.

Auch für diesen praktischen Teil ist eine Publikation vorgesehen. Eine vorläufige Version kann in [84] gefunden werden.

Den letzten Beitrag dieser Dissertation liefert ein abschließendes Kapitel (Kapitel 9) zur Sequenz Alignierung. Hier untersuchen wir einen weitverbreiteten Algorithmus zur paarweisen Alignierung von je zwei Sequenzen [59].

Wir zeigen, dass die ursprüngliche Formulierung irreführend, oder sogar fehlerhaft ist.

Wir untersuchen, wie weit verbreitet dieser Fehler heutzutage ist, indem wir Sachbücher, Universitätsvorlesungen und Softwareprogramme untersuchen. Wir zeigen auf, dass der Fehler in renommierten Büchern vorkommt

und regelmäßig in Vorlesungen gelehrt wird (etwa 50% der untersuchten Vorlesungen mit einer Vollständigen Beschreibung des Algorithmuses enthalten fehlerhaftes Material). Auch die von uns analysierte Software gibt längst nicht immer die zu erwartenden Ergebnisse.

Dieses Kapitel, bzw. die angestrebte Publikation (Vorläufige Version in [53]), sollen Nutzer und Entwickler der Software auf dieses Verhalten aufmerksam machen.

Contents

Part 0:

Introduction

1 Motivation and Related Work

Evolution The evolutionary history of species has been of interest to humanity ever since Charles Darwin first formulated his theory of evolution and natural selection in "On the Origin of Species" in 1859 [29]. The underlying idea is that species adapt to the environment they live in, and thus evolve. New species may arise as a result, while other species face extinction. Consequently, all life on earth shares a common evolutionary history.

The relationships within this so-called tree of life can provide insight as to which species are closely related to each other. Usually, these relationships are represented as a tree structure. The terminal nodes denote the current, extant species, while the inner nodes, connecting the species are hypothetical common ancestors of the respective species. We call such a tree a phylogeny (see Figure 1.1).

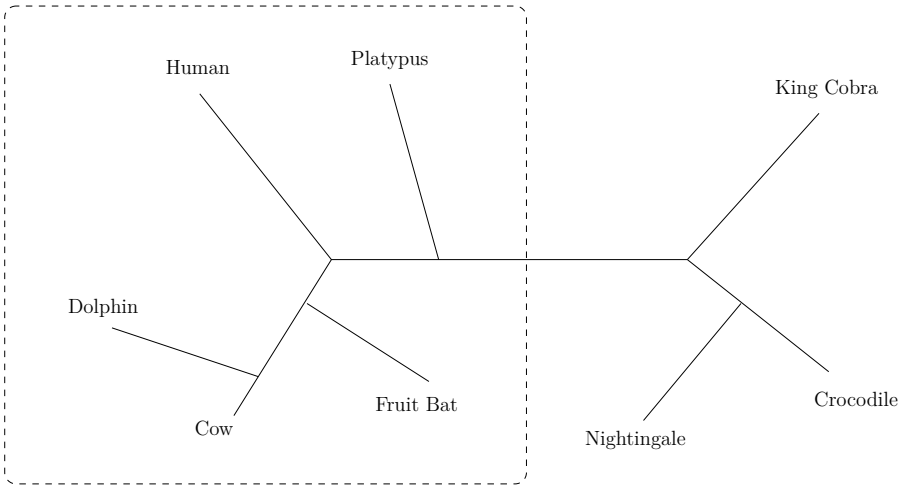


Figure 1.1: *Phylogeny of exemplary species. The framed species are mammals.*

Beyond a mere academic interest in capturing the evolutionary history of species, phylogenetic analyses have a wide range of applications (see [129], for an overview). For example, in the field of medicine, the development of new drugs can be guided by phylogenetics [85, 120]. In the field of law, phylogenetic analyses in criminal cases, can prove the innocence of the accused [32]. Epidemiologists can understand the spread of pandemics and the development of resistances [11, 88]. Conservation biologists can use these methods to show, among other questions, which species are endangered and

need protection, or face extinction [8, 139].

Original attempts at building these phylogenies compared morphological similarities and differences among different species. For example, bovines (such as cows) and primates (such as humans) have hair and give birth to live young, while birds have feathers and lay hard shelled eggs. Thus, it can be concluded that the former two, which are both mammals, are more closely related to each other than to birds. Looking at only morphological traits however can be misleading. For example, not all species of a common subgroup may share all characteristics. There exist mammals, such as the platypus, that lay leathery eggs. Thus, using this morphological trait for finding similarities is obviously not sufficient. On the other hand, a specific characteristic can be present in species from different taxonomic classes by having evolved independently. Bats, which are mammals, for example have wings. If this characteristic is used for a classification, bats might erroneously be thought of as being closely related to birds instead of other mammals.

Genetics Gregor Mendel laid the ground work for modern genetics with his discoveries about heredity among peas in 1866 [99]. In 1972, more than a century after Mendel‘s discoveries, the first DNA gene sequence was successfully decoded (sequenced) [102]. Rapid developments in this area have enabled the sequencing of full genomes for a plethora of species since then, including the human genome.

With these genetic sequences as a basis, a more targeted reconstruction of phylogenies is possible.

Among other events, random mutations at single positions within the DNA sequences of organisms can occur between one generation and the next. Other nucleotides may randomly be added (insertion) or deleted (deletion) from a sequence. Thus, species change, that is evolve, over time. See Figure 1.2 for exemplary mutation events between two sequences. Similar to the case of observing morphological similarities, we can deduce a phylogenetic relationship between species using this genetic code. Intuitively, the more similar two genetic sequences are, the fewer mutations will have occurred. From this, we deduce that less time has passed since both species had a common ancestor.

Molecular Phylogenetics Simple distance based methods can be used to reconstruct a phylogenetic tree by recursively grouping the least distant species together.

Alternatively, a score may be defined for each tree structure. Different tree structures can then be iteratively proposed and scored. The best scoring tree is then our best guess at the correct phylogeny.

A simple scoring function is the so-called parsimony score [20, 47]. Here, simply the minimum required number of mutations to explain a given tree is calculated. More sophisticated methods are the maximum likelihood (**ML**) method for tree inference [17, 43] and the Bayesian inference (**BI**) method [93]. Maximum likelihood and Bayesian analyses, are the focus of this thesis.

For ML, not only the numbers of mutations are counted, but a model of evolution is assumed in order to be able to calculate the probability of attaining the data, that is the distinct genetic sequences at the tips, from a theoretic common ancestor, given the tree structure. This probability is the so-called likelihood of the tree. The data is given by a so-called multiple sequence alignment (**MSA**) (see Figure 1.3). That is, each individual sequence of any one species (taxon) is arranged (aligned) in such a way, that the characters of all species at a given position (site) are assumed to share a common evolutionary history. The sequences in the alignment can either be composed of the four DNA characters (the nucleotides `A`, `C`, `G`, and `T`), or other symbols, such as the twenty amino acids which these DNA characters

Site:	1	2	3	4	5	6	7	8	9	10	11
Sequence 1:	T	T	A	T	G	T	A	G	C	C	-
			|		|		|	|		|	|
			|		A		G	|		|	|
			|		|		|	|		|	|
Sequence 2:	T	T	G	T	T	T	A	-	C	G	T

Figure 1.2: *Exemplary evolution between Sequence 1 and Sequence 2. Sites 3 and 10 show a simple mutation event. Site 5 shows two mutation events of which only one is observed in the presented sequences. Site 7 demonstrates two mutation events, while no difference is apparent in the resulting sequences at this site. Site 8 shows a deletion from Sequence 1 (or an insertion in Sequence 2). Similarly site 11 shows an insertion in Sequence 1 (or a deletion from Sequence 2). Note, that we can not generally distinguish whether an insertion or deletion event occurred since we do not know the original ancestral sequence.*

encode. The model of evolution determines how likely mutations from one DNA nucleotide (or amino acid) to another are within a certain amount of time. A notion of evolutionary time is given by the branch lengths in the tree (see Figure 1.4). Intuitively, the more time passes, the more likely a nucleotide changes into another.

The BI method not only calculates the likelihood, but also estimates the posterior probability [12, 92] of phylogenetic trees and chosen model parameters. The posterior probability is computed using the likelihood of the tree, as well as some prior probabilities for the tree structure and evolutionary model parameters. To actually calculate the posterior probability, the prior probability for the data, in this case the DNA sequences, must be known as well. Since this value is usually hard to obtain, Markov Chain

```
Sequence 1:  T  T  A  T  G  T  A  G  C  C
Sequence 2:  T  A  T  T  T  A  C  C  T
Sequence 3:  T  T  G  T  T  T  A  C  G  T

Sequence 1:  T  T [A] T  G  T  A  G  C  C  -
Sequence 2:  -  T [A] T  T  T  A  -  C  C  T
Sequence 3:  T  T [G] T  T  T  A  -  C  G  T
```

Figure 1.3: *Shown are three raw DNA sequences, and a MSA of these three sequences. The framed nucleotides represent one site of the alignment.*

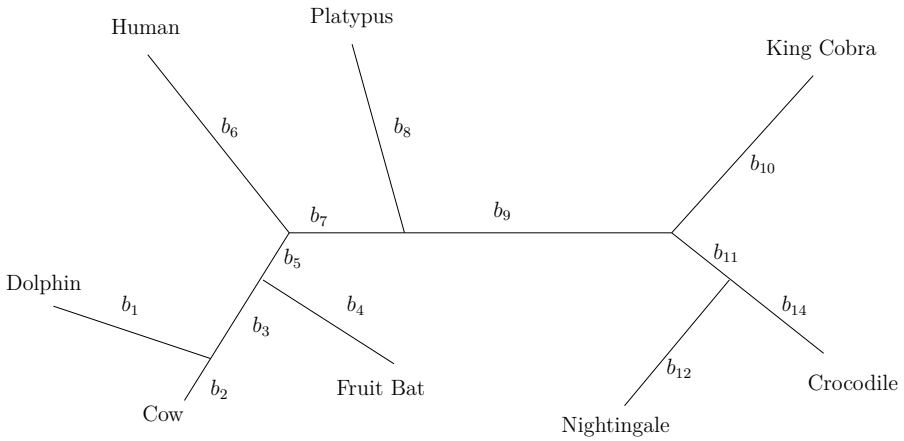


Figure 1.4: *Tree with branch lengths b_1 through b_{14}.*

Taxa:	3	4	5	6	7	8	9	10	n
Unrooted:	1	3	15	105	945	10395	135135	2027025	$\frac{(2n-5)!}{2^{n-3}(n-3)!}$
Rooted:	3	15	105	945	10395	135135	2027025	34459425	$\frac{(2n-3)!}{2^{n-2}(n-2)!}$

Table 1.1: *Numbers of unrooted and rooted bifurcating phylogenetic trees for given numbers of taxa. Note, that the number of unrooted trees for n taxa is equivalent to the number of rooted trees for $(n-1)$ taxa.*

Monte Carlo methods [66, 100] are typically employed instead, to estimate the posterior probability.

Note that, the number of possible phylogenies grows super-exponentially with the number of species [43] (see Table 1.1 for actual values). Furthermore, finding the optimal phylogenetic tree, using, for example, the parsimony or ML criterion is known to be NP-hard (see [23, 55, 112]). Thus, unless $P = NP$ one must rely on heuristics to obtain phylogenies in a reasonable time frame. Due to the huge number of possible trees, it is easy to imagine that exact BI is hard as well. Given an infinite amount of time, the Markov Chain Monte Carlo method actually yields exact results. However, due to limited computer resources, this is obviously not feasible.

Several software tools exist that implement such heuristics. For example, PhyML [63] and RAxML [130] are tools for ML tree inference, whereas ExaBayes [2] and MrBayes [114] are exemplary BI tools.

The cost of computing the likelihood function is asymptotically linear to the size of the data (length of the sequence alignment times number of species). Modern sequencing technologies provide ever growing amounts of data and continuously more and more genomes are sequenced. The human genome alone is already roughly three billion nucleotides long. For both methods, ML and BI, repeatedly calculating the likelihood for different tree topologies is thus the most time consuming task (numbers between 85% and 98% of the total runtime have been reported [6]). Efficient methods for calculating this likelihood function are thus needed.

2 Overview and Contribution

In this section we briefly give an overview over the structure of the thesis. The following chapter (Chapter 3) will provide some common notations and definitions which we will use throughout this work. Each individual chapter will provide additional notations and definitions as needed.

All results in this thesis are either already published works [54, 51, 52, 81, 82, 83], currently under review (one of which is available as a pre-print at `http://biorxiv.org/content/early/2016/01/04/035873`, [84]), or in the process of submission (again, a pre-print can be found at `http://www.biorxiv.org/content/biorxiv/early/2015/11/12/031500`, [53]). The following chapters, including parts of the introduction are based on these publications.
Other publications by me (as first or co-author) that were written during my time as a PhD student are [2, 50, 67, 151]. The works were published in internationally recognized journals on mathematics and computational theory, such as "*Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*" and "*Theoretical Computer Science*", as well as high impact journals from the field of biology and bioinformatics, such as "*Molecular Biology and Evolution*" and "*Bioinformatics*". Other papers appeared in the proceedings of internationally renowned conferences, where we presented the results.

The main body of this work is split into two distinct parts. The first part solves problems associated with so-called partitioned alignments. The second part deals with repeating structures within tree topologies, that is identical subtrees within a tree. In this part we also analyze (and point out mistakes in) a commonly used algorithm for aligning DNA sequences.

The first part is divided into three chapters.

In Chapter 4 we show that the optimal choice of an evolutionary model that maximizes the likelihood, is not simple. In fact, it is NP-hard if branch lengths are assumed to be the same for all partitions in the alignment, at least three models are present to choose from, and the data at hand has at least nine character states. Firstly, this serves as a justification for using heuristics to assign models to partitions. Secondly, in the context of this thesis, this chapter serves as an exemplary, rigorous, prove of NP-hardness. NP-hard and NP-complete problems can be found throughout the thesis and in the field of phylogenetics in general. This chapter is based on [82].

Chapter 5 analyzes the problem of optimally assigning sites of an alignment to parallel processors to ensure a near-optimal load balance for likelihood calculations. Again, this problem is NP-hard. However, we provide a polynomial time algorithm with a close to optimal approximation. In fact, if $P \neq NP$, this algorithm guarantees an optimal worst case performance, among polynomial time algorithms. This new assignment scheme was implemented in standard software tools. The new assignment significantly improves the parallel efficiency of likelihood-based inferences compared to previous implementations. We observed a performance improvement of up to 5.9 times faster inferences. This Chapter is based on [81].

The last chapter on partitioned alignments, Chapter 6, shows how to calculate the so-called internode certainty from partial gene trees. The internode certainty is a measure of confidence that not only reflects the absolute support of a clade (subtree) in a phylogenetic tree, but also takes the degree of conflict into account. The input is a reference tree, for example the ML tree, and a collection of trees with potentially fewer taxa than the reference tree. The contents of this chapter has been published in [83].

The second part of the thesis deals with repeating structures within trees, as well as the so-called pairwise global alignment between two sequences. Again, this part is organized into three chapters.

Chapter 7 demonstrates how to calculate all repeating (sub-)tree structures in an arbitrary tree. The algorithm works with rooted as well as unrooted trees, and labeled as well as unlabeled trees (that is, whether some character or symbols are given at the nodes or not). Further, subtree repeats on ordered as well as unordered trees (that is, whether the order of nodes in a (sub-)tree is important to its identity, or not) can be computed. The presented algorithm runs in linear time and requires linear space, making it time and space optimal. The chapter is based on [51].

Chapter 8 shows how to apply the results of Chapter 7 to phylogenetic trees. They can be used to speed up the likelihood calculations. An optimized algorithm for the application to phylogenetic trees is given. With this algorithm we observed a speed-up of up to more than 5 times faster execution times if repeats for full tree traversals are calculated on the fly. In fact, all tested data sets yield a speed up factor of more than two for this case.

If repeats can be precomputed and do not need to be updated (in partic-

ular, when the tree topology remains fixed) a speed up of almost ten times faster run times is observed in the best case. Run time improvements due to the new algorithm are analyzed in detail and memory savings are reported. A pre-print is available at `http://biorxiv.org/content/early/2016/01/04/035873` [84].

Finally, in Chapter 9 we take a critical look at global pairwise sequence alignment methods. We show that the original publication of the quadratic time algorithm for this problem contains several irregularities. Mistakes resulting from these irregularities can easily be overlooked, as evident by the numerous implementations that yield erroneous results. We show how to avoid these errors, and analyze a number of books, software tools and university lecture slides to asses the severity and prevalence of these errors. A pre-print with the contents of this chapter is available at `http://www.biorxiv.org/content/biorxiv/early/2015/11/12/031500` [53].

3 Common Notations, Formulas, and Definitions

We will now give a brief overview over the notations and definitions we will use throughout this thesis.

3.1 Alignment

First, we need an understanding of the data that is used for computing the likelihood function.

Multiple Sequence Alignment. Modern sequencing methods provide us with raw DNA sequences for different species or individuals in a population. However, in order to correctly compute the likelihood that some tree topology yields exactly these sequences, all nucleotides at a given position in the sequences must share the same evolutionary history. Due to insertions and deletions of nucleotides in the genetic code, which accumulate during the process of evolution, this might not be true for nucleotides at the same position in the raw sequences. For this reason, a so-called **multiple sequence alignment** (**MSA**) must first be established (see Figure 1.3 on page 6). This multiple sequence alignment is given by an $m \times n$ matrix A, where m is the number of species and n at least as large as the longest sequence of the individual species. The entries within A are the original nucleotides of the DNA sequences, possibly containing so-called **indels** (insertions or deletions, also called **gaps**). We denote such an indel by the special character "-". A **site** a_i of the alignment A, with $i = 1, \ldots, n$, is the i-th column of this matrix. Thus, each site contains at most one nucleotide character form any of the DNA sequences together with possible indels. All nucleotides at a site are then assumed to have evolved from a common ancestral nucleotide (in other words, the nucleotides at a given site are *homologous*). The actual calculation of such a multiple sequence alignment from raw DNA sequences is beyond the scope of this thesis. Several publications [60, 71] and implementations [37, 138] exist that cover this topic.

In this thesis, we are however interested in so-called *global pairwise sequence alignment*. Here, only two sequences are aligned with each other. Chapter 9 gives an overview of related work for this topic. There, we show that the original description of one of the most widely used algorithms for aligning pairwise sequences, actually contains several irregularities.

For the pairwise sequence alignment, the goal is to optimize (minimize or maximize) the score of an alignment between two sequences. For this,

a numerical value is defined for any pair of homologous characters in the sequences. For example, if both sequences have the same nucleotide at a position, a *matching* score is counted. If the two nucleotides disagree, a *miss match* is counted. If one sequence contains a gap, a *gap cost* is applied. See Figure 3.1 for an intuitive example.

Sequence 1:	-	A	A	A	-	T	A	G	C	C	-
Sequence 2:	T	A	A	A	T	T	A	-	C	C	T

Figure 3.1: *Let the* match score *be* 5*, the* miss score *be* -8*, and the* gap penalty *be* -5*. Then, the above alignment has a score of* 15*.*

Several polynomial time algorithms exist for computing the optimal pairwise sequence alignment (see for example [106]).

Instead of demanding a constant penalty for any one gap encountered in the alignment of two sequences, it is biologically reasonable to assign so-called *affine gap costs* instead. Here, a typically high penalty is invoked whenever a new gap is started (gap opening penalty), and only a small penalty is applied to each individual gap (gap extension penalty). See Figure 3.2 for an example scoring. This means, that gaps are more likely to be

Sequence 1:	A	A	A	-	-	T	A	G	C	C
Sequence 2:	T	A	A	A	T	T	A	C	C	T

Figure 3.2: *Let the* match score *be* 5*, the* miss score *be* -8*, the* gap open penalty *be* -12*, and the* gap extension *penalty be* -1*. Then, the above alignment has a score of* -13*. The alignment of the same sequences as seen in Figure 3.1 would obtain a worse score of* -17 *under this scoring scheme.*

placed consecutively in an optimal pairwise sequence alignment, than to be scattered throughout the alignment. The biological motivation behind this is, that gaps are unlikely to occur, but if a gap is encountered, more than one position of the alignment may be affected.

Gotoh's algorithm [59], which we analyze in Section 9, can find the optimal pairwise global sequence alignment under these affine gap costs in quadratic time.

Partitioned Alignment. It is often reasonable to assume that different sites in the alignment evolve according to different models of evolution. This is reasonable, for example, if sites come from different genes, or from regions

with different mutation rates, within the genome. For example, functionally important parts, such as protein coding regions, of the genome may be less likely to accumulate mutations than non-coding regions. The reason for this is, that the fitness (chance of producing offspring) drastically decreases if the mutation is lethal to the organism. Thus, different model parameters may be chosen for the different regions. For this reason we define the notion of a **partitioned alignment**.

Definition 1 (Partitioned Alignment)**.** *Let A be an alignment. Further, let p be the number of partitions. We define the p partitions, P_1, P_2,$\cdots$,P_p such that each site $a \in A$ must satisfy $a \in P_i$ for exactly one $i \in \{1, 2, ..., p\}$.*

	Gene 1:				Gene 2:					
Sequence 1:	T	T	A	T	G	T	A	G	C	C
Sequence 2:	T	A	T	T	T	A	C	C	T	
Sequence 3:	T	T	G	T	T	T	A	C	G	T

	P_1:					P_2:					
Sequence 1:	T	T	A	T	-	G	T	A	G	C	C	-
Sequence 2:	-	T	A	T	T	-	T	A	-	C	C	T
Sequence 3:	T	T	G	T	-	T	T	A	-	C	G	T

Figure 3.3: *Shown are two genes for three raw DNA sequences, and a partitioned MSA of these three sequences. Note that this alignment differs from the unpartitioned alignment of the same sequences presented in Figure 1.3 (page 6).*

We then may choose to link, or unlink, some parameters between these partitions. For example, one distinct model of evolution might be assumed per partition, while the tree structure may be required to be the same for all partitions. If the tree structure is linked across partitions we often talk about the **species tree**. If, on the other hand, a specific tree topology is analyzed for each partition, we call the resulting topologies **gene trees**. The choice of linking or unlinking parameters across partitions can either be biologically motivated, result oriented, or even a question of resource management. Choosing different mutation rates for different partitions is a biologically motivated example for unlinking the model parameters across partitions. The question whether we are interested in gene trees or species trees is a result oriented decision. Lastly, the more parameters are linked across partitions, the less parameters have to be estimated and optimized

overall. This can help to avoid over-fitting the data, as well as allow us to save computational resources.

Programs such as PartitionFinder [91] can help to decide which partitions to link together under the same evolutionary model.

3.2 Model of Evolution and Transition Probability

In order to calculate the likelihood for each of the sites of the alignment, a model of evolution is needed. That is, the probability of one nucleotide mutating into another, given a certain amount of time, must be known.

Instantaneous Rate Matrix Usually, changes from one nucleotide to another are assumed to follow a continuous-time Markov chain (also called continuous-time Markov process) with exponential waiting times. The states of the corresponding Markov chain are the actual nucleotides (or amino acids). See Figure 3.4 for an illustration. To accurately model this, a **frequency** $\Pi \in \mathbb{R}^{|\Sigma|}$, where $\Sigma = \{state_1, states_2, \ldots\}$ is the set of states and $\Pi =: (\pi_{state_1}, \ldots, \pi_{state_{|\Sigma|}})$ ($\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$, and $\Pi = (\pi_A, \pi_C, \pi_G, \pi_T)$, for DNA), and a symmetrical instantaneous transition **rate matrix** $R \in \mathbb{R}^{|\Sigma|\times|\Sigma|}$ must be given. Note, that we require $\sum_{i=1}^{|\Sigma|} \pi_{state_i} \stackrel{!}{=} 1$.

The frequencies and the rate matrix may be estimated from the data at hand, or be picked from a set of predefined models. Such predefined models are usually estimated using large amounts of data. These large amounts of data then ensure that over-fitting of the parameters is unlikely. For example, simply counting the numbers of differences in closely related sequences can give estimates for the rate matrix. Several such models have been published for protein data (see for example [77, 86, 145]).

Since the rate matrix is assumed to be time reversible and normalized, and the frequencies sum up to 1.0, the number of free parameters for a DNA model is $5 + 3 = 8$. For protein data (20 states) we similarly get $189 + 19 = 208$ free parameters. Thus, estimating the rate matrix from the data is more common for analyses with DNA sequences, while predefined models are typically applied to amino acid data sets. This is often done to avoid over parameterizing the analysis.

Picking the optimal model from a set of predefined models is the focus of Chapter 4.

Transition Probability Matrix From standard text books on stochastic processes [14, p. 268] we get the following definition for calculating the transition probabilities of nucleotides.

$$P(t) = e^{Q \cdot t}, \tag{1}$$

where $Q = R \cdot D(\Pi)$, with $D(\Pi)_{i,i} = \pi_{state_i}$ and zero else, is the so-called **Q-Matrix**. The transition probability from some state i to another state j in time t is then denoted by $P(i \to j|t) = P(t)_{i,j}$.

In order to efficiently evaluate this matrix exponential, some observations are required. First, R is a symmetric matrix. However, $Q = R \cdot D(\Pi)$ is not generally symmetric. This is unfortunate, as this matrix exponential can easily be computed for symmetrical matrices by using the so-called spectral decomposition (also called eigen decomposition). To still apply the spectral

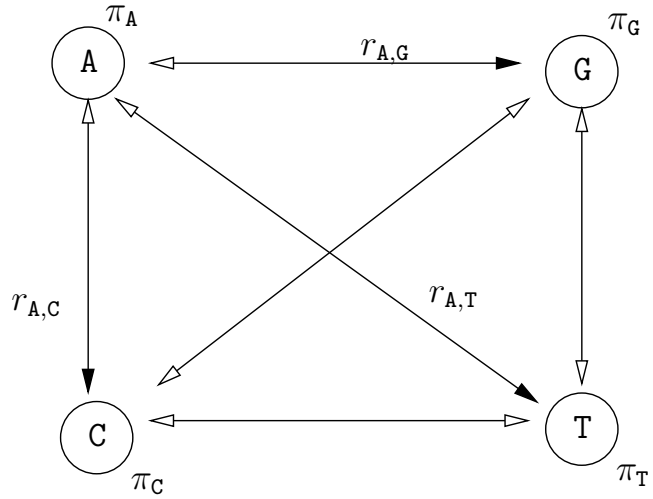


Figure 3.4: *Markov chain for mutations between DNA nucleotides. The states are the four nucleotides* A, C, G *and* T. *Frequencies* $\pi_{\mathtt{A}}$, $\pi_{\mathtt{C}}$, $\pi_{\mathtt{G}}$, *and* $\pi_{\mathtt{T}}$ *denote the probability of starting at any of these nucleotides, as well as the probability of observing any of these nucleotides after an infinite amount of time passes, regardless of the initial starting state (nucleotide). If we are at state* A, *the Markov chain waits for some time* t, *where* t *is exponentially distributed with* $\lambda = (r_{\mathtt{A,C}} + r_{\mathtt{A,G}} + r_{\mathtt{A,T}})$. *A jump is then performed to state* C *with probability* $\frac{r_{\mathtt{A,C}}}{r_{\mathtt{A,C}}+r_{\mathtt{A,G}}+r_{\mathtt{A,T}}}$, *to* G *with probability* $\frac{r_{\mathtt{A,G}}}{r_{\mathtt{A,C}}+r_{\mathtt{A,G}}+r_{\mathtt{A,T}}}$ *or to* T *with probability* $\frac{r_{\mathtt{A,T}}}{r_{\mathtt{A,C}}+r_{\mathtt{A,G}}+r_{\mathtt{A,T}}}$. *Then, a new waiting time is draw for the new state* C, G *or* T.

decomposition, some further operations are required. Observe, that $Q' := \hat{\Pi} Q \hat{\Pi}^{-1}$, with $\hat{\Pi}_{i,j} = \sqrt{\pi_i}$ if $i = j$ and $\hat{\Pi}_{i,j} = 0$ else, is, in fact, a symmetrical matrix.

Applying the spectral decomposition to Q' we get $Q' = U' \Lambda U'^T$, where Λ is the diagonal matrix with the eigenvalues $\lambda_1, \ldots, \lambda_n$ of Q' as diagonal elements. The columns of the orthogonal matrix U' and thus the rows of U'^T are the corresponding eigenvectors of Q'. Since $U' \Lambda U'^T = Q' = \hat{\Pi} Q \hat{\Pi}^{-1}$, we can decompose Q as $Q = (\hat{\Pi}^{-1} U') \Lambda (U'^T \hat{\Pi}) =: U \Lambda U^{-1}$. Given this spectral decomposition we can easily compute the transition probability (Equation (1)).

$$P(t) = e^{U \Lambda \cdot t U^{-1}} \tag{2}$$

$$= \sum_{i=0}^{\infty} \frac{(U \Lambda t U^{-1})^i}{i!} \tag{3}$$

$$= U (\sum_{i=0}^{\infty} \frac{\Lambda^i t}{i!}) U^{-1} \tag{4}$$

$$= U e^{\Lambda t} U^{-1} \tag{5}$$

$$= U \cdot E \cdot U^{-1}, \tag{6}$$

with $E_{i,j} = e^{\lambda_i \cdot t}$ if $i = j$ and $E_{i,j} = 0$ else. Thus,

$$P(t)_{i,j} = \sum_{k=1,\ldots,s} e^{\lambda_k \cdot t} \cdot U_{i,k} \cdot U^{-1}_{k,j}. \tag{7}$$

If U, U^{-1} and Λ are known, this value can be computed easily for any two states i and j, and any time t. In practice, computing the spectral decomposition to obtain U, U^{-1} and Λ takes a non-trivial amount of computational time. Since the calculations to obtain a spectral decomposition have to be applied separately for each model of evolution, this computational time is large enough to be of concern for efficient parallel likelihood function implementations.

The production-level ML based phylogenetic inference software ExaML [131] for supercomputers originally implemented two sub optimal data distribution approaches: The first is the *cyclic data distribution* scheme that does not balance the number of unique partitions per processor, but just assigns single sites to processors in a cyclic fashion. The second approach is the *whole-partition data distribution* or *monolithic distribution* scheme. Here, the individual partitions are not considered divisible and are assigned monolithically to processors using the longest processing time heuristic for

	P_1: Model 1					P_2: Model 2				P_3: Model 3		
	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}
Sequence 1:	T	T	A	T	-	G	T	A	G	C	C	-
Sequence 2:	-	T	A	T	T	-	T	A	-	C	C	T
Sequence 3:	T	T	G	T	-	T	T	A	-	C	G	T

	Processor 1	Processor 2
cyclic:	$\{a_1 , a_3 , a_5 \mid a_7 , a_9 \mid a_{11} \}$	$\{a_2 , a_4 \mid a_6 , a_8 \mid a_{10} , a_{12} \}$
monolithic:	$\{a_1 , a_2 , a_3 , a_4 , a_5 \}$	$\{a_6 , a_7 , a_8 , a_9 \mid a_{10} , a_{11} , a_{12} \}$
balanced:	$\{a_1 , a_2 , a_3 , a_4 , a_5 \mid a_6 \}$	$\{ a_7 , a_8 , a_9 \mid a_{10} , a_{11} , a_{12} \}$

Figure 3.5: *MSA divided into three partitions. Models are assumed to be unlinked across partitions. That is, each partition has its own model of evolution. Under the cyclic data distribution scheme, each of the two processors is assigned* 6 *sites to compute. Each processor also has to compute transition probabilities for* 3 *different models. Under the monolithic data distribution scheme, processor* 1 *is assigned* 5 *sites and needs to compute the probabilities from one set of model parameters. Processor* 2 *must compute* 7 *sites with two distinct sets of model parameters. The third, balanced, distribution scheme minimizes the maximum number of sites (*6*) and models (*2*) per processor.*

the 'classic' multi-processor scheduling problem [152]. This ensures that the total and maximum number of initialization steps (substitution matrix calculations) is minimized, at the cost of not being balanced with respect to the sites per processor. See Figure 3.5 for an intuitive illustration of these distribution schemes.

It is easy to construct worst case examples for each of the two existing distribution schemes that show their sub-optimal behavior. For the cyclic distribution scheme, simply assume a MAS with n partitions and n sites each. If we use the cyclic distribution scheme to distribute the sites to n processors, each processor must evaluate n sites, as well as calculate n model parameters. Other distribution schemes require only the calculation of one model per processor while retaining the number of n sites per processor.

For a worst case example of the monolithic distribution scheme, assume $(n-1)$ partitions with one site each, and one partition with $((n-1)^2+n)$ sites. Monolithically distributing these partitions to n processors results in one processor doing almost all the work. Specifically, one processor computes $((n-1)^2+n)$ out of n^2 sites. However, each processor only computes one set of model parameters. By having each processor calculate sites from at most one more model, we can drastically reduce the number of sites per processor

to n.

Given these two examples, we see, that an efficient distribution of sites to processors is performance critical.

In Chapter 5 we analyze how to minimize the time used for this step in a parallel environment for partitioned alignments. The objective there is to minimize the number of spectral decompositions each processor has to compute, while the number of sites allocated to each processor must remain balanced. Computational run time is compared to that of the cyclic and monolithic data distribution schemes.

3.3 Trees

Next we will define the actual tree structure on which a likelihood can be computed. Again, our notation is analogous to the definitions provided in standard text books (see for example [87]).

Definition 2 (Tree)**.** *A* **tree** $T = (V, E)$ *is an acyclic connected graph.* V *is the node set and* E *the set of edges with* $E \subset V \times V$*.*

Two nodes i *and* j *are connected to one another, if there exists an edge* $e_1 = (i, j) \in E$*, or* $e_2 = (j, i) \in E$*.*

Definition 3 (degree)**.** *The degree* $\delta(i)$ *of a node* i *in an undirected tree* T *is the number of edges leading to* i*. That is,* $\delta(i) = |\{e \in E | i \in e\}|$

Definition 4 (Diameter)**.** *The* **diameter** *of an unrooted tree* T *is denoted by* $d(T)$ *and is defined as the number of edges of the longest path between any two* leafs *(nodes with degree 1) of* T*.*

Definition 5 (unrooted tree)**.** *A tree* T *is unrooted iff it is undirected.*

In case of an unrooted tree $T = (V, E)$ we may write $e = \{i, j\}$ for $e \in E$. The phylogeny presented in Figure 1.1 (see page 3) is such an unrooted tree.

Definition 6 (rooted tree)**.** *A directed tree* T *is rooted iff there exists a single node* $r \in V$ *such that each other node can be reached from* r *using only edges in* E*. The node* r *is then called root of* T*.*

See Figure 3.6 for an example.

Definition 7 (Child, Parent, and Sibling)**.** *For a rooted tree* $T = (V, E)$*, we call* $u \in V$ *a* **child** *of* v iff $(v, u) \in E$*. In this case, we call* v *the* **parent** *of* u *and define parent*$(u) := v$*.*

We call $u \in V$ *and* $u' \in V$ **siblings** *iff there exists a node* $v \in V$*, such that* (v, u) *and* $(v, u') \in E$*.*

See Figure 3.7 for an illustration.

Definition 8 (Subtree)**.** *The (rooted)* **subtree** *of T that contains node v as its root node, obtained by removing edge (v, u), is denoted by $\hat{T}(v, u)$. We consider only* full subtrees*, that is, subtrees which contain all nodes and edges that can be reached from v when only the edge (v, u) is removed from the tree. The special case $\hat{T}(v, v)$ denotes the tree containing all nodes that*

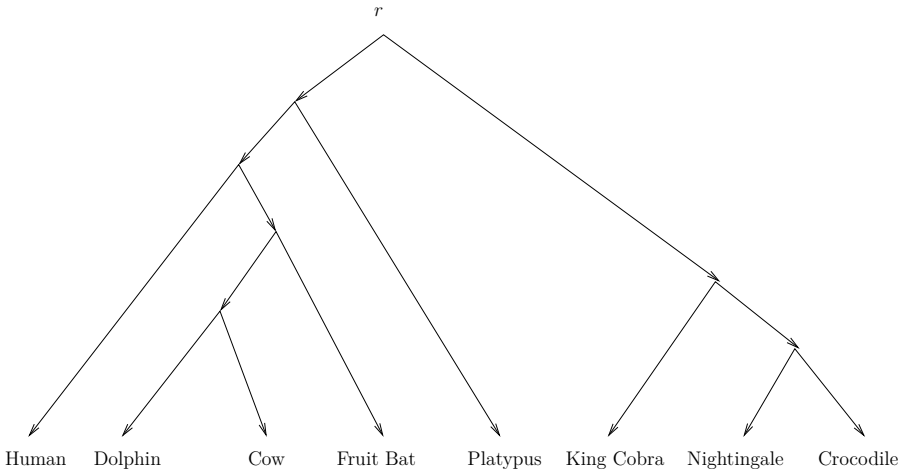


Figure 3.6: *An exemplary rooted labeled tree.*

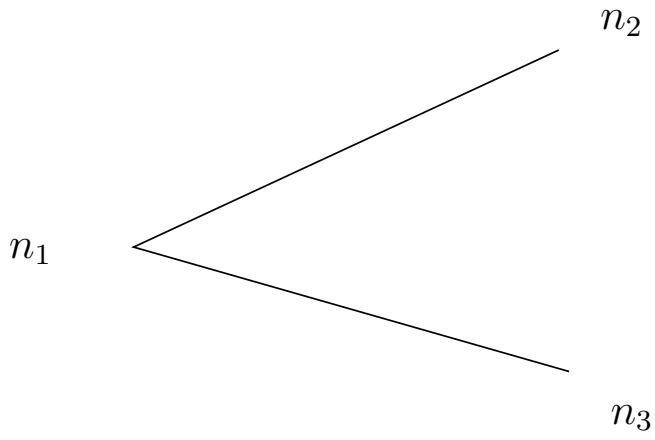


Figure 3.7: *Given is the relation between nodes n_1, n_2, and n_3 in some tree. Node n_1 is the parent of nodes n_2 and n_3. Thus, n_2 and n_3 are children of node n_1, and siblings of each other.*

is rooted in v.

For simplicity, we refer to $\hat{T}(v, parent(v))$ as $\hat{T}(v)$.

Definition 9 (Alphabet)**.** *An* **alphabet** Σ *is a finite, non-empty set whose elements are called* **symbols**.

A **string** *over an alphabet Σ is a finite, possibly empty, string of symbols of Σ.*

The **length of a string** *x is denoted by $|x|$, and the* **concatenation** *of two strings x and y by xy.*

Definition 10 (Labeled Tree)**.** *A tree T is called* **labeled** *if every node, or some nodes, of T are labeled by a symbol, or string, from some alphabet Σ (in our case DNA characters).*

Otherwise it is called **unlabeled**.

Different nodes may have the same label.

Chapter 7 explains how to find repeating structures within any of these types of trees (for example, unrooted unordered labeled trees).

Phylogenetic trees comprise additional assumptions. Here, phylogenetic trees are assumed to be fully binary and generally unrooted. (However, in order to compute the likelihood function (see Section 3.4), an artificial rooting must be assumed.)

Definition 11 (Phylogenetic Topology)**.** *Let $T = (V, E)$ be a tree, and let N be the set of species of a phylogenetic analysis. Then $V =: N \cup I$, where I is the set of* **inner nodes**. *An unrooted phylogenetic tree topology T fulfills the following properties*

$$\nexists e \in E \text{ s.t. } e \in N \times N \tag{8}$$

$$\delta(i) = 1 \ \forall i \in N \tag{9}$$

$$\delta(i) = 3 \ \forall i \in I. \tag{10}$$

Additionally, the nodes $i \in N$ are labeled with the respective genetic sequences, and/or the corresponding species names, while the nodes I typically remain unlabeled. The nodes in N are either called **tip nodes** or **taxa** (or sometimes simply species).

For phylogenetic analyses, we typically assume the species labels, as well as the sequence labels to be unique. In general, this is not the case for all labeled trees. Labels may be repeated across different nodes.

In addition to the actual topology, we also need a measure of evolutionary time between nodes. For this reason, we define the **branch lengths**. See Figure 3.8.

Definition 12 (branch lengths)**.** *Let $T = (V, E)$ be a phylogenetic tree topology. Then, $b(e) \in \mathbb{R}_{\geq 0}$, where $e \in E$, denotes the branch length for edge e.*

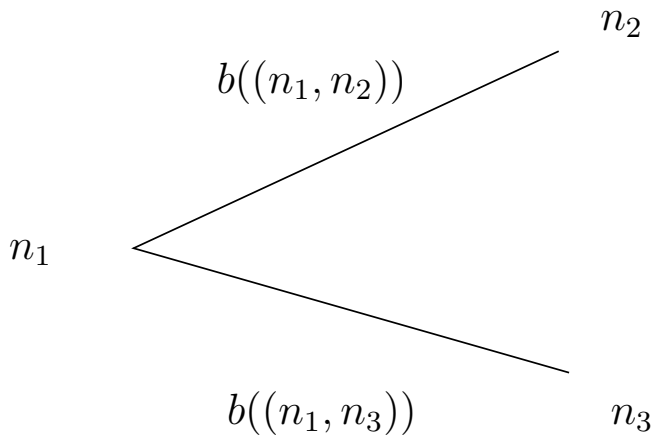


Figure 3.8: *Branch length notation for a triplet of nodes, n_1, n_2, and n_3. The branch between nodes n_1 and n_2 is denoted by $e_1 := (n_1, n_2)$. Analogously, $e_2 := (n_1, n_2)$ is the branch between nodes n_1 and n_2.*

If branch lengths are used, we may also write $T = (V, E, b)$ for a **phylogeny**, or **phylogenetic tree**.

Note, that this definition, and the interpretation of time is not exact. At least two factors affect the probability of changing from one nucleotide to another. These factors are the time, and an overall rate of change. The amount of time that passes intuitively affects how likely a mutation from one nucleotide to another is to be observed. The more time passes, the more mutations accumulate. On the other hand, mutations are not equally likely to occur in all species, nor all positions of an alignment (see the reasoning for partitioned alignments on page 12). For example, viruses accumulate mutations more quickly than other species, such as humans or insects. Thus, in the same amount of time, more mutations are expected within the viral genome. The rate of change takes exactly this into account. However, it is non-trivial to distinguish between the time and rate of change. Thus, we simply set the branch lengths b to be the product of the two. That is,

$b_i(e) = b(e) \cdot r_i$, where $e \in E$, and i denotes a partition, or corresponding model of evolution.

Finding the optimal branch length configuration for a fixed tree topology and a given evolutionary model represents a non-trivial numerical problem [44] and the solution may not be unique [22]. On real data, good approximations of the optimal branch length assignment can be computed efficiently, for example using the Newton-Raphson procedure [44, 57].

3.4 Likelihood Function

Now, we have everything we need to compute the likelihood function.

The likelihood of the data, given a tree with fixed branch lengths, and known substitution probabilities can be computed in polynomial time (with respect to the number of sequences and sites in the alignment) using the Felsenstein pruning algorithm [40]. Using this algorithm, the likelihood of a phylogenetic tree T is calculated by computing the conditional likelihoods at each inner node of T.

The conditional likelihoods are computed independently for each *site* (column in the MSA). They are computed via a post-order traversal of T, starting from a virtual root. Note that, as long as the statistical model of evolution is time-reversible (that is, evolution occurred in the same way if followed forward or backward in time) the likelihood score is invariant with respect to the location of the virtual root in T [17].

Also note that, the likelihood of a tree, given an alignment, is multiplicative across the sites of the alignment. That is, to obtain the overall phylogenetic likelihood, the individual likelihood values for each site are multiplied together. Biologically this means that we assume that the nucleotides at different positions in the alignment evolve independently of one-another. The advantage of making this assumption is that the likelihood values for each site can be computed independently. This is especially important for parallelizing likelihood calculations. Each processor can compute any number of sites independently of the likelihood values obtained by other sites.

