UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR
NEURO- UND BIOINFORMATIK

From the Institute for Neuro- and Bioinformatics of the University of Lübeck
Director: Prof. Dr. Thomas Martinetz

Fast Computation of Genome Distances

Dissertation

in Fulfillment of Requirements for the Doctoral Degree of the University of Lübeck
from the Department of Computer Sciences

Submitted by **Fabian Klötzl**, born in Hamburg, Germany

Lübeck, 2020

First referee: Prof. Dr. Bernhard Haubold

Second referee: Prof. Dr. Till Tantau

Date of oral examination: 15th October 2020

Approved for printing: Lübeck, 16th October 2020

To my father, Dr. Günther KLÖTZL
(* 29. 4. 1953 – † 13. 6. 2019)

The day you lose someone isn’t the worst. At least you’ve got something to do.
It’s all the days they stay dead.
— The Doctor

Abstract

To understand the evolutionary relationships between organisms, they are typically presented in a tree-like structure, a phylogeny. In genomic studies, phylogenies are traditionally reconstructed from a multiple sequence alignment. While most accurate, this approach is also computationally demanding. The problem is that in order to identify shared homologies, the sequences are usually first aligned nucleotide by nucleotide. This alignment step has become a bottleneck in the practice of molecular biology, where thousands of whole bacterial genomes, each a few megabases long, are sequenced and then need to be summarized as phylogenies when analyzing pathogen outbreaks.

One alternative are methods that estimate evolutionary distances directly from unaligned genomes. These pairwise distances can then be used to cluster sequences in a tree. Most of these alignment-free methods heavily rely on exact matching techniques for words of a fixed size for fast sequence comparison. However, they usually do not reflect the substitution rate, the most widely used measure of evolutionary distance.

Instead of using words of fixed size, Haubold et al. (2015) used matches of maximal length as anchors for approximate pairwise alignments. These anchor alignments then can be used to estimate the substitution rate. A first implementation, andi, quickly estimates accurate pairwise distances from hundreds of bacterial genomes on standard hardware. However, the thousands of genomes currently being collected during outbreaks again slow the program down.

Andi uses a suffix array as a full-text index for each of the input sequences. Since constructing and searching in a suffix array is slow, the aim of this thesis was to investigate, whether it might be possible to just compute a single suffix array for one of the input sequences and pile all remaining sequences onto that reference. This should produce an approximate multiple sequence alignment, from which pairwise mismatches could be counted.

This approach is implemented in the program phylonium (Klötzl and Haubold 2019). It is available via package managers or as open source at

`github.com/evolbioinf/phylonium.`

Phylonium is much faster than andi while losing little of its predecessor's accuracy. In this thesis I explain the background to phylonium, describe its implementation, and apply it to simulated and real data. In the application section I compare phylonium to its best competitors and show that it holds a reasonable position in the classical trade-off between speed and accuracy.

Kurzfassung

Um die evolutionären Beziehungen von Organismen zu verstehen, werden sie üblicherweise in einer baumartigen Struktur, einer Phylogenie, dargestellt. In Genomstudien werden Phylogenien klassischerweise mit Hilfe eines multiplen Sequenzalignments rekonstruiert. Dieser Ansatz ist sehr genau, aber auch rechnerisch anspruchsvoll, da die Sequenzen erst Nukleotid für Nukleotid aligniert werden, um gemeinsame Homologien zu finden. Dieser Alignierungsschritt ist zu einem Hindernis in der Molekularbiologie geworden. Dort werden inzwischen bei Ausbrüchen von pathogenen Bakterien tausende Genome sequenziert und zu Phylogenien zusammengefasst.

Mittlerweile gibt es Methoden, die evolutionäre Distanzen direkt von unalignierten Sequenzen schätzen können. Mittels dieser Distanzen werden die Sequenzen dann zu einer Phylogenie zusammengefasst. Dazu verwenden alignment-freie Methoden vielfach Vergleiche basierend auf Wörtern einer festen Länge. Dadurch sind sie zwar schnell, entsprechen aber nicht der Substitutionsrate, welche das übliche Maß einer evolutionären Distanz ist.

Anstelle von Wörtern fester Länge verwenden Haubold u. a. (2015) Matches maximaler Länge als Anker eines angenäherten, paarweisen Alignments. Dank dieser Ankeralignments kann dann die Substitutionsrate geschätzt werden. Eine erste Implementation names Andi, kann selbst auf Standardhardware die paarweisen Distanzen von hunderten Bakteriengenomen schnell und genau schätzen. Die Tausenden von Genomen jedoch, welche inzwischen bei Ausbrüchen gesammelt werden, stellen das Programm vor große Herausforderungen.

Für jede Eingabesequenz baut Andi einen Suffix-Array als Index. Weil das Erstellen und Suchen in einem Suffix-Array jedoch sehr langsam ist, untersuche ich in dieser Arbeit, ob ein einziger Index ausreicht. Dazu wird nur für eine Referenzsequenz der Index erstellt und alle anderen Sequenzen darauf gestapelt. Dies ergibt ein angenähertes multiples Sequenzalignment, welches dann zum Schätzen der paarweisen Substitutionsraten verwendet wird.

Dieser neue Ansatz ist in dem Programm Phylonium implementiert (Klötzl und Haubold 2019). Es ist via Paketmanager oder als freie Software verfügbar.

`github.com/evolbioinf/phylonium`

Phylonium ist schneller als Andi, ohne allzu viel Genauigkeit zu verlieren. In dieser Arbeit erläutere ich den Hintergrund zu Phylonium, beschreibe die Implementierung und wende es auf simulierte und reale Daten an. Ich vergleiche es mit Konkurrenzprogrammen und zeige, dass es einen sinnvollen Kompromiss zwischen Genauigkeit und Laufzeit darstellt.

Contents

Acknowledgments

I am sure that will get quoted back at me somewhere.

Jodie Whittaker

I love quotations.

Gyles Brandreth

There are many people who supported me throughout this PhD. First and foremost there is Bernhard who offered me the position in his group and saw me through some tough times. I am very grateful for his scientific guidance and am afraid that I might be the cause of a few gray hairs. I would also like to thank all my proofreaders: Karl Heinz, Marcel, Gustavo, Teresa, Chris, Amir, and even Stephan. Without their comments, this dissertation would not be much more than one of my usual ramblings.

The past two years have been demanding and I have to acknowledge the people that supported me emotionally, psychologically, and with their mere presence: My sister Julia, who encouraged me to seek professional advice when I needed it the most. My mother Jeanette, and her emotional support dog, Emma. I think we made the best of the situation. Jonas, who had the unfortunate position of living just around the block. He and his house mates were great hosts when I was in need of some social interaction. Theresa, who suggested trying yoga. Frederike and Tobias, who cared for me despite having a child to care for. Kıvanç, thanks to whom I now know the intricate architectural details of the Hagia Sophia. And everyone else I forgot to mention.

Thank you all very much.

Publications

1. F. Klötzl and B. Haubold (2016). “Support Values for Genome Phylogenies”. *Life* 6.1
2. L. Odenthal-Hesse et al. (2016). “hotspot: software to support sperm-typing for investigating recombination hotspots”. *Bioinformatics* 32.16, pp. 2554–2555
3. S. Möller et al. (2017). “Robust Cross-Platform Workflows: How Technical and Scientific Communities Collaborate to Develop, Test and Share Best Practices for Data Analysis”. *Data Science and Engineering* 2.3, pp. 232–244
4. F. Klötzl and B. Haubold (2019). “Phylonium: Fast Estimation of Evolutionary Distances from Large Samples of Similar Genomes”. *Bioinformatics* 36.7, pp. 2040–2046
5. B. Haubold and F. Klötzl (2020). “Fast Phylogeny Reconstruction from Genomes of Closely Related Microbes”. In: *Bacterial Pangenomics: Methods and Protocols*. Ed. by M. Fondi et al. 2nd ed. Springer. Forthcoming

1 Introduction

> It is curious because you think, if you and another person, another creature, are kind of in the same world, you must be feeling roughly similar. But one of the things you begin to realise when you look at different animals is that because of their evolutionary history, and because of the forms they have developed into, and the ways they have developed to perceiving the world, they may be inhabiting the same world, but actually a completely different universe.
>
> Douglas Adams

> A straight line may be the shortest distance between two points, but it is by no means the most interesting.
>
> The Doctor

1.1 Sequence Comparison

Organisms are best understood by comparing them to each other. In any such comparison, like needs to be compared with like, the human hand with that of the chimpanzee, with the wing of the bat, or the fin of the whale. Using such morphological comparisons, Haeckel (1897) drew one of the first phylogenies of humans depicted in Figure 1.1. It shows humans on top surrounded by the apes gorilla, chimpanzee, orangutan, and gibbon. At the bottom are the primitive forms of life like the single-celled amoebas.

However, with the advent of molecular biology in the 1950s, residues in DNA, RNA, and protein sequencing increasingly replaced the morphological traits studied by Haeckel and his successors. Zuckerkandl and Pauling (1965) were among the first to realize that the evolutionary history of an organism is written in its molecular sequences, which can hence be used as a molecular clock. Today, phylogeny reconstruction from nucleotide sequences is widely applied in biology.

Sequencing

Whole genome sequencing was part of DNA sequencing technology right from its inception. When Sanger et al. (1977) published their di-deoxy sequencing method, now simply called Sanger sequencing, they demonstrated its potential by presenting the complete 5375 bp sequence of phage ΦX174, a virus that infects the bacterium *Escherichia coli*. Sanger sequencing yields a few hundred nucleotides per reaction, a fraction of

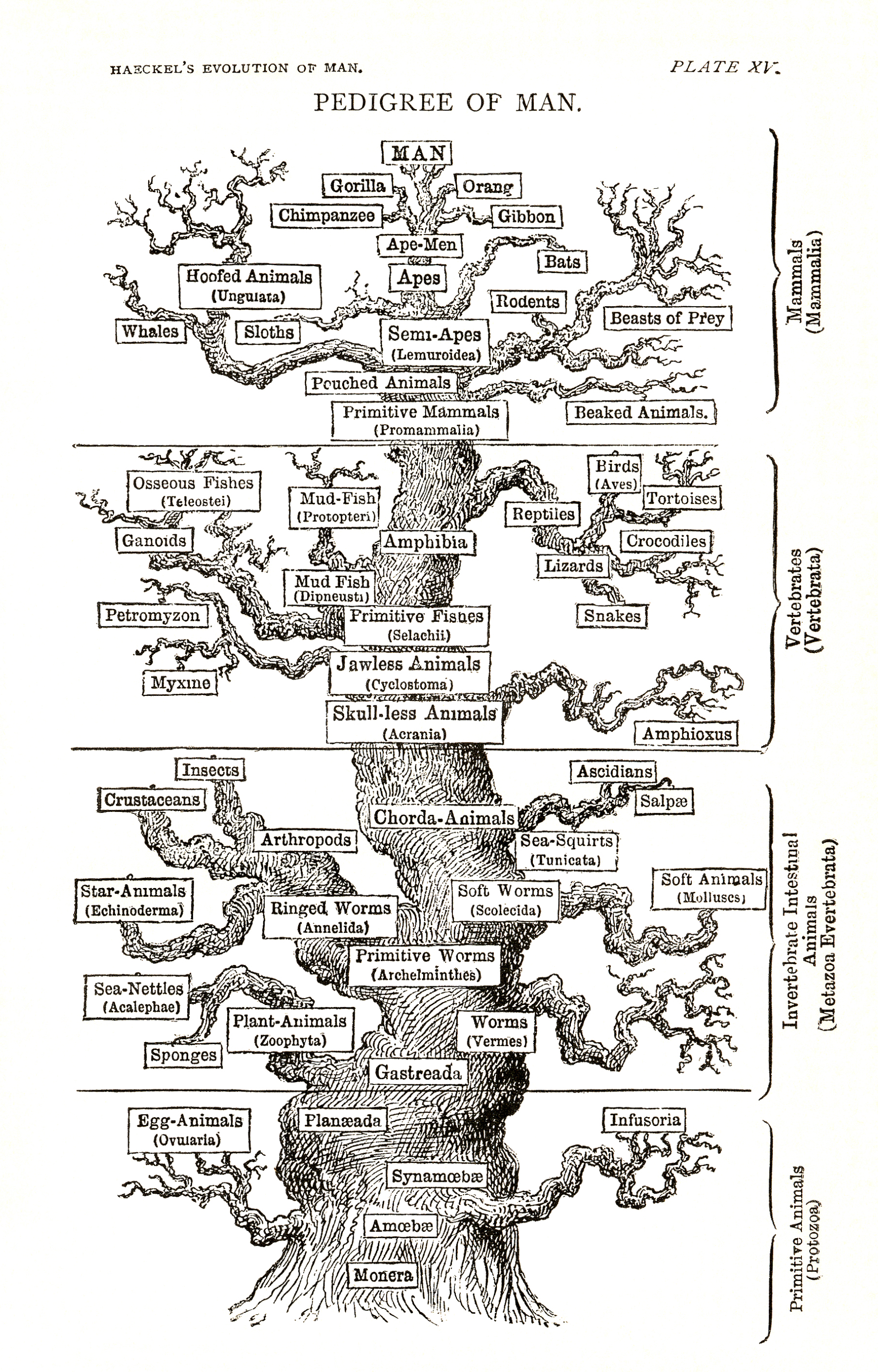


Figure 1.1: The Pedigree of Man as by Haeckel (1897).

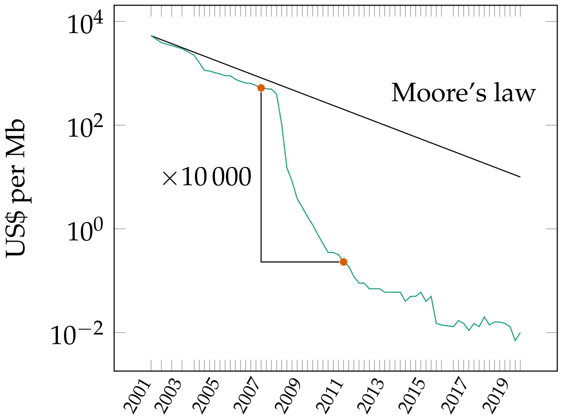


Figure 1.2: The cost of sequencing a million nucleotides (Wetterstrand 2020). The straight line represents an exponential decrease.

the genome size, even if this comprises only the 5.4 kb of phage ΦX174. Later, Sanger et al. (1982) realized that it is not necessary to sequence from one end of the DNA molecule straight to the other. Instead, a large number of copies of the target genome are smashed into random fragments, for example by sonication, which can then be sequenced individually, in parallel. This shotgun approach was first applied to the 48.5 kb genome of phage λ. The random sequences generated by shotgun sequencing are called reads and need to be assembled, which created the assembly problem (Rice and Green 2019). Assembly of Sanger sequenced reads remained the technology of choice for the next twenty years, which saw the sequencing of ever longer genomes.

In 1995, the era of phage genome sequencing turned into bacterial genome sequencing when the 1.8 Mb sequence of the human pathogen *Haemophilus influenzae* was published (Fleischmann et al. 1995; Haubold and Wiehe 2006b, p. 35). A year later the first eukaryote was sequenced, the 12 Mb yeast genome (Goffeau et al. 1996). 1998 came the first multicellular organism, the nematode and model organism *Caenorhabditis elegans* (The *C. elegans* Sequencing Consortium 1998). Two years later the first plant genome was sequenced, the 125 Mb of *Arabidopsis thaliana*, a model plant with a particularly compact genome (The Arabidopsis Genome Initiative 2000). The same year saw the sequencing of the fruit fly *Drosophila melanogaster* (120 Mb), which was the first metazoan genome sequenced by whole genome shotgun sequencing (Adams et al. 2000). A year later, the 3.1 Gb of the human genome were published (International Human Genome Sequencing Consortium 2001; Venter et al. 2001). This was the pinnacle of the classical phase of sequencing, when the experimental side of genomics was so expensive, only large labs could participate.

As shotgun sequencing is still used today, the assembly problem also persists (Myers 2014). However, from the mid-noughts onward Sanger sequencing has been replaced by a number of next generation methods. These vary widely in the length, number and quality of the reads returned (Goodwin et al. 2016). Irrespective of the details, the cost of sequencing fell 10 000 fold in the years between 2007 and 2011 (Figure 1.2) making sequencing virtually free.

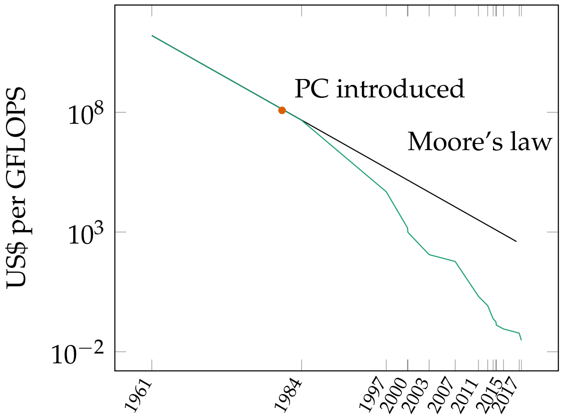


Figure 1.3: The cost of a billion floating-point operations per second in 2020 US$. The straight line represents an exponential decrease.

This has lead to the wide-spread application of sequencing in medicine, where pathogen outbreaks are increasingly tracked using whole-genome sequencing, a development called *genomic epidemiology* (Tang et al. 2017). As a result, massive samples of complete genomes from closely related bacteria are published. For example, Holt et al. (2018) sequenced 1635 isolates of *Mycoplasma tuberculosis* collected from tuberculosis patients in Ho Chi Minh City, Vietnam. Each *M. tuberculosis* genome is approximately 4.4 Mb long. By comparing their sample with 3144 isolates from elsewhere, they discovered that a lineage imported from Beijing was three times more virulent than its local relative. They were also able to pinpoint the gene most likely causing this increased in virulence (Holt et al. 2018).

Computing

A similar drop in price as just described for sequencing took place in computing three decades earlier. Forty-six years ago Ritchie and Thompson (1974) proudly announced that “UNIX can run on hardware costing as little as $40,000.” The PC-revolution of the 1980s made powerful computers widely available and lead to an accelerated decay in the price of computer hardware. Figure 1.3 shows that in 1981, when the PC was first released, to execute one billion floating-point operations per second (GFLOPS) one would have to invest one hundred million dollars.[1] Ten years later this was down to one million and in 2001 a GFLOPS cost a thousand dollars. By the year 2011 the price of a GFLOPS had collapsed to less than one dollar. Today UNIX and much more can run on a Raspberry Pi, available to consumers for 30 €.

So sequencing and computing have both become virtually free. However, as Dijkstra (1972) already predicted, increased computing power does not necessarily solve the programming problem. In the particular case of sequencing data, its analysis remains a formidable challenge. One of the most popular ways to analyze a set of genomes is

[1] Data taken from `https://en.wikipedia.org/wiki/FLOPS#Cost_of_computing`.

	1	1111111112	2222222223	3333333334	4444444445
	1234567890	1234567890	1234567890	1234567890	1234567890
Chimpanzee	TTTCTCACAC	AAGCAACTGC	GTCCATAATT	CTCCTGATAG	CTATCCTCTC
Bonobo	....	.......C..	......G...	.....A....	....
Human	..C.....G.	.......C..	A........C	..T..A....	.........T
Gorilla	..C.......	....	A........C	.....A....	.C........
Orangutan	..C.......	.......C..	A.........	T....A....	.C......CA
Gibbon	..C...GT..	.......C..	A.........	...A.A....	.C..T.....
Baboon	.....AG...	..T.T..C..	A........C	...A.A....	.A..TA.T..

					1
	5555555556	6666666667	7777777778	8888888889	9999999990
	1234567890	1234567890	1234567890	1234567890	1234567890
Chimpanzee	CAACAACATA	CTCTCCGGAC	AATGAACCAT	AACCA-ATAC	--TACCAATC
Bonobo	....	....	....	.....-.C..	--........
Human	.....G....	....	....	.....-....	--........
Gorilla	......T...	....	C	.....-..G.	--C..T....
Orangutan	....	T.T.......	.G.......C	.G...-....	--C.....C.
Gibbon	.......C..	..T.....G.	.G.....T..	.G...-.C.T	--C.....C.
Baboon	.......T..	T.A..A....	.C.....A.C	...A.C.A..	TAC.......

Figure 1.4: A multiple sequence alignment of 100 bp sequences taken from the mitochondrial genomes of seven primate species. A dot represents the same nucleotide as in the top row. The Baboon sequence contains three additional nucleotides which appear as gaps in all other sequences.

to summarize their evolutionary relationships as a phylogeny. This is also the central application of the distance computation method I present in this thesis.

1.2 Phylogeny Reconstruction

Phylogeny reconstruction from molecular data began in the 1960s and many of the methods that have been developed since are implemented in the Phylip software package (Felsenstein 2005). Its author also wrote the definitive guide to phylogeny reconstruction (Felsenstein 2004), while Yang and Rannala (2012) provide a more recent and succinct summary. Here, I give a brief overview of the most common methods. Phylogeny reconstruction typically starts from an alignment, and Figure 1.4 shows a 100 bp sample of seven mitochondrial genomes of primates, which are approximately 16 kb long. So the example data represents 0.6 % of the mitochondrial genome, which in turn is roughly 200 000 times smaller than the 3 Gb of the primate nuclear genome.

There are two classes of methods available to reconstruct a phylogeny from an alignment like that in Figure 1.4: Either the space of possible trees is searched for the tree that best fits the data as determined by some function; or the data is converted to pairwise distances, which are then transformed into a tree.

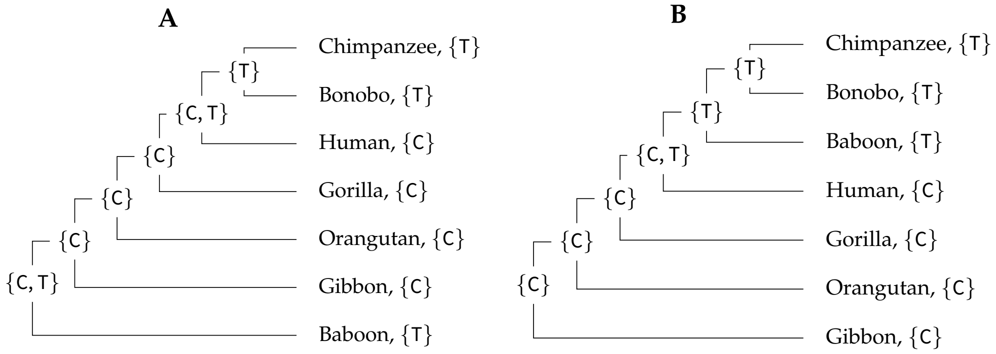


Figure 1.5: The number of mutations implied by column 3 in the alignment in Figure 1.4 given two example trees. In a bottom-up traversal, at every internal node the child sets are intersected. If the intersection is empty, a mutation has occurred and the union is formed. As a result, every set containing more than one element implies a mutation, so **A** implies two mutations, **B** just one.