For computational stability, the logarithm of the likelihood function is calculated instead of the actual likelihood. One advantage is, that the likelihood values across sites are now additive instead of multiplicative.

For a node k with child nodes i and j we compute the **conditional likelihoods** at site s for each possible state (for example, `A`, `C`, `G`, `T` for DNA

data) as follows (see also [41]):

$$L_{X_k}^{(k)}(s) = \Big(\sum_{X_i=A}^{T} P(X_k \to X_i | b((k,i)) L_{X_i}^{(i)}(s)\Big)\Big(\sum_{X_j=A}^{T} P(X_k \to X_j | b((k,j)) L_{X_j}^{(j)}(s)\Big), \tag{11}$$

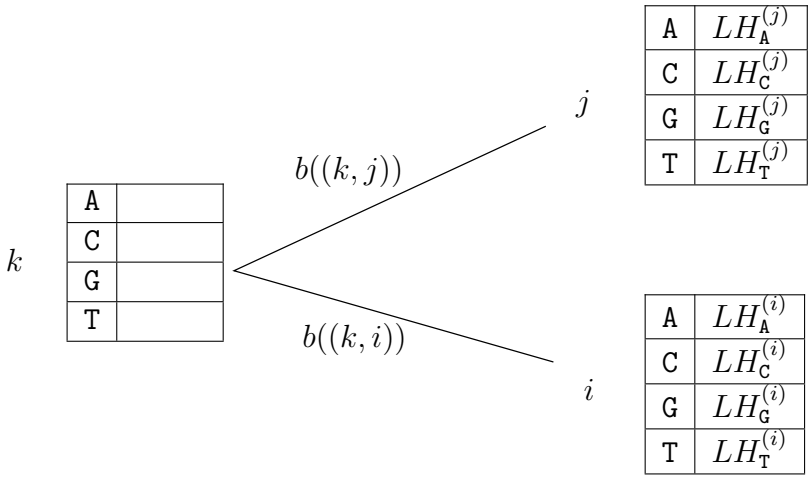


Figure 3.9: *Conditional likelihood vectors at nodes i and j. The conditional likelihood vectors at node k can be computed using the given values (and a model of evolution) by Equation (11)*

where $L_{X_k}^{(k)}(s)$ is the conditional likelihood of observing the DNA nucleotide state X_k at site s for the subtree rooted at k.
See Figure 3.9 for an illustration of the values.

The function $P(X_k \to X_i | b((k,i))$ gives the probability that nucleotide X_k evolved into nucleotide X_i after time $b((k,i))$ (the branch length between k to i). If i is a tip (leaf) and site s consists of a nucleotide, say `A`, then $L_{\mathtt{A}}^{(i)}(s) := 1.0$ and $L_{\mathtt{C}}^{(i)}(s) := L_{\mathtt{G}}^{(i)}(s) := L_{\mathtt{T}}^{(i)}(s) := 0.0$. Analogously, all of this holds for j, as well.

The so-called **conditional likelihood vector** (**CLV**) for a particular site s at a given node i is denoted by the ordered set

$$\mathcal{L}^i(s) = \bigcup_{x=\mathtt{A}}^{\mathtt{T}} L_x^{(i)}(s). \tag{12}$$

Finally, we compute the overall likelihood for a single site s at the virtual root r of the tree by multiplying the frequencies π_x of observing a nucleotide

state x with the likelihood of that state at r:

$$L^{(r)}(s) = \sum_{x=\mathtt{A}}^{\mathtt{T}} \pi_x L_x^{(r)}(s). \tag{13}$$

Once the likelihood for each site has been computed, the overall likelihood of the tree is the product over these per-site likelihoods. That is, the log-likelihood of the tree T, $L(T)$, is given by:

$$L(T) = \sum_{i=1}^{n} L^{(r)}(s_i), \tag{14}$$

where n is the number of sites in the alignment and s_i is the i-th site of the alignment.

If two sites have the same nucleotides at all tip nodes in the subtree rooted at node k, Equation (11) must, by construction, yield the same conditional probabilities for all states X_k for both sites. Avoiding these redundant operations during the likelihood computation is the focus of Chapter 8.

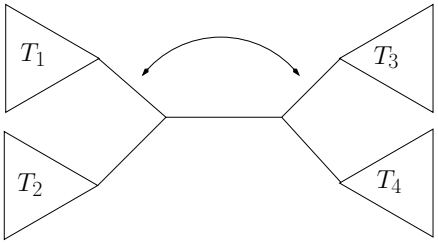
Actual tree search strategies are beyond the scope of this thesis. Thus, we will not go into detail here. However, Figure 3.10 gives an overview over two common tree search mechanisms. Interested readers may also find the following standard text book on this topic helpful [43]. Note that both ML, and BI, heavily rely on the repeated calculation of the likelihood values. Phylogenetic software tools may spend as much as 85% to 98% of the total runtime in evaluating the likelihood function [6]. Thus, efficient methods for calculating this function are paramount to speeding up phylogenetic analyses.

3.5 Bipartition Support

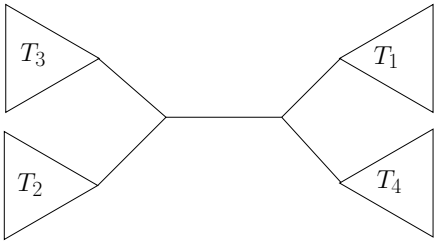
In order to evaluate the results obtained by running a phylogenetic analysis, the notion of bipartitions on trees is helpful.

Definition 13 (Bipartition)**.** *Given a taxon set S, a* **bipartition** *B of S is defined as a tuple of taxon subsets (X, Y) with X, $Y \subset S$ and $X \cup Y = S$, $X \cap Y = \emptyset$. We write, $B = X|Y = Y|X$.*

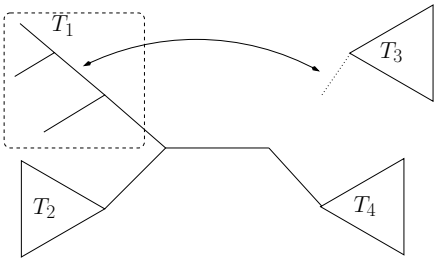
In phylogenetic trees, a bipartition is obtained by removing a single edge from the tree. Let $b = (n_1, n_2)$ be an edge connecting nodes n_1 and n_2 in some unrooted phylogenetic tree T. The bipartition that is obtained by



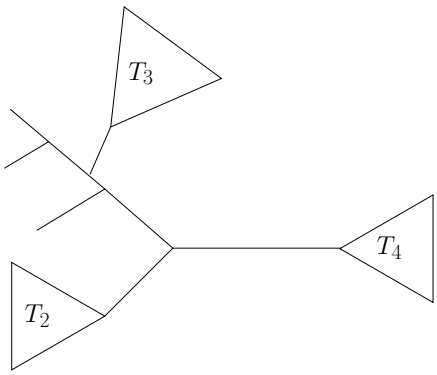
(a) *Tree with four subtrees T_1, T_2, T_3, and T_4. A Nearest neighbor interchange move is performed by switching positions of any two of these subtrees.*



(b) *Resulting topology, after the NNI move is applied.*



(c) *Tree with four subtrees T_1, T_2, T_3, and T_4. A SPR move is performed by disconnecting any of these subtrees and re-inserting it within any other.*



(d) *Tree after SPR move is performed.*

Figure 3.10: *On top (a and b), the Nearest Neighbor Interchange (NNI) move is shown. On the bottom (c and d), the Subtree Pruning and Regrafting (SPR) move is illustrated. Iterative application of these (and other topological moves) are used to search and traverse the tree space.*

removing b is denoted by $B(b)$, which we define as: $B(b) = X(n_1)|X(n_2)$, where $X(n_1)$ and $X(n_2)$ are all taxa that are still connected to nodes n_1 and n_2 respectively, if branch b is removed.

Definition 14 (Trivial bipartition)**.** *We call a bipartition $B = X|Y$* **trivial** *iff $|X| = 1$ or $|Y| = 1$.*

Trivial bipartitions are uninformative, since having only a single taxon in either X or Y means that this taxon is connected to the rest of the tree. This is trivially given for any tree containing this taxon.

Bipartitions with $|X| \geq 2$ and $|Y| \geq 2$ are called **non-trivial**. In contrast to trivial bipartitions, non-trivial bipartitions contain information about the structure of the underlying topology.
Henceforth, the term bipartition will always refer to a non-trivial bipartition.

Two bipartitions that do not have the same taxon set may still agree on the topology for the taxa included in both bipartitions. For the case that the taxon set of one bipartition is a subset of the taxon set of another bipartition, we define sub- and super-bipartition relations between them.

Definition 15 (Sub-bipartition, super-bipartition)**.** *We denote $B_1 = X_1|Y_1$ as a* **sub-bipartition** *of $B_2 = X_2|Y_2$ if $X_1 \subseteq X_2$ and $Y_1 \subseteq Y_2$, or $X_1 \subseteq Y_2$ and $Y_1 \subseteq X_2$.*
The bipartition B_2 is then said to be a **super-bipartition** *of B_1.*

A common method in phylogenetics is to measure the bipartition support for each bipartition in a reference tree. The reference tree may, for example be a ML tree for some data set. A list of alternative trees is then computed, for example by simply re-running the analysis with different random starting points, or altering the alignment in some way [95] (for example by bootstrapping [38, 42] or jackknifing [124]). Usually, not all of these trees are equivalent. Many trees, even if they are ultimately different, will share common bipartitions. The **support** for any bipartition of the reference tree can then simply be computed by counting the number of alternative trees, that also contain exactly this bipartition (as illustrated by Figure 3.11).

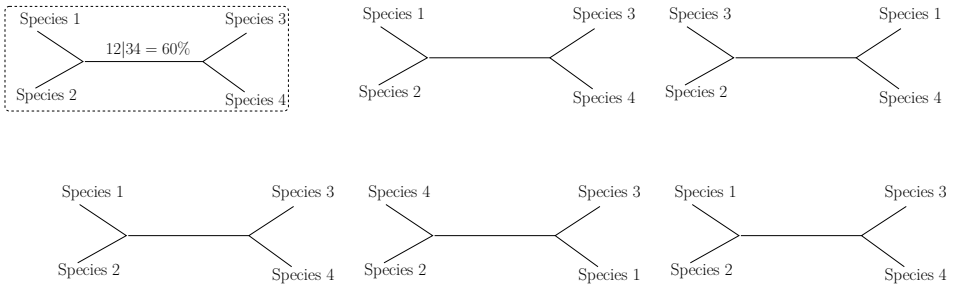


Figure 3.11: *The only (non-trivial) bipartition in the reference tree (*12|34*) is supported by* 60 *% of all other trees.*

A more invested method for measuring the support for bipartitions in a reference tree is discussed in Chapter 6. There, Shannon‘s measure of entropy is calculated to asses the support, and conflict, of bipartitions.

Another useful application of bipartitions in phylogenetics, is the Robinson Foulds (RF) distance measure of trees [111]. Here, the distance between two trees T_1 and T_2 is computed by counting how many bipartitions each tree contains, that the other does not.

Definition 16 (RF-Distance)**.** *The RF distance between two trees* $T_1 = (V_1, E_1)$ *and* $T_2 = (V_2, E_2)$*,* $RF(T_1, T_2)$ *is:*

$$\begin{aligned} RF(T_1, T_2) = &|\{B(b_1) | b_1 \in E_1, B(b_1) \text{ non-trivial}, B(b_1) \neq B(b_2) \forall b_2 \in E_2\}| \\ &+ |\{B(b_2) | b_2 \in E_2, B(b_2) \text{ non-trivial}, B(b_2) \neq B(b_1) \forall b_1 \in E_1\}|. \end{aligned} \tag{15}$$

This value is often normalized by the number of bipartitions in both trees. This normalized distance is then called **relative RF distance**.

Part I:

Tree Inference on Partitioned Alignments

4 Hardness of Model Assignment

As stated in Chapter 3, in phylogenetics, computing the likelihood that a given tree generated the observed sequence data requires calculating the probability of observing the sequenced data for a given tree (topology and branch lengths) under a statistical model of sequence evolution. Here, we focus on selecting an appropriate model for the data, which represents a generally non-trivial task. It is well-known, that an inappropriate model, which does not fit the data, can generate misleading tree topologies [18, 19, 96].

More specifically, we consider the case of partitioned protein sequence alignments (see Section 3.1, page 12), where each partition may have an individual model of evolution. That is, the model of evolution is unlinked across partitions. Our objective is to maximize the likelihood of the per-partition protein model assignments (e.g., JTT, WAG, etc. [77, 145]) when branches are linked across partitions on a given, fixed tree topology. That is, branch lengths are not estimated individually for each partition. Linked branch lengths across partitions substantially reduce the number of free parameters.

For p partitions and $|M|$ possible substitution models, there are $|M|^p$ possible model assignments. Since the number of combinations grows exponentially with p, an exhaustive search for the highest scoring assignment is computationally prohibitive for $|M| > 1$. We show that the problem of finding the optimal protein substitution model assignment under linked branch lengths on a given, tree topology, is NP-hard. Our results imply that one should employ heuristics to approximate the solution, instead of striving for the exact solution. Alternatively, the problem can be simplified by relaxing the assumptions.

This chapter was first published in the journal of *Theoretical Computer Science* as "Is the Protein Model Assignment problem under linked branch lengths NP-hard?", with Jörg Hauser and Alexandros Stamatakis as co-authors, in 2014 [82]. The paper was recognized by the journal of *Theoretical Computer Science*, as one of their top 5 downloaded papers (4^{th}) between 2010 and 2014.
The NP-hardness proof presented here is my main contribution to this topic. Alexandros Stamatakis first stated the problem, while Jörg Hauser implemented and tested different heuristic solutions. These results are analyzed in more detail in a separate publication [67]. Both Stamatakis and Hauser helped to write the paper.

4.1 Motivation and Related Work

In phylogenetics, many of the questions that we try to answer have been shown to be hard (NP-hard) to solve [3, 30, 55]. Among these are some of the most fundamental problems, such as finding the ML for a given MSA [23, 112] or even finding an optimal MSA [39], which are proven to be NP-hard. Some problems may not even have a unique solution, as is the case with finding the ML phylogeny [133]. In fact, many trees may obtain the same likelihood and thus, form a so-called terrace [117].

Here we are not interested in the actual phylogenetic tree search, but in the optimal assignment of evolutionary models to partitions of a partitioned MSA for a fixed tree. At present, a plethora of empirical protein substitution models is available, such as JTT, DAYHOFF, WAG, etc. [77, 86, 145] some of which are collections of substitution matrices that contain different matrices such as the PAM or BLOSUM families [31, 70]. They are provided in the form of an instantaneous 20×20 substitution matrix and the corresponding base frequencies (prior probabilities) of the states. Given this matrix (usually denoted as *Q-matrix*), one can calculate the transition probabilities from one state to another for a given time/branch length t. If each partition can be evaluated independently from the others, this task is almost trivial and an optimal solution can be found in polynomial time. However, if we assume that the branch lengths of the phylogenetic tree are jointly estimated over all partitions, the model choice for each partition is no longer independent from the choice of the models allocated to the other partitions. Under this assumption, the optimal assignment of models to partitions, with respect to the phylogenetic likelihood, is NP-hard, even if we assume a fixed tree topology.

When analyzing large multi-gene datasets joint branch length estimates can be used to reduce the number of free model parameters and thereby avoid over-parameterizing the model. Each set of independent per-partition branch lengths increases the number of model parameters by $2n-3$ where n is the number of taxa. Therefore, the option to link branch lengths is offered in numerous phylogenetic tools such as RAxML [128] and PartitionFinder [91]. Numerous analyses of multi-gene alignments use this feature (see for example, [61, 94, 126]. Other results suggest that branch lengths may, under certain conditions, inherently be correlated across partitions [78], which provides an additional motivation to link branch lengths across partitions.

Tests on real-world data-sets performed by Hauser *et al.* [67] revealed that suboptimal model assignments under linked branch lengths can change the final tree topologies. They carried out tests on two previously published

multi-gene data-sets [101, 150] using RAxML-Light version 1.0.5 [128]. On these datasets, a total of 150 runs were conducted, on randomly chosen subsets containing three partitions and 50 species each. Thereafter, the best model assignment (with respect to its log likelihood score on the same fixed tree) was determined for each subset using linked *and* unlinked branch lengths. In 57% of the cases these model assignments were not identical. For the cases (subsets) where the model assignments differed, tree searches with RAxML under linked branch lengths using the two alternative model assignments were conducted. For 86% of these runs, the inferred best-known ML trees were different. On average, the Robinson Foulds distance [111] (confer page 27) between different trees inferred under the optimal and suboptimal model amounted to 9%. In other words, using the optimal protein model assignment under linked branch lengths on empirical data frequently yields a different tree topology with respect to the tree obtained from a suboptimal model assignment. Thus, the Protein Model Assignment problem ($PMA^{\star}$) 'matters' since it alters the inferred tree topology. All data-sets from Hauser *et al.* are available for download at `https://github.com/Kobert/PMA`.

4.2 Problem Definition: The Protein Model Assignment Problem

We define the Protein Model Assignment problem ($PMA^{\star}$) as follows: Find the best-fit model from a set of available models for each partition of a protein alignment on some given, fixed, tree topology. Further assume that the branch lengths are linked across partitions. In other words, the branch lengths are estimated/optimized jointly across all partitions of the alignment. The following is a more formal definition:

Let M be a set of evolutionary models. Usually a model is defined by its Q-*matrix*. Here, the evolutionary models from which the Protein Model Assignment problem ($PMA^{\star}$) can choose, are regarded as probability functions whose values represent the transition probability from one state to another, given a certain amount of time t, and the equilibrium frequencies for each state. The matrix and the frequencies are required for the actual likelihood calculations. We introduce this abstract view to avoid the calculations required for obtaining the transition-probabilities from the instantaneous transition rates in Q.

We denote a given model M_i with k states as:

$$M_i = (P, \Pi), \text{ where } \Pi \in [0,1]^k, \tag{16}$$

$$P : \mathbb{R} \rightarrow [0,1]^{k \times k}. \tag{17}$$

Here $P_{X,Y}(t) := P(X \to Y|t)$ is the probability of a transition/mutation from state X to state Y in time t, and π_X is the equilibrium frequency of state X. For amino acid sequences we have 20 states, that is, $k = 20$.

Let A be an alignment for a set of taxa, divided into the p partitions P_1, P_2,$\cdots$,P_p. Let $(T, \beta) = ((V, E), \beta(m))$ be a phylogenetic tree with nodes V, edges E and edge weights (branch lengths) β. Here, the branch lengths $\beta(m)$ are given as edge weights under a chosen phylogenetic model assignment m. Formally we write $\beta : M^p \to \mathbb{R}^{|E|}$.

For this chapter, we assume that an optimal branch length configuration exists and is given for each possible model assignment via a "black box" or an "oracle". That is, $\beta(m)$ always denotes the branch length value that maximizes the tree likelihood under model assignment m ($m \in M^p$). For reasons of complexity we may also assume this function only to take approximate values that fit polynomial sized storage.

$PMA^\star$ can be formulated as follows:

Definition 17 ($PMA^\star$)**.** *Given A, M, T as defined above, find the model assignment $m \in M^p$ that maximizes the likelihood function for A, M and T. That is, maximize $P(A|(T, \beta(m)), m)$, the probability of observing the alignment, given the phylogenetic tree, with respect to m.*

To show that $PMA^\star$ is NP-hard, it suffices to show that a corresponding decision problem is NP-complete.

Definition 18 (PMA Decision problem for $PMA^\star$)**.** *We define the* PMA *decision problem as follows. For a partitioned protein alignment A, a tree T containing all n species of the alignment, and a set of possible models M, does there exist a model assignment m such that the optimal branch length configuration $\beta(m)$ yields a likelihood above some chosen threshold $\hat{b}$?*

In other words:

$$PMA(A, T, M, \hat{b}) = \begin{cases} \textit{true, } \exists\ m \in M^p \textit{ s.t. } LH(A|(T, \beta(m)), m) \geq \hat{b} \\ \textit{false, else} \end{cases}$$

where $LH(A|(T, \beta(m)), m)$ is the probability of observing the data A under the given tree $(T, \beta(m))$ and substitution models m chosen from M^p, that is, the likelihood. An instance of PMA *is uniquely defined by the choice of A, T, M, and $\hat{b}$.*

We demonstrate that the decision problem PMA is NP-complete by initially showing that it is in fact in NP. Then, we reduce the well-known boolean satisfiability problem (SAT, which is known to be NP-complete) to

the decision problem. By definition of NP-completeness, this implies that our problem is also NP-complete [27].

Obviously, the original protein model assignment optimization problem is at least as hard as PMA. If we can obtain the solution of the maximization problem from an oracle, we can verify whether the optimal solution is greater than some real value $\hat{b}$ or not.

4.3 Boolean Satisfiability Problem

SAT and 1-3-SAT One of the most well studied NP-complete problems is the boolean satisfiability problem (SAT), which has been proven to be NP-complete by Cook in 1971 [27]. Here, we show that there exists a polynomial time reduction from SAT to PMA. From this, we deduce that $PMA \in$ NP-complete since any problem in NP can first be reduced to SAT, by definition of NP-completeness, and subsequently to PMA. Again, by definition of NP-completeness, this suffices for showing that $PMA \in$ NP-complete.

For simplicity, we consider a special form of the boolean satisfiability problem called *one-in-three-SAT* (*1-3-SAT*) [119].

The $1-3-SAT$ problem is defined as follows. For *variables* v_i, $i = 1, \cdots, n$ and their *negations* $\neg v_i$, $i = 1, \cdots, n$ a true/false *assignment* a has the following form:

$$a : \{v_1, v_2, \cdots, v_n, \neg v_1, \neg v_2, \cdots, \neg v_n\} \rightarrow \{\text{true, false}\}, \tag{18}$$

where $a(v_i) \neq a(\neg v_i), \forall i = 1, \cdots, n$. Any $l \in \{v_1, v_2, \cdots, v_n, \neg v_1, \neg v_2, \cdots, \neg v_n\}$ is called a *literal*, and we define $\neg(\neg l) = l$.

A *clause* $C_j = C(l_{1,j}, l_{2,j}, l_{3,j})$ is said to be true/satisfied under a, if *exactly* one of the three literals $l_{1,j}$, $l_{2,j}$, $l_{3,j}$ is set to true in the assignment a. For *1-3-SAT* (as well as for the less restrictive *3-SAT* [79]) each clause must contain at most 3 literals. Each literal $l_{1,j}$, $l_{2,j}$, $l_{3,j}$ represents one of the variables or negated variables.

An *instance* c of *1-3-SAT* consists of a combination of clauses.

$$c = C_1 \wedge C_2 \wedge ... \wedge C_m. \tag{19}$$

The assignment a is called truthful/feasible for an instance c, *if and only if*, all clauses C_1, C_2,$\cdots$ C_m are true under a. An instance c is *satisfiable iff* there exists an assignment a, such that a is feasible for c.

1-3-unique-SAT For technical reasons, we impose one additional restriction to the *1-3-SAT* problem. We require that, each problem instance contains only clauses in which each variable appears at most once. In other words, no literal may appear twice in any clause, nor in a clause that contains its negation. Thus, $l_{i,j} \neq l_{k,j}$ and $l_{i,j} \neq \neg l_{k,j} \forall j$, $\forall k \in \{1,2,3\} \setminus \{i\}$. We denote this as *1-3-u-SAT* (*1-3-unique-SAT*) problem. Keep in mind that the clauses $C(v_1, v_1, v_2)$ and $C(v_1, \neg v_1, v_2)$ can not be part of any *1-3-u-SAT* instance.

The following observation shows that the problem is still NP-complete under this restriction.

Observation 19. *1-3-u-SAT* $\in$ NP-complete.

Proof. The *1-3-SAT* problem is known to be NP-complete [119]. What needs to be shown is that, an instance c of *1-3-SAT* can be transformed into an instance $\bar{c}$ of *1-3-u-SAT* in polynomial time, such that $\bar{c}$ is satisfiable under *1-3-u-SAT iff* c is satisfiable under *1-3-SAT*. We show that any clause of a *1-3-SAT* problem can be represented by at most 4 new clauses while adding at most 2 new variables, such that the original clause is satisfiable for an *1-3-SAT* instance *iff* the new clauses are satisfiable for *1-3-u-SAT*.

Note that, if we require some literal l to be true for any feasible true/false assignment, we can enforce this under the new setting by introducing two new variables a and b and two new clauses as follows:

$$C(\neg l, a, b) \wedge C(\neg l, \neg a, \neg b) \tag{20}$$

Furthermore, whenever some literal $\hat{l}$ appears twice in a clause, it must be set to false for any truthful assignment of *1-3-SAT* and its negation must consequently be true. This can be achieved by replacing l with $\neg\hat{l}$ in (20). If the given clause contains a third literal, it must consequently be set to true, which can again be achieved by two new clauses of the above form. If no third literal exists, the clause can never be satisfied. This can be achieved by (in addition to forcing $\hat{l}$ to be false) also requiring $\neg\hat{l}$ to be false with two clauses in the form of (20), which must result in an unsolvable instance. The case where some clause contains both, a literal $\hat{l}$ and its negation $\neg\hat{l}$ implies that a possible third literal must be set to false in any truthful assignment of *1-3-SAT*, since either $\hat{l}$ or its negation $\neg\hat{l}$ will be true. This is again ensured by Equation (20).

Thus, at most two auxiliary variables a and b have to be added, since a and b can be reused for any other clause as well. The number of clauses grows

by a factor of four at most. Using the above algorithm, any instance c of *1-3-SAT* can be transformed into an instance $\hat{c}$ of *1-3-u-SAT* in polynomial time, and c is satisfiable under 1-3-SAT *iff* $\hat{c}$ is satisfiable under *1-3-u-Sat*. □

In the following Section we show how to reduce the *1-3-u-SAT* problem to the *PMA* problem.

4.4 NP-Completeness

PMA is in NP: First we need to show that PMA is in fact in *NP*. While this seems trivial at first glance, it still warrants some consideration since we have so far allowed arbitrary real values for branch lengths and other parameters which might require us to provide non-polynomial memory for storing these values. The first observation is, that for the test parameter $\hat{b}$ we can simply choose a rational number that fits some polynomial storage. For the branch lengths we may refine the "black box", that we use to obtain the branch length values, to either return approximated values that fit the polynomial storage, or to return the approximated likelihood value for any given model assignment. The second approach is easy to validate if the likelihood approximation works in such a way, that the largest rational number to fit polynomial storage is chosen such that it is smaller than or equal to the actual likelihood. The drawback is that we can not interpret the branch length values in any way. As we will see later, this is unfortunate, since there is a clear correspondence between branch lengths of *PMA* and the true/false assignment of *1-3-SAT*. Where appropriate we will mention the changes that have to be made in order to account for approximated branch lengths, as suggested in the first approach. Given that polynomial storage is guaranteed by observing one of the afore mentioned methods, we can observe that:

Observation 20. $PMA \in NP$

Proof. By definition the class *NP* contains all problems for which a true solution can be verified in polynomial time using a deterministic Turing machine. *PMA* is in *NP*, since, as we recall from Section 3.4, the likelihood can be computed in polynomial time, using the Felsenstein pruning algorithm [40]. Thus, we can check if a solution (model assignment and corresponding branch lengths) is true in polynomial time by calculating whether it yields a likelihood larger than $\hat{b}$ or not. □

Reduction of 1-3-unique-SAT to PMA: We will now give a polynomial time algorithm to transform an arbitrary instance c of *1-3-u-SAT* into an instance $\hat{c} = \hat{c}(c)$ of *PMA* that is satisfiable *iff* the original problem c is satisfiable. More specifically, we show how the alignment, the partitions, the tree topology, and models can be constructed and how a truthful solution of *PMA* can be interpreted as a truthful solution of *1-3-u-SAT*. We require at least 9 distinct states for the proof of NP-completeness. This means that the results hold for amino acid data, which has 20 states, but no claim can be made for DNA (4 states) or binary (2 states) data. While both, DNA and binary data, are widely used in phylogenetics, models selection as we define it here is usually irrelevant for DNA and binary data. Instead of choosing from a finite set of precomputed models, as we do for protein data, one estimates the rates from the data at hand (as explained in Section 3.2, page 14). One example for this is the General Time Reversible model, GTR [136], which can be estimated from the data. In the following let k be the number of states with $k \geq 9$. We also require that at least 3 models of protein substitution are available to choose from. In practice, one can choose from the available set of empirical models (WAG, JTT, DAYHOFF, PAM, etc.). Here, we construct artificial models M_1, M_2, and M_3 to prove NP-completeness. The models M_1, M_2, and M_3 are very different from one another and different from any realistic model that would be used in practice (WAG, JTT,...). However, the results from Hauser *et al.* [67] imply that $PMA^\star$ is also not easy to solve given the standard models. None of the heuristics described in that paper (except for exhaustive search) can identify the best scoring model assignment for all test cases.

An instance of *1-3-u-SAT* consists of variables/literals and their arrangement in clauses. A solution is a true/false assignment to the variables. We can map this to a *PMA* instance as follows:

Topology and Alignment

The species in the alignment and phylogenetic tree are the variables and their corresponding negations. We therefore need $2n$ species to achieve this, where n is the number of variables in *1-3-u-SAT*. Hence, our phylogenetic tree has $2n$ taxa. We impose the following constraint on the tree topology: Each variable/species is a direct neighbor of the species representing its negation. Apart from that, an arbitrary tree topology can be constructed as long as it complies with this topological constraint (see Figure 4.1).

Let S_{origin}, S_0, S_1, S_2, S_3, S_{-1}, S_{-2}, S_{-3}, S_{int} be nine unique states

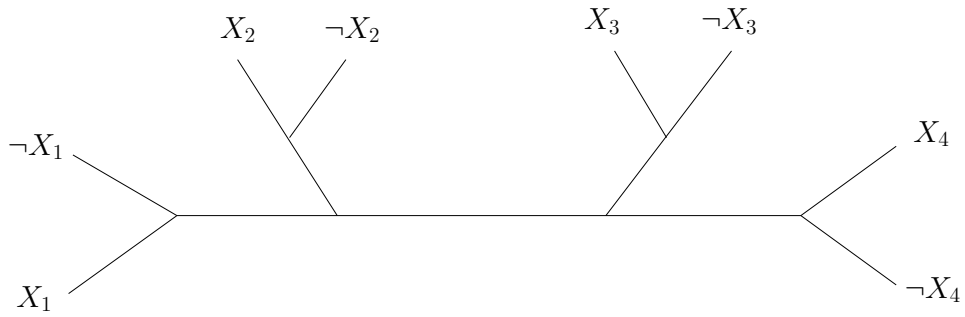


Figure 4.1: *Exemplary tree for $n = 4$ variables/species/taxa.*

(Figure 4.4 illustrates the choice of names for these states). Each clause, $C_i = C(l_{1,i}, l_{2,i}, l_{3,i})$ in c corresponds to one partition S_i in $\hat{c}$ and each partition contains *exactly* one site. To each of the species that correspond to one of the literals $l_{1,i}$, $l_{2,i}$ and $l_{3,i}$, we assign the unique state values of S_1, S_2, and S_3 at site s_i respectively. The corresponding negations, $\neg l_{1,i}$, $\neg l_{2,i}$ and $\neg l_{3,i}$ are assigned the characters S_{-1}, S_{-2}, and S_{-3}, in that order. For all other species, we assign the value S_0 at site s_i (see Figure 4.2). Each partition has exactly one site, with exactly one occurrence of S_1, S_2, S_3, S_{-1}, S_{-2}, S_{-3} for 6 different species and state S_0 for all other species. Note that, S_1, S_2, S_3, S_{-1}, S_{-2}, S_{-3} and S_0 are fixed values. We require that the literals at a position in a clause must always gain the same state and their negation the appropriate consistent counterpart.

Model Construction

The models that are assigned must be of a certain form as outlined below. We distinguish among models based on whether they allow for transitions to states S_1, S_2, S_3 and S_{-1}, S_{-2}, S_{-3} from certain other states and for certain branch lengths with 'high'/not 'near-zero' probability or not. We denote a probability as 'high', if it is greater or equal to some given real value b with $0 < b < 1/4$. We call a probability 'near-zero' or 'diminishing' when it is less than or equal to ϵ, where ϵ is defined in relation to b and π_{S_0}, where π_{S_0} is simply the frequency (as defined in Section 3.2, page 14) of some state S_0. It is chosen such that $\epsilon < \frac{(b^{2 \cdot n} \cdot \pi_{S_0})^m}{k \cdot k^{2n-2}}$, where $2n$ is the number of species, m the number of sites (or the number of clauses for *1-3-u-SAT*), k the number of states, and π_{S_0} with $0 < \pi_{S_0} \leq 1$ an arbitrary, but fixed real value. The branch lengths that we specifically consider are t_{b-}, t_{min} and t_{b+} with $t_{b-} < t_{min} < t_{b+}$. These values can not be chosen arbitrarily, but must comply with some restrictions depending on b and

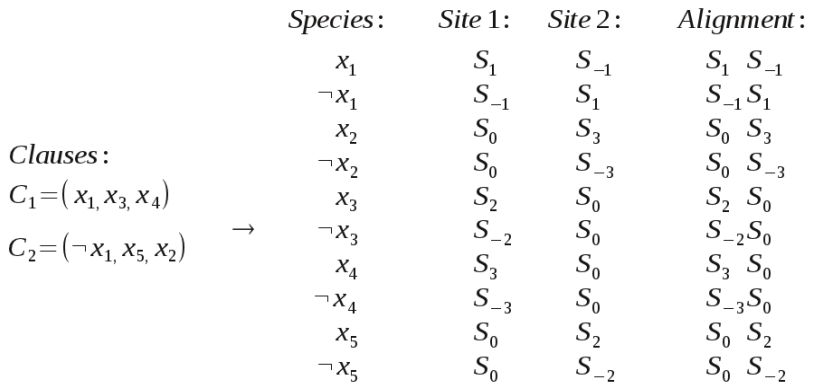


Figure 4.2: *Exemplary transformation of two clauses of* SAT *into an alignment of* PMA*. Clauses C_1 and C_2 correspond to sites* 1 *and* 2 *in the alignment respectively. Keep in mind that each site is in fact a single site partition and can thus be assigned its own model of evolution.*

π_{S_0}. All models must satisfy $\pi_{S_{origin}} \geq \pi_{S_0}$, $P(S_{origin} \to S_0 | t = t_{b+}) \geq b$ and $P(S_0 \to S_0 | t = t_{b-}) \geq b$. Where $P(X \to Y|t)$ is the probability of transitioning from state X to state Y in time t, and $\pi_{S_{origin}}$ is the equilibrium frequency of state S_{origin}.

An important property that we require from these models is that, for each of the three models, it is only possible to reach either state S_1 or S_{-1}, either state S_2 or S_{-2}, and either state S_3 or S_{-3} with non-diminishing probability for any branch length t. Moreover, only one of the three states S_1, S_2, S_3 can be reached with 'high' probability within time $t \geq t_{min}$. Analogously, only a single one of the three states S_{-1}, S_{-2} or S_{-3} can evolve from any other state X with a probability greater or equal to b on a branch shorter than t_{min}. For an illustration of this see Figure 4.3.

The following three models satisfy the aforementioned requirements.

For model $M_1 = M_1(\hat{c}(c))$ we require that:
$P(S_{origin} \to S_1 | t = t_{b+}) \geq b$, $P(X \to S_1|t) < \epsilon \ \forall t < t_{min} \forall X \neq S_1$,
$P(S_{-1} \to S_1|t) < \epsilon \ \forall t$,
$P(S_{origin} \to S_{-2} | t = t_{b+}) \geq b$, $P(X \to S_{-2}|t) < \epsilon \ \forall t < t_{min} \forall X \neq S_{-2}$,
$P(S_2 \to S_{-2}|t) < \epsilon \ \forall t$,
$P(S_{origin} \to S_{-3} | t = t_{b+}) \geq b$, $P(X \to S_{-3}|t) < \epsilon \ \forall t < t_{min} \forall X \neq S_{-3}$
$P(S_3 \to S_{-3}|t) < \epsilon \ \forall t$.
And
$P(S_{origin} \to S_{-1} | t = t_{b-}) \geq b$, $P(X \to S_{-1}|t) < \epsilon \ \forall t \geq t_{min} \forall X$,

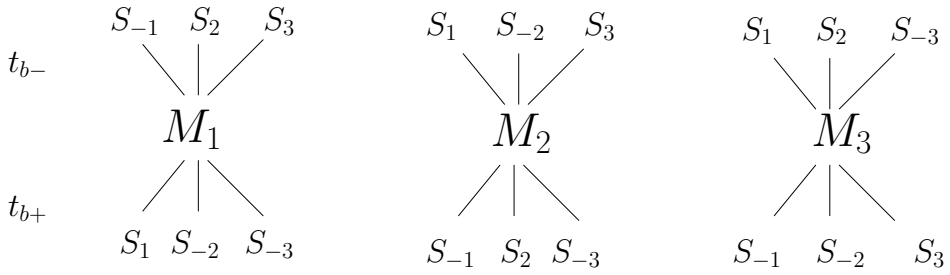


Figure 4.3: *Illustration of how the different models M_1, M_2 and M_3 allow for transitions to S_1, S_2, S_3, S_{-1}, S_{-2} and S_{-3} after time t_{b-} and t_{b+}. After time t_{b-} exactly one of S_{-1}, S_{-2} and S_{-3} can ever be reached. Similarly after time t_{b+} exactly one of S_1, S_2 and S_3 can be reached.*

$P(S_{origin} \to S_2|t = t_{b-}) \geq b$, $P(X \to S_2|t) < \epsilon \ \forall t \geq t_{min} \forall X$,
$P(S_{origin} \to S_3|t = t_{b-}) \geq b$, $P(X \to S_3|t) < \epsilon \ \forall t \geq t_{min} \forall X$.

Analogously for model $M_2 = M_2(\hat{c}(c))$:
$P(S_{origin} \to S_{-1}|t = t_{b+}) \geq b$, $P(X \to S_{-1}|t) < \epsilon \ \forall t < t_{min} \forall X \neq S_{-1}$,
$P(S_1 \to S_{-1}|t) < \epsilon \ \forall t$,
$P(S_{origin} \to S_2|t = t_{b+}) \geq b$, $P(X \to S_2|t) < \epsilon \ \forall t < t_{min} \forall X \neq S_2$,
$P(S_{-2} \to S_2|t) < \epsilon \ \forall t$,
$P(S_{origin} \to S_{-3}|t = t_{b+}) \geq b$, $P(X \to S_{-3}|t) < \epsilon \ \forall t < t_{min} \forall X \neq S_{-3}$
$P(S_3 \to S_{-3}|t) < \epsilon \ \forall t$.
And
$P(S_{origin} \to S_1|t = t_{b-}) \geq b$, $P(X \to S_1|t) < \epsilon \ \forall t \geq t_{min} \forall X$,
$P(S_{origin} \to S_{-2}|t = t_{b-}) \geq b$, $P(X \to S_{-2}|t) < \epsilon \ \forall t \geq t_{min} \forall X$,
$P(S_{origin} \to S_3|t = t_{b-}) \geq b$, $P(X \to S_3|t) < \epsilon \ \forall t \geq t_{min} \forall X$.

And for model $M_3 = M_3(\hat{c}(c))$:
$P(S_{origin} \to S_{-1}|t = t_{b+}) \geq b$, $P(X \to S_{-1}|t) < \epsilon \ \forall t < t_{min} \forall X \neq S_{-1}$,
$P(S_1 \to S_{-1}|t) < \epsilon \ \forall t$,
$P(S_{origin} \to S_{-2}|t = t_{b+}) \geq b$, $P(X \to S_{-2}|t) < \epsilon \ \forall t < t_{min} \forall X \neq S_{-2}$,
$P(S_2 \to S_{-2}|t) < \epsilon \ \forall t$,
$P(S_{origin} \to S_3|t = t_{b+}) \geq b$, $P(X \to S_3|t) < \epsilon \ \forall t < t_{min} \forall X \neq S_3$
$P(S_{-3} \to S_3|t) < \epsilon \ \forall t$.
And
$P(S_{origin} \to S_1|t = t_{b-}) \geq b$, $P(X \to S_1|t) < \epsilon \ \forall t \geq t_{min} \forall X$,
$P(S_{origin} \to S_2|t = t_{b-}) \geq b$, $P(X \to S_2|t) < \epsilon \ \forall t \geq t_{min} \forall X$,
$P(S_{origin} \to S_{-3}|t = t_{b-}) \geq b$, $P(X \to S_{-3}|t) < \epsilon \ \forall t \geq t_{min} \forall X$.

Figure 4.4 illustrates the behavior of a stochastic process by example of model M_3.

All other properties of the probability functions can be freely chosen. Models M_1, M_2 and M_3 are simplified and do not, in their stated form, comply with the assumptions we made when observing that $PMA \in NP$. If we want to accommodate approximated branch lengths that fit polynomial storage, we need to further adjust these probability requirements. Instead of requiring $P(X \to X | t = \bar{t}) \geq b$ for some state X and Y and a time $\bar{t}$, we must require $P(X \to X | t = \hat{t}) \geq b$ for all $\hat{t} \in B(\bar{t})$, where $B(\bar{t})$ is the ball around $\bar{t}$ with a radius that is large enough to accommodate the approximated branch length of $\bar{t}$. If this is obeyed, polynomial storage can be guaranteed.

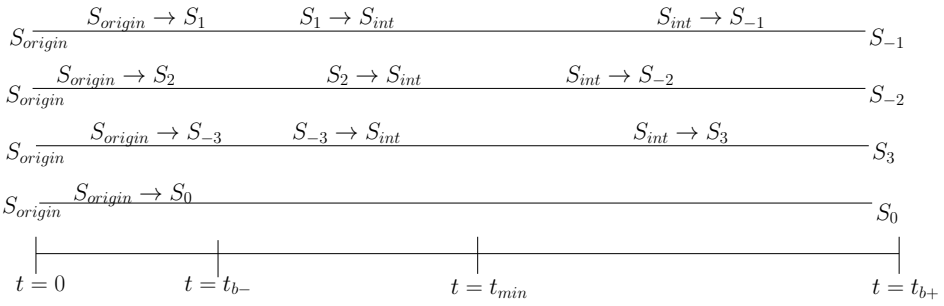


Figure 4.4: *Four exemplary runs of a stochastic process that starts in state S_{origin} and moves according to model M_3. State S_{origin} is always (with high probability) left before time t_{b-} is reached. States S_1, S_2 and S_{-3} change to S_{int} before time t_{min} with high probability. State S_{int} is left, with high probability, at some time t with $t_{min} < t < t_{b+}$. S_{origin} is called origin because all other states can be reached from it. S_{int} is an intermediate state separating S_1 and S_{-1}, S_2 and S_{-2}, and S_3 and S_{-3}, respectively.*

The models are given by explicit probabilities of transitioning from one state to another, given some time t (and the equilibrium state frequencies). In practice, a model is defined by the so called *Q-matrix* (see Section 3.2, page 14), which specifies instantaneous transition rates ($q_{i,j}$) instead of transition probabilities. The instantaneous rates are translated into probabilities [62, 148]. For the sake of simplicity, we use explicit probability functions. We could however also construct three *Q-matrices* whose corresponding probabilities satisfy the requirements of models M_1, M_2, and M_3, respectively.

One way to construct the probability functions with the above properties, requires a total of 9 states. Let b and ϵ be given. For model M_1 choose $q_{S_{origin},S_0} = q_{S_{origin},S_{-1}} = q_{S_{origin},S_2} = q_{S_{origin},S_3}$ large enough and $q_{S_{-1},S_{int}} = q_{S_2,S_{int}} = q_{S_3,S_{int}}$ small enough such that $P(S_{origin} \rightarrow X|t = t_{b-}) \geq b$ for $X \in \{S_0, S_{-1}, S_2, S_3\}$. At the same time $q_{S_{-1},S_{int}}$ must be large enough such that $P(S_{origin} \rightarrow S_{-1}|t = t_{min}) < \epsilon$. The transition rates $q_{S_{int},S_1} = q_{S_{int},S_{-2}} = q_{S_{int},S_{-3}}$ must be chosen such that $P(S_{origin} \rightarrow S_1|t = t_{min}) < \epsilon$ and $P(S_{origin} \rightarrow S_1|t = t_{b+}) \geq b$ (See Figure 4.4). For these reasons t_{b-}, t_{min} and t_{b+} can not be arbitrarily chosen, but must be far enough apart from one another. All other rates $q_{i,j}$ can be set to 0 to make the above construction feasible. Models M_2 and M_3 can be constructed analogously.