Trees are commonly scored using one of two statistics: The number of mutations implied by the tree, or the probability—also called likelihood—of the data, given the tree and a model of mutation. The first is called the parsimony criterion and since the aim is to find the tree implying the fewest mutations, the corresponding method is known as maximum parsimony. Similarly, the second method is called maximum likelihood, as the aim is to find the tree that maximizes the likelihood of the observed data.

Maximum Parsimony

Fitch (1970) developed maximum parsimony based on the idea of searching for the tree implying the fewest mutations. The number of mutations is computed as follows: Given a tree and a polymorphic column in an alignment, label the leaves by the nucleotides in the column; Figure 1.6 shows this for column 3 in the example alignment (Figure 1.4) and two example trees. Think of these leaves as sets. Then traverse the tree bottom up. At every internal node, v, form the intersection of the two sets in the child nodes. If the intersection is empty, label v with the union and count one mutation, otherwise label v with the intersection. The tree in Figure 1.5A implies two mutations, the tree B only one. Thus, the biologically true tree B is more parsimonious at alignment position 3 than the wrong tree A.

This procedure is repeated across all polymorphic positions in the alignment to give the number of mutations for the tree. Multiple trees can imply the same number of mutations and are thus equally parsimonious. For our example alignment, the smallest number of mutations is 55, which is implied by the three trees in Figure 1.6.

Maximum Likelihood

The maximum likelihood criterion is based on the probability of the data, given the tree and a mutation model. The simplest mutation model is to equate the number of

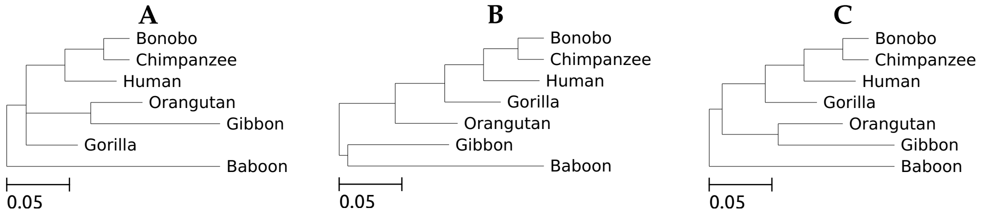


Figure 1.6: The three most parsimonious trees.

mismatches with the number of mutations; in other words, the probability of any site mutating more than once is zero. This is known as the infinite sites model often used in population genetics (Wakeley 2009, ch. 4), where very closely related sequences are compared. It is also effectively the model underlying maximum parsimony, where one mismatch is one mutation. However, the alignment in Figure 1.4 contains seven positions with more than two alleles: 42, 50, 58, 63, 72, 78, and 88. These positions must have mutated more than once since their divergence from the last common ancestor.

For DNA sequences, a mutation model is a matrix of probabilities μ_{xy} of nucleotide x changing into nucleotide y, with $x, y \in \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$.

$$\begin{pmatrix} \mu_{\mathtt{AA}} & \mu_{\mathtt{AC}} & \mu_{\mathtt{AG}} & \mu_{\mathtt{AT}} \\ \mu_{\mathtt{CA}} & \mu_{\mathtt{CC}} & \mu_{\mathtt{CG}} & \mu_{\mathtt{CT}} \\ \mu_{\mathtt{GA}} & \mu_{\mathtt{GC}} & \mu_{\mathtt{GG}} & \mu_{\mathtt{GT}} \\ \mu_{\mathtt{TA}} & \mu_{\mathtt{TC}} & \mu_{\mathtt{TG}} & \mu_{\mathtt{TT}} \end{pmatrix}$$

By definition each row sums to 1. In the simplest case all off-diagonal elements are set to the same value, which is called the one-parameter model. For this model, an equation developed by Jukes and Cantor (1969) transforms the number of mismatches per site, m, into the number of substitutions per site, K,

$$K = -\frac{3}{4} \ln \left(1 - \frac{4}{3} m \right) .$$

Table 1.1A shows the Jukes-Cantor distances based on the example data. As the baboon sequence contains insertions, all other sequences have gaps at positions 86, 91, and 92, denoted by a "-". For my analysis I excluded the whole column, which is known as *complete deletion*, so the distance is based on 97 positions. If instead a gap is only excluded in the pairwise comparison, that is called *pairwise deletion*. Complete deletion has the advantage that all distances are based on the same sites (Stecher et al. 2016).

There are a few alternatives to the one-parameter model. The next step is the 2-parameter model by Kimura (1980), which accounts for the two chemical classes of nucleotides, the purines `A` and `G`, and the pyrimidines `C` and `T`. Mutations within chemical classes, transitions, have a different rate from mutations between classes, transversions. So the 2-parameter model is based on the number of transitions per site, P, and the number of transversions per site, Q, which sum to the total number of mismatches $m = P + Q$,

$$K = -\frac{1}{2} \ln \left((1 - 2P - Q) \sqrt{1 - 2Q} \right) .$$

Table 1.1: Estimated substitution rates for the example alignment of Figure 1.4.

(A) 1-parameter model (Jukes and Cantor 1969)

	Bo	Ch	Hu	Go	Or	Gi	Ba
Bo	0	.04	.10	.14	.17	.20	.27
Ch	.04	0	.10	.11	.17	.23	.29
Hu	.10	.10	0	.12	.17	.24	.30
Go	.13	.11	.12	0	.15	.23	.27
Or	.17	.17	.17	.15	0	.16	.29
Gi	.20	.23	.24	.23	.16	0	.27
Ba	.27	.29	.30	.27	.29	.27	0

(B) 2-parameter model (Kimura 1980)

	Bo	Ch	Hu	Go	Or	Gi	Ba
Bo	0	.04	.10	.13	.17	.20	.29
Ch	.04	0	.10	.11	.17	.22	.30
Hu	.10	.10	0	.12	.17	.24	.32
Go	.13	.11	.12	0	.15	.22	.29
Or	.17	.17	.17	.15	0	.16	.31
Gi	.20	.22	.24	.22	.16	0	.28
Ba	.29	.30	.32	.29	.31	.28	0

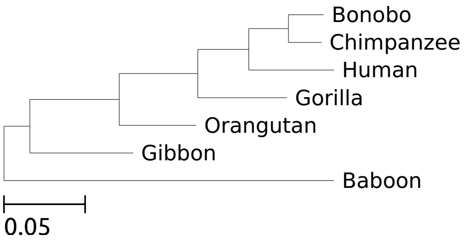


Figure 1.7: Maximum likelihood tree. The scale bar represents the number of substitutions on the branches.

Table 1.1B shows the Kimura distances for the example primate data. They are very similar to the Jukes-Cantor distances, but tend to increase for more divergent pairs of sequences. For instance, the Jukes-Cantor distance between Baboon and Bonobo is 0.27 substitutions per site, whereas the Kimura distance is 0.29 substitutions per site. Still more complex models exist, the most general one having twelve parameters (Liò and Goldman 1998), but the Jukes-Cantor model is the one I use throughout this thesis to calculate the substitution rate from mismatches.

Given a mutation model and a tree, computing the likelihood of the data is more difficult than computing the number of mutations, as the likelihood needs to be maximized across all possible branch lengths. This problem was solved by Felsenstein (1981). Figure 1.7 shows the maximum likelihood tree for the example data, which is very similar to the maximum parsimony tree in Figure 1.6B.

The great advantage of the maximum likelihood approach over other phylogeny methods is that alternative trees can be compared using the likelihood ratio test (Huelsenbeck and Rannala 1997). For instance, the log-likelihood of the maximum likelihood tree is -363.6. This can be compared to the log-likelihood of the two alternative trees from the maximum parsimony method, Figures 1.6A and C, which is -363.9. The difference between the best phylogeny and its alternatives is thus only 0.3, not enough to reject the alternatives ($p = 0.491$, test by Shimodaira and Hasegawa 1999).

Felsenstein (1981, p. 374) commented on the original implementation of his maximum likelihood procedure, "it must be acknowledged that this computer program is quite slow,

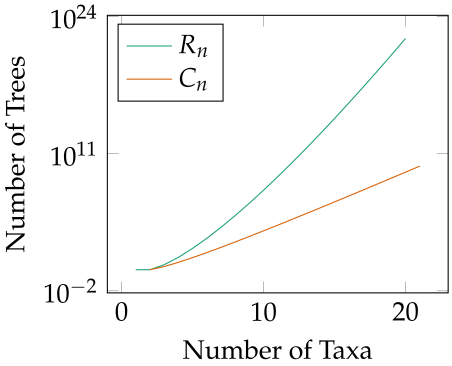


Figure 1.8: The number of binary trees, C_n, and rooted phylogenies, R_n, as a function of the number of taxa.

and could be effectively used only by someone who had free computer time available." Today, free computer time is the norm rather than the exception and maximum likelihood reconstruction is widely used in implementations such as RAxML (Stamatakis 2014).

Apart from the speed of scoring an individual tree, maximum parsimony and maximum likelihood require powerful algorithms for navigating the space of possible trees (Felsenstein 2004, ch. 4 and 5), as this grows very quickly with the number of taxa. Consider four taxa A, B, C, and D. Phylogenies of these taxa can be represented by parenthesizations, where pairs of parenthesis represent a subtree joined by an internal node.

((AB)C)D   (AB)(CD)   (A(BC))D   A((BC)D)   A(B(CD))

The number of possible parenthesizations is equivalent to the number of binary trees with ordered labels, known as the Catalan numbers (Knuth 2013b, sec. 7.2.1.6),

$$C_n = \frac{1}{n+1}\binom{2n}{n}\,.$$

However, the order of labels can affect the biological interpretation of the tree. The phylogeny (AB)(CD) is different from (BC)(DA), but not from (CD)(BA) as the left and right child position of a node are equivalent. So for our four taxa that leads to the following additional topologies.

((AC)B)D   (AC)(BD)   ((AC)D)B   ((BD)C)A   ((BD)A)C

((AD)B)C   ((AD)C)B   (AD)(BC)   ((AB)D)C   ((CD)A)B

The number of rooted topologies R_n with four taxa is thus $R_4 = 15$. So the number of rooted phylogenies, R_n for n taxa grows much quicker than the Catalan numbers C_n,

$$R_n = \prod_{i=2}^{n} 2i - 3\,.$$

Figure 1.8 shows the growth for both combinatorial problems.

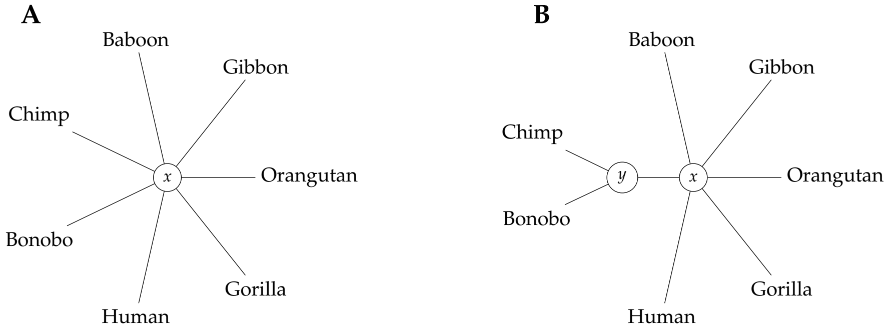


Figure 1.9: NJ starts with a star topology **(A)** and then joins the two most closely related neighbors **(B)**. The branch lengths are not drawn to scale, since they are only computed as the algorithm progresses.

Distance Methods

One of the most widely used methods to directly infer a tree from a distance matrix, D, is neighbor-joining (Saitou and Nei 1987). Neighbor-joining (NJ) starts with a star-like topology (Figure 1.9A) and, as the name implies, it then joins neighbors.