If we want to accommodate more than three models, each additional model must at least fulfill the requirements of model M_1, M_2 or M_3. Alternatively, we can use a more restrictive model where at least all those probabilities that are smaller than ϵ, for M_1, M_2 or M_3 must also be smaller than ϵ for the new model.

Proof of Correctness: We now show that the instance c of *1-3-u-SAT* is satisfiable, *iff* we can find a model assignment and respective branch lengths for the corresponding *PMA* instance $\hat{c}(c)$ as defined above, that yields a likelihood above $\hat{b}$. The value $\hat{b} = \hat{b}(\hat{c}(c))$ is defined as $\hat{b} := (b^{2 \cdot n} \cdot \pi_{S_0})^m$, where n is equal to the number of variables and m is the number of clauses in c.

Initially, we observe three properties.

Observation 21. *If some site S_i yields a likelihood of at most ϵ for some $\epsilon \in [0, 1]$, then the overall likelihood for the entire alignment must be less than or equal to ϵ.*

This holds since the likelihood function is multiplicative across sites and each site can only ever have a likelihood of at most 1, but must be greater or equal to 0.

Observation 22. *If we find a site s_i such that the probability $\hat{\epsilon}$ of reaching a tip from an ancestral node is sufficiently small for at least one tip (for given branch lengths)* and *for all possible states of the ancestral state, we observe that this site must yield a likelihood of less than $\hat{\epsilon} \cdot k \cdot k^{2 \cdot n-2}$, where k is the number of possible states.*

This holds, since we need to sum over all k possible states at the root of the tree to obtain the likelihood and over all $k^{2 \cdot n-2}$ possible state configu-

rations for the inner nodes. Each configuration can contribute at most $\hat{\epsilon}$ to the site likelihood.

By Observation 22 we get the following result. If we choose $\hat{\epsilon} = \epsilon$, with $0 < \epsilon < \frac{(b^{2\cdot n}\cdot\pi_{S_0})^m}{k\cdot k^{2\cdot n-2}}$, where k is the number of states $(b, \pi_{S_0} > 0)$, the likelihood at site s_i is strictly smaller than $(b^{2\cdot n}\cdot\pi_{S_0})^m$. By Observation 21, this means that $PMA(\hat{c}(c))$ returns false for this case.

Now we consider the case, where there exists a model configuration, where the probability of going from state S_{origin} at the respective ancestral node to the tip states in the time given by the branch length is always greater or equal to b ($0 < b < 1/4$ is chosen such that models M_1, M_2 and M_3 can be constructed).

Observation 23. *Given the above assumptions, $PMA(\hat{c}(c))$ returns true.*

Proof. We consider a contracted tree (also called *star-tree*) that is obtained by setting all branch lengths for branches that connect inner nodes, to zero, and place the virtual root for likelihood computations on one of these inner (zero length) branches (see Figure 4.5).

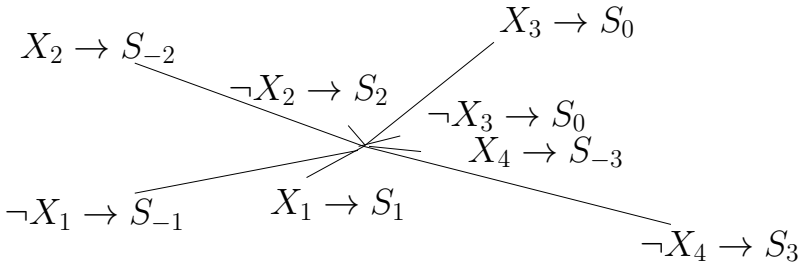


Figure 4.5: *Contracted tree with exemplary tip states for clause $C = C(x_1, \neg x_2, \neg x_4)$. The site given by clause C, yields a likelihood greater or equal to $\hat{b}$ for this tree under model M_3.*

We observe that, the resulting likelihood must be greater or equal to $(b^{2\cdot n}\cdot\pi_{S_0})$. If we consider the term for observing state S_{origin} at the root, which is used to calculate the likelihood, we observe that the probability of going from S_{origin} to S_{origin} in time $t = 0$ is 1 ($P(S_{origin} \to S_{origin}|t = 0) = 1$) and $P(S_{origin} \to X_j|t_j) \geq b$ for the states X_j at all tip-nodes j, by assumption. Hence, the above observation is true. The factor π_{S_0} is given, because the base frequency of state S_{origin}, $\pi_{S_{origin}}$, which forms part of the likelihood computation when assuming an observed state S_{origin} at the root, is always greater or equal to π_{S_0}, by construction of the models. Since this is a feasible branch length and model assignment, the optimal branch length

and model assignment must yield a likelihood that is at least as large as $(b^{2 \cdot n} \cdot \pi_{S_0})$. Thus, if all m sites yield at least this likelihood, the likelihood of the tree for the entire alignment is greater or equal to $(b^{2 \cdot n} \cdot \pi_{S_0})^m$, that is, $PMA(\hat{c}(c))$ returns true. □

Now, we need to show that these two cases for $PMA(\hat{c}(c))$, as detailed in Observations 22 and 23, actually correspond to c being satisfiable and unsatisfiable, respectively. For any given branch length assignment, let the corresponding true/false assignment be given by:
$a(l_i)$ =false *iff* the branch leading to species l_i is of length less than t_{min}.
And $a(l_i)$ =true otherwise (see Figure 4.6).

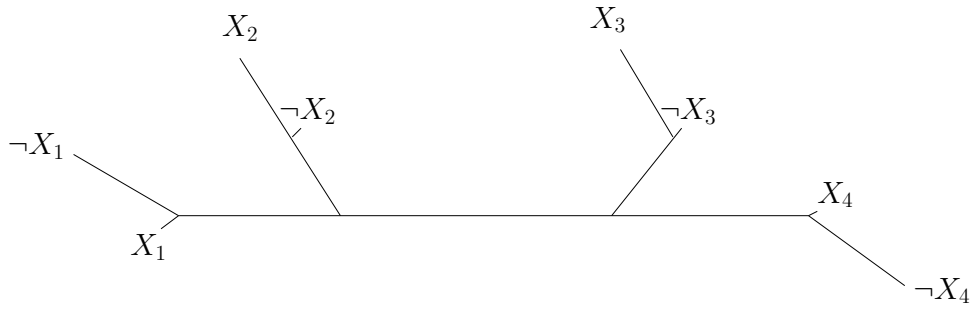


Figure 4.6: *Branch length configuration translating to the following true/false assignment: x_1 =False, x_2 =True, x_3 =True, x_4 =False.*

We show that the above branch length assignment for *PMA* allows for likelihood values greater/lower than the chosen threshold at any site i, *iff* the corresponding clause in c of *1-3-u-SAT* is satisfied/not satisfied under the true-false assignment as obtained by the process we described above.

Theorem 24. *The* 1-3-u-SAT *instance c is satisfiable,* iff *there exists a model assignment m, from the models $M_1 = M_1(\hat{c}(c))$, $M_2 = M_2(\hat{c}(c))$ and $M_3 = M_3(\hat{c}(c))$ for the partitions (sites) S_i of $\hat{c}(c)$ such that the likelihood calculated for some rooting of the tree with optimal branch lengths $b(m)$ is greater or equal to $\hat{b} = \hat{b}(\hat{c}(c))$. Here, $\hat{c}(c)$ is the* PMA *instance corresponding to c.*

In other words: 1-3-u-SAT(c) =true $\Leftrightarrow$ $PMA(\hat{c}(c))$ =true.

Proof. Note that, if the branch lengths of two branches leading to a variable x_i and its negation $\neg x_i$ are of the same length class (i.e., if both are smaller than t_{min}, or if both are greater or equal to t_{min}), the likelihood of the tree is always smaller than $\hat{b}$. This corresponds to a false assignment of the

variables in *1-3-u-SAT*, since the condition $a(x_i) \neq a(\neg x_i)$ is violated for variable x_i. Therefore, we will only consider the remaining cases, where the branch of x_i is greater or equal to t_{min} and the leading branch to $\neg x_i$ is smaller than t_{min}, and vice versa (see Figure 4.7). If a variable does not appear in any clause, this contradiction does not hold. However, in this case it does not matter whether the variable is assigned true or false in the original *1-3-SAT* problem either, such that we can discard these variables.

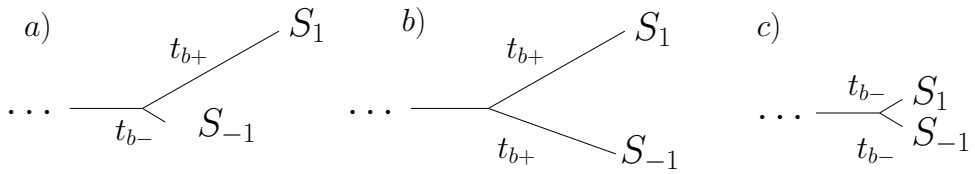


Figure 4.7: *For case a) model M_1 yields a 'high' likelihood. Models M_2 and M_3 yield a 'diminishing' probability. For case b) and case c) all of M_1, M_2 and M_3 yield a 'diminishing' likelihood as only S_1 or S_{-1} can be reached after time t_{b+} and t_{b-}.*

If any clause of c only contains literals that are set to false in the assignment obtained from the branch length solution of *PMA*, the corresponding alignment site will yield a likelihood smaller than ϵ. This holds, because the three literals are set to false, *iff* the branches leading to these literals have a length smaller than t_{min}. However, the models were chosen such that only two literals (i.e., their respective representation in the alignment (states S_1, S_2 and S_3)), can be reached with a probability greater or equal to b within at most t_{min} time. The third literal/tip-branch must contribute a probability of less than ϵ. As we have seen, this implies that *PMA* returns false. Analogously, if for a site i two branches leading to leafs that represent literals in the corresponding clause, have branch lengths exceeding t_{min}, this means that *PMA* and *1-3-u-SAT* (under the corresponding true/false assignment) return false. Again, because of the way we have defined the models, one of the two tip-branches (leading to states S_1, S_2, or S_3) with length greater than t_{min} must contribute a probability of less than ϵ. That is, the overall likelihood is smaller than $\hat{b}$. For an illustration see Figure 4.8.

Now we consider the case where *PMA* (as well as *1-3-u-SAT*) reports true. Let $\beta(m)$ be the optimal branch length configuration for tree T under the model assignment m. Let us further assume that, for each site i, exactly one branch leading to a tip with states S_1, S_2 or S_3 at site i has a length greater or equal to t_{min}. This is equivalent to requiring exactly one literal to be set to true per clause (i.e., the true/false assignment is true for our

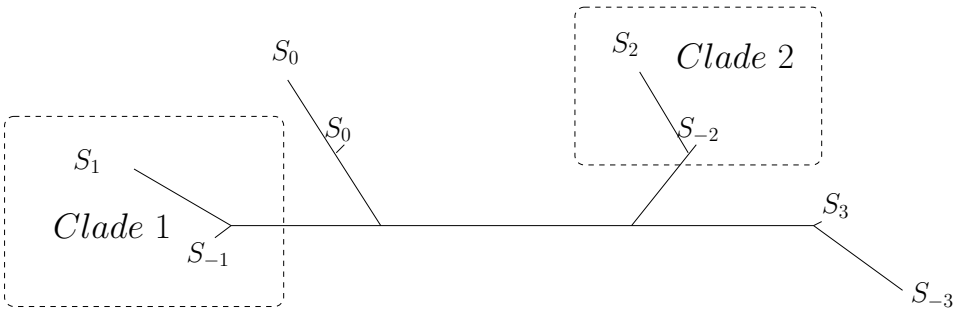


Figure 4.8: *Tree with tip-states for a site corresponding to clause* $C = C(x_1, x_3, x_4)$. *Clade* 1 *contributes a 'high' probability under model* M_1 *but a diminishing probability under models* M_2 *and* M_3. *Similarly Clade* 2 *contributes a 'high' probability under model* M_2 *but a diminishing probability under models* M_1 *and* M_3. *Thus the overall likelihood contribution is diminishing for any of the three models.*

instance of *1-3-u-SAT*), under the corresponding true/false assignment.

Let us consider an alternate branch length assignment $\beta^\star(\beta(m))$ with the following properties: Any branch leading to a tip that has length $t < t_{min}$ in β is assigned length $t^\star = t_{b-}$ in $\beta^\star$. Any branch with length $t \geq t_{min}$ in β, leading to a tip, is set to length $t^\star = t_{b+}$ in $\beta^\star$. All other branch lengths are assumed to be optimized for $\beta^\star$. Obviously, the likelihood of T under β must be greater or equal to that of T under $\beta^\star$. However, the resulting true/false assignment for c is identical in both cases.

For each site (partition) i of the alignment, we can easily decide which model to assign. If the branch leading to the species/literal that was assigned state S_1 at position i is of length t_{b+}, select model M_1. Analogously, select model M_2 or M_3 if the branch leading to S_2 or S_3 has length t_{b+}. If we apply these rules, all branches leading to one of the three literals of the clause corresponding to site i and their corresponding negations yield a probability greater or equal to b. All other branches yield a probability of at least b, independently of the model selected. This means that the overall likelihood of the tree is at least $\hat{b}$ (See Observation 23). Since the likelihood of T under β can only be greater or equal than that of T under $\beta^\star$, *PMA* reports true.

We have shown that, any branch length assignment for *PMA* translates into a true/false assignment of *1-3-u-SAT*. This true/false assignment is true for the instance c, *iff* the corresponding branch length assignment returns true under the optimal model assignment. Hence, we have shown that, *1-3-u-SAT* reduces to *PMA*. □

The proof presented above is constructed in such a way, that it is possible, not only, to verify that an instance c of *1-3-u-SAT* is solvable, *iff* the corresponding instance $\hat{c}(c)$ of *PMA* is solvable. In addition, we also present a means for interpreting the solution of a truthful *PMA* instance as a truthful assignment of *1-3-u-SAT*.

Corollary 25. *PMA* $\in$ NP-complete.

Proof. The corollary follows from Theorem 24 and $PMA \in NP$, as shown in Observation 20. □

Corollary 26. *The Protein Model Assignment Problem (*PMA**) is* NP-*hard.*

4.5 Computational Results

Here we have shown that *PMA** is NP-hard. This leads to the question of how hard this problem is to solve in practice and how good polynomial time heuristics can approximate the optimal solution. These questions are the focus of Hauser *et al.* in [67]. For that publication I was co-author, so we now give a brief summary of the results obtained therein. For more details, please refer to the afore mentioned paper.

The analysis was done on two previously published multi-gene datasets [101, 150] using RAxML-Light version 1.0.5 [128]. A total of 150 runs were conducted, on randomly chosen subsets containing 3 partitions and 50 species each. With only 3 partitions $PMA^\star$ can still be solved exhaustively, and hence exactly, within an acceptable time frame. This exact solution was compared to various heuristics, in terms of the actual model assignment and the resulting ML tree topology, when a ML search was performed under the respective model assignments.

The so called *naïve heuristic* simply optimizes the model assignment under unlinked branch lengths. In 57% of the cases these model assignments were not identical to those found during the exhaustive search. Performing a ML search (with linked branch lengths) under this model assignment resulted in a different tree topology for 86% of the samples.

Among other heuristics, the *steepest ascent* heuristic yielded 'good' results. Nonetheless, this heuristic failed to find the best scoring assignment in 7% of the cases. The relative Robinson Foulds distance [111] (see Definition 16, page 27) between the trees inferred under the optimal and suboptimal (heuristic) model assignment amount to an average of 3%.

4.6 Conclusion

We have shown that the Protein Model Assignment problem ($PMA^\star$) is NP-hard. In other words, unless $P = NP$, no polynomial time algorithm exists that solves this problem exactly.

To reduce the computational effort, one can either relax the constraints or apply heuristics to solve this problem without the guarantee of obtaining the exact solution. One intuitive way to relax the problem is to assume unlinked branch lengths instead of linked branch lengths. Our tests indicate that, this can often yield different trees compared to the optimal solution though.

With respect to potential heuristic approaches, one can, for example, employ hill-climbing methods. These can however converge to a local optimum and do not guarantee a globally optimal model assignment. Furthermore, we have shown how to obtain a solution for an instance of *1-3-u-SAT* (and by reduction, of *1-3-SAT*) by solving an instance of the Protein Assignment Problem (PMA).

The proof presented in this chapter does not make assumptions about time reversibility of the substitution models. It is an open question whether the results hold if we restrict ourselves to time-reversible models. Moreover, the proof makes use of 9 distinct states and requires a minimum of 3 models. For practical reasons, requiring 9 distinct states does not limit us in a meaningful way, since we can apply the result to protein model selection (20 states). For data with a lower number of states, such as DNA (4 states) or binary (2 states) data, model selection is usually not done by assigning precomputed empirical models, but by directly optimizing a rate matrix from the data at hand [136]. From a theoretical point of view, this question is still interesting to answer. However, it is not obvious whether the results can be broadened, for instance, whether $PMA^\star$ is NP-hard for DNA (4 states) or binary (2 states) data, or a minimum of 2 models. If the problem is still NP-hard when we allow only 2 models, the proof must likely use a different NP-hard problem than the boolean satisfiability problem for the reduction, as *2SAT* can be solved in polynomial time [9].

5 Distibution of Partitions to Parallel Processors

Motivated by load balance issues in parallel calculations of the phylogenetic likelihood function we address the problem of distributing divisible items to a given number of bins. The task is to balance the overall sum of (fractional) item sizes per bin, while keeping the maximum number of unique elements in any bin to a minimum. We show that this problem is NP-hard and give a polynomial time approximation algorithm that yields a solution where the sums of (possibly fractional) item sizes are balanced across bins. Moreover, the maximum number of unique elements in the bins is guaranteed to exceed the optimal solution by at most one element. We implement the algorithm in two production-level parallel codes for large-scale likelihood-based phylogenetic inference: ExaML [131] and ExaBayes [2]. For ExaML, we observe best-case runtime improvements of up to a factor of 5.9 compared to the previously implemented data distribution algorithms.

This chapter has been published in *Algorithms in Bioinformatics* as "The divisible load balance problem and its application to phylogenetic inference" in 2014 [81]. The publication was co-authored with Andre Aberer, Tomáš Flouri, and Alexandros Stamatakis.
My contribution to this topic are the actual algorithm for load balancing, the NP-hardness proof, and the proof for near optimality. Stamatakis first formulated the problem in a practical setting. Flouri helped finalize and formalize the algorithm. Aberer implemented and tested the algorithm. All authors were involved in the writing of the paper.

5.1 Motivation and Related Work

Motivation. Maximizing the efficiency of parallel codes by distributing the data in such a way as to optimize load balance is one of the major objectives in high performance computing.

Here, we address a specific case of job scheduling (data distribution) which, to the best of our knowledge, has not been addressed before. We have a list of N divisible jobs, each of which consists of s_i atomic tasks, where $1 \leq i \leq N$, and B processors (or bins). All jobs have an equal, constant startup latency α, and each task, regardless of the job it appears in, requires a constant amount of time β to be processed. Although these times are constant, they depend on the available hardware architecture, and hence are not known a priori. Moreover, the jobs are independent of one another. We also assume that processors are equally fast. Therefore, any task takes time β to execute, independently of the processor it is scheduled to run on.

Any job can be partitioned (or decomposed) into disjoint sets of its original tasks, which can then be distributed to different processors. However, each such set incurs its own startup latency α on the processor on which it is scheduled to run. Thus, a job of k tasks takes time $k \cdot \beta + \alpha$ to execute on any processor. The tasks (even of the same job) are independent of each other, that is, they can be executed in any order, and the sole purpose of the job configuration is to group together the tasks that require the same initialization step and hence minimize the overall startup latency.

Our work is motivated by parallel likelihood computations in phylogenetics (see [43, 148] for an overview). There, we are given a MSA that is typically subdivided into distinct partitions (as introduced in Section 3.1, page 12). Given the alignment and a partition scheme, the likelihood on a given candidate tree can be calculated. To this end, transition probabilities for the statistical nucleotide substitution model need to be calculated (start-up cost α in our context) for each partition separately because they are typically considered to evolve under different models (see Section 3.2, page 14). Note that, all alignment sites that belong to the same partition have identical model parameters.

The partitions are the divisible jobs to be distributed among processors. Each partition has a fixed number of *sites* (columns from the alignment), which denote the size of the partition. The sites represent the independent tasks a job (partition) consists of. Since alignment sites are assumed to evolve independently in the likelihood model, the calculations on a single site can be performed independently of all other sites (see Equation (14), page 24). Thus, a single partition can easily be split among multiple processors.

As we reason in Section 3.2 (page 16), the overhead α is actually performance critical.

Finally, note that, parallel implementations of the phylogenetic likelihood function now form part of several widely-used tools [49, 103, 134] and the results presented in this chapter are generally applicable to all tools.

Related work. A related problem is *bin-packing* with item fragmentation. Here, items may be fragmented, which can potentially reduce the total number of bins needed for packing the instance. However, since fragmentation incurs overhead, unnecessary fragmentations should be avoided. The goal is to pack all items in a minimum number of bins. For an overview of the fractional bin packing problem see [58, Chapter 33]. However, in contrast to our problem, the number of bins is not part of the input but is the objective

function.

The most closely related domain of research is *divisible load theory* (DLT). Here, the goal is to distribute optimal fractions of the total load among several processors such that the entire load is processed in a minimal amount of time. For a review on DLT, see [13]. However, in general DLT can accommodate more complex models, taking into account a number of factors, such as network parameters or processor speeds. Our problem falls into the category of scheduling divisible loads with start-up costs (see for instance [15, 141]). To our knowledge the problem we present has not been solved before.

There exists previous work by our group on improving the load-balance in parallel phylogenetic likelihood calculations [152]. There, for the sake of code simplicity, single partitions/jobs are assumed to be indivisible. Thus, the scheduling problem addressed there was equivalent to the 'classic' multiprocessor scheduling problem.

Overview. In Section 5.2 we formally define two variations of the problem. We then prove that the problem is NP-hard (Section 5.3). The main contribution of this chapter can be found in Section 5.4, where we give a polynomial-time approximation algorithm which yields solutions that assign at most one element more, that is, sites from one additional partition, to any processor (or bin) than the optimal solution. We analyze the algorithm complexity and prove the $OPT+1$ approximation in Section 5.5. Unless P = NP [27, 79], no polynomial time algorithm can guarantee a better worst case approximation. Finally, in Section 5.6, we present the performance gains we obtain, when employing our algorithm for distributing partitions in ExaML [131], available at `http://www.exelixis-lab.org/web/software/examl/index.html` .

5.2 Problem Definition: Load Balancing

Assume we have N divisible items of sizes $s_1, s_2, \ldots, s_N$, and B available bins. Our task is to find an assignment of the N items to the B bins, by allowing an item to be partitioned into several sub-items whose total size is the size of the original item, in order to achieve the following two goals:

1. The sum of sizes of the (possibly partitioned) items assigned to each bin is well-balanced.

2. The maximum load over all bins is minimal with respect to the number of items added.

In the rest of the text we will use the term *solid* for the items that are not partitioned, and *fractional* for those that are partitioned.

We can now formally introduce two variations of the problem; one where we only allow items of integer sizes, and one where the sizes can be represented by real numbers. In the case of integers, the problem can be formulated as the following integer program.

Problem 27 (LB$\mathbb{N}$)**.** *Given a sequence of positive integers $s_1, s_2, \ldots, s_N$ and a positive integer B,*

$$\begin{aligned}
&\textit{minimize} && \max\{\textstyle\sum_{j=1}^{N} x_{i,j} \mid i = 1, 2, \ldots, B\} \\
&\textit{subject to} \\
&&\textstyle\sum_{i=1}^{B} q_{i,j} = s_j, && 1 \le j \le N \\
&&\textstyle\sum_{j=1}^{N} q_{i,j} \ge \lfloor \sigma/B \rfloor, && 1 \le i \le B \\
&&\textstyle\sum_{j=1}^{N} q_{i,j} \le \lceil \sigma/B \rceil, && 1 \le i \le B \\
&&\sigma = \textstyle\sum_{i=1}^{N} s_i \\
&&0 \le q_{i,j} \le x_{i,j} \cdot s_j, && 1 \le i \le B,\ 1 \le j \le N \\
&&q \in \mathbb{N}_{\ge 0}^{B \times N} \\
&&x \in \{0,1\}^{B \times N}
\end{aligned}$$

By removing the imposed restriction of integer sizes, and hence allowing for positive real values as the sizes of both solid and fractional items, we obtain the following mixed integer program.

Problem 28 (LB$\mathbb{R}$). *Given a sequence of positive real values $s_1, s_2, \ldots, s_N$ and a positive integer value B,*

$$\begin{aligned}
&\textit{minimize} && \max\{\textstyle\sum_{j=1}^{N} x_{i,j} \mid i = 1, 2, \ldots, B\} \\
&\textit{subject to} \\
&&& \textstyle\sum_{i=1}^{B} q_{i,j} = s_j, && 1 \le j \le N \\
&&& \textstyle\sum_{j=1}^{N} q_{i,j} = \sigma/B, && 1 \le i \le B \\
&&& \sigma = \textstyle\sum_{i=1}^{N} s_i \\
&&& 0 \le q_{i,j} \le x_{i,j} \cdot s_j, && 1 \le i \le B,\ 1 \le j \le N \\
&&& q \in \mathbb{R}^{B \times N} \\
&&& x \in \{0,1\}^{B \times N}
\end{aligned}$$

If for some bin i and element j we get a solution with $q_{i,j} < s_j$, we say that element j is only assigned to bin i partially, or that only a fraction of element j is assigned to bin i. If $q_{i,j} = s_j$ we say that element j is fully assigned to bin i.

5.3 NP-Hardness

We now show that problems LB$\mathbb{N}$ and LB$\mathbb{R}$ are NP-hard by reducing the well-known Partition [79] problem. We reduce it to another decision problem that decides whether a set can be broken into disjoint sets of equal cardinality and equal sum of elements (see Problem ECP, Def. 30), which can be solved by the two flavors of our problem.

Definition 29 (Partition). *Is it possible to partition a set S of positive integers into two disjoint subsets Q and R, such that $Q \cup R = S$ and $\sum_{q \in Q} q = \sum_{r \in R} r$?*

Definition 30 (ECP). *Let p and k be two positive integers and S a set of positive integers such that $|S| = p \cdot k$. Is it possible to partition S into p disjoint sets $S_1, S_2, \ldots, S_p$ of k elements each, such that $\bigcup_{i=1}^{p} S_i = S$ and $\sum_{s \in S_i} s = \sum_{s \in S_j} s$, for all $1 \le i \le p$ and $1 \le j \le p$?*

Clearly, if we can solve our original optimization problems LB$\mathbb{N}$ and LB$\mathbb{R}$ for any S exactly, we can also answer whether ECP returns *true* or *false* for the same set S. Thus, if we can show that ECP is NP-Complete we know

that the original problems are NP-hard.
To show that ECP is NP-Complete, it is sufficient to show that ECP is in NP, that is the set of polynomial time verifiable problems, and some NP-Complete problem (here PARTITION) reduces to it.

Lemma 31. ECP *is NP-Complete.*

Proof. The first part, i.e., ECP $\in$ NP, is trivial. Given a solution (that is, the sets $S_1,\ldots,S_p$), we are able to verify, in polynomial time to p, that the conditions for problem ECP hold, by summing the elements of each set.

For the reduction of PARTITION to ECP consider the set S to be an instance of PARTITION.

We derive an instance $\hat{S}$ of ECP from S, such that PARTITION(S) is true *iff* ECP($\hat{S}$) is true for 2 bins (that is $p = 2$).

To this end, we define $\hat{S} = S \cup (a \cdot S)$ a set of integers, with $a = (1 + \sum_{s\in S} s)$ and $(a \cdot S) = \{\, a \cdot s \mid s \in S \,\}$.

Clearly, if there is a solution for PARTITION given S, there must also be a solution for ECP given $\hat{S}$. If $Q, R \subset S$ is a solution for PARTITION, $Q \cup (a \cdot R)$, then $R \cup (a \cdot Q)$ is a solution for ECP.

Similarly, let $\hat{Q}, \hat{R}$ be a solution for ECP given $\hat{S}$. Let $Q = \hat{Q} \cap S$, $R = \hat{R} \cap S$, $(a \cdot Q) = \hat{Q} \cap (a \cdot S)$ and $(a \cdot R) = \hat{R} \cap (a \cdot S)$.

Trivially, it holds that $Q = \{\, q \in \hat{Q} \mid q < a \,\}$, $R = \{r \in \hat{R} \mid r < a\}$ and $(a \cdot Q) = \hat{Q} \setminus Q$, $(a \cdot R) = \hat{R} \setminus R$.

Thus, we obtain $Q \cup R = S$ and $(a \cdot Q) \cup (a \cdot R) = (a \cdot S)$. We also obtain that $\sum_{q\in Q} q = \sum_{r\in R} r$ (and $\sum_{q\in(a\cdot Q)} q = \sum_{r\in(a\cdot R)} r$).

We prove that the equations hold by contradiction:
Suppose this was not the case for some solution of ECP, that is $\sum_{q\in Q} q \neq \sum_{r\in R} r$ and hence $\sum_{q\in(a\cdot Q)} q \neq \sum_{r\in(a\cdot R)} r$.

By definition, $(a \cdot Q)$ and $(a \cdot R)$, q/a and r/a are integer values for any $q \in (a \cdot Q)$ and $r \in (a \cdot R)$, and therefore:

$$\begin{aligned} |\sum_{q\in(a\cdot Q)} q - \sum_{r\in(a\cdot R)} r| &= |\sum_{q\in(a\cdot Q)} a\cdot q/a - \sum_{r\in(a\cdot R)} a\cdot r/a| \\ &= a \cdot |\overbrace{\sum_{q\in(a\cdot Q)} q/a - \sum_{r\in(a\cdot R)} r/a}^{\geq 1}| \geq a \end{aligned}$$

However, $\sum_{s\in Q\cup R=S} s < a$. Thus, $\sum_{q\in Q\cup(a\cdot Q)=\hat{Q}} q \neq \sum_{r\in R\cup(a\cdot R)=\hat{R}} r$ which contradicts the assumption of $\hat{Q}$, $\hat{R}$ being a solution for ECP($\hat{S}$,2).

Therefore, PARTITION reduces to ECP, which means that ECP is NP-Complete. □

Corollary 32. *The optimization problems* LB$\mathbb{N}$ *and* LB$\mathbb{R}$ *are NP-hard.*

This follows directly from Lemma 31 and the fact that an answer for ECP can be obtained by solving the optimization problem.

5.4 Algorithm

```
LoadBalance(N, B, S)
▷ Phase 1 — Initialization
 1.  Sort S in ascending order and let S = (s_1, s_2, ..., s_N)
 2.  σ = Σ_{i←1}^{N} s_i
 3.  c ← ⌈σ/B⌉
 4.  r ← c · B − σ
 5.  for i ← 1 to B do
 6.      size[b] ← 0; items[b] ← 0; list[b] ← ∅
 7.  full_bins ← 0; b ← 0;
▷ Phase 2 — Initial filling
 8.  for i ← 1 to N do
 9.      if size[b] + s_i ≤ c then
10.          size[b] ← size[b] + s_i
11.          items[b] = items[b] + 1
12.          Enqueue(list[b], (i, 1, s_i))
13.          if size[b] = c then
14.              full_bins ← full_bins + 1
15.              if full_bins = B − r then c ← c − 1
16.      else
17.          break
18.      b ← (b + 1)  mod B
```

Figure 5.1: *The algorithm accepts three arguments N, B and S, where N is the number of items in list S, and B is the number of bins*

As seen in Section 5.3, finding an optimal solution to this problem is hard. To overcome this hurdle, we propose an approximation algorithm running in polynomial time that guarantees a near-optimal solution. For an in-depth analysis of the complexity of the algorithm, see Section 5.5. The input for the algorithm is a list S of N integer weights (numbers of sites for

the partitions) and the number of bins B (processors) these elements must be assigned to. The idea of the algorithm can be explained by the following three steps:

1. Sort S in ascending order.
2. Starting from the first (solid) element in the sorted list S, assign elements from S to the B bins in a cyclic manner (at any time no two bins can have a difference of more than one element) until any bin can not entirely hold the proposed next item.
3. Break the remaining elements from S to fill the remaining space in the bins.

Fig. 5.1 presents the pseudo code for the first two phases, while Fig. 5.2 illustrates phase 3. The output of this algorithm is an assignment, $\mathit{list} = (\mathit{list}[1], \ldots, \mathit{list}[p])$, of (possibly fractional) elements to bins. Each entry in *list* is a set of triplets that specify which portion of an integer sized element is assigned to a bin. Let $(j, i, k) \in \mathit{list}[l]$ be one such triplet for bin number l. We interpret this triplet as follows: bin l is assigned the fraction of element j that starts at i and ends at k (including i and k).

For the application in phylogenetics, each triplet specifies which portion (how many sites) of a partition is assigned to which processor. Again, let $(j, i, k) \in \mathit{list}[l]$ be one such triplet for some processor l. We interpret this triplet as follows: processor l is assigned sites i through k of partition j.

If $i \neq 1$ or $k \neq s_j$ (recall s_j is the size of element j), we say that element j is partially assigned to bin i, that is, only a fraction of element j is assigned to bin i. Otherwise, if $i = 1$ and $k = s_j$, then the triplet represents a solid element, i.e., element j is fully assigned to bin i.

For applications that allow any fraction of an integer to be assigned to a bin, not just whole integer values (that is, problem LB$\mathbb{R}$), we redefine the variable c to be exactly σ/B, without rounding. Additionally, the output (*list*) must correctly state which ranges of the elements are assigned to which bin and not give integer lower and upper bounds.

We give two examples of how algorithm LoadBalance works on a specific set of integers.

Example 33. *Consider the set* $\{2, 2, 3, 5, 9\}$ *and three bins. During initialization (phase 1) we have* $c = 7$ *and* $r = 0$. *Phase 2 makes the following assignments:* $\mathit{list}[1] = \{(1, 1, 2), (4, 1, 5)\}$, $\mathit{list}[2] = \{(2, 1, 2)\}$, $\mathit{list}[3] = \{(3, 1, 3)\}$. *Adding the next element of size 9 is not possible since*

```
▷ Phase 3 — Partitioning items into bins
19.  low ← B; ℓ ← B; high ← 1; h ← 1
20.  while i ≤ N do
21.      while size[ℓ] ≥ c do
22.          low ← low − 1; ℓ ← low
23.      while size[h] ≥ c do
24.          high ← high + 1; h ← high
25.      if size[h] + add ≥ c then
26.          items[h] ← items[h] + 1
27.          Enqueue(list[h], (i, s_i − add + 1, s_i − add − size[d] + c))
28.          add ← size[h] + add − c
29.          size[h] ← c
30.          full_bins ← full_bins + 1
31.          if full_bins = B − r then c ← c − 1
32.      else
33.          items[ℓ] ← items[ℓ] + 1
34.          if size[ℓ] + add < c then
35.              size[ℓ] ← size[ℓ] + add
36.              Enqueue(list[ℓ], (i, s_i − add + 1, s_i))
37.              add ← 0
38.              high ← high − 1; h ← ℓ
39.              low ← low − 1; ℓ ← low
40.          else
41.              Enqueue(list[ℓ], (i, s_i − add + 1, s_i − add − size[d] + c))
42.              add ← size[ℓ] + add − c
43.              size[ℓ] ← c
44.              full_bins ← full_bins + 1
45.              if full_bins = B − r then c ← c − 1
46.      if add = 0 then
47.          i ← i + 1; add ← s_i
```

Figure 5.2: *Phase 3 of the algorithm*

$size[2] + 9 = 2 + 9 = 11 > c$. *Thus, phase 2 ends. Phase 3 splits the last element of size 9 among bins* 2 *and* 3*, and the solution is* $list[1] = \{(1,1,2),(4,1,5)\}$, $list[2] = \{(2,1,2),(5,1,5)\}$, $list[3] = \{(3,1,3),(5,6,9)\}$. *With* $\max\{|list[1]|, |list[2]|, |list[3]|\} = 2$. *This is also an optimal solution.*

Example 34. *Consider the set* $\{1,1,2,3,3,6\}$ *and two bins. During*

the initialization (phase 1) we have $c = 8$ *and* $r = 0$*. Phase 2 generates the following assignments:* $list[1] = \{(1,1,1),(3,1,2),(5,1,3)\}$, $list[2] = \{(2,1,1),(4,1,3)\}$*. The last element of size 6 can not be fully assigned to bin 2, thus phase 2 terminates. Finally, phase 3 splits the last element of size 6 among the two bins, and the solution is* $list[1] = \{(1,1,1),(3,1,2),(5,1,3),(6,1,2)\}$, $list[2] = \{(2,1,1),(4,1,3),(6,3,6)\}$*. We get* $\max\{|list[1]|,|list[2]|\} = 4$*. However, an optimal solution* $list_1^\star = \{(1,1,1),(2,1,1),(6,1,6)\}$, $list_2^\star = \{(3,1,2),(4,1,3),(5,1,3)\}$ *with* $max\{|list_1^\star|,|list_2^\star|\} = 3$ *exists.*

As we can see in Example 34, algorithm LoadBalance fails to find the optimal solution in certain cases. However in the next section we show that the difference of 1, as observed in Example 34, already represents the worst case scenario.

5.5 Algorithm analysis

We now show that the score obtained by algorithm LoadBalance, for any given set of integers and any number of bins, is at most one above the optimal solution. We then give the asymptotic time and space complexities.

Near-optimal solution: Before we start with the proof, we make three observations associated with the algorithm that facilitate the proof. We use the same notation as in the description of the algorithm. That is, $items[i]$ indicates the number of items in bin i, $size[i]$ the sum of sizes of items in bin i, and $list[i]$ is a list of records per item in bin i, describing which fraction of the particular item is assigned to bin i.

Observation 35. *During phase 2 of algorithm* LoadBalance*, it holds that*

$$size[i] > size[j]$$

for any two bins j *and* i*, such that* $items[i] = items[j] + 1$*.*

The list of integers was sorted in Phase 1 of the algorithm to a non-decreasing sequence. Hence, any item added to a bin during the i-th cyclic iteration over bins, must be smaller or equal to an item that is added during iteration $i+1$. Let s_l^k denote the item that is added to bin k during iteration l. For two bins j and i, it holds that

$$s^i_{items[j]+1} \geq s^j_{items[j]}, s^i_{items[j]} \geq s^j_{items[j]-1}, \ldots, s^i_2 \geq s^j_1.$$

Since $s_1^i > 0$, we obtain that $\sum_{l=1}^{items[j]+1} s_l^i = size[i] > size[j] = \sum_{l=1}^{items[j]} s_l^j$.

Observation 36. *For all bins i and j during phase 2 of algorithm* LoadBalance*, it holds that*

$$items[j] \leq items[i] + 1.$$

This follows directly from Observation 35.

Observation 37. *Phase 3 appends at most 2 more (fractional) items to a bin.*

Any remaining (unassigned) item of size s in this phase satisfies the condition $size[j] + s > c$, for any bin j and capacity c as computed in Fig. 5.1. Therefore, each bin will be assigned at most one fractional item that does not fill it completely, and one new element that is guaranteed to fill it up.

Lemma 38. *Let $OPT(S, B)$ be the score for the optimal solution for a set S distributed to B bins. Let list be the solution produced by Algorithm* LoadBalance *for the same set S and B bins. Then:*

$$\max\{\, |list[i] \mid i = 1, 2, \ldots, B \,\} \leq OPT(S, B) + 1$$

Proof. Let $\hat{j}$ be the bin that terminates phase 2. That is, $\hat{j}$ is the last bin considered for any assignment in phase 2. After phase 2, if there exists a bin j with $items[j] = items[\hat{j}] + 1$ we get, by Observation 35 and the *pigeonhole principle*, that $OPT(S, B) \geq items[\hat{j}] + 1$. Otherwise, if no such bin exists, $OPT(S, B) \geq items[\hat{j}]$. Let K be the number of unassigned elements at the beginning of phase 3. Let J be the number of bins j with $items[j] = items[\hat{j}]$. We distinguish between three cases. First assume that $items[j] = items[\hat{j}]$ for all bins j and $K > 0$. Clearly, $OPT(S, B) \geq items[\hat{j}] + 1$. By observation 37 we know that $items[j] \leq items[\hat{j}] + 2$. Thus the lemma holds for this case. Now consider $K > J$ and $items[j] \neq items[\hat{j}]$ for some bin j, that is, there are more unassigned elements than there are bins with only $items[\hat{j}]$ elements assigned to them. By the pigeonhole principle, $OPT(S, B) \geq items[\hat{j}] + 2$. By observation 37 we get that $items[j] \leq items[\hat{j}] + 1 + 2 = items[\hat{j}] + 3$ for all j. Thus the lemma holds for this case as well. For the last case assume $K \leq J$ and $items[j] \neq items[\hat{j}]$ for some bin j. After a process is assigned a fractional element that does not fill it completely, it is immediately filled up with the next element. Since preference is given to any bin j with $items[j] = items[\hat{j}]$ and there are at least as many such bins as remaining elements to be added ($K \leq J$), we get that $items[j] \leq items[\hat{j}] + 2$. Since we have seen above that $OPT(S, B) \geq items[\hat{j}] + 1$, the lemma holds. As this covers all cases, the lemma is proven. □

Run-time: The runtime analysis is straight forward. Phase 1 of the algorithm consists of initializing variables, sorting N items by size in ascending order and computing their sum. Using an algorithm such as MERGE-SORT, Phase 1 requires $\mathcal{O}(N \log(N))$ time. Phase 2 requires $\mathcal{O}(N)$ time to consider at most N items, and assign them to B bins in a cyclic manner. Phase 3 appends at most 2 items to a bin (see Observation 37), and hence has a time complexity of $\mathcal{O}(B)$. This yields an overall asymptotic run-time complexity of $\mathcal{O}(N \log(N) + B)$. Finally, LOADBALANCE requires $\mathcal{O}(B)$ space due to the arrays *items*, *size* and *list*, that are each of size B.

5.6 Computational Results

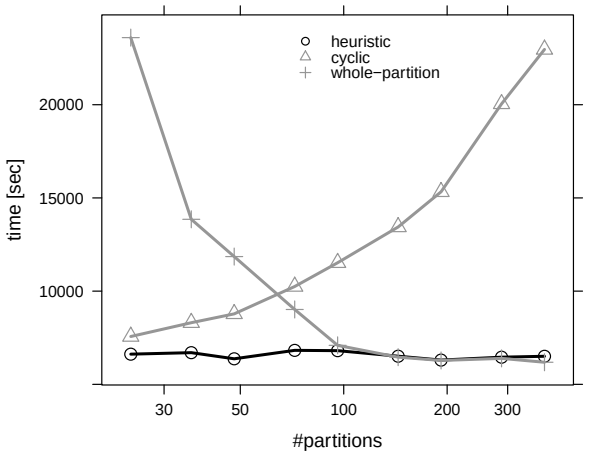
As mentioned before, the scheduling problem arises for parallel phylogenetic likelihood calculations on large partitioned multi-gene or whole-genome datasets. This type of partitioned analyses represent common practice at present (see for example [61, 94, 126]).

The number of MSA partitions, the number of alignment sites per partition, and the number of available processors are the input to our algorithm.

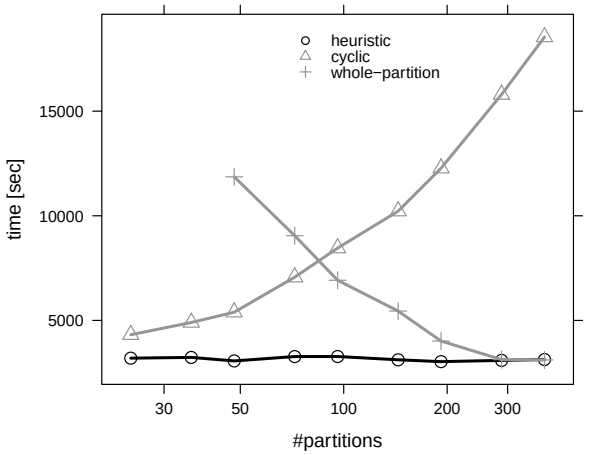
In order to evaluate the new distribution scheme, we compare it to the two original schemes presented in Section 3.2 (page 16), that is, the cyclic, and the whole-partition or monolithic data distribution schemes. The runtime is measured as the total ExaML runtime. Note that, our algorithm has also been implemented in ExaBayes[1] [2] which is a code for large-scale BI.

Methods: We performed runtime experiments on a real-world alignment. The alignment comprises 144 species and 38 400 amino acid characters (data from the 1KITE project [76, 104]). We used the alignment to create 9 distinct partitioning schemes with an increasing number of partitions. For each scheme, partition lengths were drawn at random, while the number of partitions per scheme was fixed to 24, 36, 48, 72, 96, 144, 192, 288, 384, and 768, respectively. To generate 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. The offset of 0.1 was added to random numbers to prevent partition lengths from becoming unrealistically small, since the exponential distribution strongly favors small values. Fig. 5.4 displays the distributions of the partition lengths for each of the 9 partition

[1] Available at http://www.exelixis-lab.org/web/software/exabayes/index.html



(a) Runtimes on 24 cores.



(b) Runtimes on 48 cores.

Figure 5.3: *Runtime comparison for ExaML employing algorithm* LoadBalance*, the cyclic data distribution scheme, or the monolithic partition distribution scheme.*

schemes. As expected, partition lengths are distributed uniformly on the log-scale.

We executed ExaML using 24 and 48 processes, respectively, to assess performance with our new data distribution algorithm and compare it with

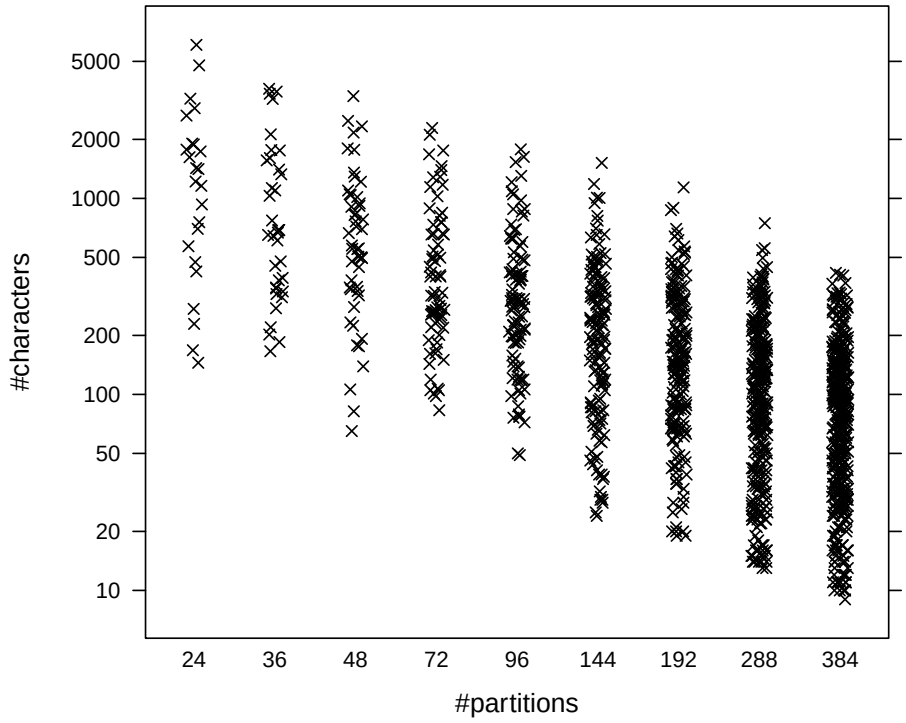


Figure 5.4: *Number of characters/sites in each partition for the various partitioning schemes.*

the cyclic and monolithic partition distribution performance. We used a cluster equipped with Intel SandyBridge nodes (2×6 cores per node) and an Infiniband interconnect. Thus, a total of 2 nodes was needed for runs with 24 processes and 4 nodes for runs with 48 processes (inducing higher inter-node communication costs). In Fig. 5.3.b, the run-times for the monolithic partition distribution approach with less than 48 partitions are omitted, since they are identical to executing the runs on 24 processes. The reason is that this method does not divide partitions and thus, in case the number of partitions is smaller than the number of available processors, the extra processors will remain unused.

Results: As illustrated by Fig. 5.3, with algorithm LoadBalance, ExaML always runs at least as fast as the two previous data distribution strate-

gies with one minor exception. Compared to the cyclic data distribution, LoadBalance is 3.5× faster for 24 processes and up to 5.9× faster for 48 processes. Using LoadBalance, ExaML requires up to 3.6× less runtime than with the monolithic partition distribution 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 monolithic partition distribution scheme converges against the runtime of LoadBalance. This is expected, since by increasing the number of partitions we break the alignment into smaller chunks and the chance of any heuristic to attain a near-optimal load/data distribution increases. However, if the same run is executed with more processes (i.e., 48 instead of 24), this break-even point shifts towards a higher number of partitions, as shown in Fig. 5.3.

The results show that, cyclic data distribution performance is acceptable for many processes and few partitions, whereas monolithic whole-partition data distribution is on par with our new heuristic for analyses with few processes and many partitions. Both Figures show, that there exists a region where neither of the previous strategies exhibits acceptable performance compared to LoadBalance and that this performance gap widens, as parallelism increases.

Finally, employing LoadBalance, 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 the cyclic data distribution, scaling factors ranged from 1.24 to 1.75 and under whole partition distribution, scaling factors ranged from 1.00 (i.e., no parallel runtime improvement) to 2.04. The slight super-linear speedups are due to increased cache efficiency.

5.7 Conclusion

We have introduced an approximation algorithm for solving a NP-hard scheduling problem with an acceptable worst-case performance guarantee. This theoretical work was motivated by our efforts to improve parallel efficiency of phylogenetic likelihood calculations. By implementing the approximation algorithm in ExaML, a dedicated code for large-scale ML-based phylogenetic analyses on supercomputers, we showed that (i) the data distribution is near-optimal, irrespective of the number of partitions, their lengths, and the number of processes used and (ii) substantial run time improvements can be achieved, thus saving scarce supercomputer resources. The data distribution algorithm is generally applicable to any code that parallelizes likelihood calculations.

6 Calculating the Internode Certainty and Related Measures on Partial Gene Trees

Lastly for tree inferences on partitioned MSA, we present, implement, and evaluate an approach to calculate the internode certainty and tree certainty on a given reference tree from a collection of partial gene trees. Previously, the calculation of these values was only possible from a collection of gene trees with exactly the same taxon set as the reference tree. An application to sets of partial gene trees requires mathematical corrections in the internode certainty and tree certainty calculations. We implement our methods in RAxML and test them on empirical data sets. These tests imply that the inclusion of partial trees *does* matter. However, in order to provide meaningful measurements, any data set should also contain comprehensive trees.

A manuscript containing the contents of this chapter has been published in *Molecular Biology and Evolution* as "Computing the Internode Certainty and Related Measures from Partial Gene Trees" in 2016 [83]. Antonis Rokas, Leonidas Salichos, and Alexandros Stamatakis helped prepare the manuscript.
Antonis Rokas and Leonidas Salichos originally defined the internode certainty in [115]. Salichos and Rokas also provided an adjusted data set for testing the internode certainty on partial gene trees for this chapter, and summarize the biological implications of the results. Stamatakis helped develop the adjustment methods and provided the framework for implementing the internode certainty calculation from partial gene trees using RAxML [130]. My contribution is the actual development of the methods for calculating the internode certainty from partial gene trees, as well as the implementation of the methods, and the evaluation on real life data sets.

6.1 Motivation and Related Work

Recently Salichos and Rokas [115] proposed a set of novel measures for quantifying the confidence for bipartitions in a phylogenetic tree. These measures are the so-called Internode Certainty (IC) and Tree Certainty (TC), which are calculated for a specific reference tree, given a collection of other trees with the exact same taxon set.
The calculation of their scores was implemented [116] in the phylogenetic software RAxML [130].
The underlying idea of Internode Certainty is to assess the degree of conflict

of each internal branch, connecting two internal nodes of a phylogenetic reference tree, by calculating Shannon's Measure of Entropy [123]. This score is evaluated for each bipartition in the reference tree independently. The basis for the calculations are the frequency of occurrence of this bipartition and the frequencies of occurrences of a set of conflicting bipartitions from the collection of trees. In contrast to classical bipartition confidence scores for the branches, such as simple bipartition support or posterior probabilities, the IC score also reflects to which degree the most favored bipartition is contested.

The reference tree itself can, for example, be constructed from this tree set, or can be a ML tree for a MSA. The tree collection may, for example, come from running multiple phylogenetic searches on the same data set, multiple bootstrap runs [38, 42], or running the analyses separately on different genes or different subsets of the genes (as done for example in [69]). While for the first two cases the assumption of having the same taxon set is reasonable, this is often not the case for different genes. Gene sequences may be available for different subsets of taxa, simply due to sequence availability or the absence of some genes in certain species.
In this chapter, we show how to compute an appropriately corrected internode certainty (IC) on collections of partial gene trees. When using partial bipartitions for calculating the IC and TC scores we need to solve two problems. First, we need to calculate their respective adjusted support (analogous to the frequency of occurrence) (Section 6.3.1). Unlike in the standard case, with full taxon sets, this information cannot be directly obtained. Then, we also need to identify all conflicting bipartitions (Section 6.3).

An alternative method for calculating these frequencies has recently been independently developed by *et al.* [125]. The method developed by Smith *et al.* is similar to what we denote as *l*ossless support (see page 75).

6.2 Definitions: Bipartitions, Internode Certainty, and Tree Certainty

Most concepts and notation that we will use throughout the chapter have been defined in the introductory chapter (see Section 3.5, page 24). In addition to the definitions there, we formally define internode certainty and tree certainty here. For this, we first need a notion of compatibility and conflict between bipartitions.

Definition 39 (Conflicting bipartitions)**.** *Two bipartitions* $B_1 = X_1|Y_1$ *and* $B_2 = X_2|Y_2$ *are* **conflicting/incompatible** *if there exists no single tree*

topology that explains/contains both bipartitions. Otherwise, if such a tree exists, they must be compatible. More formally, the bipartitions B_1 and B_2 are incompatible if and only if *all of the following properties hold (see for example [16]):*

$$\begin{aligned} & X_1 \cap X_2 && \neq \emptyset \\ \wedge\ & X_1 \cap Y_2 && \neq \emptyset \\ \wedge\ & Y_1 \cap X_2 && \neq \emptyset \\ \wedge\ & Y_1 \cap Y_2 && \neq \emptyset. \end{aligned}$$

This definition of conflict and compatibility is valid irrespective of whether the taxon sets of B_1 and B_2 are identical or not.

Definition 40 (Internode certainty)**.** *The* **Internode certainty** *(IC) score (as defined in [115]) is calculated using Shannon's measure of entropy [123]. For a branch b we define $IC(b)$ as follows:*

$$IC(b) = 1 + X_{B(b)} \cdot log_2(X_{B(b)}) + X_{B^\star} \cdot log_2(X_{B^\star}), \tag{21}$$

where $B(b)$ is the bipartition induced by removing branch b, and $B^\star$ is the bipartition from the tree collection that has the highest frequency of occurrence and is incompatible with $B(b)$. The terms denoted by X are the relative frequencies of the involved bipartitions. More formally, we define $X(B(b))$ as,

$$X_{B(b)} := \frac{f(B(b))}{f(B(b)) + f(B^\star)},\ X_{B^\star} := \frac{f(B^\star)}{f(B(b)) + f(B^\star)}, \tag{22}$$

where f simply denotes the frequency of occurrence of a bipartition in the tree set.

For the standard case of IC calculations (without partial gene trees), the frequency of occurrence f is simply the number of observed bipartitions in the tree set. In Section 6.3.1 we will show how to calculate the support (adjusted frequencies) for bipartitions from partial gene trees. We compute this support using the observed frequencies of occurrence. The support for partial bipartitions can then be used analogously to the frequency of occurrence in Equation 22 for calculating the IC scores.
Similarly to the IC score, Salichos *et al.* [116] also introduced the ICA (**internode certainty all**) value for each branch.

Definition 41 (Internode certainty all)**.**

$$ICA(b) = 1 + \sum_{B^c \in C(b)} X_{B^c} \cdot log_n(X_{B^c}), \tag{23}$$

where $C(b)$, as defined in [116]*, is $B(b)$ union with a set of bipartitions that conflict with $B(b)$ and with each other, while the sum of support for elements in $C(b)$ is maximized and n is defined as $n = |C(b)|$. Note that $C(b)$ has a slightly different definition in* [115].

Again, the terms denoted by X are the relative support of the bipartitions involved in Equation 23. That is,

$$X_{\hat{B}} = \frac{f(\hat{B})}{\sum\limits_{B^c \in C(b)} f(B^c)}$$

for all involved bipartitions $\hat{B} \in C(b)$.
The set $C(b)$ however is not easy to obtain. In fact, as we show in the following observation, maximizing the sum of supports for elements in $C(b)$ renders the search for an optimal choice of $C(b)$ NP-hard.

Observation 42. *Finding the optimal set $C(b)$ is NP-hard.*

This can easily be seen by considering the related, known to be NP-hard, maximum weight independent set problem [56]. Alternatively, the similarity to the problem of constructing the asymmetric median tree, which is also known to be NP-hard [108], can be observed.
For the maximum weight independent set problem, we are confronted with an undirected graph whose nodes have weights. The task is then to find a set of nodes that maximize the sum of weights, such that no two nodes in this set are connected via an edge. A reduction from this problem, to finding $C(b)$ is straight-forward. Let (W, E) be an undirected graph with weighted nodes W and edges E. Let $B(b) = xy|vz$. First, we introduce one bipartition $xz|vy$ for every node in W, with support equal to the node weight. Then, for every pair of bipartitions where the corresponding nodes in W do not share an edge in E, we add four taxa that are unique to those bipartitions, in such a way that they can never be compatible (consider $\ldots ab|cd \ldots$ and $\ldots ac|bd \ldots$). If we find $C(b)$ for the newly introduced bipartitions, the corresponding nodes yield a maximum weight independent set.
For this reason, the definition of the ICA, used and implemented in [116],

which we also use here, does not guarantee $C(b)$ to contain the set of conflicting bipartitions that maximize the sum of support. Instead $C(b)$ is constructed via a greedy addition strategy.

Additionally, Salichos and Rokas [115] advocate to use a threshold of 5% support frequency for conflicting bipartitions in $C(b)$. That is, $C(b)$ may only take elements $\hat{B}$ that have support

$$f(\hat{B}) \geq 0.05. \tag{24}$$

This is done to speed up calculations. Under this restriction, the problem of maximizing the support for $C(b)$ is no longer NP-hard. However, the search space is still large enough to warrant a greedy addition strategy, instead of searching for the best solution exhaustively.
Furthermore, if $B(b)$ does not have the largest frequency among all bipartitions in $C(b)$, the $IC(B)$ and $ICA(b)$ scores are multiplied with -1 to indicate this. This distinction is necessary since we may have $|ICA(\hat{b})| = |ICA(b)|$ for some $\hat{b} \in C(b)$. So an artificial negative value denotes that the bipartition in the reference tree is not only strongly contested, but not even the bipartition with the highest support. This can for example occur when the reference tree is the maximum-likelihood tree, and the tree set contains bootstrap replicates.

From the IC scores and ICA scores the respective Tree Certainties TC and TCA can be computed. These are defined as follows:

Definition 43 (Tree certainty)**.** *The TC (***tree certainty***) and TCA (***tree certainty all***) scores are simply the sum over all respective IC or ICA scores. That is,*

$$TC = \sum_{\substack{b \text{ internal branch} \\ \text{in reference tree}}} IC(b) \tag{25}$$

$$TCA = \sum_{\substack{b \text{ internal branch} \\ \text{in reference tree}}} ICA(b). \tag{26}$$

Furthermore, the *relative* TC and TCA scores are defined as the respective values normalized by the number of internal branches b, that is, branches for which $B(b)$ is a non-trivial bipartition.

As we can see, all we need to calculate the IC, TC, ICA, and TCA scores is to calculate $f(\hat{B})$ (Section 6.3.1) and $C(b)$ (Section 6.3.2).

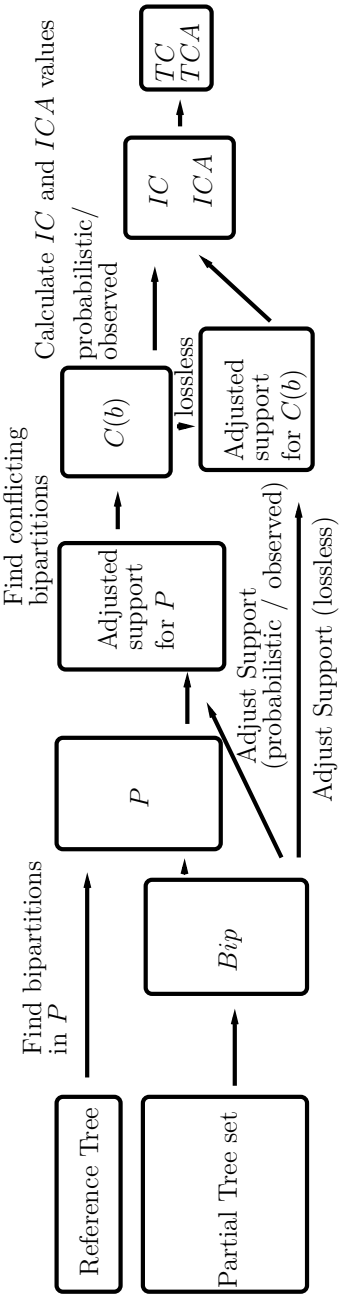


Figure 6.1: *Overview of the proposed methods.*

6.3 Adjusting the Internode Certainty

Now we must consider how to obtain the relevant information, that is the sets C and corrected support f, from partial bipartitions.
First, we formally define the input. We are given a so-called reference tree T with taxon set $S(T)$, node set $V(T) \supseteq S(T)$, and a set of branches $E(T) \subset V(T) \times V(T)$ connecting the nodes of $V(T)$. Let $\hat{E}(T) \subset E(T)$ be the set of internal branches. That is, for $b \in \hat{E}$ the bipartition $B(b)$ is non-trivial.
Additionally, we are given a collection of trees $\hat{T}$. From this collection we can easily extract the set of all non-trivial bipartitions Bip. The bipartitions in Bip are used to adjust the frequencies of other bipartitions. The taxon sets of the bipartitions in Bip are subsets of, or equal to, $S(T)$. We call a bipartition with fewer than $|S(T)|$ taxa a partial bipartition. A bipartition that includes all taxa from $S(T)$ is called comprehensive or full bipartition. From Bip and the bipartitions in the reference tree, we can construct a set of bipartitions P, for which we will adjust the score.

Figure 6.1 gives an overview of the steps explained in the following sections.

6.3.1 Correcting the Support

We aim to measure the support, the given set of partial trees $\hat{T}$ (or bipartition set Bip), induces for any of the bipartitions in P. We call this the **adjusted frequency** or **adjusted support**. If Bip and P only contain comprehensive bipartitions, the support for any given bipartition is simply equal to its frequency of occurrence.
In case of partial bipartitions, some thought must be given to this process. Imagine a comprehensive bipartition $B = X|Y$ in P, and a sub-bipartition D of B in Bip. Even though D does not exactly match B, it also does not contradict it. More so, it supports the super-bipartition, by agreeing on a common sub-topology.

We distinguish whether the observed sub-bipartition D from Bip is allowed to support any possible bipartition, even those not observed in Bip and P, or just those we observe in P. There seems to be no clear answer as to which of these assumptions is more realistic. The choice is thus merely a matter of definition or biological interpretation. If we allow the support to be divided among all possible partitions we assume that any bipartition is as likely to have occurred in reality as any other. If we distribute the

support only among observed bipartitions, we imply that we have observed the truth, and other bipartitions (those not observed) cannot have occurred.

Support of all possible bipartitions: Probabilistic Support If we assume that an observed sub-bipartition from Bip supports all possible super-bipartitions, not just those in P, with equal probability, the impact on the adjusted support of each such super-bipartition from P ($C(b)$) quickly becomes negligible. Consider the following example:
Let $B = X|Y \in P$, be a super-bipartition of $D = x|y \in Bip$ with $|X \setminus x| + |Y \setminus y| = k$. That is, B contains k taxa that D does not contain. There are 2^k distinct bipartitions with taxon set $X \cup Y$ that also contain the constraints set by D. For $k = 10$ we already obtain $2^{10} = 1024$. That is, the support of D will only increase (adjust) the support of B by less than one permille. More formally, let R_B be the set of sub-partitions in Bip of the comprehensive bipartition B in P and f_D the support for a partial bipartition D in Bip. Then the adjusted support for B, f_B is

$$f_B = \sum_{D \in R_B} \frac{f_D}{2^{(|S(T)| - n_D)}},$$

where n_D is the number of taxa D, and $|S(T)|$ the number of taxa in the reference tree. We use $|S(T)|$ in this formula, since any bipartition in P is implicitly a comprehensive bipartition. That is, even though we do not explicitly assign the remaining taxa from a partial bipartition $B = X|Y$ in P to X or Y, they must belong to one of these sets. The missing taxa in D thus have $\frac{1}{2}$ probability to belong to the same set (X or Y) each.

The effect of such an adjustment scheme is that partial bipartitions in Bip with fewer taxa affect the TC and IC scores substantially less than bipartitions with more taxa. This can also be observed in our computational results in Section 6.5. Since f_B is the sum over the observed frequency, times the probability of constructing the actual bipartition implied by B we call this the *probabilistic* adjustment scheme.

The motivation behind the probabilistic adjustment scheme is that a partial bipartition can stem from any full bipartition that complies with the constraints induced by this partial bipartition. Furthermore, a frequency $f > 1$ for a partial bipartition can emerge due to the existence of several different, implied full bipartitions. Consider the following example: let $B_1 = ABY|XCD$ and $B_2 = ABX|YCD$ be two bipartitions from two distinct gene trees. Now, assume that taxa X and Y are not present in these gene trees (e.g., due to incomplete species sampling). In this case, the respective

trees of these two gene trees only contain the same partial bipartition $B_p = AB|CD$.

By re-distributing the frequency of B_p via the probabilistic adjustment scheme to all possible bipartitions, we distribute the corresponding support among B_1 and B_2, as well as $B_3 = ABXY|CD$ and $B_4 = AB|XYCD$.

Support of observed bipartitions: Observed Support Now suppose that B_1 and B_2 are in P since they are present in some comprehensive, or partial gene trees. Further suppose, that the bipartitions B_3 and B_4 (as defined above) are not in P since they were never observed in the tree set. Due to missing data, other partial gene trees may produce bipartition B_p. In the above example for the probabilistic support, the support of B_p is not only distributed solely among B_1 and B_2, but also among B_3 and B_4, even though B_3 and B_4 were not observed in the tree set.

Thus, if we do not want to discard some of the frequency of occurrence when calculating the adjusted support from partial bipartitions, we can distribute their frequency of occurrence uniformly among comprehensive bipartitions in P. When we assume the prior distribution of bipartitions in P to be uniform, this process is simple. For a given partial bipartition D in Bip, with support f_D, let S_D be the set of bipartitions in P that are super-bipartitions of D. Then, D contributes $\frac{f_D}{|S_D|}$ support to any $B \in S_D$. In other words, the adjusted support for each full bipartition B is

$$f_B = \sum_{D \text{ s.t. } B \in S_D} \frac{f_D}{|S_D|}. \tag{27}$$

Since this distribution scheme splits the support for each sub-bipartition among bipartitions that we observed in the tree set only, we call this the *observed* support distribution scheme.

Support of conflicting bipartitions: Lossless Support One problem with the adjustment strategy explained above is that trees with more taxa typically have more bipartitions in P than trees with fewer taxa. For an intuitive understanding of why this can be problematic consider the example illustrated in Figure 2. Let bipartitions B_1 and B_2 come from the same tree. Further, let bipartition B_3 be the only, and exclusive, sub-bipartition of B_1 and B_2 in Bip. Similarly, let bipartition B_4 be the only super-bipartition of B_5. Let the sub-bipartitions B_3 and B_5 both have a frequency of occurrence of f and let B_1 and B_2 be conflicting with B_4. If we apply the above distribution scheme, bipartitions B_1 and B_2 have an adjusted frequency of

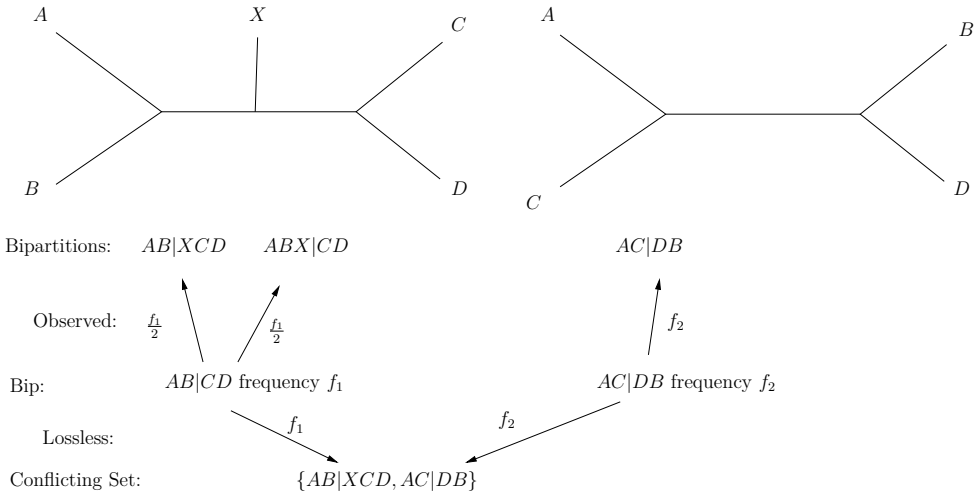


Figure 6.2: *Distribution of adjusted support for the observed and lossless adjustment scheme.*

$f/2$, while B_4 has an adjusted frequency of f. Penalizing bipartitions from trees with larger taxon sets however seems unwarranted. Thus, we propose a correction method that takes this into account. In order to circumvent this behavior we choose to distribute the frequency of any sub-bipartition only to a set of conflicting super-bipartitions (namely bipartitions in $C(b)$). That is:

$$f_B^b = \sum_{D \text{ s.t. } B \in S_D} \frac{f_D}{|S_D \cap C(b)|}. \tag{28}$$

Where S_D is defined as before. Note that, the adjusted support now depends on the set of conflicting bipartitions $C(b)$ which is defined by a branch b. This means that, the adjusted support for a given (conflicting) bipartition must be calculated separately for each reference bipartition $B(b)$.

This distribution scheme allocates the entire frequency of sub-bipartitions exclusively to these conflicting bipartitions. Thus, the sum of adjusted frequencies for all conflicting bipartitions is exactly equal to the sum of frequencies of occurrence of the found sub-bipartitions. For this reason we call this the *lossless* adjustment scheme.

Note that, $C(b)$ is obtained via a greedy addition strategy, depending on the adjusted support of bipartitions. Since the adjusted support according to the lossless adjustment scheme depends on $C(b)$ we obtain a recursive

definition. To alleviate this, we simply precompute the above explained probabilistic adjustment scheme to obtain an adjusted support for each bipartition. The set of conflicting bipartitions $C(b)$ is then found with respect to the probabilistically adjusted support values. Then, using $C(b)$, the actual lossless support adjustment is calculated and replaces the probabilistic support in the calculation of IC and ICA values.

For the above example we get the following. Let $\{B_1, B_4\}$ be the set of conflicting bipartitions. Then, the support for B_1 and B_4 after applying the lossless distribution scheme is f for both bipartitions, which is the desired behavior for this distribution scheme.

6.3.2 Finding Conflicting Bipartitions

From Bip we construct a set of maximal bipartitions P. That is, bipartitions that are not themselves sub-bipartitions of any other bipartition in Bip. Once we have constructed P, we can calculate the internode certainty $IC(b)$ as before. The construction of P is trivial. The set P simply contains all bipartitions that are not themselves strict sub-bipartitions of other bipartitions in Bip. We do this step, since any information contained in a sub-bipartition is also contained in the super-bipartition. That is, the implied gene tree (or species tree) for the super-bipartition can also explain the gene tree for all taxa in the sub-bipartition. How the frequency of occurrence of the sub-bipartition affects the frequency of occurrence of the super-bipartition has been explained in Section 6.3.1.

We implicitly assume that each bipartition in P should actually contain all taxa from $S(T)$. To achieve this, we keep the placement of the missing taxa ambiguous. That is, we assume that, each missing taxon has a uniform probability to fall into either side of the bipartition.

To construct $C(b)$ greedily as proposed above, the support of the bipartitions must be known. However, the lossless support adjustment scheme explained above is only reasonable on a set of conflicting bipartitions (that is, $C(b)$ itself). To avoid this recursive dependency, we first compute an adjusted support that does not depend on $C(b)$ for this case. (Here we use the probabilistic adjusted support, as explained in Section 6.3.1, to obtain an initial adjusted support.) Then, a greedy algorithm is used to approximate the set $C(b)$ with the highest sum of adjusted support, with respect to the initial adjustment. Once $C(b)$ is obtained, the support for all bipartitions in $C(b)$ is adjusted using the new method, which depends on a set of conflicting bipartitions. These new values then replace the initial estimate via the first adjustment scheme.

Keeping the above in mind, we can easily construct $C(b)$ from P for every branch b in $\hat{E}(T)$. Note that, we also defined the reference bipartition $B(b)$ to be in $C(b)$. Thus, we simply start with $B(b)$ and iterate through the elements of P in decreasing order of adjusted support (that is, the probabilistic adjusted support if we are to apply the probabilistic or lossless distribution scheme, and the observed adjusted support if this distribution is desired) and add every bipartition that conflicts with all other bipartitions added to $C(b)$ so far. During this process the threshold given in Equation 24 is applied.

Given $B(b)$, $C(b)$, and Bip we can calculate the IC and ICA values as defined in Equations 21 and 23 under the *probabilistic* or *observed* adjustment schemes. For the *lossless* adjustment scheme, the actual adjusted frequencies have to be calculated separately for each bipartition in $C(b)$ for all reference bipartitions b in this step.

6.4 Example

We now present a simple example for calculating the IC score under the different adjustment schemes. To this end, we analyze the tree set shown in Figure 6.3. From these trees we initially extract the following bipartition lists:

$$\begin{aligned} Bip = \{&AB|CDEF,\ ABE|CDF,\ ABED|CF,\\ &AB|CD,\ AC|BEF,\ ACB|EF,\ AC|FBE,\\ &ACF|BE\} \end{aligned}$$

$$\begin{aligned} P =&\{AB|CDEF,\ ABCD|EF,\ ABEF|CD,\\ &ABE|CDF,\ ABED|CF,\ AC|BEF,\\ &ACF|BE\}\\ =&\{R_1,\ R_2,\ R_3, B_2,\ B_3,\ B_5,\ B_8\}. \end{aligned}$$

We can now immediately calculate the probabilistic and observed support for bipartitions in P. As mentioned before, the lossless adjustment can only be calculated on sets of conflicting bipartitions. Let f_B^p and f_B^o be the probabilistic and observed support of a bipartition B. Further, let $f_B := (f_B^p, f_B^o)$.

Then, as B_1 in the Figure is exactly identical to R_1, and B_4 is a sub-bipartition of R_1 with 2 missing taxa, $f_{R_1}^p = f_1 + \frac{1}{4} f_2$. At the same time, R_1 is the only super-bipartition of B_1. However, two other bipartitions, namely

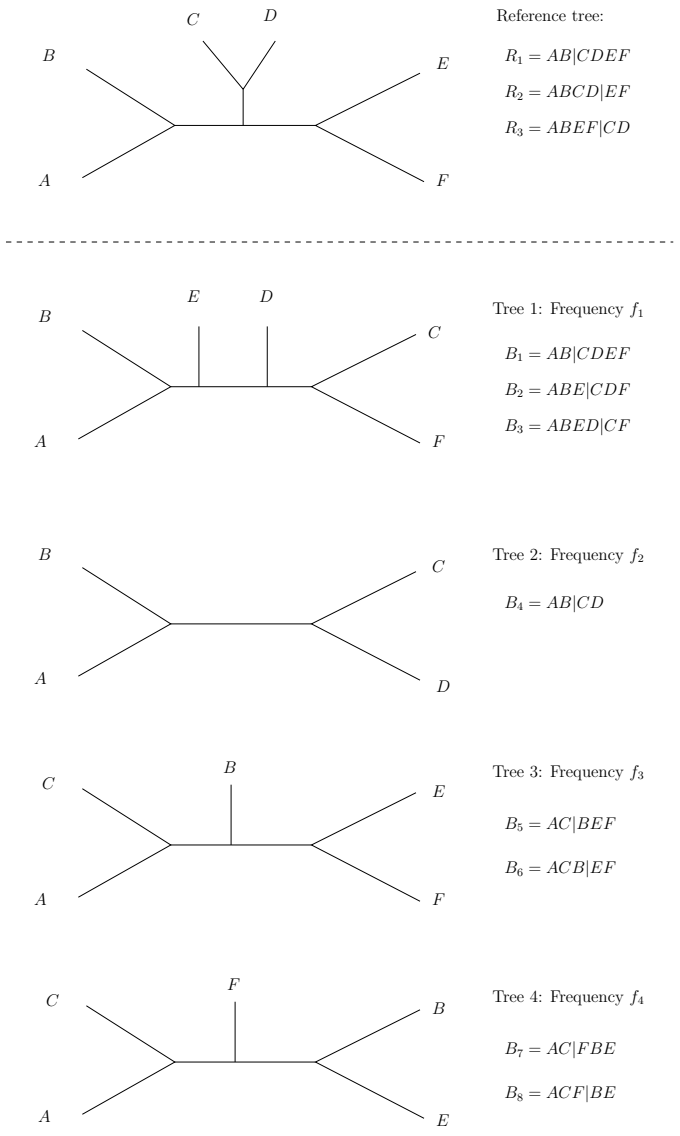


Figure 6.3: *Example tree set for IC calculations.*

R_3 and B_2, are super-bipartitions of B_4. Thus, we obtain $f^o_{R_1} = f_1 + \frac{1}{3}f_2$. All other bipartitions in P can be scored analogously to obtain the following

probabilistic and observed support value pairs:

$$
\begin{aligned}
f_{R_1} =& (f_1 + \frac{1}{4}f_2,\ f_1 + \frac{1}{3}f_2) \\
f_{R_2} =& (\frac{1}{2}f_3,\ f_3) \\
f_{R_3} =& (\frac{1}{4}f_2,\ \frac{1}{3}f_2) \\
f_{B_2} =& (f_1 + \frac{1}{4}f_2,\ f_1 + \frac{1}{3}f_2) \\
f_{B_3} =& (f_1,\ f_1) \\
f_{B_5} =& (\frac{1}{2}f_3 + \frac{1}{2}f_4,\ f_3 + f_4) \\
f_{B_8} =& (\frac{1}{2}f_4,\ f_4).
\end{aligned}
$$

Given the above, we can now calculate the IC scores for bipartitions R_1, R_2, and R_3. Assume that we have the following frequencies, $f_1 = 3$, $f_2 = 4$, $f_3 = 6$, and $f_4 = 6$. Bipartition $R_1 = AB|CDEF$ conflicts with both, $B_5 = AC|BEF$, and $B_8 = ACF|BE$. However, since B_5 and B_8 do not conflict with each other, only one of them is included in the list of conflicting bipartitions. Since B_5 has a higher adjusted support than B_8, we include B_5. If b is the branch that gives rise to bipartition R_1 in the reference tree, then $C(b) = \{R_1, B_5\}$. Under the probabilistic adjustment scheme we obtain:

$$
\begin{aligned}
-IC(b) = \quad & 1 + \frac{f_1 + \frac{1}{4}f_2}{(f_1 + \frac{1}{4}f_2) + (\frac{1}{2}f_3 + \frac{1}{2}f_4)} log_2(\frac{f_1 + \frac{1}{4}f_2}{(f_1 + \frac{1}{4}f_2) + (\frac{1}{2}f_3 + \frac{1}{2}f_4)}) \\
& + \frac{\frac{1}{2}f_3 + \frac{1}{2}f_4}{(f_1 + \frac{1}{4}f_2) + (\frac{1}{2}f_3 + \frac{1}{2}f_4)} log_2(\frac{\frac{1}{2}f_3 + \frac{1}{2}f_4}{(f_1 + \frac{1}{4}f_2) + (\frac{1}{2}f_3 + \frac{1}{2}f_4)}) \\
= \quad & 1 + \frac{3 + \frac{1}{4}4}{(3 + \frac{1}{4}4) + 3 + 3} log_2(\frac{3 + \frac{1}{4}4}{(3 + \frac{1}{4}4) + 3 + 3}) \\
& + \frac{6}{(3 + \frac{1}{4}4) + 6} + log_2(\frac{6}{(3 + \frac{1}{4}4) + 6}) \\
\approx \quad & 0.0290
\end{aligned}
$$

The negative value of $IC(b)$ is due to the fact that, under the observed adjustment scheme, B_5 has a higher adjusted support than R_1. Similarly,

under the observed adjustment scheme we obtain:

$$\begin{aligned}
-IC(b) = \quad & 1+ \quad \frac{f_1+\frac{1}{3}f_2}{(f_1+\frac{1}{3}f_2)+(f_3+f_4)} log_2(\frac{f_1+\frac{1}{3}f_2}{(f_1+\frac{1}{3}f_2)+(f_3+f_4)}) \\
& + \quad \frac{(f_3+f_4)}{(f_1+\frac{1}{3}f_2)+(f_3+f_4)} log_2(\frac{(f_3+f_4)}{(f_1+\frac{1}{3}f_2)+(f_3+f_4)}) \\
= \quad & 1+ \quad \frac{3+\frac{1}{3}4}{(3+\frac{1}{3}4)+6+6} log_2(\frac{3+\frac{1}{3}4}{(3+\frac{1}{3}4)+6+6}) \\
& + \quad \frac{6+6}{(3+\frac{1}{3}4)+6+6} + log_2(\frac{6+6}{(3+\frac{1}{3}4)+6+6}) \\
\approx \quad & 0.1653 \qquad .
\end{aligned}$$

Given $C(b)$, we can now also compute the lossless adjusted support. We obtain a support of $f_1 + f_2 = 7$ for R_1, and a support of $f_3 + f_4 = 6 + 6$ for B_5. With these numbers at hand, we can calculate the IC score under lossless adjustment as:

$$-IC(b) = 1 + \frac{7}{7+12} log_2(\frac{7}{7+12}) + \frac{12}{7+12} log_2(\frac{12}{7+12}) \approx \quad 0.0505.$$

This can be done analogously for bipartitions R_2 and R_3. For $R_2 = ABCD|EF$ we observe three conflicting bipartitions: $B_2 = ABE|DCF$, $B_3 = ABED|CF$, and $B_8 = ACF|BE$. The corresponding frequencies for the above bipartitions are:

$$\begin{aligned}
f_{R_2} &= (\frac{1}{2}f_3,\ f_3) && = (3,\ 6) \\
f_{B_2} &= (f_1 + \frac{1}{4}f_2,\ f_1 + \frac{1}{3}f_2) && = (4,\ 4 + \frac{1}{3}) \\
f_{B_3} &= (f_1,\ f_1) && = (3,\ 3) \\
f_{B_8} &= (\frac{1}{4}f_4,\ f_4) && = (1 + \frac{1}{2},\ 6).
\end{aligned}$$

Under the probabilistic support, we thus obtain $C(b) = \{R_2,\ B_2\}$, where b is the branch that corresponds to the reference bipartition with $R_2 = B(b)$. However, the set of conflicting bipartitions is different for the observed adjustment scheme. Here, $C(b) = \{R_2,\ B_8\}$. As a consequence we obtain the following IC scores:

$$-IC(b) = 1 + \frac{3}{3+4} log_2(\frac{3}{3+4}) + \frac{4}{3+4} log_2(\frac{4}{3+4}) \approx \quad 0.0148$$

under the probabilistic scheme, and

$$IC(b) = 1 + \frac{6}{6+6} log_2(\frac{6}{6+6}) + \frac{6}{6+6} log_2(\frac{6}{6+6}) = \qquad 0$$

under the observed adjustment scheme. The adjusted frequencies for bipartitions R_2 and B_2, under the lossless adjustment scheme, are $f_3 = 6$ and $f_1 + f_2 = 7$, respectively. Thus, the IC score is

$$-IC(b) = 1 + \frac{6}{6+7} log_2(\frac{6}{6+7}) + \frac{7}{6+7} log_2(\frac{7}{6+7}) \approx \qquad 0.0043.$$

For reference bipartition $R_3 = ABEF|CD$, there is only one conflicting bipartition in P, namely $B_3 = ABED|CF$. Thus, the calculation of $IC(b)$ is straight-forward (as before b is the branch inducing the reference bipartition: R_3). Under the probabilistic scheme we obtain:

$$-IC(b) = 1 + \frac{1}{1+3} log_2(\frac{1}{1+3}) + \frac{3}{1+3} log_2(\frac{3}{1+3}) \approx \qquad 0.1887.$$

Under the observed adjustment we get:

$$-IC(b) = 1 + \frac{\frac{4}{3}}{\frac{4}{3}+3} log_2(\frac{\frac{4}{3}}{\frac{4}{3}+3}) + \frac{3}{\frac{4}{3}+3} log_2(\frac{3}{\frac{4}{3}+3}) \approx \qquad 0.1095.$$

Finally, under the lossless adjustment scheme we obtain:

$$IC(b) = 1 + \frac{4}{4+3} log_2(\frac{4}{4+3}) + \frac{3}{4+3} log_2(\frac{3}{4+3}) \approx \qquad 0.0148.$$

6.5 Results and Discussion

For implementing the methods described in Section 6.3, we used the framework of the RAxML [130] software (version 8.1.20).

The resulting proof of concept implementations, and all data sets used for our experiments in Sections 6.5.1 and 6.5.2 (as well as the above example of Section 6.4) are available at `https://github.com/Kobert/ICTC`. Usage of the software is explained there as well. The probabilistic and lossless distribution schemes are also included in the latest production level version of RAxML (`https://github.com/stamatak/standard-RAxML`, version 8.2.4). We chose to omit the implementation for the observed support adjustment from the official RAxML release, as it does not seem to offer any advantages over the other two methods.

6.5.1 Accuracy of the Methods

In this section we asses the accuracy of the proposed adjustment schemes. For this reason, we re-analyze the yeast data set originally presented in [115]. The comprehensive trees in the data set contain 23 taxa. After applying some filtering techniques to the genes, we obtained a set of 1275 gene trees. In the filtering step, genes are discarded, if (i) the average sequence length is less than 150 characters or (ii) more than half the sites contain indels after alignment. In [115], a slightly smaller subset of 1070 trees is used.

To understand which adjustment scheme better recovers the underlying truth, we randomly prune taxa from this comprehensive tree set and compare the results between adjustment schemes. Evidently, a "good" adjustment scheme will yield IC and ICA values that are as similar as possible to the IC/ICA values of the comprehensive tree set. Thus we consider the IC/ICA on the comprehensive tree set as the correct values.