Whenever two neighbors are to be joined, a new node is created, y in Figure 1.9B, and a new edge. Of all node pairs, NJ picks the one (a, b) that minimizes the overall branch length. The new node is inserted and the neighbors attached to it. The distance of the new node to all others is then calculated by averaging.

$$D_{yz} = (D_{az} + D_{bz})/2$$

The algorithm stops when there are only three nodes left. It then places a last new node in the middle. NJ recovers the true tree if the distances are additive, meaning that the distance between two leaves is the sum of the edge lengths separating them. This can also be checked on the distance matrix if for all quartets there exist a labeling so that

$$D_{ab} + D_{cd} \leq D_{bc} + D_{ad} = D_{ac} + D_{bd} \ . \tag{1.1}$$

I use neighbor-joining throughout this thesis to cluster distances, and Figure 1.10 shows the NJ tree for the example primate data. However, the NJ algorithm has two problems: First, it does not quantify the discrepancy between the tree and the data. Second, the tree is unrooted, but it is usually more convenient to read rooted trees. The first problem was addressed by Fitch and Margoliash (1967), who proposed a sum-of-squares measure to quantify the fit of the tree to the data. In Klötzl and Haubold (2016) we instead proposed to use the discrepancy between the tree and the distance matrix to gain confidence in individual splits. The four-point criterion (1.1) specifies the optimal topology of a quartet. Any quartet not in that arrangement thus lowers the confidence in the tree (Section 2.3).

The second problem with NJ, lack of a root, is often solved by placing the root midway between the two most distant leaves. For instance, in Figure 1.10 the root is placed halfway between baboon and human. There are more sophisticated methods for rooting

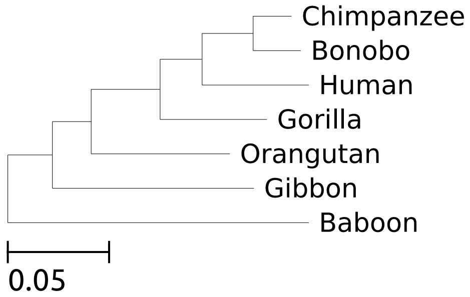


Figure 1.10: A neighbor-joining tree summarizing the distances of Table 1.1A with midpoint rooting.

trees (Tria et al. 2017), but I have used midpoint rooting in this thesis as it is still widely applied in biology.

Tree scores like maximum likelihood, maximum parsimony or sum of squares are computed across all clades. However, the reliability of a phylogeny often varies considerably between clades. The most widely-used method for quantifying clade reliability is based on a statistical resampling method called the bootstrap (Efron 1979).

Bootstrapping Phylogenies

A central question in statistics is, given a measurement, for instance the average height of a sample of wheat plants, how would that measurement vary if another sample was drawn from the same field? This can be answered by modeling the distribution of the statistics using mathematics or computer simulations. Among computer simulations, the bootstrap has become particularly popular due to its simplicity: Given a sample for n measurements, generate a pseudosample by drawing n elements with replacement, and recalculate the statistic from that pseudosample. If repeated many times, this allows the quantification of the uncertainty of the original measurement (Efron 1979).

Felsenstein (1985) introduced the bootstrap into phylogeny inference: Given an alignment of length l, such as that shown in Figure 1.4, where $l = 100$, draw with replacement l columns from it and recalculate the phylogeny from this pseudosample. When repeated, this allows the computation of the frequency with which each clade in the original tree appears in trees based on the resampled data. The higher this frequency, the higher the bootstrap support, that is the reliability, of that clade.

If this procedure is applied to the alignment of the mitochondrial sequences and the neighbor-joining tree in Figure 1.10, the bootstrap frequencies shown in Figure 1.11 are generated. A support value above 70 is generally considered significant (Hillis and Bull 1993). That applies to the clade Chimpanzee/Bonobo and the group of Chimpanzee/Bonobo/Human/Gorilla. The low support of the clade Chimpanzee/Bonobo/Human on this small sample of 100 bp mirrors the fact that for 30 % of the human DNA the Gorilla is the closest living relative, not the Chimpanzee (Scally et al. 2012).

Having established how to convert distances to a tree, I describe how to get distances in the first place.

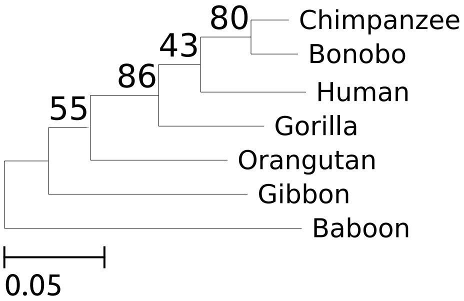


Figure 1.11: Neighbor joining phylogeny with support values computed via classical bootstrapping.

1.3 Distance Computation

Given an alignment, evolutionary distances can be easily calculated, for instance using the Jukes-Cantor model of sequence evolution to transform pairwise mismatches to substitution rates, as done with the mitochondrial genomes of seven primates in Section 1.2. However, finding the optimal multiple sequence alignment (MSA) is NP-complete and thus becomes computationally demanding with growing sequence length (Wang and Jiang 1994). An assessment of common MSA tools showed that most take hours on whole-genome data sets (Earl et al. 2014). For instance, on a simulated 120 Mb data set of four primates, the program progressiveMauve took over four hours (Darling et al. 2010). On a bigger data, set it ran for several weeks, using 200 GB of RAM, before being abruptly terminated by a power outage. Earl et al. (2014) conclude that "it's fair to say this implementation doesn't scale reasonably to such datasets." To overcome the computational restrictions of alignment-based phylogeny reconstruction, *alignment-free* methods have recently become popular (Haubold 2014).

Alignment-free tools skip the computation of an MSA and instead estimate evolutionary distances directly from the sequences. These distances can then be used as input for a tree reconstruction algorithm such as neighbor-joining (Saitou and Nei 1987). Alignment-free methods have the advantage of being much faster than alignment-based approaches. For instance, on a data set of 29 whole *Escherichia coli* and *Shigella* genomes, each about 5 Mb long, the alignment-free tool mash took one second, whereas the MSA program mugsy took almost 2 hours (Ondov et al. 2016; Angiuoli and Salzberg 2011, Section 4.3). Extrapolating this to a data set a hundred times as large, gives an expected run time of 2.7 years (Seidel 2017, p. 72), where mash only takes seven minutes, single-threaded.

Figure 1.12 shows a classification of different phylogeny reconstruction methods. To the left are the alignment-based approaches, including the ones already discussed, maximum likelihood, maximum parsimony, and distance based. Alignment-free methods are similar to the latter category as they tend to produce distance matrices. Thus, I consider computing an MSA, counting mismatches, and estimating distance for neighbor-joining the gold standard in this thesis, which all alignment-free tools are compared against. Figure 1.13 shows a phylogeny of the whole mitochondrial genomes of the seven primate species computed using mugsy, an MSA program (Angiuoli and Salzberg 2011).

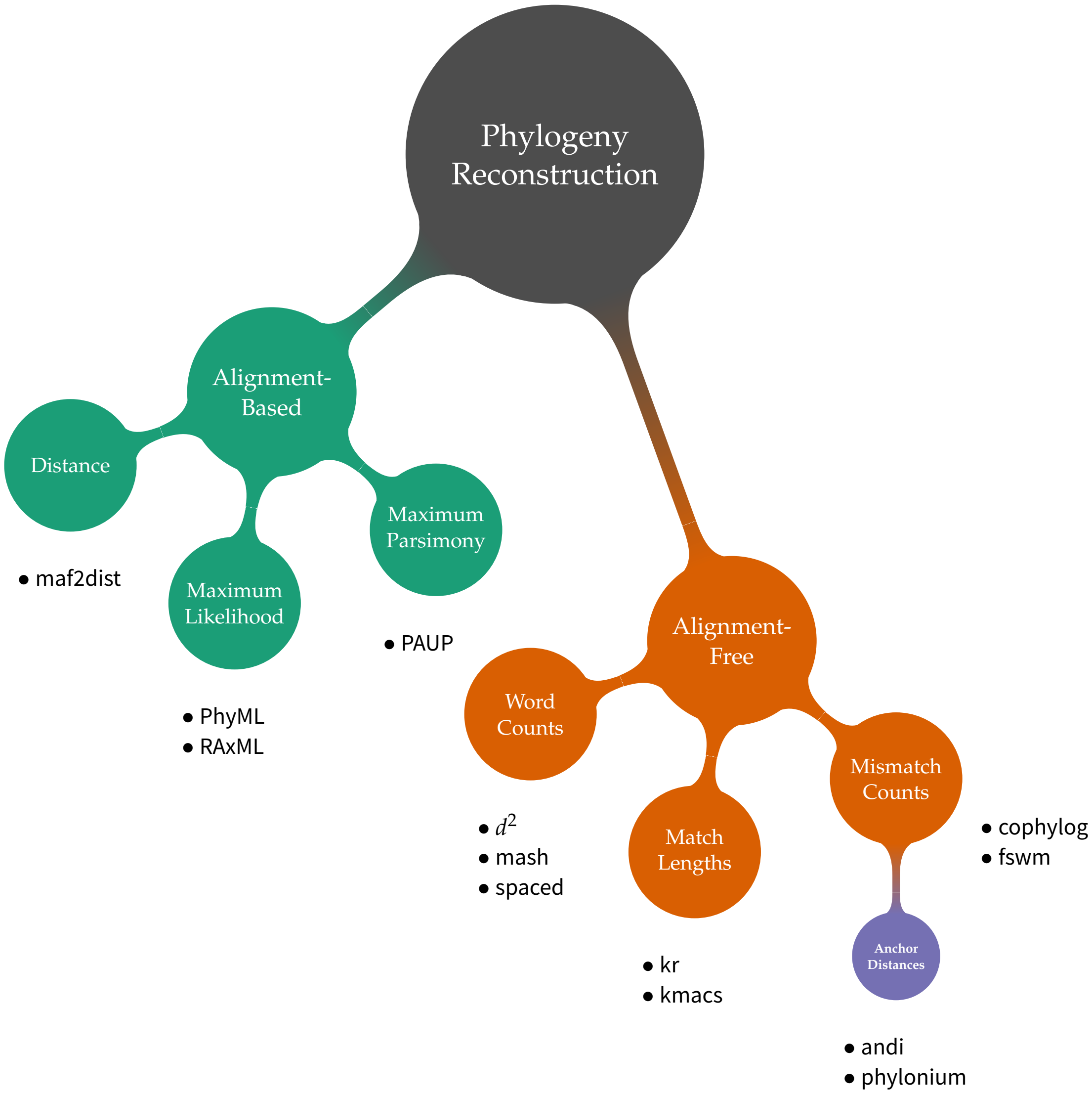


Figure 1.12: Classification of phylogeny reconstruction methods with commonly used implementations. Adapted and extended from Haubold (2014).

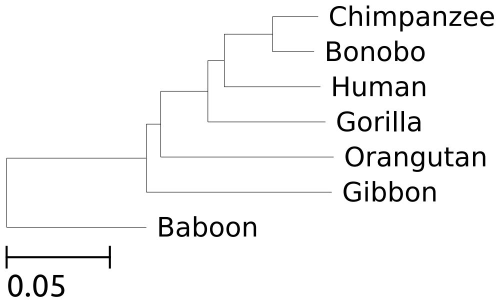


Figure 1.13: The phylogeny of seven primates, computed from a multiple sequence alignment rooted using the Baboon as an outgroup.