For each of the 1275 trees, we select and prune a random number of taxa. We draw the numbers of taxa to prune per tree from a geometric distribution with parameter p. We use a geometric distribution because the expectation is that thereby we will retain $p \cdot 1275$ comprehensive trees, for which 0 taxa have been pruned. An additional restriction is that each pruned tree must comprise at least 4 taxa to comprise at least one non-trivial bipartition. Given the number of taxa we wish to prune, we select taxa to prune uniformly at random using the *n*ewick-tools toolkit[2].

[2] https://github.com/xflouris/newick-tools

	IC				ICA			
	$p = 0.1$	$p = 0.3$	$p = 0.5$	$p = 0.7$	$p = 0.1$	$p = 0.3$	$p = 0.5$	$p = 0.7$
Probabilistic	0.31	0.20	0.18	0.08	0.26	0.18	0.18	0.12
Observed	0.42	0.27	0.15	0.07	0.39	0.25	0.19	0.08
Lossless	0.65	0.44	0.24	0.17	0.60	0.44	0.28	0.15

Table 6.1: *Differences D in IC/ICA scores, between the scores calculated by the adjustment schemes and the reference scores for the comprehensive tree set.*

Using different values for p we generate four partial tree sets. For each of these tree sets we conduct analyses including all 1275 trees (comprehensive and partial). We compare the results to the IC/ICA scores for 1275 comprehensive trees.

Similarly, in a second round of experiments we compare the results obtained by removing *all* comprehensive trees from the tree sets, to the reference IC and ICA scores for the comprehensive tree set.

To quantify, which correction method yields more accurate results, we define the following distance/accuracy measure. Let $IC(b)$ be the inter node certainty for branch b if no taxa are pruned. Similarly, let $IC^A(b)$ be the internode certainty for the same branch b under an adjustment scheme for a data set *with* partial gene trees. The accuracy D of an adjustment scheme is then defined as:

$$D = \frac{1}{N} \sum_{\substack{\text{b internal branch} \\ \text{in reference tree}}} \frac{||IC(b)| - |IC^A(b)||}{max\{|IC(b)|, |IC^A(b)|\}}, \tag{29}$$

where N is the number of internal branches in the reference tree ($N = 20$ for our test data set). The measure D is the average, weighted, component-wise difference between the two results. A low value of D indicates high similarity between the results. Furthermore, by definition, D ranges between 0 and 1.

Table 6.1 depicts this distance D for the different tree sets and adjustment schemes we tested. As we can see, both, the probabilistic, and observed adjustment methods are more accurate than the lossless method.

In Table 6.2 we observe that the probabilistic and observed adjustment schemes are not more accurate than the lossless method for tree sets that only contain partial gene trees. From Table 6.3 it also becomes evident that the lossless adjustment scheme tends to overestimate the IC and ICA values less frequently than the two alternative methods.

	IC				ICA			
	$p = 0.1$	$p = 0.3$	$p = 0.5$	$p = 0.7$	$p = 0.1$	$p = 0.3$	$p = 0.5$	$p = 0.7$
Probabilistic	0.50	0.52	0.53	0.53	0.47	0.48	0.50	0.50
Observed	0.50	0.51	0.53	0.53	0.45	0.48	0.50	0.49
Lossless	0.61	0.48	0.50	0.52	0.46	0.43	0.47	0.49

Table 6.2: *Differences D in IC/ICA scores, between the pruned tree sets* only *containing partial gene trees and the reference values.*

	IC				ICA			
All trees	$p = 0.1$	$p = 0.3$	$p = 0.5$	$p = 0.7$	$p = 0.1$	$p = 0.3$	$p = 0.5$	$p = 0.7$
Probabilistic	0.4	0.35	0.35	0.15	0.25	0.25	0.2	0.15
Observed	0.15	0.3	0.4	0.2	0.2	0.2	0.2	0.1
Lossless	0.1	0.25	0.15	0.25	0.2	0.2	0.25	0.1
Partial trees	$p = 0.1$	$p = 0.3$	$p = 0.5$	$p = 0.7$	$p = 0.1$	$p = 0.3$	$p = 0.5$	$p = 0.7$
Probabilistic	0.8	0.8	0.85	0.85	0.8	0.8	0.85	0.85
Observed	0.65	0.75	0.8	0.85	0.65	0.75	0.8	0.85
Lossless	0.3	0.65	0.75	0.8	0.25	0.65	0.75	0.8

Table 6.3: *Fraction of branches for which the adjusted IC/ICA scores are higher than the IC/ICA reference scores. The top table contains values for all three adjustment schemes if all trees (comprehensive and simulated partial) are included in the analysis. The bottom table shows the values for all three methods if* only *partial trees are analyzed.*

Another important observation is that, in most cases, accuracy decreases for any adjustment scheme when analyzing tree sets that exclusively contain partial gene trees. Intuitively, this can be explained by the fact that (i) we have less trees to base our analysis on, and (ii) only the reference bipartitions now contain all 23 taxa. Since a partial bipartition distributes its frequency among all its super-bipartitions in P, it is intuitively clear that, bipartitions with more taxa are more likely to accumulate distributed frequencies from more sub-bipartitions than bipartitions with fewer taxa. Conflicting bipartitions (with less than 23 taxa) are thus not assigned sufficient support to compete with the reference bipartitions. This behavior can be observed in Table 6.3. There, we display the numbers of times the certainty in a branch under the different adjustment schemes was higher than the certainty obtained from the comprehensive trees.

6.5.2 Empirical Data Analyses

In this section we present an additional, yet different, analysis of the above yeast data set. We do not only use the 1275 comprehensive trees, but now also include additional partial gene trees. After applying the aforementioned filters again (6.5.1), the tree set set comprises 2494 trees. The comprehensive trees are the same 1275 trees as in Section 6.5.1. The remaining 1219 trees are partial trees. The number of taxa in these partial trees ranges from 4 to 22 (see Figure 6.4 for the exact distribution of taxon numbers over partial gene trees). Unlike in Section 6.5.1, these partial trees are not simulated, but the result of phylogenetic analyses on the corresponding gene alignments.

In addition, we also analyze a gene tree set from avian genomes. The data was previously published in [76]. Here, we analyze a subset of 2000 gene trees with up to 48 taxa. Of these trees, 500 contain the full 48 taxa while the remaining trees contain either 47 taxa (500 trees) or 41-43 taxa (1000 trees). The taxon number distribution over trees is provided in Figure 6.4.

First, we report the results for the yeast data set. We present the IC and ICA scores for all internal branches under the three adjustment schemes and compare them to the scores obtained for the subset of comprehensive trees. Figure 6.5 shows the topology of the reference tree. Tables 6.4 and Table 6.5 show the respective IC and ICA values.

Taxa	Adjustment	Bip. 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
23 Taxa	None	95	29	9	3	48	27	5	95	2	14	1	56	94	75	71	71	7	1	<1	99
4-23 Taxa	Probabilistic	89	28	8	3	46	28	6	91	2	15	1	52	92	72	65	70	7	2	<1	92
4-23 Taxa	Observed	89	12	12	3	52	24	4	58	1	14	2	36	91	69	64	69	7	2	1	57
4-23 Taxa	Lossless	82	2	15	2	39	26	5	41	<1	10	3	15	89	61	56	65	7	1	<1	68

Table 6.4: *IC scores for all non-trivial bipartitions multiplied by* 100 *and rounded down. The bipartition labels are shown in Figure 6.5. The data set can either consist of only full trees (*23 *taxa), or partial and full trees (*4-23 *taxa).*

Taxa	Adjustment	Bip. 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
23 Taxa	None	95	23	7	8	48	25	14	95	3	12	2	45	94	75	71	71	7	8	9	98
4-23 Taxa	Probabilistic	89	21	6	13	46	26	14	91	3	11	1	38	92	72	60	70	25	7	11	92
4-23 Taxa	Observed	89	15	9	12	52	24	12	58	2	11	11	34	91	69	59	69	24	7	11	57
4-23 Taxa	Lossless	82	13	10	7	39	27	13	46	3	9	8	29	89	61	49	65	7	5	5	68

Table 6.5: *ICA scores for all non-trivial bipartitions multiplied by* 100 *and rounded down. The bipartition labels are shown in Figure 6.5. The data sets again either consist of only full trees (*23 *taxa), or partial and full trees (*4-23 *taxa).*

The values for the individual IC and ICA scores *can* be higher for the lossless adjustment scheme than for the probabilistic adjustment scheme

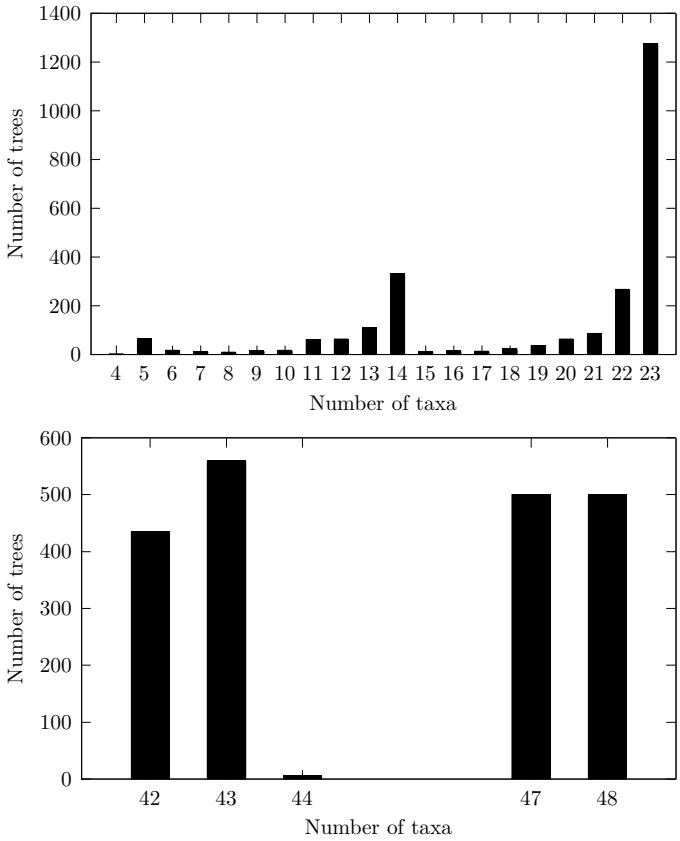


Figure 6.4: *Distribution of taxon number over trees in the yeast data set (top) and the avian data set (bottom).*

and the observed adjustment scheme. However, the relative TC and TCA values suggest, that the lossless adjustment attributes a lower certainty to individual bipartitions as well as the entire tree. The actual values are 0.298 for the relative TC score and 0.322 for the relative TCA score for the lossless adjustment; 0.389 and 0.399 for the probabilistic adjustment; and 0.339 and 0.364 for the observed adjustment scheme.

By comparing the 23-taxa yeast species tree values without adjustment against the three approaches that contain both complete and missing data (probabilistic, observed and lossless), we can conclude that, overall, the values appear very similar and they tend to provide additional support for the reference topology. Among the adjustment strategies, the probabilistic adjustment yields values that are closest to those obtained by the analysis

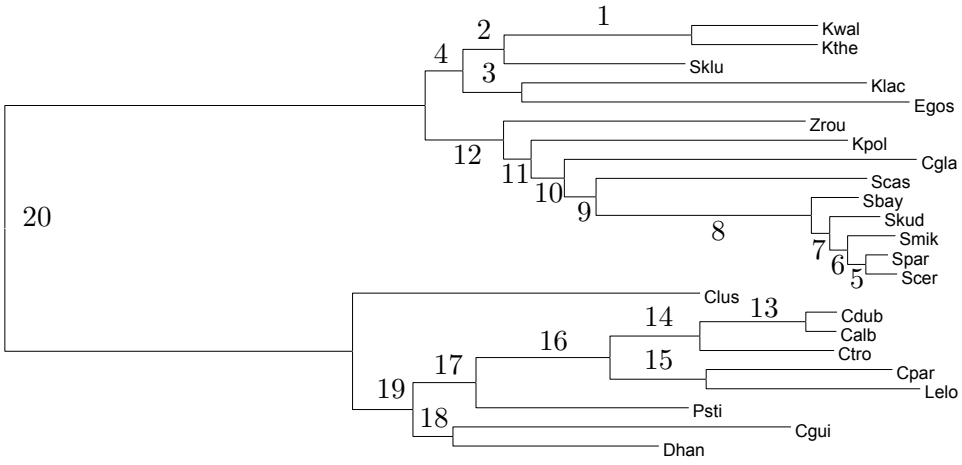


Figure 6.5: *Bipartition numbers corresponding to the presented tables, for the yeast data set.*
Taxon key: Kwal: Kluyveromyces waltii, Kthe: Kluyveromyces thermotolerans, Sklu: Saccharomyces kluyveri, Klac: Kluyveromyces lactis, Egos: Eremothecium gossypii, Zrou: Zygosacharomyces rouxii, Kpol: Kluyveromyces polysporus, Cgla: Candida glabrata, Scas: Saccharomyces castellii, Sbay: Saccharomyces bayanus, Skud: Saccharomyces kudriavzevii, Smik: Saccharomyces mikatae, Spar: Saccharomyces paradoxus, Scer: Saccharomyces cerevisiae, Clus: Candida lusitaniae, Cdub: Candida dubliniensis, Calb: Candida albicans, Ctro: Candida tropicalis, Cpar: Candida parapsilosis, Lelo: Lodderomyces elongisporus, Psti: Pichia stipitis, Cgui: Candida guilliermondii, Dhan: Debaryomyces hansenii

of only comprehensive trees. This is expected, since for the probabilistic adjustment, smaller bipartitions contribute less to the overall scores than larger bipartitions. Full bipartitions/trees are thus affecting the outcome most under this adjustment scheme.

Previous ambiguous bipartitions, concerning for example the placement of species like *S. castelii* (conf. bipartitions 9 and 8), *C. lusitaniae* (conf. bipartitions 20 and 19), *D. hansenii* (bipartition 18), and *K. lactis* (bipartition 3), remain equally uncertain, showing very similar (close to 0) IC and ICA values.

The split between the *Candida* and *Saccharomyces* clade (bipartition 20) is well documented in the literature [34, 48, 115]. The same holds for bipartition 8, the *Saccharomyces sensu stricto* clade [89, 113, 115]. Thus, a high certainty for these bipartitions is expected. As we can see in Table 6.4, the analysis of only comprehensive trees supports these two biparti-

tions with IC values of 0.99 for bipartition 20, and 0.95 for bipartition 8. However, the generally conservative lossless distribution approach, as well as the observed support adjustment scheme, provide reduced certainty for these two bipartitions; the divergence of *Candida* from the *Saccharomyces* clade (bipartition 20) is, for the lossless distribution scheme, depicted with an IC value of 0.68, and the *Saccharomyces sensu stricto* clade (bipartition 8) obtains an IC score of 0.41; the observed adjusted support for these bipartitions is reduced to 0.57 for bipartition 20, and 0.58 for bipartition 8. The probabilistic adjusted IC values for the branches inducing these splits are 0.92 for bipartition 20, and 0.91 for bipartition 8. A similar behavior can be seen for the ICA values.

In addition, under the lossless adjustment, the previously resolved placement of *Z. rouxii* (a clade with relatively low gene support frequency of 62% in [115]) remains unresolved with IC and ICA values of 0.15 and 0.29 respectively.

Next, we analyze the behavior of the adjustment schemes if *only* partial trees are provided. See Tables 6.6 and 6.7.

Taxa	Adjustment	Bip. 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
23 Taxa	None	95	29	9	3	48	27	5	95	2	14	1	56	94	75	71	71	7	1	<1	99
4-22 Taxa	Probabilistic	93	64	61	58	72	66	59	85	39	46	43	64	95	77	83	78	56	49	47	93
4-22 Taxa	Observed	89	23	58	36	80	75	70	80	1	1	<1	20	93	79	82	78	54	13	16	43
4-22 Taxa	Lossless	80	24	58	12	66	57	32	68	24	12	12	2	88	54	42	49	43	12	38	7

Table 6.6: *IC scores for all non-trivial bipartitions multiplied by* 100 *and rounded down. The bipartition labels are shown in Figure 6.5. Here, the data set only contains trees with partial taxon sets.*

Taxa	Adjustment	Bip. 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
23 Taxa	None	95	23	7	8	48	25	14	95	3	12	2	45	94	75	71	71	7	8	9	98
4-22 Taxa	Probabilistic	93	64	54	51	72	66	59	85	40	46	34	58	95	77	83	78	56	43	45	93
4-22 Taxa	Observed	89	23	48	33	80	75	70	80	17	20	18	20	93	79	82	78	54	29	24	43
4-22 Taxa	Lossless	80	27	58	24	66	57	29	68	24	11	12	2	88	54	42	49	43	12	38	22

Table 6.7: *ICA scores for all non-trivial bipartitions multiplied by* 100 *and rounded down. The bipartition labels are shown in Figure 6.5. Again, the data set only contains trees with partial taxon sets.*

The relative TC (and TCA) that result from these calculations are 0.668 (0.651) for the probabilistic distribution, 0.499 (0.532) for the observed distribution, and 0.394 (0.407) for the lossless distribution scheme. The relative TC and TCA without correction (obtained from the values shown in Tables 6.4 and 6.5), for trees with full taxon sets, are 0.406 and 0.409. The higher

TC and TCA values obtained for the former two adjustment methods suggest that these approaches are not providing the conflicting bipartitions with a sufficiently adjusted support, to compare to the reference bipartition. The reference bipartitions always contain 23 taxa for this data set. Now however, no conflicting bipartition can have that many taxa, as comprehensive trees are not included in the above analysis of only partial trees.

Analyzing the second data set with a total of 2000 trees yields similar results. See Table 6.8 for the TC and TCA values for this data set. Again, the values of the analysis restricted to a comprehensive tree set are compared to the results obtained when including partial gene trees, and restricting the analysis to partial gene trees. Specifically, we see that, the probabilis-

Taxa	adjustment	TC	TCA
48 taxa	None	-3.14	-3.17
41-48 Taxa	Probabilistic	-2.44	7.72
41-48 Taxa	Lossless	-5.05	-1.35
41-47 Taxa	Probabilistic	9.34	15.75
41-47 Taxa	Lossless	6.01	6.01

Table 6.8: *IC and ICA scores for different subsets of the data set for the probabilistic and lossless distribution schemes.*

tic support for analyzing the full data set, of 2000 trees, again gives TC values more closely in accordance with the values obtained for the analysis restricted to the 500 full trees, than the lossless adjustment scheme.

Here, the tree set does not support the reference tree well (as evident by the negative TC). At the same time, the TCA under the probabilistic adjustment scheme is actually positive.

For this data set, the discrepancy can be explained by the fact that the most frequent conflicting bipartitions not supported by much more than the second most supported conflicting bipartition. If the support for the reference bipartition is much smaller than that of the most frequent conflicting bipartition, the internode-certainty will approach -1. Let the support for the most frequent conflicting bipartition be f. As the support of the second most frequent conflicting bipartition approaches f, the ICA value tends towards 0.0. If the reference bipartition is the bipartition with the highest adjusted support in $C(b)$, this effect is less pronounced.

For the analysis of partial bipartitions only, we again see that the conflicting bipartitions are not as well supported under any tested adjustment

scheme. Again, the lossless adjustment scheme yields decreased certainty. Thus, we advocate that this adjustment scheme is used if one wants to reduce the risk of overestimating certainties.

6.6 Conclusion

We have seen that, the inclusion of partial trees into any certainty estimation is beneficial, as the partial trees do contain information that is not necessarily contained in the full/comprehensive trees. This is evident by the different TC and TCA scores we obtained for the empirical data sets.

Further, the selection of the most appropriate adjustment scheme depends on the data at hand. The lossless adjustment scheme is most appropriate, for tree sets that do not contain any comprehensive trees, since it yields more conservative certainty estimates. For gene tree sets that contain both, comprehensive, as well as partial trees, the probabilistic and observed adjustment schemes yield results that are more accurate with respect to the reference IC and ICA values.

In general, we advocate the inclusion of (some) comprehensive trees in any analysis that also includes partial trees. This is motivated by the fact that the pruned data sets that contained comprehensive trees generally yielded more accurate results than tree sets not containing comprehensive trees.

Part II:

Detecting Repetitive Patterns in Trees and Strings

7 Calculating Subtree Repeats on General Trees

Given a labeled tree T, as defined in Section 3.3 (page 18), our goal is to group repeating subtrees of T into equivalence classes with respect to their topologies and the node labels. We present an explicit, simple, and time-optimal algorithm for solving this problem for unrooted unordered labeled trees, and show that the running time of our method is linear with respect to the size of T. Unordered means, that the order of the adjacent nodes (children/neighbors) of any node of T is irrelevant. An unrooted tree T does not have a node that is designated as root and can also be referred to as an undirected tree. Further we show how the presented algorithm can easily be modified to operate on trees that do not satisfy some or any of the aforementioned assumptions on the tree structure; for instance, how it can be applied to rooted, ordered or unlabeled trees.

We sequentially published three papers on this topic. First, in [54], we solved this problem for rooted trees only. The solution for general trees (rooted and unrooted) was presented in [52]. An extended version of [52] appeared in the "*Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*", in 2014, under the title of "An optimal algorithm for computing all subtree repeats in trees" [51]. The following chapter is based on this publication. Besides me, Tomáš Flouri, Solon Pissis, and Alexandros Stamatakis (co-) authored and wrote these three papers. Solon Pissis and Tomáš Flouri first formulated the problem for rooted trees. Together, Flouri, and I devised the forward stage of the algorithm (for rooted trees). Flouri proved the linear runtime, while I provided correctness proofs. I, with the help of Tomáš Flouri, contributed the backwards stage (for unrooted trees) and the related proofs and observations.

7.1 Motivation and Related Work

Tree data structures are among the most common and well-studied of all combinatorial structures. They are present in a wide range of applications, such as, in the implementation of functional programming languages [74], term-rewriting systems [72], programming environments [10], code optimization in compiler design [5], code selection [46], theorem proving [80], and computational biology [98]. Thus, efficiently extracting the repeating patterns in a tree structure, represents an important computational problem.

Recently, Christou *et al.* [25] presented a linear-time algorithm for computing all subtree repeats in *rooted ordered unlabeled* trees. In [24], Christou *et al.* extended this algorithm to compute all subtree repeats in *rooted or-*

dered labeled trees in linear time *and* space. The authors considered only *full subtrees*, that is, subtrees which contain *all* nodes and edges that can be reached from their root.

The limitation of the aforementioned results is that they cannot be applied to *unordered* nor *unrooted* trees. By unrooted, we mean that the input tree does not have a dedicated root node; and, by unordered, we mean that the order of the adjacent nodes (children/neighbors) of any node of the tree is irrelevant. Such trees are a generalization of rooted ordered trees, and, hence, they arise naturally in a broader range of real-world applications. For instance, unrooted unordered trees are used in the field of (molecular) phylogenetics [43, 148].

Biological motivation. As we explain in the Introduction (Chapter 1, page 5), the field of molecular phylogenetics deals with inferring the evolutionary relationships among species using molecular sequencing technologies and statistical methods. Phylogenetic inference methods typically return unrooted unordered labeled trees that represent the evolutionary history of the organisms under study. These trees depict evolutionary relationships among the molecular sequences of extant organisms (living organisms) that are located at the tips (leaves) of those trees and hypothetical common ancestors at the inner nodes of the tree. With the advent of so-called next-generation sequencing technologies, large-scale multi-national sequencing projects such as, for instance, 1KITE [76, 104] emerge. In these projects, large phylogenies that comprise thousands of species and massive amounts of whole-transcriptome or even whole-genome data need to be reconstructed.

In phylogenetic inference software, a common technique for optimizing the likelihood function, which typically consumes $\approx 95\%$ of total execution time, is to eliminate duplicate *sites* (equivalent columns in the MSA). This is achieved by compressing identical sites into site patterns and assigning them a corresponding weight. This can be done because duplicate sites yield exactly the same likelihood *iff* they evolve under the same statistical model of evolution. When two sites are identical, this means that the leaves of the tree are labeled equally. Consider a forest of trees with the same topology, where, for each tree, the labels are defined by the molecular data stored at a particular site of the MSA and the position of the tips. Knowing equivalent subtrees within such a forest would allow someone to minimize the number of operations required to compute the likelihood of a phylogenetic tree. This can be seen as a generalization of the site compression technique. This application to phylogenetics is the focus of Chapter 8.

Our Contribution. In this chapter, we extend the series of results presented in [25] and [24] by introducing an algorithm that computes all subtree repeats in *unrooted unordered labeled* trees in linear time and space. The importance of our contribution is underlined by the fact that the presented algorithm can be easily modified to work on trees that do not satisfy some or any of the above assumptions on the tree structure; for example, it can be applied to rooted, ordered, or unlabeled trees.

7.2 Definitions: Central Points, Tree Rooting, and Heights

As explained in Section 3.3 (page 18), an unrooted unordered tree is an undirected unordered acyclic connected graph $T = (V, E)$ where V is the set of nodes and E the set of edges such that $E \subset V \times V$ with $|E| = |V| - 1$. The number of nodes of a tree T is denoted by $|T| := |V|$.

Some additional definitions are needed here.

Definition 44 (Tree Center, Central and Bicentral Tree)**.** *The* **tree center** *of an unrooted tree $T = (V, E)$ is the set of all vertices such that the greatest node distance to any leaf is minimal.*

If an unrooted tree T has one node that is a tree center, it is called a **central tree***.*

If two adjacent nodes are contained in the tree center, it is called a **bicentral tree** *[65].*

Definition 45 (Rooting an Unrooted Tree)**.** *For an unrooted tree $T = (V, E)$, let $\hat{T}(T) = (\hat{V}, \hat{A})$ be the rooted tree on $\hat{V} = V \cup \{r\}$, where $\hat{A}$ is defined such that $|\hat{A}|$ is minimal with $(u, v) \in \hat{A}$ only if $\{u, v\} \in E$ and each node other than r is reachable from one central point. If T is a bicentral tree, we add the additional root node r to V and add two edges to $\hat{A}$, namely (r, v) and (r, u), where v and u are the central points of T. The edge between its two central points is not added. Otherwise, if T is a central tree, with tree center u, we set $r := u$ and thus $\hat{V} = V$.*

Note that under the definitions given in Section 3.3 (page 18) two central points of a bicentral tree are thus siblings of each other.

Definition 46 (Height)**.** *The* **height of a rooted (sub)tree** *$\hat{T}(v, u)$ of some tree T, denoted by $h(v, u)$, is defined as the number of edges on the longest path from the root v to some leaf of $\hat{T}(v, u)$. The* **height of a node** *v, denoted by $h(v)$, is defined as the length of the longest path from v to some leaf in $\hat{T}(T)$.*

For simplicity, in the rest of the text, we denote: a rooted unordered labeled tree by $\hat{T}$; an unrooted unordered labeled tree by T; and the rooted (directed) version of T by $\hat{T}(T)$, as defined above.

7.3 Problem Definition: Subtree Repeats

Two trees $\hat{T}_1 = (V_1, A_1)$ and $\hat{T}_2 = (V_2, A_2)$ are *equal*, denoted by $\hat{T}_1 = \hat{T}_2$, if there exists a bijective mapping $f : V_1 \to V_2$ such that the following two properties hold

$$(v_1, v_2) \in A_1 \Leftrightarrow (f(v_1), f(v_2)) \in A_2$$

$$\mathit{label}(v) = \mathit{label}(f(v)), \forall v \in V_1.$$

A *subtree repeat* R in a tree T is a set of node tuples $(u_1, v_1), \ldots, (u_{|R|}, v_{|R|})$, such that $\hat{T}(u_1, v_1) = \cdots = \hat{T}(u_{|R|}, v_{|R|})$. We call $|R|$ the repetition *frequency* of R. If $|R| = 1$ we say that the subtree $\hat{T}(u_1, v_1)$ does not repeat. An *overlapping* subtree repeat is a subtree repeat R, where at least one node v is contained in all $|R|$ trees. If no such v exists, we call it a *non-overlapping* subtree repeat. A *total repeat* R is a subtree repeat that contains all nodes in T, that is, $R = \{(u_1, u_1), \ldots, (u_{|R|}, u_{|R|})\}$. See Fig. 7.1 in this regard.

In the following, we consider the problem of computing all such subtree repeats of an unrooted tree T.

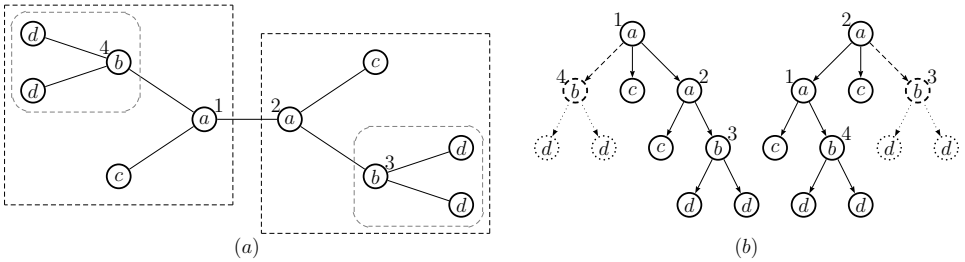


Figure 7.1: *(a) An* unrooted tree T *consisting of 10 nodes; a* non-overlapping subtree repeat $R = \{(3,2),(4,1)\}$ *is marked with dashed rounded rectangles; another* non-overlapping subtree repeat *containing the trees* $\hat{T}(1,2), \hat{T}(2,1)$ *is marked with dashed rectangles (b) An* overlapping subtree repeat $R = \{(2,3),(1,4)\}$ *of* T *resulting from the deletion of the dashed edge and its corresponding dotted subtree. This is an overlapping subtree repeat since nodes 1 and 2–and the node labeled by c–are in both subtrees. A* total repeat $R = \{(1,1),(2,2)\}$ *of* T *can be obtained by keeping all the edges and rooting* T *in node* 1 *(*$\hat{T}(1,1)$*) and* 2 *(*$\hat{T}(2,2)$*), respectively.*

7.4 Algorithm

The algorithm for finding all subtree repeats works in two stages: the forward/non-overlapping stage and the backward/overlapping stage. The forward stage finds all non-overlapping subtree repeats of some tree T. The backward stage uses the identifiers assigned during the forward stage to detect all overlapping subtree repeats, including total repeats.

The forward/non-overlapping stage: We initially present a brief description of the algorithmic steps. Thereafter, we provide a formal description of each step in Algorithm 1. This algorithm is related to that of [4] for deciding tree isomorphism.

In the following, we identify each node in the tree by a unique integer in the range of 1 to $|T|$. Such a unique integer labeling can be obtained, for instance, by a pre- or post-order tree traversal.

The basic idea of the algorithm can be explained by the following steps:

1. Partition nodes by height.
2. Assign a unique identifier to each label in Σ.
3. For each height level starting from 0 (the leaves).
    - i For each node v of the current height level construct a string containing the identifier of the label of v and the identifiers of the subtrees that are attached to v.
    - ii For each such string, sort the identifiers within the string.
    - iii Lexicographically sort the strings (for the current height level).
    - iv Find non-overlapping subtree repeats as identical adjacent strings in the lexicographically sorted sequence of strings.
    - v Assign unique identifiers to each set of repeating subtrees (equivalence class).

We will explain each step by referring to the corresponding lines in Algorithm 1.

Partitioning the nodes according to their height requires time linear with respect to the size of the tree, and is described in line 2 of Algorithm 1. This is done using an array H of queues, where $H[i]$, for all $0 \leq i \leq \lfloor d(T)/2 \rfloor$, contains all nodes of height i. Thereafter, we assign a unique identifier to each label in Σ in lines 3-7. The main loop of the algorithm starts at line

Algorithm 1: FORWARD-STAGE

Input : Unrooted tree $T = (V, E)$ labeled from Σ
Output: Sets $\mathcal{R}_{reps}$ of non-overlapping subtree repeats of T

▷ Partition tree nodes by height
for *all* $v \in V$ **do** Compute $h(v)$ and ENQUEUE($H[h(v)], v$)
$cnt \leftarrow 0$
▷ Assign a number from 1 to $|\Sigma|$ to each label
for *all labels* $\ell \in \Sigma$ **do**
    $cnt \leftarrow cnt + 1$
    $L[\ell] \leftarrow cnt$
▷ Compute subtree repeats
$reps \leftarrow 0$
for $i \leftarrow 0$ **to** $\lfloor d(T)/2 \rfloor$ **do**
    $S \leftarrow \emptyset$
    ▷ Construct a string of numbers for each node v and its children
    foreach $v \in H[i]$ **do**
        Let $children(v) = \{ u \mid \{u, v\} \in E \} \setminus \{parent(v)\}$ and $c_v = |children(v)|$
        $s_v \leftarrow L[label(v)]K[u_1]K[u_2] \dots K[u_{c_v}]$, if $children(v) = \{u_1, u_2, \dots, u_{c_v}\}$
        $S \leftarrow S \cup \{s_v\}$
    ▷ Remap numbers $[1, |T| + |\Sigma|)$ to $[1, |H[i]| + \sum_{v \in H[i]} c_v]$
    $R \leftarrow$ REMAP(S)
    ▷ Bucket sort strings
    Bucket sort the (unique) numbers of all strings in R.
    Let R' be the set of individually sorted strings that have been extracted from the respective sorted list from the previous step.
    Lexicographically sort the strings in R' using radix sort and obtain a sorted list R'' of strings $r_1, r_2, \dots, r_{|R''|}$.
    Let each r_i be of the form $k_1^i k_i^2 \dots k_{|r_i|}^i$ and the corresponding, original unsorted string s_i of the form $L[v_1^i]K[v_2^i] \dots K[v_{|r_i|}^i]$.
    $reps \leftarrow reps + 1$
    $\mathcal{R}_{reps} \leftarrow \{(v_1^1, parent(v_1^1))\}$
    $K[v_1^1] \leftarrow reps + cnt$
    for $j \leftarrow 2$ **to** k **do**
        if $r_j = r_{j-1}$ **then**
            $\mathcal{R}_{reps} \leftarrow \mathcal{R}_{reps} \cup \{(v_1^j, parent(v_1^j))\}$
        else
            $reps \leftarrow reps + 1$
            $\mathcal{R}_{reps} \leftarrow \{(v_1^j, parent(v_1^j))\}$
        $K[v_1^j] \leftarrow reps + cnt$

8 and processes the nodes at each height level starting bottom-up from the leaves towards the central points. The main loop consists of four steps. First, a string is constructed for each node v which comprises the identifier for the label at v followed by the identifiers assigned to $u_1, u_2, \ldots, u_{c_v}$. The identifiers of $u_1, u_2, \ldots, u_{c_v}$ represent the subtrees $\hat{T}(u_1), \hat{T}(u_2), \ldots, \hat{T}(u_{c_v})$, where $u_1, u_2, \ldots, u_{c_v}$ are the children of v (lines 11-16). Assume that this particular step constructs k strings $s_1, s_2, \ldots, s_k$.

In the next step, we sort the identifiers within each string. To obtain this sorting in linear time, we first need to remap individual identifiers contained as letters in those strings to the range $[1, m]$. Here, m is the number of unique identifiers in the strings constructed for this particular height, and the following property holds: $m \leq \sum_{i=1}^{k} |s_i|$. We then apply a bucket sort to these remapped identifiers and reconstruct the ordered strings $r_1, r_2, \ldots, r_k$ (lines 17-20).

The next step for the current height level is to find the subtree repeats as identical strings. To achieve this, we lexicographically sort the ordered strings $r_1, r_2, \ldots, r_k$ (line 22), and check neighboring strings for equivalence (lines 23-33). For each equivalence class $\mathcal{R}_i$ we choose a new, unique identifier, that is assigned to the root nodes of all the subtrees in that class (lines 26 and 33). Finally, each set $\mathcal{R}_i$ contains exactly the tuples of those nodes that are the roots of a particular non-overlapping subtree repeat of T and their respective parents.

Remapping from $\mathcal{D}_1 = [1, |T| + |\Sigma|)$ to $\mathcal{D}_2 = [1, |H[i]| + \sum_{v \in H[i]} c_v]$ can be done using an array A of size $|T| + |\Sigma|$, a counter m, and a queue Q. We read the numbers of the strings one by one. If a number x from domain $\mathcal{D}_1$ is read for the first time, we increase the counter m by one, set $A[x] := m$, and place m in Q. Subsequently, we replace x by m in the string. In case a number x has already been read, that is, $A[x] \neq 0$, we replace x by $A[x]$ in the string. When the remapping step is completed, only the altered positions in array A will be cleaned up, by traversing the elements of Q.

Theorem 47 (Correctness)**.** *Given an unrooted tree T, Algorithm 1 correctly computes all non-overlapping subtree repeats.*

Proof. First note that if any two subtrees $\hat{T}_1$ and $\hat{T}_2$ are repeats of each other, they must, by definition, be of the same height. So the algorithm is correct in only comparing trees of the same height. Additionally, non-overlapping subtree repeats of a tree T can only be of height $\lfloor d(T)/2 \rfloor$ or less, where $d(T)$ is the diameter of T. Therefore, the algorithm is correct in stopping after processing all $\lfloor d(T)/2 \rfloor + 1$ height classes, in order to extract all the non-overlapping subtree repeats. Since the algorithm only extracts

non-overlapping repeats, we define repeats to mean non-overlapping repeats for the rest of this proof. In addition, for simplicity, we consider the rooted version of T for the rest of this proof.

We show that the algorithm correctly computes all repeats for a tree of any height by induction. For the base case we consider an arbitrary tree of height 1 (trees with height 0 are trivial). Any tree of height 1 only has the root node and any number of leafs attached to it. At the root we can never find a subtree repeat, so we only need to consider the next lower (height) level, that is, the leaf nodes. Any two leafs with identical labels will, by construction of the algorithm, be assigned the same identifiers and thus be correctly recognized as repeats of each other.

Now, assume that all (sub)trees of height $m-1$ have correctly been assigned with identifiers by the algorithm *and* that they are identical for two (sub)trees *iff* they are unordered repeats of each other.

Consider an arbitrary tree of height $m+1$. The number of repeats for the tree spanned from the root (node r) is always one (the whole tree). Now consider the subtrees of height m. The root of any subtree of height m must be a child of r. For any child of r that induces a tree of height smaller than m, all repeats have already been correctly calculated according to our assumption.

Two (sub)trees are repeats of each other *iff* the two roots have the same label and there is a one-to-one mapping from subtrees induced by children of the root of one tree to topologically equivalent subtrees induced by children of the root of the second tree. By the induction hypothesis, all such topologically equivalent subtrees of height $m-1$ or smaller have already been assigned identifiers that are unique for each equivalence class. Thus, deciding whether two subtrees are repeats of each other can be done by comparing the root labels and the corresponding identifiers of their children, which is exactly the process described in the algorithm. The approach used in the algorithm correctly identifies identically labeled strings since the order of identifiers has been sorted for a given height class. Thus the algorithm finds all repeats of height m (and $m+1$ at the root). □

Theorem 48 (Complexity)**.** *Algorithm 1 runs in time and space* $\mathcal{O}(|T|)$.

Proof. We prove the linearity of the algorithm by analyzing each of the steps in the outline of the algorithm. Steps 1 and 2 are trivial and can be computed in $|T|$ and $|\Sigma|$ steps, respectively. Notice that $|\Sigma| \leq |T|$.

The main for loop visits each node of T once. For each node v a string s_v is constructed which contains the identifier of the label of v and the

identifiers assigned to the child nodes of v. Thus, each node is visited at most twice: once as parent and once as child. This leads to $2n-1$ node traversals, where n is the number of nodes of T, since the root node is the only node that is visited exactly once. The constructed strings for a height level i are composed of the nodes in $H[i]$ and their respective children. In total we have $c(i) := \sum_{v \in H[i]} c_v$ child nodes at a height level i, where c_v is the number of children of node v. Therefore, the total size of all constructed strings for a particular height level i is $|H[i]| + c(i)$. Step 3ii runs in linear time with respect to the number of nodes at each height level i *and* their children. This is because the remapping is computed in linear time with respect to $|H[i]| + c(i)$. By the remapping, we ensure that the identifiers in each string are within the range of 1 to $|H[i]| + c(i)$. Using bucket sort we can then sort the remapped identifiers in time $|H[i]| + c(i)$ for each height level i. Consequently, the identifiers in each string can be sorted in time $|H[i]| + c(i)$ by traversing the sorted list of identifiers and positioning the respective identifier in the corresponding string on a first-read-first-place basis. This requires additional space $|H[i]| + c(i)$ to keep track which remapped identifier corresponds to which strings.

After remapping and sorting the strings, finding identical strings as repeats requires a lexicographical sorting of the strings. Strings that are identical form classes of repeats. Lexicographical sorting (using radix sort) requires time $\mathcal{O}(|H[i]| + c(i))$ and at most space for storing $|T| + |\Sigma|$ elements since the identifiers are in the range of 1 to $|T| + |\Sigma|$. This memory space needs to be allocated only once. Moreover, the elements that have been used are cleared/cleaned-up at each step via the queue Q in linear time.

By summing over all height levels we obtain $\sum_{i=0}^{\lfloor d(T)/2 \rfloor} (|H[i]| + c(i)) = 2n-1$. Thus the total time over all height levels for each step described in the loop is $\mathcal{O}(|T|)$. The overall time and space complexity of the algorithm is thus $\mathcal{O}(|T|)$. □

We conclude this section with an example demonstrating Algorithm 1. Consider the tree T from Fig. 7.2. The superscript indices denote the number associated with each node, which, in this particular example, correspond to a pre-order traversal of $\hat{T}(T)$ by designating node 1 as the root. Lines 1-2 partition the nodes of T in $\lfloor d(T)/2 \rfloor + 1$ sets according to their height. The sets $H[0] = \{3, 5, 6, 7, 8, 10, 11, 13, 14, 15, 17, 19, 20, 23, 25, 26, 28\}$, $H[1] = \{4, 12, 18, 22, 24, 27\}$, $H[2] = \{2, 9, 21\}$ and $H[3] = \{1, 16\}$ are created. Lines 5-7 create a mapping between labels and numbers. $L[a] = 1$, $L[b] = 2$, $L[c] = 3$, and $L[d] = 4$.

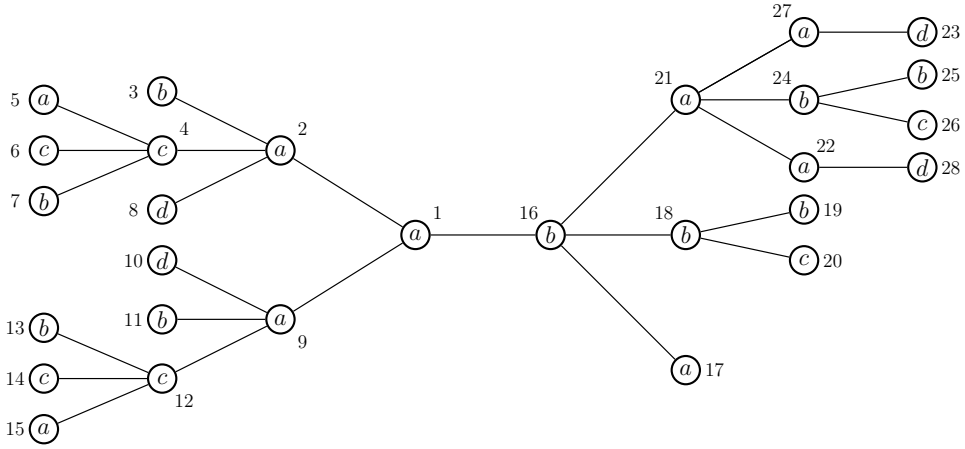


Figure 7.2: *Graphical representation of tree T. The superscript indices denote the unique identifier assigned to each node by traversing T.*

Height	Step	Process	Repeats
0	Strings: S	$2,1,3,2,4,4,2,2,3,1,1,2,3,4,2,3,4$	$\mathcal{R}_1 = \{3,7,11,13,19,25\}$
	Remapping: R	$1,2,3,1,4,4,1,1,3,2,2,1,3,4,1,3,4$	$\mathcal{R}_2 = \{5,15,17\}$
	Sorting: R'	$1,2,3,1,4,4,1,1,3,2,2,1,3,4,1,3,4$	$\mathcal{R}_3 = \{6,14,20,26\}$
	Repeats: R''	$\underbrace{1,1,1,1,1,1}_{5},\underbrace{2,2,2}_{6},\underbrace{3,3,3,3}_{7},\underbrace{4,4,4,4}_{8}$	$\mathcal{R}_4 = \{8,10,23,28\}$
1	Strings: S	3 6 7 5, 3 5 7 6, 2 5 7, 1 8, 2 5 7, 1 8	
	Remapping: R	1 2 3 4, 1 4 3 2, 5 4 3, 6 7, 5 4 3, 6 7	$\mathcal{R}_7 = \{22,27\}$
	Sorting: R'	1 2 3 4, 1 2 3 4, 3 4 5, 6 7, 3 4 5, 6 7	$\mathcal{R}_5 = \{4,12\}$
	Repeats: R''	$\underbrace{1\ 2\ 3\ 4,1\ 2\ 3\ 4}_{9},\underbrace{3\ 4\ 5,3\ 4\ 5}_{10},\underbrace{6\ 7,6\ 7}_{11}$	$\mathcal{R}_6 = \{18,24\}$
2	Strings: S	1 5 9 8, 1 8 5 9, 1 11 10 11	
	Remapping: R	1 2 3 4, 1 4 2 3, 1 5 6 5	$\mathcal{R}_8 = \{2,9\}$
	Sorting: R'	1 2 3 4, 1 2 3 4, 1 5 5 6	$\mathcal{R}_9 = \{21\}$
	Repeats: R''	$\underbrace{1\ 2\ 3\ 4,1\ 2\ 3\ 4}_{12},\underbrace{1\ 5\ 5\ 6}_{13}$	
3	Strings: S	2 6 10 13, 1 12 12	
	Remapping: R	1 2 3 4, 5 6 6	$\mathcal{R}_{10} = \{16\}$
	Sorting: R'	1 2 3 4, 5 6 6	$\mathcal{R}_{11} = \{1\}$
	Repeats: R''	$\underbrace{1\ 2\ 3\ 4}_{14},\underbrace{5\ 6\ 6}_{15}$	

Table 7.1: *State of lists S, R, R', R'' for each height level and resulting sets $\mathcal{R}_{reps}$ of non-overlapping subtree repeats*

Table 7.1 shows the state of lists S, R, R', R'' during the computation of the main loop of Algorithm 1 for each height level, where S is the list of string identifiers, R is the list of remapped identifiers, R' is the list of indi-

vidually sorted remapped identifiers, and R'' is the list R' lexicographically sorted. Fig. 7.3 depicts tree T with the respective identifiers for each node as assigned by Algorithm 1.

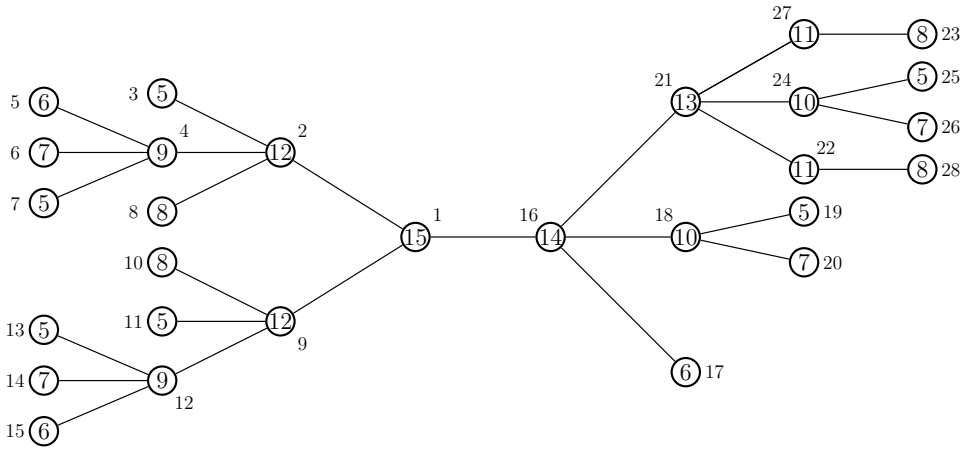


Figure 7.3: *Graphical representation of tree T with the associated identifier for each node as assigned by Algorithm 1.*

The backward/overlapping stage: Now we show how to calculate the overlapping and total subtree repeats. For this, we first introduce some additional definitions.

Definition 49 (Sibling repeat)**.** *Given an unrooted tree T, two equal subtrees of $\hat{T}(T)$ whose roots have the same parent are called a* sibling repeat*.*

Definition 50 (Child repeat—recursively defined)**.** *Given an unrooted tree T, two subtrees of $\hat{T}(T)$ whose roots have the same identifiers and whose root's respective parents are roots of trees in the same sibling or child repeat, are called a* child repeat*.*

Note that, with these definitions we get that two trees with roots u and v, respectively, are child or sibling repeats of each other *iff* the unique path between nodes u and v is symmetrical with respect to the node identifiers of the nodes traversed on the path.

The two following lemmas illustrate why it is necessary and sufficient to know the identifiers from the forward stage to compute all overlapping subtree repeats.

Lemma 51 (Sufficient conditions)**.** *Let r be the parent of u and v, where u and v are roots of a sibling repeat. Then the trees $\hat{T}(u,u)$ and $\hat{T}(v,v)$ are elements of the same total repeat. The trees $\hat{T}(r,u)$ and $\hat{T}(r,v)$ are elements of the same overlapping subtree repeat.*

Let u and v be roots of a child repeat. Furthermore let r_u and r_v be the parents of u and v, respectively. Then the trees $\hat{T}(u,u)$ and $\hat{T}(v,v)$ are elements of the same total repeat, and the trees $\hat{T}(r_u,u)$ and $\hat{T}(r_v,v)$ are elements of the same overlapping subtree repeat.

Proof. Trivial, by inspection; see Fig. 7.2. □

In Fig. 7.2, the trees $\hat{T}(2,1)$ and $\hat{T}(9,1)$ form a sibling repeat, thus the trees $\hat{T}(4,2)$ and $\hat{T}(12,9)$ form a child repeat. From the sibling repeat, we get that $\hat{T}(2,2)$ and $\hat{T}(9,9)$ are elements of a total repeat, while $\hat{T}(1,2)$ and $\hat{T}(1,9)$ are within the same overlapping repeat. Analogously, for the child repeat we get the trees $\hat{T}(4,4)$ and $\hat{T}(12,12)$ as total repeats and $\{(2,4),(9,12)\}$ as an overlapping repeat.

Note that Lemma 51 implies that all nodes of a subtree that is element of an overlapping subtree repeat with repetition frequency $|R|$ are roots of trees in overlapping repeat classes of frequency at least $|R|$.

Lemma 52 (Necessary conditions)**.** *Any two trees that are elements of a total repeat must have been assigned the same identifiers at their respective roots during the forward stage, and be rooted in roots of either sibling or child repeats.*

Any two trees that are elements of an overlapping subtree repeat, but not of a total repeat, must have been assigned the same identifiers at their respective roots during the forward stage, and be rooted in parents of roots of either sibling or child repeats.

Proof. We first look at the case of total repeats. Let $\hat{T}(u,u) = \hat{T}(v,v)$. We now consider the unique path p between u and v. Obviously, for equality among these two trees to hold, the path must be symmetrical, which by recursion implies that u and v are roots of either sibling or child repeats; see Fig. 7.4.

The case of other overlapping subtree repeats works analogously. Let $\hat{T}(r_u,u) = \hat{T}(r_v,v)$ not be total, but overlapping subtree repeat. These trees are obtained by removing a single edge from the tree: $\{r_u,u\}$ and $\{r_v,v\}$, respectively. Let p be the path between u and v. Since the trees are elements of an overlapping subtree repeat, r_u and r_v must lie on this path. Additionally, since r_u and r_v are on the path from u to v, $h(v) = h(u)$, and

since any tree is acyclic, then r_u and r_v must be closer to the central points than u and v, respectively. Since there is an edge connecting r_u with u and r_v with v this means that r_u and r_v are parents of u and v, respectively. Again, the path p is symmetrical with respect to the node labels of nodes along the path, so u and v are roots of either sibling or child repeats.

□

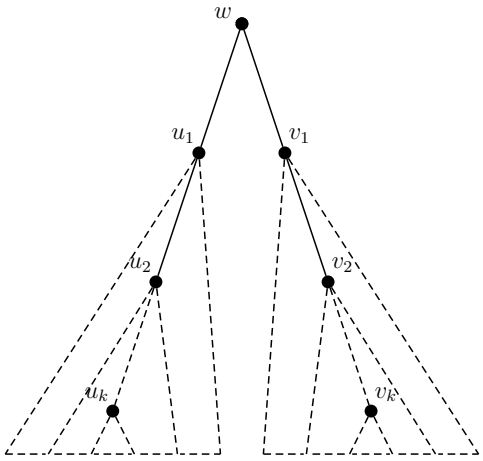


Figure 7.4: *$T(v_2, v_k) = T(u_2, u_k)$ is an overlapping repeat* iff *$T(u_k, u_2) = T(v_k, v_2)$ is a child repeat, which is true* iff *identifier(u_k) = identifier(v_k), identifier(u_2) = identifier(v_2), identifier(u_1) = identifier(v_1).*

Given these two lemmas, we can compute all overlapping subtree repeats by checking for sibling and child repeats. This can be done by comparing the identifiers assigned to nodes in the forward stage. The actual procedure of computing all overlapping subtree repeats is described in Algorithm 2. Algorithm 2 takes as input an unrooted tree T that has been processed by Algorithm 1; that is, each node of tree T has already been assigned an identifier according to its non-overlapping repeat class. First, the algorithm considers the rooted version of T, that is $\hat{T}(T)$. This is done since many operations and definitions rely on $\hat{T}(T)$. Next, we define a queue Q, whose elements are sets of nodes. Initially, Q contains only the root node (more specifically, a set containing only the root node) of $\hat{T}(T)$ (line 2). Processing Q is done by dequeuing a single set of nodes at a time (lines 5-16). For a given set U of Q, the algorithm creates a set I containing the identifiers of children of all the nodes in U. Then, the algorithm remaps these identifiers to the range of $[1, |I|]$ constructing a new set I' (line 12). Then, we construct

Algorithm 2: BACKWARD-STAGE

Input : Unrooted tree $T = (V, E)$ labeled from Σ with identifiers assigned by Algorithm 1

Output: Sets $\mathcal{R}'_{reps}$ of overlapping subtree repeats of T

▷ Initialize queue Q with the root node r of $\hat{T}(T)$
ENQUEUE$(Q, \{r\})$
▷ Compute overlapping subtree repeats
while QUEUE-NOT-EMPTY(Q) **do**
  $U \leftarrow$ DEQUEUE(Q)
  ▷ Get the identifiers of the children of the nodes in U
  Let $cod(U)$ be the cumulated out degree of all the nodes in U
  Let $children(U) = \{u_1, u_2, \ldots, u_{cod(U)}\}$ be the children of the nodes in U
  Let $ids(children(U)) = \{i_1, i_2, \ldots, i_{cod(U)}\}$ be the identifiers of $\{u_1, u_2, \ldots, u_{cod(U)}\}$
  $I \leftarrow ids(children(U))$
  ▷ Remap numbers $[1, |T| + |\Sigma|)$ to $[1, |I|]$
  $I' \leftarrow$ REMAP(I)
  Let $I' = \{i'_1, i'_2, \ldots, i'_{cod(U)}\}$ be the remapped identifiers of $\{u_1, u_2, \ldots, u_{cod(U)}\}$
  Let $C =< i'_1, u_1 >, < i'_2, u_2 >, \ldots, < i'_{cod(U)}, u_{cod(U)}) >$ be a list of tuples
  ▷ Bucket sort the remapped identifiers
  Bucket sort the list C by $i'_1, i'_2, \ldots, i'_{cod(U)}$.
  ▷ Extract the equivalence classes
  foreach $E = \{v_1, v_2, \ldots, v_k\}$ *of nodes with equivalent identifiers in* C **do**
    ENQUEUE(Q, E)
    for $i \leftarrow 1$ **to** k **do**
      $\mathcal{R}'_{reps} \leftarrow \mathcal{R}'_{reps} \cup \{(parent(v_i), v_i)\}$
      $\mathcal{R}'_{reps+1} \leftarrow \mathcal{R}'_{reps+1} \cup \{(v_i, v_i)\}$
    $reps \leftarrow reps + 2$

a list C of tuples, such that each tuple contains the remapped identifier of a child and the corresponding node. Therefore, we can use bucket sort to sort these tuples by the remapped identifiers in time linear in the cardinality of I.

We are now in a position to apply Lemmas 51 and 52. By Lemma 52, finding sibling and child repeats is done by creating sets of nodes with equivalent identifiers in C (line 18). This can be easily done due to the sorting part of the algorithm. These sets are then enqueued in Q, and, by Lemma 51 and 52, all resulting subtree repeats (overlapping and total) are, thus, created (lines 21-22). Hence we immediately obtain the following result.

Theorem 53 (Correctness)**.** *Given an unrooted tree T with identifiers assigned by Algorithm 1, Algorithm 2 correctly computes all overlapping subtree repeats, including total repeats.*

Algorithm 2 enqueues each node of T once. For each enqueued node, a constant number of operations is performed. Therefore we get the following result.

Theorem 54 (Complexity)**.** *Algorithm 2 runs in time and space $\mathcal{O}(|T|)$.*

7.5 Properties of Subtree Repeats

In this section, we provide properties which could potentially be used to speed-up an implementation of the above algorithms.

Property 7.1 (Trivial path)**.** *If we find a non-overlapping subtree with repetition frequency* 1*, no node that lays on the path from the root of that subtree to the farthest central point (including the central point itself) can have a repetition frequency other than 1 for non-overlapping subtree repeats rooted in this node. We call this path the* trivial path*.*

Proof. The proof is trivial. Assume some node v on the trivial path would induce a non-overlapping subtree repeat with frequency higher than 1. By definition, all subtrees of the subtree rooted in v must be contained in all subtrees in this repeat. In particular the original subtree with repetition frequency 1. □

The implications for implementations are obvious. Any time we encounter a subtree with repetition frequency 1, we can mark all nodes on the trivial path as *trivial*, and add them to their own repeat class.

Property 7.2 (Inclusion of trivial path)**.** *All trees from overlapping subtree repeats with repetition frequency higher than* 1 *must contain all nodes that lay on any trivial path.*

Proof. The proof is trivial. We prove the property by contradiction. Let v be a node on the trivial path. Then, by construction of overlapping subtree repeats only a single subtree in the repeat contains v. However, since v is on the trivial path, there must be a subtree without repeats induced by it. That is, no other subtree in the same overlapping repeat can have this subtree included, which contradicts the equality among trees. □

7.6 Conclusion

We presented a simple and time-optimal algorithm for computing all full subtree repeats in unrooted unordered labeled trees; and showed that the running time of our method is linear with respect to the size of the input tree.

The presented algorithm can easily be modified to operate on trees that do not satisfy some or any of the aforementioned assumptions on the tree structure.

- *Rooted trees*: In a rooted tree $\hat{T}$, only non-overlapping repeats can occur. Therefore it is sufficient to apply Algorithm 1 with the following modifications: first, we define $\hat{T}(\hat{T}) := \hat{T}$; second, the main for loop must iterate over the height of $\hat{T}$, instead of depending on its diameter.

- *Ordered trees*: If for a node the order of its adjacent nodes is relevant, that is, the tree is ordered, the bucket sort procedures in Algorithms 1 and 2 must be omitted. Additionally, sibling repeats must not be merged in line 19 of Algorithm 2 but rather be enqueued separately.

- *Unlabeled trees*: Trivially, an unlabeled tree can be seen as a labeled tree with a single uniform symbol assigned to all nodes.

Algorithm 1 can also be used to compute subtree repeats over a *forest* of rooted unordered trees. The method is the same as for the case of a single tree. The method reports all subtree repeats by clustering the identifiers of equal subtrees from all trees in the forest into an equivalence class. The correctness of this approach can be trivially obtained by connecting the roots of all trees in the forest with a virtual root node, and applying the algorithm to this single tree. This solves the problem involved in the concrete application scenario discussed in Section 7.1. However, the application to

likelihood calculations on phylogenetic trees is studied in more detail in Chapter 8.

Algorithm 1 can also be directly applied to solve the *maximal leaf-agreement isomorphic descendant subtree* (MLAIDS) problem [73]. MLAIDS is defined as follows: given a set of k phylogenetic (evolutionary) trees, find k maximal subtrees from the given trees, such that the leaves as well as the structure of the subtrees are equal. Thus, in a biological context, it is easy to find out which bipartitions (for example in a reference tree) are supported by all trees from a given tree set (for example bootstrap trees) (see Section 3.5 ,page 26).

8 Application of Subtree Repeats to Phylogenetic Trees

The phylogenetic likelihood function is the major computational bottleneck in ML phylogenetic inferences and BI. We present and implement a new method for identifying *and* omitting redundant operations in phylogenetic likelihood calculations. We assess the performance improvement that can be attained by comparing our new approach to one of the fastest and most highly tuned implementations of the phylogenetic likelihood function. Furthermore, we also report on the memory savings that can be attained via our method. Our method is generic, that is, it can seamlessly be integrated into any phylogenetic likelihood implementation.

We intend to publish a manuscript, containing the contents of this chapter. The intended title for this manuscript is "Efficient detection of repeating sites to accelerate phylogenetic likelihood calculations". A pre-print can be obrained at `http://biorxiv.org/content/early/2016/01/04/035873` [84]. Tomáš Flouri and Alexandros Stamatakis co-authored this paper with me.
Tomáš Flouri and I devised the algorithm for an efficient calculation of the likelihood function, and jointly implemented the software. Together we set up, and conducted, the experiments for evaluating the run time improvements and memory saving due to our algorithm. Alexandros Stamatakis provided additional expertise on implementation details, and provided the data sets. All authors were involved in the writing of this manuscript.

8.1 Motivation and Related Work

As discussed before, in phylogenetic tree analyses, such as ML based tree searches or BI, the by far most costly operation is the repeated evaluation of the *phylogenetic likelihood function* (PLF). It is already known, that many operations performed during the PLF evaluation in popular ML tools such as PhyML [63] and RAxML [130] or BI tools such as ExaBayes [2] or MrBayes [114], are redundant and can be omitted to accelerate the PLF.

Savings can be achieved by taking into account that (sub-)trees with identical leaf labels (in our case nucleotides), identical branch lengths and the same model parameters always yield the same likelihood score or conditional likelihood values.

Therefore, we can save computations by detecting repeating site patterns in the MSA for a given (sub-)tree topology. We will refer to those

repeating site patterns as *repeats*.

A commonly used method exploiting this property consists in only evaluating the likelihood of unique sites (columns) of a MSA. Assuming that only one set of model parameters is used for the particular MSA (i.e. unpartitioned analysis), identical sites yield the same likelihood. Therefore, the likelihood can still be accurately calculated by assigning a weight to each unique site. These weights correspond to the site frequency in the original MSA. Felsenstein refers to this method as *aliasing* in the documentation of PHYLIP [45].

Another standard technique for accelerating the PLF for *inner* nodes whose descendants are *tips* (or *leaves*) is to precompute the conditional likelihood for any combination of two states. Since there is small, finite number of character states, those precomputed entries can be stored in a lookup table, and queried when needed, instead of repeatedly re-computing them.

These two techniques are standard methods and are incorporated in virtually all PLF implementations. The benefits are faster computation times and often, in the case of the first method, considerable memory savings in the order of $d \cdot s \cdot r \cdot (t-2) \cdot c$, where d is the number of duplicate sites, s the number of states, r the number of rate categories, t the number of taxa (tips), and c a constant size for storing a conditional likelihood entry (typically 8 bytes for double precision). For example, on a phylogeny of 200 taxa with 100 000 duplicate sites, 4 states (nucleotide data) and 4 rate categories, the memory savings could be as high as 2.5 gigabytes, not to mention the savings in redundant PLF computations.

We present a general method for speeding up PLF calculations which, at the same time, reduces memory usage to a minimum. We aim to detect *all* conditional likelihood vectors at any node in the tree, that yield identical likelihood values. Computing these likelihood entries only once is thus sufficient. However, certain considerations must be made.

First, the algorithm must allow for the efficient detection of repeats. That is, the overhead incurred by finding repeats must be small to allow for a faster overall PLF execution. Furthermore, hardware related issues, such as non-linear cache accesses have to be considered. For this reason, to test the speed of the new algorithm, the performance is measured against one of the fastest and highly optimized software for PLF calculations. Additionally, the bookkeeping overhead must be small such that it does not substantially increase the PLF memory footprint.

Second, the algorithm, and the corresponding data structures, must be

flexible enough to allow for so-called *partial* tree traversals. When proposing new tree topologies via some tree rearrangement (e.g. *nearest neighbor joining*, *subtree pruning and regrafting*, as illustrated on page 25) , not all conditional likelihood vectors need to be updated. An efficient method for calculating repeats must take this into account and analogously only update the necessary data structures for the partial traversal (i.e., subset of conditional likelihoods). Thus, the overall goal is to minimize the book-keeping cost for detecting repeats such that the trade-off is favorable.

Our results. We present a new, *simple* algorithm that generalizes the common PLF optimization techniques explained above and satisfies the efficiency properties; it detects identical sites at *any* node of the phylogenetic tree and not only at the (selected) root, and thus minimizes the number of operations required for likelihood evaluation.

It is based on our linear-time linear-space (on the size of tree) algorithm for detecting repeating patterns in general rooted non-phylogenetic trees [51] (see Chapter 7). In order to obtain the desired run-time improvements, we present an adapted version of this algorithm for the PLF that reduces book-keeping overhead and relies on two additional properties of phylogenies as opposed to general *multifurcating* (or $n-$ary) trees.

First, we assume a *bifurcating* (binary) tree. This assumption can be relaxed to allow multifurcating trees by using a bifurcating tree that arbitrarily resolves the multifurcations.

Second, the calculation of the so-called conditional likelihood depends on the transition probability of one state to another. These probabilities are not generally the same for different branches in the tree. Thus, we only consider identical nucleotide patterns to be repeats if they appear at the tips of the same (ordered) subtree.

We show that even a rudimentary sub-optimal implementation of the PLF, that makes use of our method, consistently outperforms one of the most efficient PLF implementations available with a 2- to 10- fold speedup. In addition, the memory requirements are always significantly lower than for all widely used PLF implementations, in some cases up to 4 times **less** memory is required.

For the examples of the theoretical part of this chapter, and for the sake of simplicity, we assume that genetic sequences only contain the four DNA bases (that is, `A`, `C`, `G`, `T`). The approach we present can be easily adapted

to any other number and set of states (e.g., degenerate DNA characters and gaps and protein sequence data). The data sets we use for benchmarking our method in Section 8.4 are empirical DNA and protein data sets that *do* contain gaps and ambiguous characters.

Related work. Sumner *et al.* presented a method that relies on so-called partial likelihood tensors [135]. There, for each site of the alignment, the nucleotides at each tip node are iteratively included in the calculations. Let s_i be the nucleotide for site s at tip node i. The values are first calculated for (s_1), then (s_1, s_2), (s_1, s_2, s_3) and so on, until $(s_1, s_2, s_3, \ldots, s_m)$ has been processed, where m is the number of tip nodes. If the likelihood for another site $s\prime$ with $s\prime_1 = s_1$, $s\prime_2 = s_2$ and $s\prime_3 \neq s_3$ is to be computed, the results for s restricted to the first two tip nodes (s_1, s_2) can be reused for this site. A lexicographical sorting of the sites is applied in order to approximately maximize the number of operations that can be saved with this method. The authors report run-time improvements for data sets with up to 16 taxa. For more than 16 taxa, the performance of the method is reported to degrade significantly. In addition, the authors measured the relative speedup of the PLF with respect to their own, unoptimized implementation and not the absolute speedup with respect to the fastest implementation available at that time.

In [75] the idea of using general subtree site repeats for avoiding redundant PLF operations is mentioned, but dismissed as not practical because of the high bookkeeping overhead. Instead, only repeating subtree patterns consisting entirely of gaps are considered since they can be easily identified by using and updating bit vectors, that is, the bookkeeping overhead is low. In so-called gappy MSAs, with a high percentage of gaps, the authors report a speedup of 25-40% and 65% resp. 68% memory savings on gappy alignments consisting of 81.53% resp. 83.4% gaps. The underlying data structure used for identifying such repeating subtree sites is called subtree equality vector (SEV) and was originally introduced in [132]. There, only homogeneous subtree columns are considered. That is, a repeat is only stored as such, if all nucleotides in this subtree column are the same. This is done to avoid the perceived complexity associated with finding general (heterogeneous) subtree site repeats. In [132] a speedup of 19-22% is reported for the PLF computation.

Similar to [135], the authors of [109] devised a method for accelerating the likelihood computation of a site by storing and reusing the results obtained for a preceding site. Since only the results for one single site (the preceding

site) are retained, an appropriate sorting of the sites is required. This column sorting approach is reported to yield speedups in settings where the PLF is evaluated multiple times for the same topology. The authors showed that, sorting the sites in order to maximize the saving potential, can lead to run-time reductions of roughly 10% to over 80%. This corresponds to a more than 5-fold speedup. However, the authors also note, that an ideal algorithm for PLF calculations would reuse all previously computed values from all sites and not just the neighboring ones. Furthermore, the optimal column sorting relies on solving the NP-hard traveling salesman problem and relies on the tree topology. Thus, in order to maintain a polynomial time execution of the algorithm, a search heuristic, that can yield sub-optimal results, is used. This means, that the chosen column sorting may not yield the maximum amount of savings.

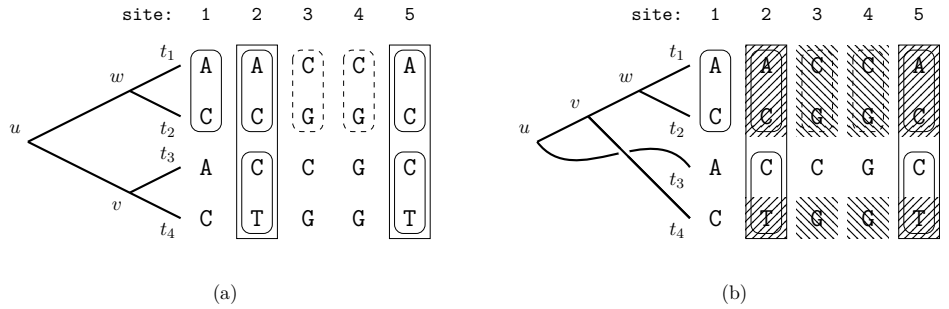
The most similar method to what we describe here also deploys a flavor of subtree repeats to accelerate the PLF has been presented in [140]. There, the PLF is used for a positive selection test. However, the authors focus on the performance for a fixed tree topology only that is repeatedly traversed. Thus, the overhead for detecting repeats is negligible, since repeats need to be computed only once. Here, we present a general method for dynamically changing trees. Performance was tested against the well known CODEML software of the PAML package [149].

8.2 Definition of Site Repeats and Observations

First, we introduce, and review the notations which we will use throughout the chapter.

Trees As usual (see Section 3.3, page 18), a tree $T = (V, E, b)$ is a connected acyclic graph where V is the set of nodes and E the set of *edges* (or *branches*), such that $E \subset V \times V$. We use the notation $e = (u, v) \in E$ to denote an edge e with end-points $u, v \in V$ and $b(e)$ to denote the associated branch length. The set $L(T)$ denotes the tip nodes. As defined in the introduction (Section 3.3, page 19) we use $\hat{T}(u)$ to denote a subtree of a (rooted) tree T, rooted at node u.

The phylogenetic likelihood function Before we introduce our method, it is necessary to review description of PLF computations. As defined in Section 3.4 (page 22), the likelihood is a function of the states Σ, the transition probabilities P for all branches, and the equilibrium frequencies of the states Π.



(a) (b)

Figure 8.1: *(a) Sites* 1, 2 *and* 5 *form repeats at node* w *as they share the same pattern* AC*.Another repeating pattern is located at sites* 3 *and* 4 *(*CG*) for the same node. Note that, node* u *also induces a subtree with pattern* AC *at the tips. However, since branch lengths can be different than for the subtree rooted at node* w*, the conditional likelihoods may differ as well. Analogously, sites* 2 *and* 5 *are site repeats for node* v *as they have the same pattern* CT*, and hence the conditional likelihood is the same for those two sites. Finally, sites* 2 *and* 5 *form repeats for node* u *(*ACCT*). (b) Repeats are not necessarily substrings of MSA sites. For this particular tree topology, node* v *has two sets of repeats: sites* 2 *and* 5 *(*ACT*) and sites* 3 *and* 4 *(*CGG*). The repeats are not contiguous in the alignment columns.*

To evaluate the likelihood, using the Felsenstein *pruning* algorithm, we iteratively evaluates Equation 11 (Section 3.4, page 23) for all internal nodes. Recall that Equation 11 defines the conditional likelihood of a node v for state s at site i as

$$L_s^{(v)}(i) = (\sum_{x \in \Sigma} P(s \to x | b((v,w)) L_x^{(w)}(i)) (\sum_{x \in \Sigma} P(s \to x | b((v,u)) L_x^{(u)}(i)),$$

where u and w are the two descendants of v. Further recall that the conditional likelihood vector (CLV) (Equation 12, page 23) is defined as

$$\mathcal{L}^v(i) = \bigcup_{\forall s \in \Sigma} L_s^{(v)}(i).$$

Site repeats We now introduce *site repeats.*

Definition 55 (Site repeat)**.** *Let* $\hat{T}(u)$ *be a subtree of* T *rooted at node* u*, which represents the relations among* $|L(\hat{T}(u))|$ *taxa (tip nodes). We denote the sequence of the* i*-th taxon* $x^i = x_1^i x_2^i \dots x_n^i$*.*

Two sites j *and* k *are called* repeats *of one another* iff $x_j^i = x_k^i$ *for all taxa* i*,* $1 \leq i \leq |L(\hat{T}(u))|$*, in* $\hat{T}(u)$*.*

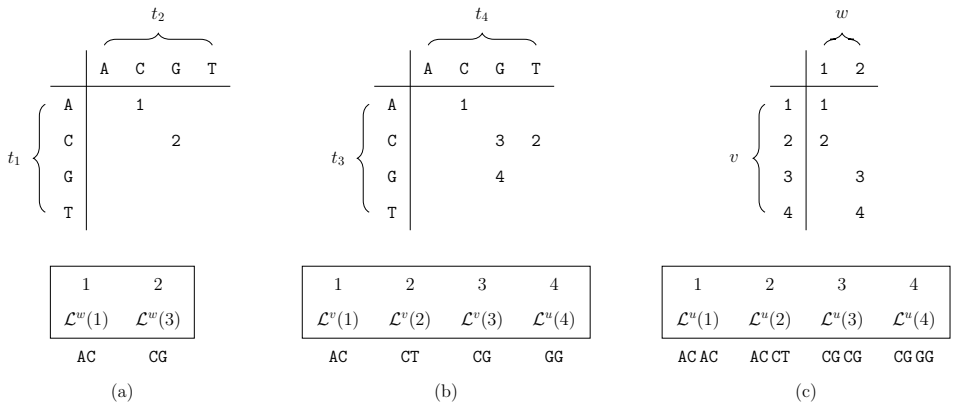


Figure 8.2: *Tables with identifier associations of nodes w (a),v (b), and u (c). The respective lists at the bottom store the corresponding CLVs that must be computed for each unique identifier. Table (a) shows that two likelihood computations need to be performed for node w (sites 1 and 3), while the rest of the sites are repeats of those two. Tables (b) and (c) show the corresponding information for nodes v and u.*

Next, we make two observations.

Observation 56. *If two sites j and k are not repeats in some tree $\hat{T}(u)$, then they are not repeats in any tree that has $\hat{T}(u)$ as a subtree.*

Observation 57. *Let u be a node whose two direct descendants (children) are nodes v and w. If two sites j and k are repeats in both $\hat{T}(v)$ and $\hat{T}(w)$, then j and k are also repeats in $\hat{T}(u)$.*

With these two observations we can formulate the algorithm for detecting site repeats in binary phylogenetic trees.

Before we formalize the algorithm though, let us consider Figure 8.1 again. From Observations 56 and 57, we see that the only repeating sites at the root node (node u), are sites 2 and 5. This is obviously correct, since both have the nucleotide pattern `ACCT` at the tips.

8.3 Algorithm

The method we propose identifies site repeats at each node in a bottom-up (post-order) traversal of the tree, meaning that a node can only be processed once the repeats for both its two children have been identified. Tip nodes

only have the trivial repeats of all sites that show a common character (for DNA, `A`, `C`, `G`, or `T`, respectively). Therefore, the method always starts at an inner node whose two children are tip nodes. By construction, such a node always exists in any binary tree and assuming four nucleotide states, there are 16 possible combinations of homologous nucleotide pairs in the sequences of its two child nodes. Let $\hat{\sigma}$ be the set of observed states (4 for nucleotides, or 16 when considering ambiguities and gaps). We use a bijective mapping $\tau : \hat{\sigma} \times \hat{\sigma} \to \{1, 2, \ldots, \hat{\sigma}^2\}$ to assign a unique identifier to each nucleotide pair. The problem of identifying repeats is thus reduced to filling, and querying the corresponding entries in a list of CLVs.

Tip–Tip case. Assuming that x^v resp. x^w are the sequences at the two children v and w of the parent node u, site i of u is assigned the *identifier* $\phi_u(i) = \tau(x_i^v, x_i^w)$. This function assigns the same identifier to sites which are repeats in $\hat{T}(u)$. Figure 8.2 illustrates the assignment of identifiers to combinations of nucleotides at the tips for the example given in Figure 8.1. The CLV entries are computed only once for each identifier (for example, the first time it is encountered) at the parent node u. By Observation 57, if a site i is a repeat of site j, that is, they were assigned the same identifier, then the method can either (a) copy the CLV from site j (run time saving), or (b) completely omit the likelihood value, since it can always be retrieved from site j (run time *and* memory saving). Furthermore, by Observation 56, we know that each repeat is found by this method.

Tip–Inner and Inner–Inner cases. We proceed analogously to detect repeats at nodes for which at least one child node is not a tip node. Again, let u be the parent node and v and w the two child nodes for which all repeats have already been computed. Further, let $\phi_v(i)$ and $\phi_w(i)$ be the respective identifiers of v and w at site i. We define the maximum over all $\phi_v(i)$ and $\phi_w(i)$ as *vmax* and *wmax* respectively. The values *vmax* and *wmax* are also the numbers of unique repeats at nodes v and w. Now, finding repeats at u is again simply a matter of filling the appropriate lists/matrices. Given *vmax* and *wmax*, there are at most $vmax \times wmax$ combinations at the sites. See Figure 8.2c for the identifier calculation at node u for the example tree and MSA in Figure 8.1. Figure 8.3 shows the combined overall result.

Figure 8.4 outlines algorithm Repeats(u, v, w, ϕ), which calculates the CLV for a given node u, with child nodes v and w. Using algorithm Repeats we can now design the overall method by conducting a post-order

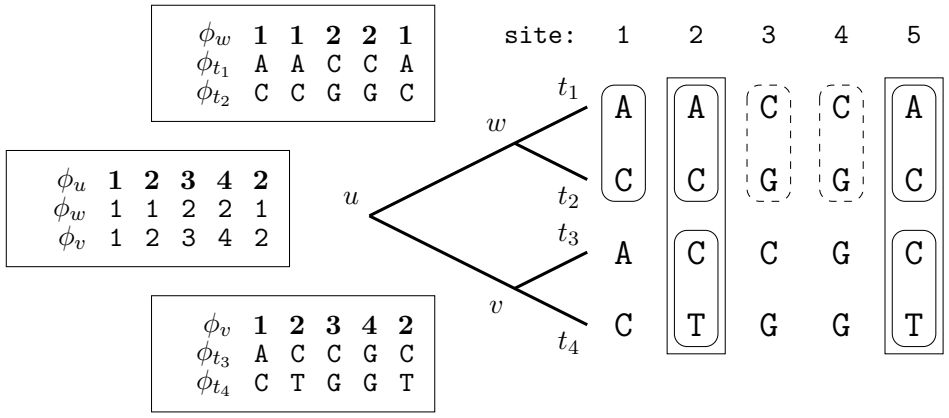


Figure 8.3: *Identifiers (here ϕ_x) are shown for every site of the alignment at every node in the tree. As we have already observed, sites* 2 *and* 5 *are repeats at node u, and thus, have been assigned the same identifier.*

traversal on all nodes of a tree T. For this, the tip nodes t_j can be assigned constant identifier sequences that correspond to their respective DNA sequences. The actual nucleotides `A`, `C`, `G`, and `T` can simply be mapped to integers. Note that, in most (if not all) phylogenetic inference tools, nucleotides are encoded using the *one-hot* (also called *1 out of N*) encoding, which ensures that the binary representations of their identifiers have exactly one bit set (e.g., `A` $\mapsto$ 1, `C` $\mapsto$ 2, `G` $\mapsto$ 4 and `T` $\mapsto$ 8). This is beneficial because the identifiers of ambiguities which are typically represented as disjoint unions of nucleotides, can be encoded as the bit-wise OR of the identifiers of the respective nucleotides. To simplify the method description, we discard ambiguities and consider only the four nucleotide bases. Hence, we use the encoding

$$\texttt{A} \rightarrow 1, \texttt{C} \rightarrow 2, \texttt{G} \rightarrow 3, \texttt{T} \rightarrow 4.$$

Lookup Table Since our focus is on an efficient implementation of the algorithm, we need to consider some technical issues in more detail. First, matrix M (defined in algorithm Repeats) can, in the worst case, become quadratic in size with respect to the number of sites in the alignment. This is unfortunate, since filling M affects overall asymptotic run-time. However, in terms of practical space requirements, M needs to be allocated only once and can be re-used for each inner node. For that, a linear list *clean* with one entry per MSA site, can be used to keep track of which entries are *valid*, that

```
REPEATS(u, v, w, φ)
    ▷ Initialization
 1. vmax ← 0
 2. wmax ← 0
 3. for i ← 1 to n do                          Get max identifier of node v
 4.     if φ_v(i) > vmax then vmax ← φ_v(i)
 5. for i ← 1 to n do                          Get max identifier of node w
 6.     if φ_w(i) > wmax then wmax ← φ_w(i)
 7. M ← {0}^(vmax × wmax)                      Initialize matrix M
 8. LH ← {0}^n
 9. ident ← 0
    ▷ Computation
10. for i ← 1 to n do                          Iterate over sites
11.     if M[φ_v(i), φ_w(i)] = 0 then          Check if site is not a repeat
12.         ident ← ident + 1                  Increase identifier count
13.         M[φ_v(i), φ_w(i)] ← ident          Set an identifier for the site
14.         LH[ident] ← L^u(i)                 Compute likelihood entries for site
15.         CLV[i] ← LH[ident]                 Place likelihood entry in CLV
16.     else                                   Site is a repeat
17.         CLV[i] ← LH[M[φ_v(i), φ_u(i)]]     Copy likelihood entry from repeat
18.     φ_u(i) ← M[φ_v(i), φ_w(i)]             Set site identifier
19. return CLV                                 return CLV
```

Figure 8.4: *Algorithm to compute the CLV of a parent node p. The most costly operation is the calculation of the CLVs, here, denoted by $\mathcal{L}^u(i)$. The algorithm thus minimizes the number of calls to this function.*

is, contain identifiers assigned to sites of current node, and which entries are *invalid* and contain identifiers assigned to sites for a previous node. After assigning an identifier i to a site of a node u, which we store in the array M, for example at position d, we also store the pair (d, u) in array *clean* at position i. Later on, when we process a different node, say v, and by chance, decide to give the same identifier i to some site, and again, by chance, the location for which we have to query matrix M is d, the element *clean*$[i]$ helps us to distinguish between valid and invalid records in M. Invalid records are equivalent to empty records and are overwritten. Further, in the actual implementation we limit the size of M to a constant maximum size. We implement this limit to avoid the impact of the quadratic complexity for filling M. Table 8.2 of Section 8.4 gives an overview of the size of M for different data sets. Since dynamically tuning the size of M to the data set can have a negative impact on the run-time and memory performance, the size of M is an input parameter. In addition, as M grows larger (i.e., we move closer to the root of the tree), it is less likely to encounter repeats in

the alignment. Note that, at the CLV of the root, there will be no repeats at all, since they have been removed by compressing MSA sites into patterns. One may also consider the following alternative view. If M is an $n \times n$ matrix, where n is the number of sites in the alignment, there can be no repeats, as every site must, by construction, have a unique identifier. If at least two sites were repeats of another, the maximal identifier would be strictly less that n and thus, M would not be a $n \times n$ matrix. Thus, if the product of maximum identifiers for two child nodes at some node u (that is, $vmax \times wmax$) exceeds our threshold for the size of M, we do not calculate repeats any more. Instead, the CLV entries are calculated separately for all sites. In other words, if calculating repeats becomes disadvantageous, repeat calculations are omitted. This allows for tuning the trade-off between repeat detection overhead, and PLF efficiency.

Memory savings Notice that, given algorithm REPEATS, not all entries in the CLVs of the child nodes v and w are needed to calculate the CLV at the parent node u. In particular, the CLV entries at site i for nodes v *and* w are only needed if the CLV at site i must be computed for u (see Figure 8.5). In fact, only the CLVs in array LH of algorithm REPEATS must be stored. Let $LH[u, j]$ be the CLV computed by Algorithm 8.4 for node u and site with identifier j. Then, the CLV for any site i is simply $LH[\phi_u(i)]$. In practice, this observation enables us to reduce the memory footprint of the PLF. Each CLV entry stores more than one single or double precision floating point values. For example, RAxML holds one double precision floating point number per DNA character and per evolutionary rate at each such CLV entry. Typically, the Γ model of rate heterogeneity is used (see [147]) with 4 rate categories. Thus, the memory footprint of a standard PLF algorithm for a MSA with n sequences of length m is $8 \times 4 \times 4 \times (n-2) \times m$ bytes. On the other hand, storing the site identifiers at each node only requires a single, unsigned integer per site. Thus, the memory required for storing CLVs without compression is $4 \cdot 4 = 16$ times higher than that of the site identifier list.

Thus, despite the fact that, we need additional data structures, and hence space for keeping track of the site identifiers at nodes, the memory requirements (if we do not store unnecessary CLV entries) are smaller than those of standard production level tools [49, 130]. While the identifiers are not the only additional data structures required for the actual implementation of the algorithm, the above argument illustrates that storing fewer CLV entries can help to save substantial amounts of RAM.