1.4 Alignment-Free Distance Estimation

Among the first alignment-free methods invented were approaches based on counting subsequences of a fixed length called words, *k*-mers or *q*-grams (Vinga and Almeida 2002). These methods would split the given sequences into *k*-mers, compute a vector of occurrences for each *k*-mer and then estimate a distance between sequences producing a matrix as in Table 1.1A. For instance, the d^2 distance is based on a frequency word profile m for different word lengths and compares them between sequences (Hide et al. 1994). Here u, is an upper bound on k, and for each k, as there are four nucleotides, the number of occurrences of 4^k words ω_i are compared.

$$d^2 = \sum_{k=1}^{u} \sum_{i=1}^{4^k} (m_S(\omega_i) - m_Q(\omega_i))^2$$

Because of their simplicity, many variations were developed, with a recent review bundling twenty-four into one web-based application (Zielezinski et al. 2017). However, most distances produced by *k*-mer methods do not reflect the substitutions per site, the most widely used evolutionary distances. For instance, Figure 1.14 shows the phylogeny computed for the primate data using d^2. Not only does it have an incorrect topology, but the scale bar and branch lengths are also wrong. Moreover, the choice of k can greatly influence the computed phylogeny (Leimeister et al. 2014). Despite over a decade of research, the choice of the optimal k is still unclear (Reinert et al. 2000; Zuo et al. 2014). Only a few methods from this category accurately estimate substitution rates, the most important of which are described in the following.

Spaced Words Most of the *k*-mer based methods for phylogeny reconstruction measure a quantity that grows monotonously with time but is not equal to the substitution rate, the most widely used evolutionary distance. In contrast, Morgenstern et al. (2014) use an explicit model of sequence evolution for accurate estimation of the substitution rate. Furthermore, their *k*-mers are *spaced words*, that is, they contain *don't care* positions, where the characters are allowed to vary.

For instance, consider the sequence $S =$ `AACTT`. It contains the 3-mers `AAC`, `ACT` and `CTT`.

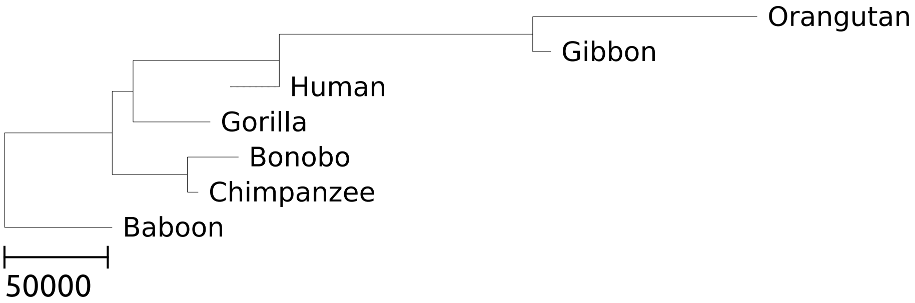


Figure 1.14: The phylogeny of seven primates, computed with d^2. Note the incorrect topology as well as the idiosyncratic scale. The negative branch length at human is a symptom of bad distances.

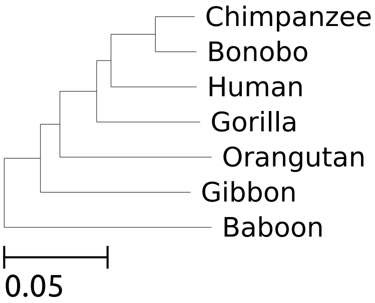


Figure 1.15: The phylogeny of seven primates, computed with spaced.

However, it also contains the 4-mers AA∗T and AC∗T, where the star denotes an ignored character. A second sequence $Q =$ AAGTT shares none of the 3-mers, but one of the *spaced* 4-mers. This way, spaced k-mers can increase the sensitivity of the distance estimation. Figure 1.15 shows the tree computed by spaced, which is much more accurate than that of exact words (Figure 1.14).

The authors of spaced use the Jensen-Shannon divergence on the word frequency profiles to estimate evolutionary distances (J. Lin 1991). Using multiple patterns for spaced words, they found their implementation could reconstruct phylogenies more accurately than other alignment-free approaches (Leimeister et al. 2014). Further, they developed a hill-climbing approach to pick the best patterns for even more accurate results (Sohrabi-Jahromi et al. 2017).

Hashed Words A hash function reduces data of arbitrary size to a fixed-size hash value (Knuth 2013a, pp. 513). Most hash functions spread their input values as evenly as possible over the output range. One technique called MinHash, however, maps similar data to similar sets of hashes (Broder 1997). Mash makes use of this property by splitting a sequence into k-mers, computing a hash for each, and then assembling the k-mers with the lowest hash value into a set (Figure 1.16). This set, termed *sketch*, is a random, fixed-size sample of all k-mers from a sequence. Given two sequences A, B and their respective

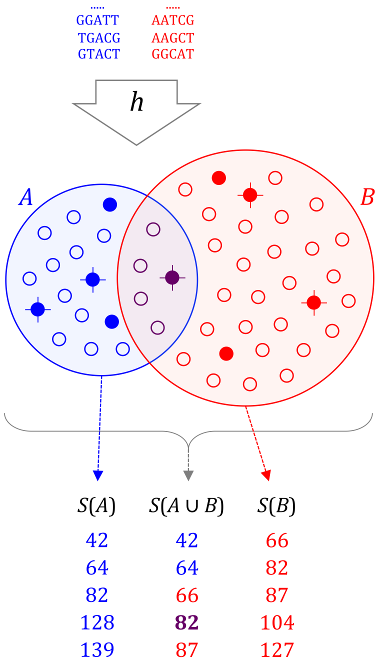


Figure 1.16: Estimating Evolutionary Distances from Genome Sketches; Taken from Ondov et al. (2016). Each sequence is split into k-mers, which are hashed into sets A, B. Of these sets only the—in this case 5—smallest k-mers are stored in sketches $S(A), S(B)$. These sketches are then compared.

sketches $S(A), S(B)$ the amount of overlap of their k-mer sets can be approximated by the overlap of the sketches (eq. (3) Ondov et al. 2016):

$$\frac{|A \cap B|}{|A \cup B|} \approx \frac{|S(A \cup B) \cap S(A) \cap S(B)|}{|S(A \cup B)|}$$

This property is used to approximate the number of shared k-mers and thus, the mutation rate between the two sequences (eq. (4) Ondov et al. 2016). Figure 1.17 shows the phylogeny for the seven primates. Mash puts no restrictions on the input sequences and hence also works on raw sequencing reads or metagenomes. The latter describes situations where a sequenced genome is actually a mix of sequences from several different genomes. This issue arises when a species cannot be cultured individually as for instance some human skin microbiota. In these cases, mash can be used to estimate the amount of one sequence contained in another (Ondov et al. 2019).

One central feature of mash is that the sketches are of a predetermined, fixed size. Thus, instead of comparing two k-mer sets of size $O(|A| + |B|)$, the analysis is limited to two small samples. By default, mash uses a thousand hashes as the sketch size. When comparing a large number of genomes, the reduction in dimensions leads to a significant increase in speed (Section 4.6). Other authors have picked up this idea and provide similar tools using different data structures for sketching (Zhao 2018). For instance, Baker

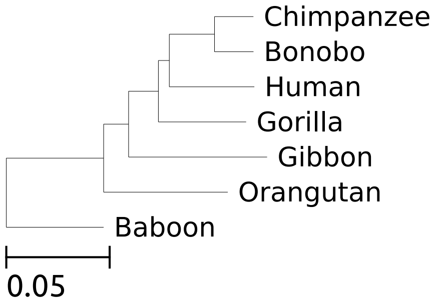


Figure 1.17: The phylogeny of seven primates computed using mash.

and Langmead (2019) use HyperLogLog, which is a compact way of counting elements in a set, to estimate the intersection of the sequences (Flajolet et al. 2007).

Match Lengths Consider two aligned sequences. As the number of mismatches varies, so does the distance between them, but in an inverse manner: For small substitution rates there are long stretches of exact matches between two mutations. Conversely, as exact matches end in a mismatch, they become shorter as the substitution rate increases. Haubold et al. (2009) related the substitution rate to the match length of unaligned sequences. Here X^* denotes the prefix length of one sequence that appears somewhere in the subject sequence S, and d is the number of mismatches (eq. (6) Haubold et al. 2009):

$$P(X^* \leq x) \approx (1 - e^{-xd}) \cdot \sum_{k=0}^{x} 2^x \binom{x}{k} p^k \left(\frac{1}{2} - p\right)^{x-k} \left(1 - p^k \left(\frac{1}{2} - p\right)^{x-k}\right)^{|S|} . \quad (1.2)$$

The average match length can be used to estimate the substitution rate, with Figure 1.18 depicting this idea on the primate data. In the first implementation of this approach, kr, pairs of sequences are stored in one generalized enhanced suffix array (GESA) (Haubold et al. 2009; Abouelhoda et al. 2002). This is then traversed to compute the average match length. Domazet-Lošo and Haubold (2009) later realized that a single GESA of multiple sequences is sufficient to compute all pairwise distances. This reduces the comparison of n sequences with a length of L from time $O(n^2L)$ to just $O(nL)$ (Ohlebusch 2013, pp. 228).

The approach, as implemented in kr, is limited to substitution rates less than 0.55 (Haubold et al. 2009). To improve the accuracy beyond that threshold, modifications were suggested that allow up to k mismatches in each match (Leimeister and Morgenstern 2014; Pizzi 2016). However, the best known algorithm takes time $O(kL^2/\log L)$ per pairwise comparison and thus implementations again fall back to heuristics to achieve reasonable run times (Apostolico et al. 2016).

Counting Words with Mismatches Counting the number of mismatches is assumed to be the gold standard for the estimation of an evolutionary distance in this thesis. On the other hand, counting words requires neither an alignment nor assembly. The tool cophylog combines counting mismatches with k-mers (Yi and Jin 2013). It uses spaced words with

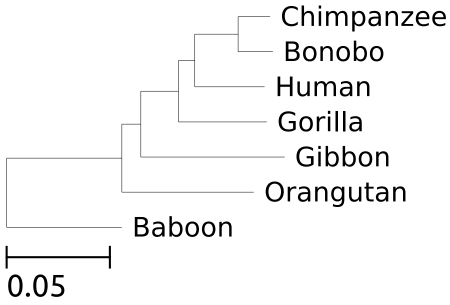


Figure 1.18: The phylogeny of seven primates computed using kr.

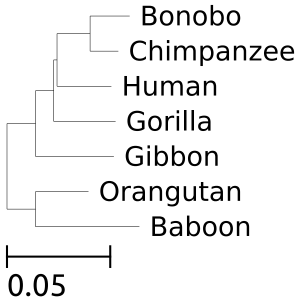


Figure 1.19: The phylogeny of seven primates computed using cophylog.

just matching positions except for one *don't care* in the middle. For all input sequences, the set of spaced words with a certain length is computed. Next, the algorithm compares sequences by looking at all spaced words featuring the same matching parts; they form a *micro-alignment*. If the position in the middle varies, it is counted as a mismatch.

Using this approach, the mismatch rate is estimated (Figure 1.19). Unfortunately, cophylog suffers from the problem that—again—the optimal length of the words is unknown and it also does not have an explicit substitution model built-in. Furthermore, the applicability of cophylog was initially limited as it was single-threaded. On a set of 3085 whole *Streptococcus pneumoniae* genomes it ran for 36 days (Haubold et al. 2015). To speed it up, I parallelized the program. On 32 virtual cores it can now analyze the same data in 17 hours. The faster cophylog can now be found online (Chapter A).