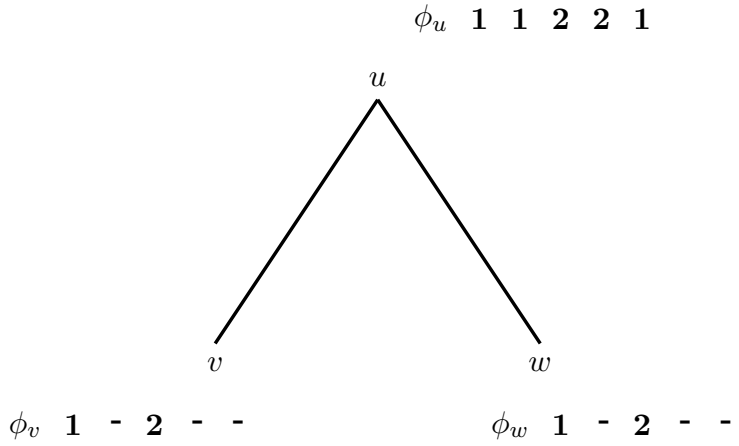


Figure 8.5: *Not all sites are needed for the likelihood calculation at parent node u. According to the identifiers of this example, sites 2 and 5 are repeats of site 1, and site 4 is a repeat of site 3. Therefore, the CLVs at sites 2, 4, and 5 do not need to be computed nor stored, as the CLV for sites 2 and 5, and site 4, of node u can be copied from sites 1, and 3, respectively.*

The overall algorithm, with memory savings and a bounded M, is given by algorithm REPEATS-FULL in Figure 8.6. One main difference to the snippet of Figure 8.4, is the introduction of a new array (*maxid*) which stores the maximal identifier assigned to each of the $2m-1$ nodes of the rooted tree T (assuming T has m tip nodes). Thereby, we eliminate the run time $\mathcal{O}(n)$ required for finding the maximal identifiers of the two children nodes (lines 3-6 in Figure 8.4) at the cost of $\Theta(m)$ memory. The second difference is that, we can no longer use the original set $\mathcal{L}^u(i)$ for the CLV entries for site i at a node u. This is due to the memory saving technique which omits the computation and storage of unnecessary CLVs as illustrated in Figure 8.5. The problem is that the CLV of the two children may not reside at entries i because repeats might have occurred. Therefore, the new ordered set $\hat{\mathcal{L}}^u(i)$ is defined as

$$\hat{\mathcal{L}}^u(i) = \bigcup_{\forall s \in \Sigma} \left(\sum_{\forall s_v \in \Sigma} P(s \to s_v | b((u,v))) L_{s_v}^{(v)}(\phi_v(i)) \right) \left(\sum_{\forall s_w \in \Sigma} P(s \to s_w | b((u,w))) L_{s_w}^{(w)}(\phi_w(i)) \right)$$

and the conditional likelihoods $L_x^{(v)}(\phi_v(i))$ resp. $L_x^{(w)}(\phi_w(i))$ can be obtained for all states x, from $CLV[v, \phi_v(i)]$ respectively $CLV[w, \phi_w(i)]$.

```
REPEATS-FULL(T, τ, tsize, x, n, m)
    ▷ Initialization
 1. M ← {0}^tsize                                  Initialize matrix M
 2. clean ← {0,0}^n                                Initialize clean array
 3. maxid ← {0}^(2m−1)                             Initialize maxid array
 4. P ← {u_1, u_2, ..., u_{m−1}}                   Post-order traversal of inner nodes
 5. ▷ Map nucleotides at tips to integers
 6. for u in L(T) do                               Iterate over all tip nodes u
 7.   for i ← 1 to n do                            Iterate over all sites of sequence x^u
 8.     φ_u(i) ← τ(x_i^u)                          Use mapping τ to encode nucleotide x_i^u
    ▷ Traverse all inner nodes in post-order
 9. for u in P do                                  Iterate through inner nodes
10.   v ← LEFT-CHILD(u)                           Set v as the left child of u
11.   w ← RIGHT-CHILD(u)                          Set w as the right child of u
12.   vmax ← maxid(v)                             Get maximal identifier of v
13.   wmax ← maxid(w)                             Get maximal identifier of w
14.   if vmax × wmax > tsize then                 Check if table size reached
15.     for i ← 1 to n do
16.       CLV[u, i] ← L̂^u(i)                      L̂^u(i) uses CLV[v, φ_v(i)] and CLV[w, φ_w(i)]
17.       φ_u(i) ← i
18.   else                                        we can still use site repeats
19.     ident ← 0
20.     for i ← 1 to n do
21.       mpos ← (φ_v(i) − 1) × wmax + φ_w(i)     linearize two coordinates into one index
22.       if M[mpos] = 0 or                       If matrix entry is empty or contains invalid
            clean[M[mpos]] ≠ (mpos, u) then       data, then compute likelihood from scratch
23.         ident ← ident + 1
24.         M[mpos] ← ident
25.         clean[M[mpos]] ← (mpos, u)
26.         CLV[u, ident] ← L̂^u(i)                L̂^u(i) uses CLV[v, φ_v(i)] and CLV[w, φ_w(i)]
27.       φ_u(i) ← M[mpos]
28.     maxid(u) ← ident                          Store max identifier for u
29. return CLV                                    return CLV
```

Figure 8.6: *Full description for computing all CLVs of a tree T with the memory saving technique and site repeat detection. Input parameters are the tree T of m taxa, the sequences (of size n) for each of the m taxa (denoted x^u for the sequence at tip node u), a mapping τ for encoding the MSA data to integer values, and the size tsize of the matrix used for computing site repeats. The algorithm computes only the necessary CLVs required for evaluating the likelihood of tree T, avoiding PLF calls on site repeats.*

Observation 58 (Runtime)**.** *Algorithm 8.6 computes all subtree repeats, and the corresponding CLVs, in linear time, with respect to the size of the alignment (number of sites times number of nodes).*

This trivially holds by inspection.

8.4 Computational Results

We can now compare the performance of the PLF implementation using our algorithm, against a standard implementation for this task. We implemented a rudimentary version of our algorithm in a new, low-level implementation of the *Phylogenetic Likelihood Library* PLL [49] (which we refer to as LLPLL) that does not make use of the highly optimized PLF of PLL, but allows for a straight-forward implementation of our method.

We used two implementations of our method. First, SRDT which computes the site repeats assuming a dynamically changing topology. That is, repeats are computed before any likelihood computation. The second utilization (SRCT) assumes a constant tree topology and therefore precomputes all site repeats once and uses that information every time the likelihood function is called without re-computing repeats.

For assessing the performance of our methods, we use the AVX-vectorized, sequential PLF implementation from PLL which uses the same, highly optimized PLF as RAxML. We selected the PLL/RAxML because (i) it is our own code and (ii) it is currently among the fastest and most optimized PLF implementations available. This thus guarantees a fair comparison, and ensures that our method can truly be used in practice for speeding up state-of-the-art inference tools. We use two flavors of PLL in our experiments; the plain version (we refer to it as PLL) and the memory saving SEV-based implementation of PLF (accessible using the `-U` switch in RAxML) which we refer to as PLL-SEV.

To obtain an accurate speedup estimate, we also used AVX intrinsics in our low-level implementation (LLPLL). However, it is still sub-optimal as PLL is faster with a speedup of approximately 1.40 - 1.45 as we show further.

Experimental setup. We performed four types of experiments for assessing the performance of our method. The results indicate that the sub-optimal implementation enhanced by a proof-of-concept utilization of our method outperforms the PLL likelihood function by a factor of up to 10. The four experiments cover the typical PLF use cases.

First, we exhaustively assess the performance of *full traversals* for all possible rootings of the trees on two data sets. Second, we assess the performance of full traversals on all selected datasets for a limited number of rootings drawn at random. Third, we evaluate the performance for *partial*

traversals. Finally, we assess PLF performance for fixed tree topologies. In this setting, preprocessing of site repeats is done only once and not for each likelihood evaluation.

For the experiments we used a 4-core Intel i7-2600 multi-core system with 16 GB RAM. In order to provide meaningful results, we always executed several (usually 10 000) independent likelihood computations.

Data sets. We used a mixture of empirical and simulated nucleotide data sets which are summarized in Table 8.1. The data sets contain gaps and ambiguous DNA characters. Table 8.1 also reports the percentages of gaps and site repeats in the alignments. The number of gaps are important, since they relate to the performance of the PLL-SEV implementation. The percentages of site repeats are given for an arbitrary root of the parsimony trees calculated for the data sets using RAxML [130]. Different trees, as well as different rootings usually give different values. We present only the values for the parsimony trees, as they are the tree topologies used for evaluating the performance of the SRDT and SRCT methods. The data set with 2 000 taxa has the lowest percentage of repeats, however, this data set still produces 86.95% repeats (which directly translate to identical conditional likelihood entries). We want to emphasize that we did not choose these data sets for their high numbers of repeats. In fact, the fraction of site repeats for each data set was previously unknown to us. The data sets and software used for testing, are available on-line[3,4]. For the run-time comparisons we focus purely on the PLF evaluation. Branch lengths and model parameters are fixed, and remain unchanged as they do not impact the run-time of PLF. The underlying trees are parsimony trees inferred with RAxML [130]. Since the calculation of the PLF takes up up to 85%-98% of the total run-time [6] of ML phylogenetic tree inferences, accelerating the performance of the PLF significantly impacts the overall execution time of ML analyses.

The memory savings due to calculating site repeats, as well as the actual size of the look up table to allow the computation of all repeats, are presented in Table 8.2. The size of the look up table was bounded by 200 MB. This corresponds to roughly 50 million entries (namely 4 byte *unsigned integer* values). The actual memory for the look up table was only allocated as needed. For most data sets, less than 200 MB of memory were thus used

[3] `https://github.com/stamatak/test-Datasets/`

[4] `http://sco.h-its.org/exelixis/web/software/site-repeats/`

Sequences [-]	59	128	354	404	500	994	1 512	2 000	3 782	7 764
Sites [-]	6 951	29 198	460	13 158	1 398	5 533	1 577	1 251	1 371	851
Repeats [%]	92.04	91.78	94.65	96.49	89.43	94.63	90.09	86.95	94.18	87.62
Gaps [%]	44.24	32.48	14.71	78.92	2.25	71.39	3.02	12.65	2.70	20.60

Table 8.1: *Nucleotide data sets summary. Sequences denotes the number of taxa in the data set. Sites is the length of the provided MSA. Repeats denotes the amount (in percentage) of sites in the MSA which are repeats of another at any node, and can thus be copied or omitted. This amount depends on the chosen root of the tree structure and the tree topology itself. The (unrooted) trees were obtained by running a maximum parsimony tree search for each of the data sets, and we chose one random node as the root to estimate the number and the table indicates the amount of repeats for that particular rooting. Gaps indicates the amount of gaps in the alignment.*

Sequences	59	128	354	404	500	994	1 512	2 000	3 782	7 764
Sites	6951	29 198	460	13 158	1 398	5533	1 577	1 251	1 371	851
Memory PLL [MB]	53	474	24.5	680	93	707	312	328	678	875
Memory PLL-SEV [MB]	46	403	21.5	326	93	256	308	297	674	819
Memory SRCT [MB]	32	303	7.5	202	34	164	104	120	171	298
Table size	5.3	220.1	0.07	23.5	0.81	6.6	2.9	2.4	2.8	0.86

Table 8.2: *Memory requirements for the different methods. Table size denotes the size of the lookup table M of Algorithm* Repeats-Full*, in millions of entries, that are needed to compute* all *possible repeats. Memory requirements for M, in MB, are thus four times as high as the presented numbers, since all entries are implemented as* unsigned integers.

(confer Table 8.2). The notable exception is the data set containing 128 taxa. For this data set, 220.1 million entries (roughly 880 MB) in M are needed in the worst case. Since we bound the size of M to 200 MB, this means that not all repeats were found when analyzing this particular data set.

Exhaustive evaluation of all rooting First, we evaluate the run-time impact of distinct rootings. The two data sets we used for this experiment have 59 and 354 taxa. Our implementation with site repeats enabled (*SRDT*), with site repeats disabled (*LLPLL*), as well as PLL and PLL-SEV were executed to perform PLF calculations (full tree traversals) for rootings on tip nodes. All of these implementations make use of an AVX vectorized function for calculating conditional likelihoods. This choice of rootings was selected because PLL requires that likelihood evaluations on unrooted trees based on full traversals are performed only on *terminal* edges, that is edges

whose one end-point is a tip node. Hence in this experiment we exhaustively evaluate the PLF for all possible rootings on tip nodes. For each rooting, we executed 10 000 independent PLF computations. The size of matrix M is limited to 200 MB for all data sets. As we can see in Table 8.2, this is sufficient to find all repeats for these two data sets. This initial analysis helps us understand the effect of root placement on the number of site repeats present in a full tree traversal and as a consequence on run-times.

For the 354 taxon data set, SRDT had an average run-time of 11.714 seconds (for 10 000 iterations) and reached maximum and minimum run-times of 15.207 and 10.211 seconds, respectively. The standard of run-times among all rootings was 0.94. PLL needed, on average, 58.112 seconds for this data set with maximum and minimum run-times of 61.067 and 57.573 respectively and a standard deviation of 0.54. Enabling the SEV method, PLL-SEV reduced the average run-time to 54.449 seconds, with minimum and maximum run-times of 55.128 and 57.299 seconds respectively. The standard deviation lowered to 0.411162.

For the 59 taxon data set, SRDT had an average runtime of 37.491 seconds. The respective maximum and minimum run-times were 44.787 and 31.515 seconds, and the standard deviation 3.197. PLL needed, on average, 134.769 seconds with a maximum run-time of 141.031 and minimum of 132.727 seconds. The standard deviation was 1.66. The average run-time of PLL-SEV was 124.733 seconds, with minimum and maximum run-times at 122.558 and 132.341, respectively, and a standard deviation of 2.05294. From this we see that, while the mean was higher for the PLL than the SRDT method, the standard deviation was lower for the PLL.

To get an initial estimate of how the original LLPLL implementation performs in comparison to PLL (and PLL-SEV) we measured its run-times by disabling site repeats. For the 354 taxon data set, the implementation averaged to 81.283 seconds, and and 194.932 seconds for the 59 taxon. The standard deviation was 0.231 and 0.783 for the two data sets, respectively. For these two specific data sets (354 and 59), PLL and PLL-SEV are on average faster by a factor of 1.4 and 1.45 (for PLL) and a factor of 1.49 and 1.56 (for PLL-SEV) respectively. The differences in speed between LLPLL and PLL can be explained by two factors. First of all, PLL is a highly optimized software for PLF calculations that directly derived from RAxML, which has been developed and optimized for over 10 years. Second, the standard optimization method explained in the introduction, namely, the look up table for tip-tip cases is not implemented for the LLPLL method yet. The reason for this is that the look up at a tip-tip node is replaced by the general repeats method implemented in SRDT, which has been disabled

Summary of speedups obtained using SRDT for a sample of rootings										
Data set	59	128	354	404	500	994	1512	2000	3782	7764
Speedup over PLL	3.46	3.27	4.96	5.31	2.78	4.5	3.03	2.47	4.06	2.63
Speedup over PLL-SEV	3.15	2.97	4.74	2.99	2.93	1.91	3.18	2.39	4.25	2.65

Table 8.3: *Speedup obtained when using SRDT over PLL and PLL-SEV for each of the ten data sets. SRDT is consistently faster than both methods.*

for this test.

Evaluation of a sample of rootings For the actual comparison of run times for full tree traversals between SRDT and PLL, we use the nucleotide data sets with taxon numbers ranging from 59 taxa to 7764 taxa (see Table 8.1). The run times were measured for 10 different rootings. These rootings were randomly chosen, and are not necessarily the same for the SRDT and PLL methods. Given the standard deviation, as demonstrated above, for different run times of the PLL under different rootings, this is a reasonable comparison. For the PLL method, the nodes for the different rootings were again restricted to be tip nodes only. For each rooting, we again conducted 10 000 full tree traversals and calculated the ratio of the time needed by SRDT, divided by the time needed by the PLL. The presented overall speedup of our new method per data set is then the average speedup over all 10 rootings. Table 8.3 shows the run-time improvements. As we can see, the SRDT implementation is always at least more than twice as fast as the PLL. In fact, the lowest observed average speedup (over the general PLL method) was 2.47. The maximal speed up was obtained for the data set containing 404 taxa. Here, the SRDT implementation was 5.31 times faster than the PLL. In table 8.1 we also see that this particular data set has the highest relative number of repeats among all analyzed data sets. This reinforces the initial intuitive assumption that the amount of repeats positively influences the runtime improvement. On the other hand, the largest decrease in speedup when comparing to PLL-SEV was for data sets 404 and 994 which contain over 70% gaps. Note also, that for data sets with a lower amount of gaps, run-time for PLL-SEV increased compared to PLL (data sets 1512, 3782 and 7764).

Partial traversal performance In phylogenetic inferences, to calculate the overall likelihood of the tree, it is not always necessary to conduct full tree traversals, in particular when conducting BI or ML tree searches that deploy local topological updates using, for instance nearest neighbor inter-

change (NNI) or subtree pruning and re-grafting (SPR) moves (confer Figure 3.10, Section 3.4, page 25). We need to assess the performance of our approach for this type of partial CLV updates as well, since less CLVs are updated and they might be located at the inner part of the tree where the number of repeats is lower. Therefore, we also assess performance, by emulating partial CLV updates. To this end, for each rooting where we evaluate the overall likelihood, we pick a random path into two directions away from it. At each node on this path we take a randomized decision on whether to stop the traversal (with probability $(1 - p)$), or continue to a randomly chosen child node with probability p. The traversal stops if both directions of the path have either been stopped with probability p, or a tip node has been reached. This pattern of CLV updates emulates the topological moves described in [90] for BI. As mentioned before, additionally to the time spend in the PLF, other factors such as optimizing branch lengths and model parameters for ML, also contribute to the overall execution time. Here we concentrate only on measuring the time for calculating the PLF. Figure 8.7

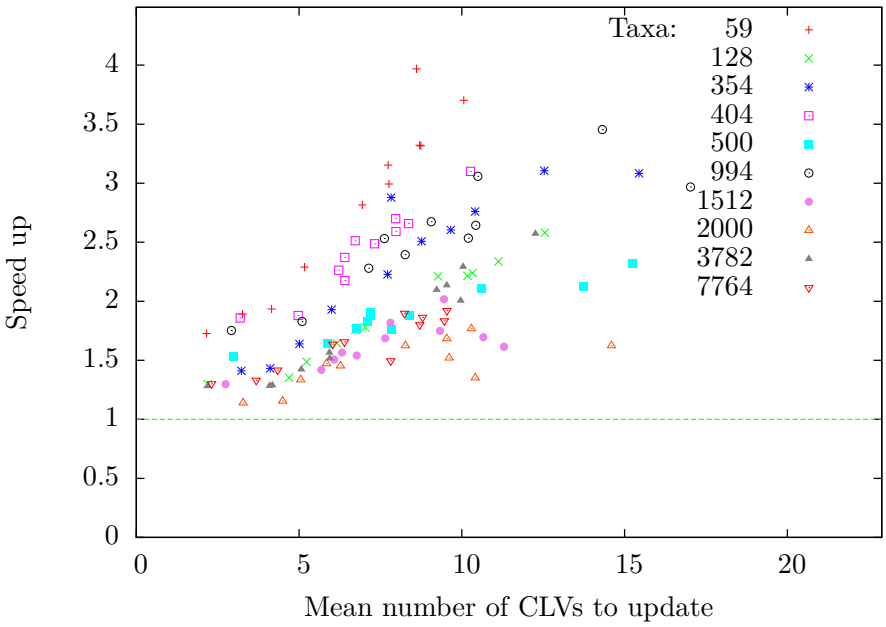


Figure 8.7: *Plotted are the runtime improvements of the SRDT method over the LLPLL method against the average number of updated CLVs. The colors distinguish the different data sets. Each data set is represented by eleven measurements for eleven different nodes.*

shows the run-time improvements of the SRDT method over LLPLL. For each data set, eleven nodes are chosen at random. For each of those nodes, 10 000 partial updates are simulated and timed by recalculating the CLVs along the path chosen by the above method (with $p = 0.95$). We present the individual average speed ups of each rooting for each data set, plotted against the number of average nodes that are updated for this particular rooting. We choose to only compare the SRDT method to LPLL, since for comparing it to the PLL it would be very hard to implement exactly identical partial traversals because of the different internal structures of PLL and SRDT. Thus, the speedup for the partial updates is *not* the absolute speedup for PLF implementations. Instead, our results demonstrate the relative speedup that can be achieved by incorporating repeats into any PLF implementation. For the full traversals over all possible rootings, the PLL was 1.4-1.45 times faster than the LLPLL method (see page 129). Assuming that these values are representative, the SRDT method still allows for faster PLF computations than the PLL for most, if not all, data sets, under this setting. Furthermore, as we discussed before, the speed difference is partially due to the lack of a dedicated tip-tip evaluation scheme for the LLPLL method. However, given the experimental set up here, a node for which both children are tips is included in the path, which is to be updated, at most twice. Thus, it remains to be evaluated, whether the speed difference between the LLPLL and PLL method persist under this setting.

Performance on fixed topologies Many phylogenetic tools use a fixed tree topology on which the likelihood is repeatedly calculated. Divergence time estimates [68] and model selection [1] are examples of this.

Under this setting, repeats can be pre-computed once and then reused for subsequent PLF invocations. Table 8.4 shows the run time improvement of the SRCT method over PLL and PLL-SEV under this setting.

Noticeably, for the data set containing 404 taxa we again observe the largest run time improvement of all analyzed data sets. Here, we observe an almost ten fold speed up (9.96 times faster run times).

8.5 Conclusion

We have seen that taking into account repeating patterns in the alignment *does* matter for an efficient PLF implementation. On fixed topologies, where repeats are only pre-processed once, we obtain an almost 10 fold run time improvement. When repeats need to be computed on the fly for changing

Summary of speedups obtained using SRCT when considering fixed topologies										
Sequences	59	128	354	404	500	994	1512	2000	3782	7764
Sites	6951	29198	460	13158	1398	5533	1577	1251	1371	851
Speedup over PLL	5.71	4.64	8.59	9.96	4.29	8.16	4.66	3.62	6.86	3.91
Speedup over PLL-SEV	5.22	4.23	8.22	9.96	5.44	3.30	4.88	3.48	7.18	3.93

Table 8.4: *Speedups of the new SRCT method which considers a fixed topology over the PLL and PLL-SEV.*

topologies, we still observe a run time improvement of up to more than five times faster execution times.

All of this can be achieved without requiring more memory overhead than standard production level software for calculating the PLF. In fact, the memory footprint of our presented method is less than that of the standard software by a factor of up to more than four times smaller memory consumption.

9 Are all Global Alignment Methods Correct?

Pairwise sequence alignment as explained in Section 3.1 (page 11) is one of the most fundamental operation in bioinformatics. Gotoh's algorithm [59] for this purpose is widely used and perhaps more importantly, implemented. A plethora of distinct formulations exist for this algorithm which, as we show, causes confusion in the field. More importantly, this confusion leads to numerous implementation issues and errors, of which typical end-users are not aware. Foremost, we point out two mathematical irregularities in Gotoh's 1982 paper. First, there are minor issues in the indices of the dynamic programming algorithm. Second, we describe a more critical problem in the initialization of the dynamic programming matrix for global sequence alignment. While the *index issue* becomes apparent immediately when implementing the procedure, the *initialization issue* is more involved and can easily be overseen. This observation is corroborated by several text books and lecture notes, where the initialization issue is either present, or circumvented via additional assumptions. As we show, the above two issues can and *do* generate incorrectly aligned sequences. Five out of ten implementations we analyzed yield sub-optimal sequence alignments. Finally, we provide a correct version and implementation of the algorithm.

We have prepared a manuscript titled "Are all Global Alignment Methods Correct?", with the contents of this chapter. A pre-print is available at `http://www.biorxiv.org/content/biorxiv/early/2015/11/12/031500` [53]. Tomáš Flouri, Torbjørn Rognes, Alexandros Stamatakis, I (co-) authored the manuscript.
Tomáš Flouri first recognized the irregularities in the original Gotoh publication [59]. Flouri and I provided a formal description of the problems and alternative formulations (and an implementation) to avoid them. Together, we analyzed the existing tools, lecture slides, books, and scientific papers to asses the spread and persistence of the mistakes. Rognes and Stamatakis confirmed the mistakes and helped write the paper.

9.1 Motivation and Related Work

The *Needleman-Wunsch* (NW) [106] and *Smith-Waterman* [127] algorithms for computing optimal global and local alignments are among the most important algorithms in bioinformatics and computational biology. They are typically presented in undergraduate lectures at many computer science and bioinformatics departments around the globe.

Although Needleman and Wunsch described their algorithm in their sem-

inal paper in 1970, the algorithm had already been discovered several times before. In fact, Damerau and Levenshtein independently described the algorithm in 1964 [28] and 1965 [97].

Analogous algorithms with quadratic run-times were also independently developed by Vintsyuk in 1968 for speech processing [142], and in 1974 by Wagner and Fischer for string matching [143].

In 1972, Sankoff presented an improved dynamic programming algorithm with quadratic time complexity for this problem by making additional assumptions [118]. The algorithm by Sankoff maximizes the number of matches between two sequences, without penalizing gaps.

Needleman and Wunsch described their algorithm in terms of maximizing similarity between two sequences. Levenshtein described the problem in terms of minimizing the *edit distance*, that is, the cost of edit operations (insertion, deletion, substitution) for transforming one sequence into another. In 1974, Sellers showed that these two variations are in fact equivalent [121].

Finally, in 1982 Gotoh presented a quadratic time algorithm to compute global sequence alignments with affine gap penalties [59]. Note that, Gotoh's approach also reduces the time complexity of the Smith-Waterman *local* alignment algorithm.

While the underlying idea of Gotoh's algorithm is valid and can yield the optimal pairwise sequence alignment, there are two issues that can lead to erroneous, that is, sub-optimal, alignments based on Gotoh's original description. The first issue (*index issue*) is straight-forward and simply a case of ambiguous indices. However, the second issue (*initialization issue*), which affects global alignments only, has a more substantial impact on alignment optimality and correctness.

There exist several distinct formulations based on Gotoh's original algorithm. Some of these are equivalent to Gotoh's algorithm, while others require additional assumptions to yield correct results. For instance, Durbin describes an algorithm that, by design, only computes alignments where an insertion can not be directly followed by a deletion and vice versa [35]. The algorithm is correct, if some restrictions are imposed on the affine gap penalty function and scoring matrix values. A sufficient condition is that the highest mismatch penalty is at most twice the gap extension penalty. Incidentally, on page 31 of [35], Durbin states this condition. On page 30 however, a different condition is given. For the latter, it is easy to show, that the condition is *not* sufficient for ensuring that insertions can not be followed by deletions in the optimal alignment.

All of the above generates confusion in the implementation of global alignment methods. Gotoh's initialization description is presented wrongly in standard textbooks (such as [64]) and in a plethora of online teaching material. Of the implementations we analyzed, some yield erroneous results, while others implicitly place additional assumptions on the alignment (e.g., no insertion can follow a deletion). This means that, the same two sequences can yield different alignments, depending on the software that is being used.

Overview. First, we give a description of Gotoh's algorithm (Section 9.2), as it represents the cornerstone for constructing pairwise sequence alignments. Then, we present a detailed analysis of the problems that were introduced in the original paper and show how to avoid them (Section 9.3). Last, we assess the impact of these ambiguities by listing books, implementations, and online lecture slides that either contain a mistaken interpretation of Gotoh's algorithm (books and lecture slides) or yield sub-optimal alignments (implementations). For lecture slides, we quantify the impact of the error, by the ratio of correct to incorrect presentations, and to lecture slides, where a formal initialization is missing altogether.

To illustrate the two error types, we first recapitulate Gotoh's algorithm for alignments with affine gap penalties. We use the same notation as in Gotoh's original paper.

9.2 Gotoh's Algorithm

Let $w_k = uk + v$ $(u \geq 0, v \geq 0)$ be the gap penalty for a gap of length k, where v is the *gap opening* penalty and u is the *gap extension* penalty. Let $A = a_1 a_2 \dots a_M$ and $B = b_1 b_2 \dots b_N$ be the two sequences we want to align. Further, assume that a weighting function $d(a_m, b_n)$ is given to score an aligned pair of residues a_m and b_n. Typically, $d(a_m, b_n) \leq 0$ if $a_m = b_n$, and $d(a_m, b_n) > 0$ if $a_m \neq b_n$. The NW algorithm calculates the cells of a dynamic programming matrix $D_{m,n}$ using the recursion:

$$D_{m,n} = \min(D_{m-1,n-1} + d(a_m, b_n), P_{m,n}, Q_{m,n}) \tag{30}$$

where

$$P_{m,n} = \min_{1 \leq k \leq m} (D_{m-k,n} + w_k) \tag{31}$$

and

$$Q_{m,n} = \min_{1 \leq k \leq n} (D_{m,n-k} + w_k) \tag{32}$$

Here, $D_{m,n}$ is the score of a globally optimal alignment of the first m residues of A with the first n residues of B. $P_{m,n}$ is the score of an optimal alignment of the first m residues of A with the first n residues of B that ends with a deletion of at least one residue from A, such that a_m is aligned with the gap symbol. Finally, $Q_{m,n}$ is the score of an optimal alignment of the first m residues of A with the first n residues of B that ends with an insertion of at least one residue from B, such that b_n is aligned with the gap symbol. Although, at first sight, $P_{m,n}$ and $Q_{m,n}$ appear to require $m-1$ (or $n-1$) steps, they can be obtained in a single step via the following expansion of the recursive formulation:

$$P_{m,n} = \min\{D_{m-1,n} + w_1, \min_{2\leq k\leq m}(D_{m-k,n} + w_k)\} \tag{33}$$

$$= \min\{D_{m-1,n} + w_1, \min_{1\leq k\leq m-1}(D_{m-1-k,n} + w_{k+1})\} \tag{34}$$

$$= \min\{D_{m-1,n} + w_1, \min_{1\leq k\leq m-1}(D_{m-1-k,n} + w_k) + u\} \tag{35}$$

$$= \min(D_{m-1,n} + w_1, P_{m-1,n} + u) \tag{36}$$

The same applies analogously to $Q_{m,n}$:

$$Q_{m,n} = \min(D_{m,n-1} + w_1, Q_{m,n-1} + u) \tag{37}$$

9.3 Original problems with Gotoh's Algorithm

We have found two issues with the original Gotoh paper [59]. With respect to the initialization, Gotoh states:

"At the beginning of the induction, one may set $D_{m,0} = P_{m,0} = w_m (1 \leq m \leq M)$, and $D_{0,n} = Q_{0,n} = w_n (1 \leq n \leq N)$. Alternatively, $D_{m,0} = P_{m,0} = 0$ and $D_{0,n} = Q_{0,n} = w_n$, or $D_{m,0} = P_{m,0} = 0$ and $D_{0,n} = Q_{0,n} = 0$ may be chosen in searching for the most locally similar subsequences ...".

Note that, the second sentence (at least the second part of it) refers to local alignments which are *not* affected by the error. Apart from the two issues we present in this Chapter, there are additional issues in Gotoh's paper, particularly in the description of the matrix traceback. In 1986, Altschul gave a detailed description of traceback issues introduced by Gotoh which can lead to sub-optimal alignments as well. For more information and examples see [7].

Index Issue. The first apparent issue is that wrong indices are used for initializing the P and Q matrices. Initially, the entries $P_{m,0}$ and $Q_{0,n}$, as well

as $D_{m,0}$ and $D_{0,n}$ (for $1 \leq m \leq M$, $1 \leq n \leq N$) are assigned some values. However, this is inconsistent with the recursions defined in equations 33 and 37. Consider computing the following entry $P_{1,1}$ of P (or $Q_{1,1}$ of Q). Equation 33 then reads as follows:

$$P_{1,1} = \min(D_{0,1} + w_1, P_{0,1} + u).$$

Here $D_{0,1}$ is defined but $P_{0,1}$ is *not* defined. However, $P_{1,0}$ *is* defined, so this may be a simple case of flipped indices. The same applies to matrix Q.

Initialization Issue. The more substantial problem are the actual values that are assigned to initialize P and Q. For global alignments, Gotoh proposes to initialize $D_{0,n} = Q_{0,n} = w_n$ and $D_{m,0} = P_{m,0} = w_m$ (for $1 \leq m \leq M$, $1 \leq n \leq N$). Correcting the indices for P and Q we obtain $D_{0,n} = P_{0,n} = w_n$ and $D_{m,0} = Q_{m,0} = w_m$. The value $D_{0,0}$ is defined as $D_{0,0} = 0$. Let us consider $P_{1,i}$ as defined in Equation 33 for some $i \in [1, N]$:

$$\begin{aligned} P_{1,i} &= \min(D_{0,i} + w_1, P_{0,i} + u) && (38)\\ &= \min(w_i + w_1, w_i + u) && (39)\\ &= \min(w_i + u + v, w_i + u) && (40)\\ &= w_i + u. && (41) \end{aligned}$$

Similarly, for $j \in [1, M]$:

$$Q_{j,1} = w_j + u. \quad (42)$$

To illustrate why this result is wrong, we consider a simple one-nucleotide example. Let $A = a_1$ and $B = b_1$. Further let $d(a_1, b_1) := 5$, the gap opening penalty $v := 2$, and the gap extension penalty $u := 1$. Now

$$D_{0,1} = D_{1,0} = P_{0,1} = Q_{1,0} = w_1 = v + u = 2 + 1 = 3.$$

Thus, by equations 38 and 42 we obtain,

$$\begin{aligned} P_{1,1} &= w_1 + u = v + u + u = 2 + 1 + 1 = 4\\ Q_{1,1} &= w_1 + u = v + u + u = 2 + 1 + 1 = 4. \end{aligned}$$

Plugging these values into Equation 30 we obtain

$$\begin{aligned} D_{1,1} &= \min(D_{0,0} + d(a_1, b_1), P_{1,1}, Q_{1,1})\\ &= \min(0 + 5, 4, 4)\\ &= 4. \end{aligned}$$

This implies that, the best alignment for A and B is:

$$\begin{array}{ll} A\text{:} & \text{-} \quad a_1 \\ B\text{:} & b_1 \quad \text{-} \end{array}$$

or

$$\begin{array}{ll} A\text{:} & a_1 \quad \text{-} \\ B\text{:} & \text{-} \quad b_1. \end{array}$$

However, the actual correct score for both of these alignments is $w_1 + w_1 = 3 + 3 = 6 \neq 4$. Aligning A and B as

$$\begin{array}{ll} A\text{:} & a_1 \\ B\text{:} & b_1. \end{array}$$

yields a score of $d(a_1, b_1) = 5 < 6$. Thus, conducting the initialization as proposed by Gotoh yields a sub-optimal solution for this simple example. Nonetheless, there is a straight-forward solution to this problem. We need to initialize the values for P and Q as $P_{0,n} \geq w_n + v$ and $Q_{m,0} \geq w_n + v$ (for $1 \leq n \leq N$, $1 \leq m \leq M$) to obtain the correct, optimal alignment score. If $P_{0,n} = w_n + v$ we can re-state Equation 38 as:

$$\begin{aligned} P_{1,i} &= \min(D_{0,i} + w_1, P_{0,i} + u) \\ &= P_{0,i} + u \\ &= w_i + v + u. \end{aligned}$$

For $P_{0,n} > w_n + v$ we get

$$\begin{aligned} P_{1,i} &= \min(D_{0,i} + w_1, P_{0,i} + u) \\ &= D_{0,i} + w_1 \\ &= w_i + v + u \end{aligned}$$

as well. A popular choice for $P_{0,n}$, in publications by authors that seem to be aware of this issue, is $P_{0,n} := \infty$ (see for example [7, 122]). A similar choice can be made for Q.
Using the corrected formula for our simple example of $A = a_1$, $B = b_1$, $d(a_1, b_1) = 5$, $v = 2$, and $u = 1$, we see that the values are correctly computed.

$$P_{1,1} = Q_{1,1} = w_1 + v + u = v + u + v + u = 2 + 1 + 2 + 1 = 6$$

By Equation 30 we get

$$\begin{aligned} D_{1,1} &= \min(D_{0,0} + d(a_1, b_1), P_{1,1}, Q_{1,1}) \\ &= \min(0 + 5, 6, 6) \\ &= 5 \end{aligned}$$

which is the correct result.

The values of $P_{1,k}$ (and analogously $Q_{k,1}$) need to contain two gap opening penalties. By definition, they should represent the score of an optimal alignment of the first residue of A with the first k residues of B and end with a deletion of a_1, that is, an alignment of a_1 with the gap symbol. The resulting alignment will then always start with an insertion of the k first symbols of B followed by a deletion of the first symbol of A. However, according to Gotoh's description, only a single gap opening penalty will be included.

9.4 Impact of the Errors

Even though Gotoh's paper was published over thirty years ago, the above error still persists in many papers and bioinformatics lectures. Furthermore, we are not aware of any previous work that specifically addresses the issues we have identified. Note that, there do exist publications that explain and/or implement a working or corrected version of the algorithm (e.g., [7, 33, 105, 110]). Other works either ignore this problem (e.g., [137]) or restrict values of v, u, and $d(a, b)$ such that the issue disappears. For example, in 1972 Sankoff [118] originally solved the problem only for $u = v = 0$, and Durbin [35] gives an algorithm that performs well if $2u$ is greater than the highest value of d.

Even though, some authors corrected these mistakes on their own, numerous other publications, textbooks, and lecture notes still use the initial, incorrect, description. In the following, we list textbooks and lecture slides that contain the error. Further, we list software packages that yield suboptimal alignments due to the issues described here or because of other conceptual errors. Note that, all open source software packages and implementations listed are available at `http://www.exelixis-lab.org/web/software/alignment/`.

Books: The following two standard text books contain the initialization error.

- "Algorithms on Strings, Trees, and Sequences" by Gusfield, 2009 [64],
- "Introduction to Computational Biology" by Waterman, 1995 [144].

Fortunately, several books exist that contain a correct description of a global alignment algorithm, for instance [122].

Software:

NW-align. The alignment program **NW-align**[5] (e.g. discussed in [146]) shows the behavior described in Section 9.3 when aligning `GGTGTGA` with `TCGCGT`. **NW-align** assigns a score of -11 for gap opening and -1 for gap extension. Note that, the interpretation of affine gap costs is slightly different from Gotoh's definition. Here, a gap of length k contributes a penalty of "$-11-(k-1)$" instead of "$-11-k$" as defined in Section 9.2. **NW-align** produces the following alignment:

```
-  G  G  T  G  T  G  A
         .  |  .  |  .
T  -  -  C  G  C  G  T
```

where the mismatch penalties are defined as $d(T,C) := -1$, $d(A,T) := 0$ and $d(G,T) := -2$. The score for the matches is defined as $d(G,G) := 6$. Thus, the score for this alignment is $-11-11-1-1+6-1+6+0 = -13$. Considering the alignment

```
G  G  T  G  T  G  A
   .  .  |  .  |  .
-  T  C  G  C  G  T
```

we can see that the result obtained by **NW-align** is sub-optimal, since the above alignment has a better score of $-11-2-1+6-1+6+0 = -3$.

Bio++. Bio++[36] is a `C++` library for Bioinformatics that includes methods for sequence comparison. The implementation of the Needleman-Wunsch-Gotoh method in the library can also generate sub-optimal alignments. Aligning the sequences `AAAGGG` and `TTAAAAGGGGTT` by assigning 0 for a match, -1 for a mismatch, -5 for gap opening, and -1 for gap extension with the command

```
./bpp AAAGGG TTAAAAGGGGTT 0 -1 -5 -1
```

yields the following alignment with a score of -20:

```
-  -  -  -  -  A  A  A  -  G  G  G
               |  .  .     |  .  .
T  T  A  A  A  A  G  G  G  G  T  T
```

However, the following alignment has a better score of -15:

[5] Y. Zhang, `http://zhanglab.ccmb.med.umich.edu/NW-align`

```
A  A  A  -  -  -  -  -  -  G  G  G
.  .  |                    |  .  .
T  T  A  A  A  A  G  G  G  G  T  T
```

The sequences and parameters used here, are the same as used by Altschul [7] to demonstrate the error in Gotoh's description of the traceback method. Interestingly, we observed another irregularity using Bio++. Running the implementation with the following options:

```
./bpp AAATTTGC CGCCTTAC 10 -30 -40 -1
```

where the third argument (10) is the match score, the forth argument (-30) is the mismatch score and the last two arguments are the gap opening (-40) and extension costs (-1), yields the alignment.

```
A  A  A  T  T  T  G  C  -  -  -  -  -  -  -  -
                    |
-  -  -  -  -  -  -  C  G  C  C  T  T  A  C
```

Surprisingly, flipping the input sequences

```
./bpp CGCCTTAC AAATTTGC 10 -30 -40 -1
```

yields a different alignment with a different score:

```
C  G  C  C  T  T  A  C  -  -  -  -  -  -  -  -

-  -  -  -  -  -  -  -  A  A  A  T  T  T  G  C
```

Nonetheless, both alignments are sub-optimal, since the alignment

```
C  G  C  C  T  T  A  -  -  -  -  -  -  -  C
                  |                       |
-  -  -  -  -  -  A  A  A  T  T  T  G  C
```

yields a better score of -72 (compared to -84 and -96 respectively).

T-Coffee. The **T-Coffee** package [107] for sequence alignment also implements the Gotoh algorithm. The command line used to produce the results below is

```
./t_coffee al.fa -dp_mode gotoh_pair_wise -gapopen -40 ...
... -gapext -1 -tg_mode=0 -matrix=score.mat
```

where `al.fa` contains the sequences `TAAATTTGC` and `TCGCCTTAC`. The gap opening penalty is -40, the gap extension penalty -1. The file `score.mat` defines a match score of 10 and a uniform mismatch score of -30. The resulting alignment as computed with T-Coffee is:

```
T  A  A  A  T  T  T  G  -  -  -  -  -  C
|              .  |                    |
T  -  -  -  -  -  C  G  C  C  T  T  A  C
```

This alignment is sub-optimal. Consider the following alternative alignment:

```
-  -  -  -  -  -  T  A  A  A  T  T  T  G  C
                  |  |                    |
T  C  G  C  C  T  T  A  -  -  -  -  -  -  C
```

For the given parameters, the alignment returned by **T-Coffee** has a score of -90. However, the alternative alignment above, has a score of -62.

It might well be that the error in the pair-wise alignment also affects the MSA algorithm in T-Coffee. However, T-Coffee does not only execute sequence-sequence, profile-sequence, or profile-profile alignments steps in the progressive MSA algorithm, but also uses additional concepts (e.g., the alignment information library). Therefore, it was not possible to reliably assess if this errors also affects the MSA procedure.

FOGSAA. The authors in [21] describe a branch-and-bound algorithm for global alignment that outperforms (in terms of speed) any optimal global alignment method including the widely used NW algorithm. Upon request via email, the authors provided us their implementation. To assess the correctness and speed of FOGSAA, the authors compared it to their own re-implementation of the NW algorithm. However, we obtained sub-optimal solutions when using this NW implementation to globally align sequences with affine gap penalties. For instance, given the sequences `AAATTTGC` and `CGCCTTAC` with the parameters *match* 10, *mismatch* -30, *gap opening* -40 and *gap extension* -1, we obtain the following alignment:

```
A  A  A  T  T  T  G  C  -  -  -  -  -
.              |  |
C  -  -  -  -  -  G  C  C  T  T  A  C
```

with a score of -100. The command we used is:

```
./nw s1.txt s2.txt 1 1 10 -30 -40 -1
```

However, the following alignment is the optimal solution for this example:

```
-  -  -  -  -  -  A  A  A  T  T  T  G  C
                  |                    |
C  G  C  C  T  T  A  -  -  -  -  -  -  C
```

with a score of -72.

HUSAR, MATLAB & BioPython. Several implementations make the assumption that an insertion can not be followed directly by a deletion (or vice versa) in the optimal alignment. An algorithm that performs well (i.e., generates optimal alignments) under this assumption is the one by Durbin [35]. **HUSAR** is the information system of the DKFZ (German Cancer Research) and comprises several applications for sequence analysis. One such application is **GAP**, which performs pairwise sequence alignment and allows for affine gaps. While experimenting with it, we found that, **GAP** yields optimal alignments under the assumption that an insertion cannot follow a deletion (or vice versa). For instance, given a match score of 10, a mismatch of -30, gap opening -25, and gap extension -1, it generates the following alignment

```
-  -  A  G  A  T
      .        |
C  T  C  -  -  T
```

with score -74. The parameters are passed with:

```
gap -MATRix=score.cmp -ENDWeight
```

where `-MATRix` is the substitution matrix file name and `-ENDWeight` ensures that end gaps are also penalized. Assuming that, insertions and deletions can not reside immediately next to each other, this *is* the optimal solution. However, if we omit this assumption, the optimal alignment is

```
-  -  -  A  G  A  T
                  |
C  T  C  -  -  -  T
```

with a score of -46.