Filtering Spaced Words One ubiquitous problem in genome comparison is the distinction between homologous sections and random matches (Devillers and Schbath 2012). Distinguishing between the true signal and the background matches requires an explicit model of evolution. For example, blast assigns each alignment an E-value, the number of alignments one would expect by chance between two random sequences of the same length and composition (Karlin and Altschul 1990; Altschul et al. 1990).

Leimeister et al. (2017) have extended this model to spaced words. By default, their software fswm uses words with only twelve fixed positions on one hundred *don't care* positions. It then compares the initial *don't care* positions using a scoring matrix designed

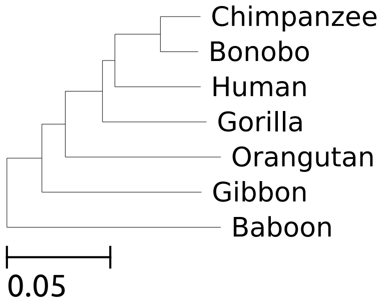


Figure 1.20: The phylogeny of seven primates computed using fswm.

for distantly related sequences (Chiaromonte et al. 2001). Comparisons with a negative value (too many mismatches) are rejected. The remaining homologous spaced words are used to estimate the number of mismatches between two sequences that are then turned into an evolutionary distance using the Jukes-Cantor substitution model. Figure 1.20 shows the phylogeny for the primate data. Interestingly, fswm can estimate the substitution rate accurately up to values of 0.8. Thereby it exceeds the range of many other alignment-free approaches (Section 4.1). However, this accuracy comes at the cost of run time; in the worst case fswm takes time $O(L^2)$ for each pairwise comparison.

1.5 Anchor Distances

Fast exact matching is conceptually simpler than inexact matching. Altschul et al. (1990) use exact matches of a fixed size, k-mers, as seeds for local alignments in blast. Given a long subject sequence S and a shorter query Q, they split the query into k-mers and create a keyword tree from them. Using the algorithm of Aho and Corasick (1975) all occurrences of the k-mers in S can be found in time $O(|S|)$ with $O(|Q|)$ preprocessing time. As only the query is preprocessed, the algorithm requires $O(|Q|)$ memory.

Where blast uses k-mers as seeds for local alignments, in Haubold et al. (2015), we use maximal matches instead. A maximal match is guaranteed to be flanked by a mismatch on either side and can be quickly found using full-text indices, such as suffix trees (Section 1.6).

These maximal matches are the basis of anchor distances as implemented in andi (Haubold et al. 2015). In Section 1.6, I explain how maximal matches are found, but assume for a moment one can quickly determine the longest prefix of a query sequence Q that also appears in a subject S. If this prefix is unique and longer than a threshold t, it is termed an *anchor*. If a match does not fit the requirements, the search continues beyond until an anchor is found. Given a left anchor, the program continues to search for a second one, the right anchor. Two anchors form a pair if they are equidistant (Figure 1.21). For anchor distances, we assume that such a pair does not contain an insertion or deletion and thus is an approximate local gap-free alignment for which one can count the mismatches in between to estimate the evolutionary distance using the Jukes-Cantor model of sequence evolution (Section 1.2, Jukes and Cantor 1969). This idea

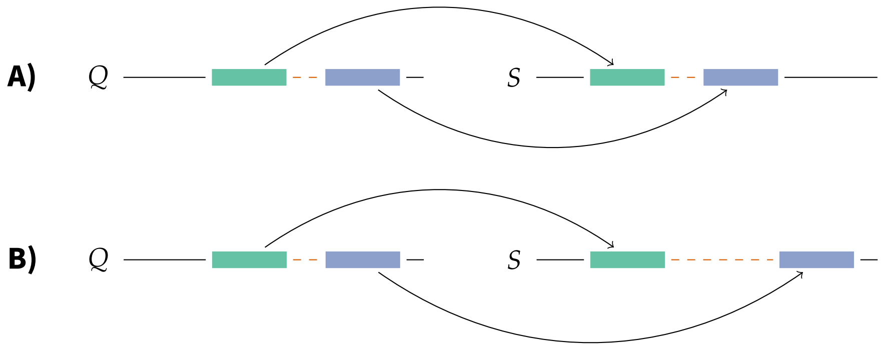


Figure 1.21: Anchors as Exact Matches Between Two Sequences. **A)** Two maximal matches span the same distance in Q as in S; thus, they form an anchor pair. **B)** The dashed segment between the anchors is unequally spaced on the two sequences and thus ignored in distance estimation.

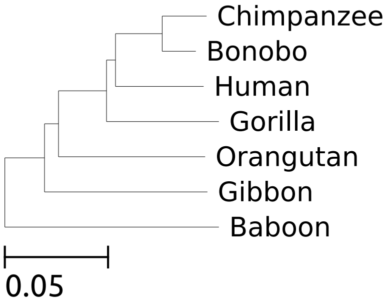


Figure 1.22: The phylogeny of seven primates computed used andi.

is implemented in the program andi and Figure 1.22 applies it to the primates (Haubold et al. 2015).

Multiple queries can be streamed against the same subject and all algorithms involved are linear in the length of the sequences, L, comparing a whole set of sequences thus takes time $O(Ln^2)$ (Klötzl 2015). For andi we use algorithms that are slower in theory but more efficient in practice. For instance, instead of the linear-time DC3 algorithm for suffix array construction, we use libdivsufsort with a worst-case run time of $O(L \log L)$ (Kärkkäinen et al. 2006; Fischer and Kurpicz 2017).

While being state-of-the-art at the time, andi had some limitations: As the uniqueness criterion for anchors is only with respect to the subject, not the query, duplications can lead to varying mismatch counts depending on the labeling. To make estimates symmetric, both sequences in a comparison have to serve once as a subject and once as a query. Thus, for a set of n sequences, n full text indexes are computed for $n^2 - n$ comparisons. Each single comparison is quick, but as the data set grows linearly, the run time grows quadratically. So for the new tool described in this thesis, phylonium,

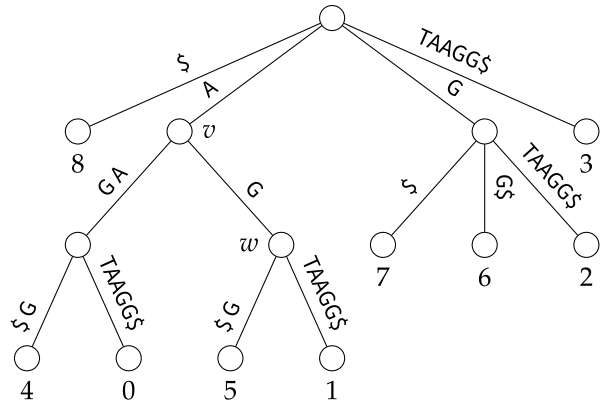


Figure 1.23: Suffix tree for the sequence $S =$ AAGTAAGG$.

we focused on reducing the quadratic portion of the distance estimation by piling all sequences in a sample onto a single reference. Further, phylonium handles duplications differently so the distances become symmetric and thus fewer comparisons have to be performed. This is described in detail in Chapter 3.

1.6 Finding Maximal Matches

Anchor distances require maximal matches. Given the two sequences S and Q, what is the longest prefix of Q that is also contained anywhere in S? We call this prefix a *maximal match*.

Definition 1 (Maximal Match Problem) *Let Σ be an alphabet and $S \in \Sigma^*, Q \in \Sigma^*$ two words over that alphabet. Find the largest q so that for all $i < q : S[i+k] = Q[i]$ holds for a fixed k. The maximal match then starts at position k in S and has length q.*

For instance, for the subject $S =$ AAGTAAGG$ and the query $Q =$ TACC$ the maximal match is TA. It starts at position 3 in the subject and is two characters long, $S[3..4] = Q[0..1]$.

With anchor distances we repeatedly match many different queries against the same subject. Thus, we are willing to spend more time preprocessing the subject if that reduces the time for finding maximal matches. One solution here is to use a *suffix tree* (Weiner 1973; Gusfield 1997). Figure 1.23 depicts the suffix tree for the string $S =$ AAGTAAGG$. Each leaf is labeled with a number i, $0 \leq i < |S|$ and each path from the root to a leaf spells out the suffix starting at that position, for instance $S[6..] =$ GG$.

Each outgoing edge from a node begins with a different letter from the fixed size alphabet Σ. For example, there is one edge of the root labeled A leading to an internal node labeled v. The four leaves on the subtree of v each give the position of the letter A in S. Continuing from v there is an edge labeled AG. All suffixes on leaves below it not only start with A, but with AAG.

The properties noted above can be adapted into a matching algorithm. Recursively iterating down from the root one can quickly find all occurrences of a pattern P in the

i	SA	$S[SA[i]..]$
0	8	$\texttt{\$}$
1	4	$\texttt{AAGG\$}$
2	0	$\texttt{AAGTAAGG\$}$
3	5	$\texttt{AGG\$}$
4	1	$\texttt{AGTAAGG\$}$
5	7	$\texttt{G\$}$
6	6	$\texttt{GG\$}$
7	2	$\texttt{GTAAGG\$}$
8	3	$\texttt{TAAGG\$}$

Figure 1.24: The suffix array for the string $S = \texttt{AAGTAAGG\$}$. Next to each entry is also the associated suffix.

tree. Consider $P = \texttt{AG}$. Starting from the root, follow the edge labeled with the first letter $P[0] = \texttt{A}$. Continue from there with the next letters until the pattern is exhausted or no suitable edge is available. In the former case, every leaf below the last match gives the starting position of the pattern in the subject. In our case, below node w are the leaves 5 and 1, which indeed give the starting positions of $\texttt{AG}$ in S. An optimal search algorithm on a suffix tree runs in time $O(|P|)$ (Gusfield 1997, p. 92). Thus, the time to search for a pattern in the index is independent of the size of the index. This is particularly useful when the index has to be built only once and many patterns are matched against it. The algorithm by Weiner (1973) and its simplification by Ukkonen (1995) create a suffix tree from a sequence S in $O(|S|)$ time using $O(|S|)$ memory.

While the suffix tree solves the exact matching problem in the same asymptotic time as the algorithm by Knuth et al. (1977), it is a data structure that can be reused. It thus provides a speed up when searched repeatedly. However, suffix trees have been criticized for their practical memory demands (Manber and Myers 1993; Abouelhoda et al. 2004). Manber and Myers (1993) proposed a simpler data structure with similar capabilities, the suffix array (SA). The suffix array of a string S contains the integers 0 through $|S|$ representing the lexicographic order of the suffixes. Figure 1.24 gives the suffix array, SA, for the same string $S = \texttt{AAGTAAGG\$}$. The sentinel character $\texttt{\$}$ is considered smaller than any other character, thus the suffix $S[8..]$ comes first in the SA.

A suffix array can be constructed from a suffix tree by visiting the edges in alphabetic order. Note how, read from left to right, the indices on the leaves of the suffix tree in Figure 1.23 correspond to the entries in the suffix array (Figure 1.24). Thus, a suffix array can be computed in time $O(|S|)$ with $O(|S|)$ memory. Since its invention, many algorithms have been discovered that directly construct a suffix array in linear time (Kärkkäinen and Sanders 2003; Ko and Aluru 2003). The most used library in practice, libdivsufsort, computes a SA in time $O(|S| \log |S|)$ (Fischer and Kurpicz 2017).