The corresponding function (*nwalign*()) in **MATLAB**[6] yields an equivalent (in terms of alignment score) solution to **GAP**:

```
-  -  C  T  C  T
      .  |
A  G  A  T  -  -
```

The **MATLAB** call is:

[6] ©2015 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See `www.mathworks.com/trademarks` for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

```
nwalign('CTCT','AGAT', 'Alphabet', 'NT', 'ScoringMatrix', M,...
```

```
... 'GapOpen', 25, 'ExtendGap', 1)
```

Note that, **MATLAB** returns a score of -72 for this alignment. This is due to the different possible interpretations of affine gap scores. That is, a gap of length k can contribute to the score with $v + (k-1)u$ instead of $v + ku$. Alternatively, one can apply a gap opening penalty of -26 to get the score of -74 reported by **GAP** for this alignment. The module **pairwise2**[7] of the Biopython library [26] behaves analogously. The function

```
alignments = pairwise2.align.globalms("AGAT", "CTCT",...
```

```
...10, -30, -25, -1)
```

also yields alignments (including those found by **GAP** and **MATLAB**) with a score of -72. All three software packages do apparently not allow for insertions that are immediately followed by deletions. However, they do accept input values for which the optimal alignment does not exhibit this property.

nwalign. The **nwalign**[8] implementation is a python library (actually written in C) which implements global alignment with affine gaps. In some cases, it produces sub-optimal alignments as well. Again, consider the example of `AGAT` and `CTCT`. Given the same setup that we used for **HUSAR** (**GAP**), that is, a match score of 10, mismatch of -30, gap opening -25, and gap extension -1. The command:

```
./nwalign -gap_open -25 -gap_extend -1 -match 10 ...
```

```
...-matrix MATRIX AGAT CTCT
```

generates the correct alignment:

```
-  -  -  A  G  A  T
                  |
C  T  C  -  -  -  T
```

However, changing the scoring scheme to penalize opening a gap with -30 instead of -25 generates the following sub-optimal alignment:

```
-  -  A  G  A  T
      .        |
C  T  C  -  -  T
```

[7] Available at `http://biopython.org/DIST/docs/api/Bio.pairwise2-module.html`

[8] This program (*nwalign*) is available at `https://pypi.python.org/pypi/nwalign/`

Lecture slides: To further quantify the impact of the problem, we classified 31 lecture slides reported as the most popular results of Google search for the terms *global alignment*, *affine gaps*, *Needleman-Wunsch*, *Gotoh Algorithm*, into three distinct categories: *Correct*, *incomplete* and *wrong*.

We observed that the majority (≈52.6%) of the slides (16 lectures) are incomplete, since the initialization of the matrices is not explicitly given. Of course, lecture slides are only a part of the actual lectures. Hence, from the available resources we can not judge with certainty, whether an initialization (correct or incorrect) was presented to the students, for example orally, or via additional course material.

Approximately 25.8% of the slides (8 lectures) are correct. That is, a quadratic time algorithm is presented and a correct initialization is given. Slides that describe algorithms which make additional assumptions (e.g., Durbin [35]) are classified as correct if the initialization is correct for that particular case.

The remaining 22.6% of the slides (7 lectures) are wrong, that is, an incorrect initialization as described in Section 9.3 is provided. Other mistakes, such as stating incorrect conditions for avoiding subsequent insertions and deletions in the optimal alignment, are not counted as mistakes here.

Slides that only describe the algorithm for locally aligning two sequences, without giving an algorithm for globally aligning sequences were discarded.

List of incomplete lectures (16):

"http://www.cs.utoronto.ca/~brudno/csc2427/Lec8Notes.pdf"
"ftp://statgen.ncsu.edu/pub/thorne/bioinf2/gotoh.pdf"
"http://www.cs.umd.edu/class/fall2011/cmsc858s/Gap_Scores.pdf"
"http://math.mit.edu/classes/18.417/Slides/alignment.pdf"
"http://users.ece.utexas.edu/~hvikalo/ee381v/lecture5h.pdf"
"http://ls11-www.cs.uni-dortmund.de/people/rahmann/teaching/ws2008-09/GrundlegendeBioinformatik/skript.pdf"
"http://www.csie.ntu.edu.tw/~kmchao/bioinformatics13spr/alignment.ppt"
"http://labs.bio.unc.edu/Vision/courses/162F02/03.pair.align.ppt,http://labs.bio.unc.edu/Vision/courses/162F02/04.mult.align.ppt"
"http://web.calstatela.edu/faculty/nwarter/courses/bioinfo/Bioinformatics_Sequence_Align_003.ppt"
"http://robotics.stanford.edu/~serafim/cs262/Slides/Lecture3.

ppt"
"http://bioinfo.ict.ac.cn/~dbu/AlgorithmCourses/Lectures/
Lec6-EditDistance.pdf"
"http://thor.info.uaic.ro/~ciortuz/SLIDES/pairAlign.pdf"
"http://www.cs.rice.edu/~nakhleh/COMP571/Slides/
SequenceAlignment-PairwiseDP.pdf"
"http://www.cs.tau.ac.il/~bchor/CG09/CG2-alignment.ppt"
"http://www.cs.bilkent.edu.tr/~calkan/teaching/cs481/slides/
cs481-Week4.2.pdf"
"http://angom.myweb.cs.uwindsor.ca/teaching/cs558/
558-Lecture3.pptx"

List of correct lectures (8):

"http://ab.inf.uni-tuebingen.de/teaching/ws06/albi1/script/
pairalign_script.pdf"
"http://users-cs.au.dk/cstorm/courses/AiBS_e14/slides/
AffineGapcost.pdf"
"http://www3.cs.stonybrook.edu/~rp/class/549f14/lectures/
CSE549-Lec04.pdf"
"http://www.bioinf.uni-freiburg.de/Lehre/Courses/2014_SS/V_
Bioinformatik_1/gap-penalty-gotoh.pdf"
"http://www.comp.nus.edu.sg/~ksung/algo_in_bioinfo/slides/Ch2_
sequence_similarity.pdf"
"http://www.cs.cmu.edu/~ckingsf/class/02-714/Lec08-gaps.pdf"
"http://www.csie.ntu.edu.tw/~kmchao/seq11spr/Presentation_
Sequence-final.pptx"
"http://wwwmayr.informatik.tu-muenchen.de/lehre/2009SS/cb/
slides/CB1-2009-06-19.pdf"

List of lectures containing mistake (7):

http://math.ucdenver.edu/~billups/courses/ma5610/lectures/
lec4.pdf
http://www.cise.ufl.edu/~cap5510fa13/02-CAP5510-Fall13.pptx
http://www.cs.uku.fi/~kilpelai/BSA05/lectures/print10.pdf
http://www.cse.msu.edu/~torng/Classes/Archives/cse960.01/
Lectures/SequenceAlignment.ppt
http://www.haverford.edu/biology/GenomicsCourse/manduchi.ppt

http://www.site.uottawa.ca/~lucia/courses/5126-10/lecturenotes/03-05SequenceSimilarity.pdf
https://www.site.uottawa.ca/~turcotte/teaching/csi-5126/lectures/04/handouts.pdf

9.5 Conclusion

We have seen that, even though Gotoh published his findings more than thirty years ago, the initial ambiguous definition still result in wrong interpretations today. Reputed Books, widely used software, and university lecture slides on this topic contain this error. Adding to the confusion in this field is the availability of a plethora of different formulations for this algorithm.

We pointed out the mistakes in detail, and give a correct formulation, as well as an exemplary implementation, to avoid this problem.

Part III:

Addendum

10 Outlook and Future Work

Hardness of Model Assignment For the NP-hardness proof presented in Chapter 4 we made a few necessary assumptions.

First, the proof makes use of 9 distinct states and requires a minimum of 3 models. As we argued before, requiring 9 distinct states does not limit us in practice. Model selection is usually applied to protein data sets with 20 states. For data sets with lower numbers of states, such as 4 states for DNA or 2 states for binary data, a rate matrix is usually estimated from the data at hand.

For completeness, it is still interesting to know, for which number of states and models this problem remains NP-hard.

In particular, knowing, for what combination of states and models the question is polynomial time solvable remains important. For these cases, exact polynomial time algorithms may be devised to actually find an optimal model assignment.

The other open question is whether the results hold if we restrict ourselves to time-reversible models. For the proof in Chapter 4 we constructed three models which were in fact not time reversible. However, time reversibility is often assumed by phylogenetic software tools such as RAxML [130].

Again, if the model assignment problem is actually polynomial time solvable under this restriction, optimal model assignments may be used in practical tree inferences.

Given the NP-hardness proof presented here, further effort may also be invested into developing efficient heuristics for approximating the optimal model assignment (as done for example in [67]). This may allow for a more accurate phylogenetic tree reconstructions in the future.

Distribution of Partitions to Parallel Processors The algorithm presented in Chapter 5 is rather mature. There is little room for improvement on the actual algorithm for the stated problem.

Future effort may be invested into deriving related problems and applying the presented algorithm to them. Alternatively, more involved problem descriptions may be contrived, which build up on the original algorithm.

For example, computational costs for sites may not be constant. Instead, the cost may depend on which processor a site is assigned to, or what other sites of the specific partition are computed by the same processor. One such example is the application of subtree repeats to phylogenetic likelihood calculations, as done in Chapter 8. Repeating site patterns may then only

be calculated for the same partition on the same processor. Obviously, if two repeating sites are assigned to the same processor, the computations for one such site may be skipped entirely. If, however, both sites are computed by different processors, both processors may calculate each site in full (or only partially, depending on other sites assigned to the processors).

Calculating the Internode Certainty and Related Measures on Partial Gene Trees There is no clear indicator of which method is the best (let alone which method is correct) for calculating the internode certainty (and related measures) on partial gene trees. Thus, many more variations for distributing the frequencies may be devised and tested.

One possible improvement is to incorporate the correlation of placements of taxa in a tree set. Let two taxa appear in the same bipartition with high frequency (frequency close to 1.0) throughout all trees that contain both taxa in the tree set. Intuitively, the probability of placing these taxa together in a bipartition where at least one is missing could be assumed to be higher than 0.5 (that is, not uniform for both sets of the bipartition).

Calculating Subtree Repeats on General Trees The problem of calculating subtree repeats for arbitrary trees in linear time is addressed rather exhaustively in Chapter 7.

Future work may include the application to different scientific fields. For example, applications in language processing come to mind. Repeating words, phrases, or sentences could quickly be recognized. Automatic content detection or plagiarism detection are possible applications for this. Furthermore, pattern, or file compression methods may be developed using our algorithms.

Another generalization of subtree repeats is the detection of repeating structures in general graphs. Special graph structures, such as planar graphs, bipartite graphs, or directed connected acyclic graphs may potentially be analyzed using similar methods.

Application of Subtree Repeats to Phylogenetic Trees The main concern of Chapter 8 is an efficient calculation of the likelihood function. Thus, improvements and optimizations are always possible for any implementation.

Furthermore, actual tree search algorithms must be implemented to work in conjunction with our method for efficiently computing the conditional

likelihood vectors. Actual maximum likelihood or Bayesian inferences are beyond the implementation we present here. The algorithm we developed can, in future work, be incorporated into existing software tools for these tasks.

Additionally, the implementation of subtree repeat detection must keep up with current developments in computer hardware architectures. Implementation improvements may come in the form parallel processing capabilities or more efficient vectorization schemes.

Other topics Beyond the topics presented in this dissertation, there are many other exciting open questions in the field on molecular evolution, in particular phylogenetics, and next generation sequencing.

For example, an abstract view of the tree space that discards the notion of topology can be envisioned. This may allow us to avoid the process of random tree generation, for example via NNI or SPR moves (see Figure 3.10, Section 3.4, page 25). Instead, only distances between taxa are estimated. Obviously, not all combinations of distances between pairs of taxa are compatible. Thus, the distances are not independent of one another. In fact, many such combinations will not result in valid phylogenies.

However, iterative decision making for the distances between taxa, and certain limitations on the distances can guarantee an underlying assumed topology. Optimization methods, similar to those used for branch length optimization, such as the Newton-Raphson method may be used to obtain reasonable results. Alternatively, simple distance-based methods may yield a starting phylogeny, to which the above mentioned optimization (Newton-Raphson or similar) is iteratively applied.

For a data set with m taxa we may thus reduce the problem of finding the optimal tree topology to the $m \times m$ dimensional, possibly m dimensional, Euclidean space.

References

[1] Federico Abascal, Rafael Zardoya, and David Posada. ProtTest: selection of best-fit models of protein evolution. *Bioinformatics*, 21(9):2104–2105, 2005.

[2] Andre J Aberer, Kassian Kobert, and Alexandros Stamatakis. ExaBayes: Massively Parallel Bayesian Tree Inference for the Whole-Genome Era. *Molecular Biology and Evolution*, 31(10):2553–2556, 2014.

[3] Louigi Addario-Berry, Benny Chor, Mike Hallett, Jens Lagergren, Alessandro Panconesi, and Todd Wareham. Ancestral Maximum Likelihood of Evolutionary Trees is Hard. *Journal of Bioinformatics and Computational Biology*, 2(2), 2004.

[4] Alfred V. Aho, John E. Hopcroft, and Jeffrey Ullman. *Data Structures and Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1983.

[5] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: principles, techniques, and tools*. Addison Wesley, 2 edition, 2006.

[6] N. Alachiotis and A. Stamatakis. A generic and versatile architecture for inference of evolutionary trees under maximum likelihood. *Signals, Systems and Computers (ASILOMAR), 2010 Conference Record of the Forty Fourth Asilomar Conference on*, pages 829–835, 2010.

[7] Stephen F. Altschul and Bruce W. Erickson. Optimal sequence alignment using affine gap costs. *Bulletin of Mathematical Biology*, 48(5/6):603–616, 1986.

[8] M. J. Alves, H. Coelho, M. J. Collares-Pereira, and M. M. Coelho. Mitochondrial DNA variation in the highly endangered cyprinid fish Anaecypris hispanica: importance for conservation. *Heredity*, 87(4):463–473, 2001.

[9] Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A Linear-Time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas. *Information Processing Letters*, 8(3):121–123, 1979.

[10] David R. Barstow, Howard E. Shrobe, and Erik Sandewall. *Interactive Programming Environments*. McGraw-Hill, Inc., 1984.

[11] C Basler, A Reid, J Dybing, T Janczewski, T Fanning, H Zheng, M Salvatore, D Perdue, M andand Swayne, A Garcia-Sastre, P Palese, and J Taubenberger. Sequence of the 1918 pandemic influenza virus nonstructural gene (NS) segment and characterization of recombinant viruses bearing the 1918 NS genes. *Proceedings of the National Academy of Sciences of the United States of America*, 98:2746–51, 2001.

[12] T. Bayes and R. Price. An Essay towards Solving a Problem in the Doctrine of Chances. By the Late Rev. Mr. Bayes, F. R. S. Communicated by Mr. Price, in a Letter to John Canton, A. M. F. R. S. *Philosophical Transactions*, 53:370–418, 1763.

[13] Veeravalli Bharadwaj, Debasish Ghose, and ThomasG. Robertazzi. Divisible Load Theory: A New Paradigm for Load Scheduling in Distributed Systems. *Cluster Computing*, 6(1):7–17, 2003.

[14] R.N. Bhattacharya and E.C. Waymire. *Stochastic Processes with Applications*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 1990.

[15] Jacek Błażewicz and Maciej Drozdowski. Distributed Processing of Divisible Jobs with Communication Startup Costs. *Discrete Applied Mathematics*, 76(1-3):21–41, June 1997.

[16] D. Bryant. A classification of consensus methods for phylogenies. In M. Janowitz, F.-J. Lapointe, F.R. McMorris, B. Mirkin, and F.S. Roberts, editors, *Bioconsensus*, Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, pages 163–184, 2003.

[17] D. Bryant, N. Galtier, and M.-A. Poursat. Likelihood calculations in molecular phylogenetics. In O. Gascuel, editor, *Mathematics of evolution and phylogeny*, pages 33–62. Oxford University Press, 2005.

[18] T. R. Buckley. Model misspecification and probabilistic tests of topology: evidence from empirical data sets. *Systematic Biology*, 51(3):509–523, 2002.

[19] T.R. Buckley and C.W. Cunningham. The effects of nucleotide substitution model assumptions on estimates of nonparametric bootstrap support. *Molecular Biology and Evolution*, 19(4):394–405, 2002.

[20] Joseph H. Camin and Robert R. Sokal. A Method for Deducing Branching Sequences in Phylogeny. *Evolution*, 19(3):311–326, 1965.

[21] Angana Chakraborty and Sanghamitra Bandyopadhyay. FOGSAA: Fast optimal global sequence alignment algorithm. *Scientific Reports*, 3, 2013.

[22] Benny Chor, Michael D. Hendy, Barbara R. Holland, and David Penny. Multiple Maxima of Likelihood in Phylogenetic Trees: An Analytic Approach. *Molecular Biology and Evolution*, 17(10):1529–1541, 2000.

[23] Benny Chor and Tamir Tuller. Finding the Maximum Likelihood Tree is Hard. In *Journal of the ACM*, 2005.

[24] Michalis Christou, Maxime Crochemore, Tomas Flouri, Costas S. Iliopoulos, Jan Janoušek, Borivoj Melichar, and Solon P. Pissis. Computing all subtree repeats in ordered trees. *Information Processing Letters*, 112(24):958–962, 2012.

[25] Michalis Christou, Maxime Crochemore, Tomáš Flouri, Costas S. Iliopoulos, Jan Janoušek, Bořivoj Melichar, and Solon P. Pissis. Computing All Subtree Repeats in Ordered Ranked Trees. In Roberto Grossi, Fabrizio Sebastiani, and Fabrizio Silvestri, editors, *String Processing and Information Retrieval*, volume 7024 of *Lecture Notes in Computer Science*, pages 338–343. Springer, 2011.

[26] Peter J A Cock, Tiago Antao, Jeffrey T Chang, Brad A Chapman, Cymon J Cox, Andrew Dalke, Iddo Friedberg, Thomas Hamelryck, Frank Kauff, Bartek Wilczynski, and Michiel J L de Hoon. Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25(11):1422–3, 2009.

[27] Stephen A. Cook. The complexity of theorem-proving procedures. *Proceedings of the 3rd annual ACM Symposium on Theory of Computing, STOC*, pages 151 – 158, 1971.

[28] Fred J. Damerau. A Technique for Computer Detection and Correction of Spelling Errors. *Communications of the ACM*, 7(3):171–176, March 1964.

[29] Charles Darwin. *On the Origin of Species by means of natural selection; or, The preservation of favoured races in the struggle for life.* New York :D. Appleton and Co., 1859.

[30] W. H. E. Day and D. Sankoff. Computational complexity of inferring phylogenies by compatibility. *Systematic Zoology*, 35(2), 1986.

[31] M. O. Dayhoff, R. M. Schwatz, and B. C. Orcutt. A model of evulutionary change in proteins. *Atlas of protein sequence and structure*, 5(3):345–352, 1978.

[32] Tulio de Oliveira, Oliver G. Pybus, Andrew Rambaut, Marco Salemi, Sharon Cassol, Massimo Ciccozzi, Giovanni Rezza, Guido C. Gattinara, Roberta D'Arrigo, Massimo Amicosante, Luc Perrin, Vittorio Colizzi, Carlo F. Perno, and Benghazi Study Group. Molecular Epidemiology: HIV-1 and HCV sequences from Libyan outbreak. *Nature*, 444(7121):836–837, 2006.

[33] Andreas Doring, David Weese, Tobias Rausch, and Knut Reinert. SeqAn An efficient, generic C++ library for sequence analysis. *BMC Bioinformatics*, 9(1):11, 2008.

[34] Bernard Dujon. Yeast evolutionary genomics. *Nature Reviews Genetics*, 11(7):512–524, 2010.

[35] Richard Durbin, Sean Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge University Press, 1998.

[36] Julien Dutheil, Sylvain Gaillard, Eric Bazin, Sylvain Glemin, Vincent Ranwez, Nicolas Galtier, and Khalid Belkhir. Bio++: a set of C++ libraries for sequence analysis, phylogenetics, molecular evolution and population genetics. *BMC Bioinformatics*, 7(1):188, 2006.

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

[38] Bradley Efron, Elizabeth Halloran, and Susan Holmes. Bootstrap confidence levels for phylogenetic trees. *Proceedings of the National Academy of Sciences*, 93(23), 1996.

[39] Isaac Elias. Settling the Intractability of Multiple Alignment. In *Proceedings of the 14th Annual International Symposium on Algorithms and Computation (ISAAC*, pages 352–363. Springer, 2003.

[40] J. Felsenstein. Maximum likelihood and minimum-steps methods for estimating evolutionary trees from data on descrete characters. *Sytematic Zoology*, 22:240–249, 1973.

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

[42] J. Felsenstein. Confidence limits on phylogenies: an approach using the bootstrap. *Annals of Statistics*, 39:783–791, 1985.

[43] J. Felsenstein. *Inferring Phylogenies*. Sinauer Associates, 2004.

[44] J. Felsenstein and G. A. Churchill. A Hidden Markov Model approach to variation among sites in rate of evolution. *Molecular Biology and Evolution*, 13:93–104, 1996.

[45] Joe Felsenstein. PHYLIP (Phylogeny Inference Package) version 3.5c, 1993.

[46] Christian Ferdinand, Helmut Seidl, and Reinhard Wilhelm. Tree automata for code selection. *Acta Informatica*, 31:741–760, 1994.

[47] Walter M. Fitch. Toward Defining the Course of Evolution: Minimum Change for a Specific Tree Topology. *Systematic Zoology*, 20(4):406–416, 1971.

[48] David Fitzpatrick, Mary Logue, Jason Stajich, and Geraldine Butler. A fungal phylogeny based on 42 complete genomes derived from supertree and combined gene analysis. *BMC Evolutionary Biology*, 6(1):99, 2006.

[49] T. Flouri, F. Izquierdo-Carrasco, D. Darriba, A.J. Aberer, L.-T. Nguyen, B.Q. Minh, A. von Haeseler, and A. Stamatakis. The phylogenetic likelihood library. *Systematic Biology*, 2014.

[50] Tomas Flouri, Emanuele Giaquinta, Kassian Kobert, and Esko Ukkonen. Longest common substrings with k mismatches. *Information Processing Letters*, 115(6-8):643 – 647, 2015.

[51] Tomáš Flouri, Kassian Kobert, Solon P Pissis, and Alexandros Stamatakis. An optimal algorithm for computing all subtree repeats in trees. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 372(2016):20130140, 2014.

[52] Tomáš Flouri, Kassian Kobert, SolonP. Pissis, and Alexandros Stamatakis. An Optimal Algorithm for Computing All Subtree Repeats in Trees. In Thierry Lecroq and Laurent Mouchard, editors, *Combinatorial Algorithms*, volume 8288 of *Lecture Notes in Computer Science*, pages 269–282. Springer Berlin Heidelberg, 2013.

[53] Tomáš Flouri, Kassian Kobert, Torbjørn Rognes, and Alexandros Stamatakis. Are all global alignment algorithms and implementations correct? *bioRxiv*, 2015.

[54] Tomáš Flouri, Kassian Kobert, Solon P. Pissis, and Alexandros Stamatakis. A simple method for computing all subtree repeats in unordered trees in linear time. In *Festschrift for Borivoj Melichar*, pages 145–152. Czech Technical University in Prague, 2012.

[55] L. R. Foulds and R. L. Graham. The Steiner Problem in Phylogeny is NP-Complete. *Advances in Applied Mathematics*, 3, 1982.

[56] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[57] P.E. Gill, W. Murray, and M.H Wright. *Practical Optimization*. Academic Press, London, 1981.

[58] Teofilo F. Gonzalez. *Handbook of Approximation Algorithms and Metaheuristics*. Chapman & Hall/CRC, 2007.

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

[60] Osamu Gotoh. Multiple sequence alignment: Algorithms and applications. *Advances in Biophysics*, 36:159 – 206, 1999.

[61] Marc Gottschling, Alexandros Stamatakis, Ingo Nindl, Eggert Stockfleth, Angel Alonso, and Ignacio G. Bravo. Multiple Evolutionary Mechanisms Drive Papillomavirus Diversification. *Molecular Biology and Evolution*, 24(5):1242–1258, 2007.

[62] G.R. Grimmett and D.R. Stirzaker. *Probability and Random Processes*. Clarendon Press, Oxford, 1992.

[63] Stephane Guindon, Jean-Francois Dufayard, Vincent Lefort, Maria Anisimova, Wim Hordijk, and Olivier Gascuel. New Algorithms and

Methods to Estimate Maximum-Likelihood Phylogenies: Assessing the Performance of PhyML 3.0. *Systematic Biology*, 59(3):307–321, 2010.

[64] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, New York, NY, USA, 2009.

[65] Frank Harary. *Graph Theory.* Addison Wesley Publishing Company, 1994.

[66] W. K. Hastings. Monte Carlo Sampling Methods Using Markov Chains and Their Applications. *Biometrika*, 57(1):pp. 97–109, 1970.

[67] Jörg Hauser, Kassian Kobert, Fernando Izquierdo-Carrasco, Karen Meusemann, Bernhard Misof, Michael Gertz, and Alexandros Stamatakis. Heuristic Algorithms for the Protein Model Assignment Problem. In *Bioinformatics Research and Applications*, volume 7875 of *Lecture Notes in Computer Science*, pages 137–148. Springer, 2013.

[68] Tracy A. Heath, Mark T. Holder, and John P. Huelsenbeck. A dirichlet process prior for estimating lineage-specific substitution rates. *Molecular Biology and Evolution*, 2011.

[69] Andreas Hejnol, Matthias Obst, Alexandros Stamatakis, Michael Ott, Greg W. Rouse, Gregory D. Edgecombe, Pedro Martinez, Jaume Baguñà, Xavier Bailly, Ulf Jondelius, Matthias Wiens, Werner E. G. Müller, Elaine Seaver, Ward C. Wheeler, Mark Q. Martindale, Gonzalo Giribet, and Casey W. Dunn. Assessing the root of bilaterian animals with scalable phylogenomic methods. *Proceedings of the Royal Society of London B: Biological Sciences*, 2009.

[70] S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences of the United States of America*, 89(22), 1992.

[71] Paul G. Higgs and Teresa K. Attwood. *Sequence Alignment Algorithms*, pages 119–138. Blackwell Publishing Ltd., 2004.

[72] Christoph M. Hoffmann and Michael J. O'Donnell. Programming with Equations. *ACM Transactions on Programming Languages and Systems*, 4:83–112, 1982.

[73] Sun-Yuan Hsieh. Finding maximal leaf-agreement isomorphic descendent subtrees from phylogenetic trees with different species. *Theoretical Computer Science*, 370(1-3):299–308, 2007.

[74] Paul Hudak. Conception, evolution, and application of functional programming languages. *ACM Computing Surveys*, 21:359–411, 1989.

[75] Fernando Izquierdo-Carrasco, StephenA Smith, and Alexandros Stamatakis. Algorithms, data structures, and numerics for likelihood-based phylogenetic inference of huge trees. *BMC Bioinformatics*, 12(1), 2011.

[76] ErichD Jarvis, Siavash Mirarab, AndreJ Aberer, Bo Li, Peter Houde, Cai Li, SimonYW Ho, BrantC Faircloth, Benoit Nabholz, JasonT Howard, Alexander Suh, ClaudiaC Weber, RuteR da Fonseca, Alonzo Alfaro-Nunez, Nitish Narula, Liang Liu, Dave Burt, Hans Ellegren, ScottV Edwards, Alexandros Stamatakis, DavidP Mindell, Joel Cracraft, EdwardL Braun, Tandy Warnow, Wang Jun, MThomasPius Gilbert, and Guojie Zhang. Phylogenomic analyses data of the avian phylogenomics project. *GigaScience*, 4(1), 2015.

[77] D. T. Jones, W. R. Taylor, and J. M. Thorton. The rapid generation of mutation data matrices from protein sequences. *Computer Applications in the Biosciences*, 8:275–282, 1992.

[78] I King Jordan, Fyodor A Kondrashov, Igor B Rogozin, Roman L Tatusov, Yuri I Wolf, and Eugene V Koonin. Constant relative rate of protein evolution and detection of functional diversification among bacterial, archaeal and eukaryotic proteins. *Genome Biology*, 12(2), 2001.

[79] Richard Karp. Reducibility Among Combinatorial Problems. *Complexity of Computer Computations*, pages 85–103, 1972.

[80] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebra. In J. Leech, editor, *Computational problems in abstract algebra*, pages 263–297. Pergamon Press, 1970.

[81] Kassian Kobert, Tomáš Flouri, Andre Aberer, and Alexandros Stamatakis. The divisible load balance problem and its application to phylogenetic inference. In *Algorithms in Bioinformatics*, pages 204–216. Springer, 2014.

[82] Kassian Kobert, Jörg Hauser, and Alexandros Stamatakis. Is the Protein Model Assignment problem under linked branch lengths NP-hard? *Theoretical Computer Science*, 524:48–58, 2014.

[83] Kassian Kobert, Leonidas Salichos, Antonis Rokas, and Alexandros Stamatakis. Computing the internode certainty and related measures from partial gene trees. *Molecular Biology and Evolution*, 33(6):1606–1617, 2016.

[84] Kassian Kobert, Alexandros Stamatakis, and Tomáš Flouri. Efficient detection of repeating sites to accelerate phylogenetic likelihood calculations. *bioRxiv*, 2016.

[85] Katsuko Komatsu, Shu Zhu, Hirotoshi Fushimi, Tran Kim Qui, Shaoqing Cai, and Shigetoshi Kadota. Phylogenetic Analysis Based on 18S rRNA Gene and matK Gene Sequences of Panax vietnamensis and Five Related Species. *Planta Medica*, 67:461–465, 2001.

[86] C. Kosiol and N. Goldman. Different versions of the Dayhoff rate matrix. *Molecular Biology and Evolution*, 22:193–199, 2005.

[87] Sven Oliver Krumke and Hartmut Noltemeier. *Graphentheoretische Konzepte und Algorithmen*. Springer DE, 2009.

[88] N. Pradeep Kumar, K.P. Patra, S.L. Hoti, and P.K. Das. Genetic variability of the human filarial parasite, wuchereria bancrofti in south india. *Acta Tropica*, 82(1):67 – 76, 2002.

[89] Cletus P Kurtzman and Christie J Robnett. Phylogenetic relationships among yeasts of the 'Saccharomyces complex' determined from multigene sequence analyses. *FEMS Yeast Research*, 3(4):417–432, 2006.

[90] Clemens Lakner, Paul van der Mark, John P. Huelsenbeck, Bret Larget, and Fredrik Ronquist. Efficiency of Markov Chain Monte Carlo Tree Proposals in Bayesian Phylogenetics. *Systematic Biology*, 57(1):86–103, 2008.

[91] Robert Lanfear, Brett Calcott, Simon Y. W. Ho, and Stephane Guindon. PartitionFinder: Combined Selection of Partitioning Schemes and Substitution Models for Phylogenetic Analyses. *Molecular Biology and Evolution*, 29(6):1695–1701, 2012.

[92] Pierre Simon Laplace. Memoir on the Probability of the Causes of Events. *Statistical Science*, 1(3):364–378, 1986.

[93] B Larget and DL Simon. Markov Chasin Monte Carlo Algorithms for the Bayesian Analysis of Phylogenetic Trees. *Molecular Biology and Evolution*, 16(6):750, 1999.

[94] Ernest K. Lee, Angelica Cibrian-Jaramillo, Sergios-Orestis Kolokotronis, Manpreet S. Katari, Alexandros Stamatakis, Michael Ott, Joanna C. Chiu, Damon P. Little, Dennis Wm. Stevenson, W. Richard McCombie, Robert A. Martienssen, Gloria Coruzzi, and Rob DeSalle. A Functional Phylogenomic View of the Seed Plants. *PLoS Genetics*, 7(12), 12 2011.

[95] P. Lemey, M. Salemi, and A.M. Vandamme. *The Phylogenetic Handbook: A Practical Approach to Phylogenetic Analysis and Hypothesis Testing*. Cambridge University Press, 2009.

[96] A.R. Lemmon and E.C. Moriarty. The importance of proper model assumption in Bayesian phylogenetics. *Systematic Biology*, 53(2):265–277, 2004.

[97] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Doklady Akademii Nauk SSSR*, 163(4):845–848, 1965.

[98] Giancarlo Mauri and Giulio Pavesi. Algorithms for pattern matching and discovery in RNA secondary structure. *Theoretical Computer Science*, 335(1):29–51, 2005.

[99] Gregor Mendel. Versuche über Pflanzen-Hybriden. *Verhandlungen des naturforschenden Vereines in Brünn*, 42:3–47, 1866.

[100] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[101] K. Meusemann, B.M. von Reumont, S. Simon, F. Roeding, S. Strauss, P. Kück, I. Ebersberger, M. Walzl, G. Pass, S. Breuers, et al. A phylogenomic approach to resolve the arthropod tree of life. *Molecular Biology and Evolutioni*, 27(11):2451–2464, 2010.

[102] W. Min Jou, G. Haegeman, M. Ysebaert, and W. Fiers. Nucleotide sequence of the gene coding for the bacteriophage MS2 coat protein. *Nature*, 237(5350):82–88, 1972.

[103] Bui Quang Minh, Le Sy Vinh, Arndt von Haeseler, and Heiko A. Schmidt. pIQPNNI: parallel reconstruction of large maximum likelihood phylogenies. *Bioinformatics*, 21(19):3794–3796, 2005.

[104] Bernhard Misof, Shanlin Liu, Karen Meusemann, Ralph S. Peters, Alexander Donath, Christoph Mayer, Paul B. Frandsen, Jessica Ware, Tomas Flouri, Rolf G. Beutel, Oliver Niehuis, Malte Petersen, Fernando Izquierdo-Carrasco, Torsten Wappler, Jes Rust, Andre J. Aberer, Ulrike Aspöck, Horst Aspöck, Daniela Bartel, Alexander Blanke, Simon Berger, Alexander Böhm, Thomas R. Buckley, Brett Calcott, Junqing Chen, Frank Friedrich, Makiko Fukui, Mari Fujita, Carola Greve, Peter Grobe, Shengchang Gu, Ying Huang, Lars S. Jermiin, Akito Y. Kawahara, Lars Krogmann, Martin Kubiak, Robert Lanfear, Harald Letsch, Yiyuan Li, Zhenyu Li, Jiguang Li, Haorong Lu, Ryuichiro Machida, Yuta Mashimo, Pashalia Kapli, Duane D. McKenna, Guanliang Meng, Yasutaka Nakagaki, Jose Luis Navarrete-Heredia, Michael Ott, Yanxiang Ou, Günther Pass, Lars Podsiadlowski, Hans Pohl, Björn M. von Reumont, Kai Schütte, Kaoru Sekiya, Shota Shimizu, Adam Slipinski, Alexandros Stamatakis, Wenhui Song, Xu Su, Nikolaus U. Szucsich, Meihua Tan, Xuemei Tan, Min Tang, Jingbo Tang, Gerald Timelthaler, Shigekazu Tomizuka, Michelle Trautwein, Xiaoli Tong, Toshiki Uchifune, Manfred G. Walzl, Brian M. Wiegmann, Jeanne Wilbrandt, Benjamin Wipfler, Thomas K. F. Wong, Qiong Wu, Gengxiong Wu, Yinlong Xie, Shenzhou Yang, Qing Yang, David K. Yeates, Kazunori Yoshizawa, Qing Zhang, Rui Zhang, Wenwei Zhang, Yunhui Zhang, Jing Zhao, Chengran Zhou, Lili Zhou, Tanja Ziesmann, Shijie Zou, Yingrui Li, Xun Xu, Yong Zhang, Huanming Yang, Jian Wang, Jun Wang, Karl M. Kjer, and Xin Zhou. Phylogenomics resolves the timing and pattern of insect evolution. *Science*, 346(6210):763–767, 2014.

[105] Eugene W. Myers and Webb Miller. Optimal alignments in linear space. *Computer Applications in the Biosciences*, 4(1):11–17, 1988.

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

[107] C. Notredame, D.G. Higgins, and J Heringa. T-Coffee: A novel method for fast and accurate multiple sequence alignment. *Journal of Molecular Biology*, 302(1):205 – 2017, 2000.

[108] Cynthia Phillips and Tandy J. Warnow. The asymmetric median tree - A new model for building consensus trees. *Discrete Applied Mathematics*, 71(1-3):311 – 335, 1996.

[109] Sergei L. Kosakovsky Pond and Spencer V. Muse. Column Sorting: Rapid Calculation of the Phylogenetic Likelihood Function. *Systematic Biology*, 53(5):685–692, 2004.

[110] P. Rice, I. Longden, and A. Bleasby. EMBOSS: the European Molecular Biology Open Software Suite. *Trends Genet*, 16(6):276–7, 2000.

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

[112] Sebastien Roch. A Short Proof that Phylogenetic Tree Reconstruction by Maximum Likelihood Is Hard. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(1), 2006.

[113] A. Rokas, B. L. Williams, N. King, and S. B. Carroll. Genome-scale approaches to resolving incongruence in molecular phylogenies. *Nature*, 425(6960):798–804, 2003.

[114] Fredrik Ronquist, Maxim Teslenko, Paul van der Mark, Daniel L Ayres, Aaron Darling, Sebastian Höhna, Bret Larget, Liang Liu, Marc a Suchard, and John P Huelsenbeck. MrBayes 3.2: efficient Bayesian phylogenetic inference and model choice across a large model space. *Systematic Biology*, 61(3):539–42, 2012.

[115] Leonidas Salichos and Antonis Rokas. Inferring ancient divergences requires genes with strong phylogenetic signals. *Nature*, 2013.

[116] Leonidas Salichos, Alexandros Stamatakis, and Antonis Rokas. Novel Information Theory-Based Measures for Quantifying Incongruence among Phylogenetic Trees. *Molecular Biology and Evolution*, 2014.

[117] Michael J. Sanderson, Michelle M. McMahon, and Mike Steel. Terraces in Phylogenetic Tree Space. *Science*, 333(6041):448–450, 2011.

[118] David Sankoff. Matching Sequences under Deletion/Insertion Constraints. *Proceedings of the National Academy of Sciences*, 69(1):4–6, 1972.

[119] T. J. Schaefer. The complexity of satisfiability problems. *Proceedings of the 10th annual ACM Symposium on Theory of Computing, STOC*, pages 216 – 226, 1978.

[120] David B. Searls. Pharmacophylogenomics: genes, evolution and drug targets. *Nature Reviews Drug Discovery*, 2:613–623, 2003.

[121] P. Sellers. On the Theory and Computation of Evolutionary Distances. *SIAM Journal on Applied Mathematics*, 26(4):787–793, 1974.

[122] João Carlos Setubal and J. Meidanis. *Introduction to Computational Molecular Biology*. Computer Science Series. PWS Pub., 1997.

[123] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27, 1948.

[124] Jian Shi, Yiwei Zhang, Haiwei Luo, and Jijun Tang. Using jackknife to assess the quality of gene order phylogenies. *BMC Bioinformatics*, 11(1):168, 2010.

[125] Stephen Smith, Michael Moore, Joseph Brown, and Ya Yang. Analysis of phylogenomic datasets reveals conflict, concordance, and gene duplications with examples from animals and plants. *BMC Evolutionary Biology*, 15(1):150, 2015.

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

[127] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

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

[129] Alexandros Stamatakis. Phylogenetics: Applications, Software and Challenges. *Cancer Genomics - Proteomics*, 2(5):301–305, 2005.

[130] Alexandros Stamatakis. RAxML Version 8: A tool for Phylogenetic Analysis and Post-Analysis of Large Phylogenies. *Bioinformatics*, 2014.

[131] Alexandros Stamatakis and Andre J. Aberer. Novel Parallelization Schemes for Large-Scale Likelihood-based Phylogenetic Inference. In *IPDPS Workshops*, pages 1195–1204, 2013.

[132] A.P. Stamatakis, T. Ludwig, H. Meier, and M.J. Wolf. AxML: a fast program for sequential and parallel phylogenetic tree calculations based on the maximum likelihood method. In *Bioinformatics Conference, 2002. Proceedings. IEEE Computer Society*, pages 21–28, 2002.

[133] Mike Steel. The Maximum Likelihood Point for a Phylogenetic Tree is not Unique. *Systematic Biology*, 43(4), 1994.

[134] C.A. Stewart, D. Hart, D.K. Berry, G.J. Olsen, E.A. Wernert, and W. Fischer. Parallel implementation and performance of fastdnaml - a program for maximum likelihood phylogenetic inference. In *Supercomputing, ACM/IEEE 2001 Conference*, pages 32–32, 2001.

[135] J.G. Sumner and M.A. Charleston. Phylogenetic estimation with partial likelihood tensors. *Journal of Theoretical Biology*, 262(3):413 – 424, 2010.

[136] S. Tavaré. Some Probabilistic and Statistical Problems in the Analysis of DNA Sequences. *In American Mathematical Society: Lectures on Mathematics in the Life Sciences*, 17, 1986.

[137] Philip Taylor. A fast homology program for aligning biological sequences. *Nucleic Acids Research*, 12(1):447–455, 1984.

[138] J. D. Thompson, D. G. Higgins, and T. J. Gibson. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22(22):4673–4680, 1994.

[139] Tanya L. Trepanier and Robert W. Murphy. The coachella valley fringe-toed lizard (uma inornata): Genetic diversity and phylogenetic relationships of an endangered species. *Molecular Phylogenetics and Evolution*, 18(3):327 – 334, 2001.

[140] Mario Valle, Hannes Schabauer, Christoph Pacher, Heinz Stockinger, Alexandros Stamatakis, Marc Robinson-Rechavi, and Nicolas Salamin. Optimization strategies for fast detection of positive selection on phylogenetic trees. *Bioinformatics*, 30(8):1129–1137, 2014.

[141] B. Veeravalli, X. Li, and Chi-Chung Ko. On the influence of start-up costs in scheduling divisible loads on bus networks. *Parallel and Distributed Systems, IEEE Transactions on*, 11(12):1288–1305, Dec 2000.

[142] T.K. Vintsyuk. Speech discrimination by dynamic programming. *Cybernetics*, 4(1):52–57, 1968.

[143] Robert A. Wagner and Michael J. Fischer. The String-to-String Correction Problem. *J. ACM*, 21(1):168–173, January 1974.

[144] M. S. Waterman. *Introduction to Computational Biology: Maps, Sequences and Genomes*. Chapman & Hall, London, 1995.

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

[146] Renxiang Yan, Dong Xu, Jianyi Yang, Sara Walker, and Yang Zhang. A comparative assessment and analysis of 20 representative sequence alignment methods for protein structure prediction. *Scientific Reports*, 3, 2013.

[147] Ziheng Yang. Maximum likelihood phylogenetic estimation from dna sequences with variable rates over sites: Approximate methods. *J. Mol. Evol.*, 39(3):306–314, 1994.

[148] Ziheng Yang. *Computational Molecular Evolution*. Oxford University Press, 2006.

[149] Ziheng Yang. Paml 4: Phylogenetic analysis by maximum likelihood. *Molecular Biology and Evolution*, 24(8):1586–1591, 2007.

[150] N. Yutin, P. Puigbò, E.V. Koonin, and Y.I. Wolf. Phylogenomics of Prokaryotic Ribosomal Proteins. *PloS one*, 7(5), 2012.

[151] Jiajie Zhang, Kassian Kobert, Tomáš Flouri, and Alexandros Stamatakis. PEAR: a fast and accurate Illumina Paired-End reAd mergeR. *Bioinformatics*, 30(5):614–620, 2014.

[152] Jiajie Zhang and Alexandros Stamatakis. The Multi-Processor Scheduling Problem in Phylogenetics. In *IPDPS Workshops*, pages 691–698. IEEE Computer Society, 2012.