Searching for a pattern in the suffix array can be done by binary search. This leads to a

i	SA	LCP	CL	CR	FVC	$S[SA[i]..]$	lcp-intervals		
0	8	-1		9		$	0		
1	4	0		5	**A**	**A**AGG$		1	3
2	0	3			**T**	AAG**T**AAGG$			
3	5	1	2	4	**G**	A**G**G$			2
4	1	2			**T**	AG**T**AAGG$			
5	7	0	3	8	**G**	**G**$		1	
6	6	1		7	**G**	G**G**$			
7	2	1			**T**	G**T**AAGG$			
8	3	0	6		**T**	**T**AAGG$			
9		-1	1						

Figure 1.25: An enhanced suffix array for the sequence $S =$ AAGTAAGG$ using the same example as Klötzl (2015).

run time of $O(|P| \log |S|)$ (Manber and Myers 1993). To arrive at a run time independent of the size of the array, we need to *guide* the binary search along specific intervals. These intervals are LCP-intervals and the guides are child tables. Here, LCP stands for longest common prefix. In Figure 1.25 the suffix array has been enhanced by an *LCP* column. Each LCP value gives the length of the prefix shared with the suffix above.

$$LCP[i] = \max \{k \mid \forall j < k : S[SA[i] + j] = S[SA[i-1] + j]\}$$

For instance, AAGTAAGG$ and AAGG$ have a longest common prefix of 3. This LCP value is given in the third row in Figure 1.25. The first and the last LCP values are set to -1. Both work as sentinels in the following algorithms.

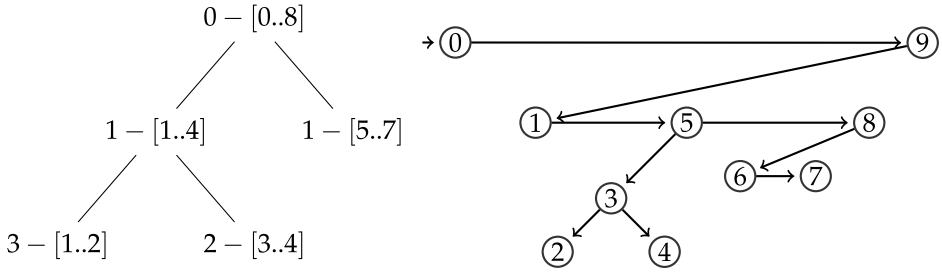
The LCP array requires $O(n)$ memory and can naïvely be computed in time $O(n^2)$. The algorithm by Kasai et al. (2001) reduces the run time to $O(n)$ but requires an additional $\Theta(n)$ memory. It is based on the following observation. The suffix $S[0..] =$ AAGTAAGG$ has an LCP of 3 with its upper neighbor $S[4..] =$ AAGG$. Thus, $S[1..] =$ AGTAAGG$ has to have an LCP value of at least 2. (Simply removing the first character of both suffixes leaves the rest of the LCP intact). Therefore these characters do not need to be re-compared. Listing 1.1 gives the algorithm in detail. As the suffixes are processed in text order, the inverse suffix array (ISA) is required to give the position in the suffix array.

Note how the LCP values in Figure 1.25 form intervals. For instance, the suffixes at position 1 through 4 all start with the letter A. Further, the first two of them share AAG. Following Abouelhoda et al. (2004), I use the notation $\ell - [i..j]$ to denote such an interval where ℓ is the length of the shared prefix of all suffixes between $SA[i]$ and $SA[j]$. So the LCP-interval of A is $1 - [1..4]$ and for AAG it is $3 - [1..2]$. Figure 1.26A displays the tree of lcp-intervals. Note how it corresponds to the suffix tree just without the leaves. For many algorithms traversing the suffix tree node-by-node via an lcp-tree is hence just as good. However, fast exact matching requires more work.

The local minima of the LCP array define the boundaries of the intervals. We next use

Listing 1.1: The Kasai Algorithm for LCP Construction (Kasai et al. 2001)

```
input S, SA

n ← size(S)
for i = 0 to n:
  isa[sa[i]] ← i

lcp[0] ← -1
lcp[|S|] ← -1
l ← 0
for i = 1 to n - 1:
  j ← isa[i]
  k ← sa[j - 1]
  while S[k + l] = S[i + l]:
    l ← l + 1
  lcp[j] ← l
  l ← max(l-1, 0)

output lcp
```



(A) An LCP tree for S.

(B) Super Cartesian tree for S

Figure 1.26: Two tree representations of the LCP intervals on the suffix array for the sequence $S =$ AAGTAAGG$.

two pointers, one for left, CL, and one for right, CR, to implicitly iterate the tree. The two global minima in Figure 1.25 are at positions 0 and 9. These are linked in order, i. e. the right pointer, CR, at position 0 is set to 9. There is no further minimum beyond position 9, thus it only gets a left pointer. The left pointer, $CL[9]$, points to the first local minimum in the current LCP-interval $0 - [0..8]$, namely position 1. At positions 1, 5, and 8 the LCP values are 0. Thus, $CR[1]$ points to 5, and $CR[5]$ points to 8. Further, 5 and 8 also have left pointers leading to deeper subintervals. Figure 1.26B visualizes this relationship.

This super cartesian tree (Ohlebusch 2013, Section 4.3.4) represents the structure of the lcp-intervals and also resembles the suffix tree. This structure has even been called a *linearized suffix tree* (D. K. Kim et al. 2008). It consists of only two pointers per element, CR and CL. These two pointers can be computed from the LCP array in time $O(n)$ with $O(n)$ memory (Ohlebusch 2013, Section 4.3; Frith and Shrestha 2018). Particularly, both pointers can be stored in a single child array. The value $CL[i]$ can be stored at $CR[i-1]$ as i is the border of an interval, and thus, $i-1$ cannot have a right pointer (Abouelhoda et al. 2004). See how for every CL value in Figure 1.25 the cell above right in CR is empty.

The child table now gives us a way to iterate the enhanced suffix array (ESA) for finding maximal matches as if it were a suffix tree. This requires two functions. Listing 1.2 contains the algorithm for `get_interval`, which given an lcp-interval finds the subinterval for a certain character. For instance, for the input $[0..8]$ and the letter `G` it produces $[5..7]$ which is the subinterval for all suffixes starting with the letter `G`. This method now has to be employed recursively (Listing 1.3). The function `get_match` takes the first character of the query and finds its subinterval by calling `get_interval`. It then checks whether the LCP value of the interval increased by more than one. For instance the interval $1 - [1..4]$ has the child interval $3 - [1..2]$. This extra character thus gets checked in the `for` loop.

The function `get_interval` iterates all subintervals of the current one, taking time $O(|\Sigma|)$. It gets called by `get_match` at most once per character until the maximal match is found. Thus, the total time to find the longest match q of a query Q in S is $O(|q||\Sigma|)$. As in our case the alphabet is small, the ESA is at par with the suffix tree.

With additional arrays the practical performance of the ESA can be improved. Reconsider the function `get_interval` which iterates the super cartesian tree horizontally. For each subinterval it has to check whether it corresponds to the currently searched character. Like in a suffix tree these edge labels are only stored implicitly and have to be retrieved via indexing such as `S[SA[i] + l]`. As the `l` value for each `i` can be computed in advance, this expression can be replaced by a single array `FVC[i]`. In Figure 1.25 the first variant character (FVC) follows immediately after the LCP and is highlighted in bold font. If more than one character is stored explicitly they are known as the discriminating characters or fringe (Wu 2016; Sinha et al. 2008). These methods provide a speed up of about 16 % at the cost of $O(|S|)$ memory (Klötzl 2015, Section 3.5 and p. 46).

The ESA is our data structure of choice to repeatedly solve the maximal match problem. However, each new search starts at the root of the LCP interval tree (Listing 1.3, line 5). Thus, the first few intervals are visited constantly leading to repeated work. Instead, bucket tables can be used to flatten the first d layers of the tree (Manber and Myers 1993; Abouelhoda et al. 2004). Each bucket stores the result of searching a string Σ^d in the ESA. Thereby, for a new query Q one can use the first d characters to jump to the correct interval in the index with just one table lookup. The size of the bucket table is

Listing 1.2: Finding a subinterval; adapted from Ohlebusch (2004, alg. 4.10 and 5.1).

```
fn get_interval
requires S, SA, LCP, CR, CL // ESA
input [i..j] // current interval
input char // the character to look for

if i = j:
  if S[SA[i] + l] != char:
    output ⊥
  else
    output i

m ← CL[j + 1]
l ← LCP[m]
do
  if S[SA[i] + l] = char:
    // found
    output [i..m-1]

  i ← m
  if i == j:
    break

  m ← CR[m]
while LCP[m] = l

if S[SA[i] + l] = char:
  output [i..j]
else
  output ⊥
```

Listing 1.3: Finding the maximal match; adapted from Ohlebusch (2004, alg. 5.2).

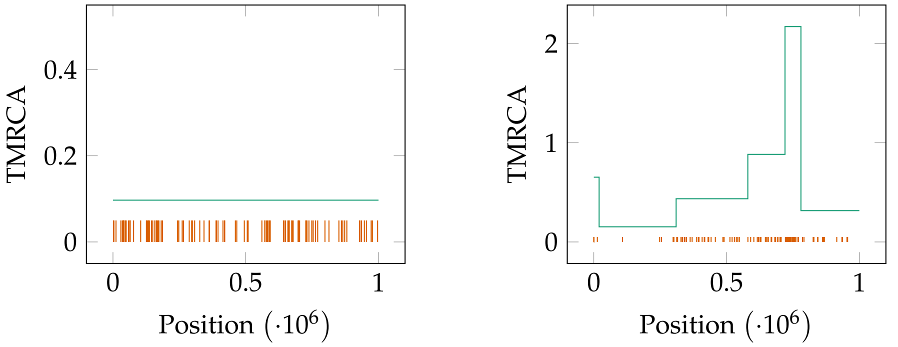
```
fn get_match
requires S, SA, LCP, CR, CL // ESA
input Q

in ← [0..|S|]
k ← 0
do
  sub ← get_interval(in, Q[k])
  if sub = ⊥:
    output in

  [i..j] ← sub
  l ← min(Q, LCP[CL[j]])
  for w = k + 1 to l:
    if S[SA[i] + w] != Q[w]:
      output in

  k ← l
  in ← sub
while k < |Q|

output in
```

thus determined by considering the trade-off between the speed gained by jumping an extra level in the tree, and the space taken up by the Σ^d entries. I have found a value of 10 for d to be practical for andi (Klötzl 2015, p.36).

1.7 Other Applications of Maximal Matches

Maximal matches not only form the basis of anchor distances, but have long been used by Haubold and colleagues in the analysis of genomes. Initially, Haubold et al. (2005) noticed that in human and mouse, the shortest unique substrings were 11 bp long, while in random 3 Gb sequences their expected length is 16 bp. Haubold and Wiehe (2006a) proposed an index of repetitiveness based on the length of unique substrings and used it to find that highly regulated regions in the human genome, such as the four *Hox* clusters of developmental genes, have very low values of repetitiveness compared to the rest of the genome. This makes sense as these regions are almost devoid of transposons, while mammalian genomes are generally littered with them. By inverting this observation it might be possibly to use complexity to find interesting genes, an idea Pirogov et al. (2019) returned to thirteen years later.

Going from shortest unique substrings to maximal matches allowed Haubold et al. (2009) to measure the similarity between genomes. If the probability of a substitution per site is π, then the expected waiting time or distance to the next substitution is $1/\pi$. The average match length is thus the inverse of the mismatches per site. Domazet-Lošo and Haubold (2009) implemented this measure of genetic distance in their program kr (Section 1.4, p. 17) using a single enhanced suffix array for all sequences. In theory suffix



(A) TMRCA without recombination.

(B) TMRCA with recombination.

Figure 1.27: Time to the most recent common ancestor (TMRCA) along a pair of sequences. Vertical bars indicate a substitution.

array construction scales linearly in the size of the input data, but in practice suffix array construction can be memory intensive so this approach of placing all data into a single suffix array was later abandoned.

DNA sequences of closely related organisms tend to recombine to form new combinations of alleles. In eukaryotes, this usually takes place during mitosis, but might also be due to introgression. In prokaryotes and viruses, recombination is called horizontal gene transfer. Regardless of the terminology, all forms of recombination lead to a variation in the time to the most recent common ancestor (TMRCA) along two or more sequences. Figure 1.27A shows the case without recombination, where the TMRCA is flat and the mutations plotted along the x-axis are uniformly distributed. In contrast, Figure 1.27B shows the case with recombination, where the TMRCA varies along the sequence. As the local mutation rate is proportional to the local TMRCA, recombination leads to clustering of mutations.

In kr, match lengths were looked up at every position in the genome. This is simple and hence fast, but has the disadvantage that a match of length l is counted almost l times, giving greater weight to long matches than to short matches. Haubold and Pfaffelhuber (2012) mitigate this problem by basing the estimation of π on the full distribution of mismatch lengths rather than just the mean.

Instead of correcting for recombination, it can also be investigated directly. Haubold et al. (2013) used the fact that the distribution of match lengths is sensitive to recombination to devise a statistical test for recombination implemented in their program rush. Having detected the presence of recombination, biologists are often interested in localizing the recombination breakpoints. Horizontal gene transfer in bacteria leads to fluctuations in the most closely related sequence when comparing a query to a set of subjects. Domazet-Lošo and Haubold (2011) used maximal matches to reconstruct what is known as the mosaic structure of genomes.

In humans and other mammals recombination is not uniformly distributed along a chromosome, but instead happens at so-called hotspots. Quantifying the rate of recombination can be achieved by counting allele-specific PCR-reactions across hotspots, a technique known as sperm typing. Sperm typing is often accompanied by enrichment of target fragments using oligos that bind the target in all individuals but no other region in the genome. Odenthal-Hesse et al. (2016) devised a complexity measure to pick the oligos with the lowest microsatellite content and hence the lowest likelihood of false positives via cross-hybridization.

Pirogov et al. (2019) implemented a sliding window version of this complexity measure in their program macle. Macle allows separate index construction and index traversal, because traversals are often repeated and can be orders of magnitude faster than construction in large genomes. Application of macle to genomes of human and mouse uncovered the *Hox* clusters detected 13 years prior using the less efficient index of repetitiveness of Haubold and Wiehe (2006a). In addition Pirogov et al. (2019) found that high-complexity regions are strongly enriched for developmental genes.

Other groups have also used maximal matches to great effect in genome comparison. For instance, in the years 1997 and 1999 two different strains of *Helicobacter pylori* became publicly available (Tomb et al. 1997; Alm et al. 1999). In order to compare these sequences Delcher et al. (1999) used maximal unique matches (MUMs). A MUM is a maximal match that cannot be extended to either side and is unique on both sequences. These can be efficiently found using a suffix tree (Ohlebusch 2013, pp. 184). Just like maximal matches are used as anchors in anchor alignments, MUMs can serve as the basis for pairwise alignments (MUMmer by Kurtz et al. 2004). Unlike andi, however, the MUMmer suite contains a component that creates pairwise alignments including gaps (Chapter A). Angiuoli and Salzberg (2011) chose MUMmer as one of the central components in their multiple sequence alignment tool, mugsy. For each pair of sequences a pairwise alignment is formed, all of which are then assembled into one big multiple sequence alignment. Owing to the performance of MUMs and MUMmer, mugsy is fast enough to produce alignments even for larger sets of bacterial genomes and hence I use it in this thesis to reconstruct reference phylogenies.

1.8 Structure of this Thesis

Phylogenies are at the heart of much biological research, ranging from classical species trees to genomic epidemiology. This chapter introduced how a phylogeny can be computed from sequenced genomes: via an alignment-based method such as maximum parsimony, maximum likelihood or using distances; or via alignment-free methods. I explained a number of approaches falling into the latter category with a particular focus on those employing maximal matches. I explained how maximal matches can be found using an enhanced suffix array and how they then serve as anchors for approximate local alignments in andi.

In Chapter 2, I explain improvements made to andi since its initial publication by Haubold et al. (2015). This includes improved performance made possible by a deeper analysis of the behavior of matches that do not serve as anchors. The next improvement is the addition of support values to phylogenies computed by andi. These values measure the uncertainty of each clade.

Chapter 3 focuses on the new program developed in my dissertation, phylonium. It first gives a broad overview and then explains individual steps in more detail. Particularly, it shows how maximal matches can serve as anchors for approximate multiple sequence alignments, called *anchor alignments*. These anchor alignments then allow applications previously not possible with alignment-free methods such as complete deletion. Further, this chapter describes some technical advances made to sequence comparison on the implementation level.

In Chapter 4 phylonium is compared to the most important alternative alignment-free and alignment-based approaches with respect to phylogeny reconstruction accuracy and run time. For that, I use simulated and real data commonly used in benchmarking.

Finally, in Chapter 5 I discuss the limitations of phylonium and explain how it might be further improved. I also point to additional applications of the anchor alignment underlying phylonium.

2 Improving Andi

> DNA is source code for the most complex machine in the known universe. Each chromosome contains a staggering amount of information, and the interaction between DNA and the cell machinery around it is incredibly complicated, with countless moving parts and Mousetrap-style feedback loops. Even calling DNA "source code" sells it short—compared to DNA, our most complex programming projects are like pocket calculators.
>
> Randall Munroe

> The traditional wall to statistical knowledge is blocked, for most, by a formidable wall of mathematics.
>
> Bradley Efron

2.1 Faster Matching

Reconsider the matching framework from Section 1.6. Given an indexed subject sequence S and a query Q, we look for the longest prefix of Q that appears anywhere in S. If this match is long, it is likely to arise from common evolutionary descent. If the sequences are unrelated, however, or the homology is broken by a mutation early on Q, the match will be short. Peter Pfaffelhuber summarized this relation in the following equation (eq. (4) Haubold et al. 2009). It is the limit for increasing divergence d of Equation 1.2 in Section 1.4.

$$P(X^* > l) = 1 - \sum_{k=0}^{l} 2^l \binom{l}{k} p^k \left(\frac{1}{2} - p\right)^{l-k} \left(1 - p^k \left(\frac{1}{2} - p\right)^{l-k}\right)^{|S|} \tag{2.1}$$

Here, $P(X^* > l)$ gives the probability that a random maximal match is longer than l. By default, we use the 97.5 % quantile of this distribution as the minimum length, t, of an anchor. Conversely, an anchor of this length has a p-value of 2.5 % of not being homologous. For example, for a 100 kb sequence (twice if the reverse complement is included), and a GC-content of 0.5, the threshold t is 12 nucleotides.

Figure 2.1 visualizes the distribution of match lengths for different substitution rates on simulated sequences. As sequences become more related, the peak becomes less pronounced and more mass of the distribution is in the long tail. Conversely, given a sample distribution of match lengths, one can estimate the substitution rate. This approach was taken by kr (Haubold et al. 2009, Section 1.4).

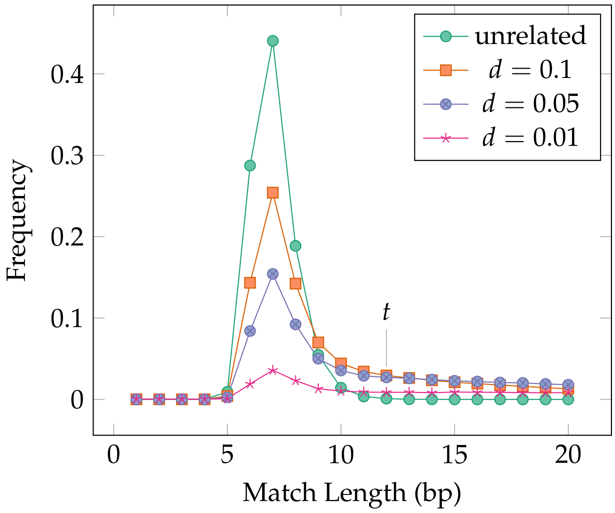


Figure 2.1: The Distribution of Match Lengths. Simulated were two 100 kb sequences, repeatedly matched one against the other, with match lengths counted. If the plot were not based on simulated sequences, but on Equations 2.1 and 1.2, it would look virtually identical. Figure 2.2 proves how well the theory models the behavior of anchor distances.

Equation (2.1) approximates the length of maximal matches for unrelated sequences. Analog to Equation 1.2, this can be adopted to related sequences with an average substitution rate of d. I write M to distinguish it from the unrelated case.

$$P(M = m) = (1 - e^{-dm})P(X^* \leq m) - (1 - e^{-d(m-1)})P(X^* \leq m - 1) \tag{2.2}$$

Consider a region flanked by two equidistant anchors which is thus homologous. After the left anchor, all subsequent maximal matches are too short to serve as a right anchor until one is found, eventually. Calling the flanked region a *gap* and its length g, I sum over all possible combinations of too short matches to calculate the probability of such a gap.

$$P(G = g) = \sum_{i=0}^{t} P(M = i)P(G = g - i) \tag{2.3}$$

Using Equation (2.2), the initial value $P(G = 1) = P(M \geq t)$, and a dynamic programming scheme, one can compute the theoretical distribution of gaps between anchors. Figure 2.2 depicts the virtually indistinguishable theoretical distribution and measurements on simulated sequences side by side. The peak at $G = 1$ shows that a pair of anchors commonly flanks only a single mismatch. For lower substitution rates this is even more pronounced.

As anchor pairs tend to flank only single mismatches, it makes sense to optimize for this configuration: I decided to relax the uniqueness criterion for right anchors. A left

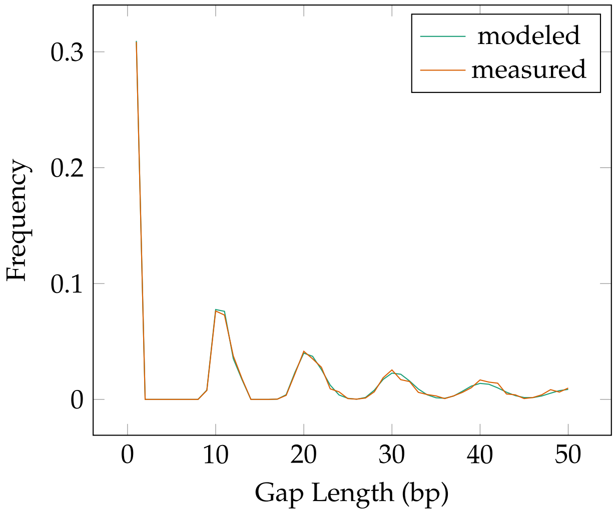


Figure 2.2: The Gap of Anchor Pairs. Two 100 kb sequences with a substitution rate 0.1 were used. At a p-value of 0.025 the minimum length t is 12 nucleotides. Approximately 30 % of anchor pairs span just one intermediate nucleotide.

anchor always ends in a mismatch, however, one could continue comparing characters beyond it and, with a reasonable probability, find at least t more matching characters. The anchor matching procedure of andi (Listing B.1) references a function, `lucky_match`, which does exactly that: given the position of the last match, it checks whether the next match starts immediately beyond the terminating mismatch, leading to a *lucky anchor*. If the search fails, andi falls back to a new search using the ESA. For two 100 kb sequences with a 0.1 substitution rate the quick check is viable 30 % of all anchor pairs (Figure 2.2). With this new trick, andi is about 15 % faster.

2.2 Limits

Since its initial release, andi has become faster through repeated optimization such as the *lucky anchors*. However, the design of andi limits its performance. For instance, to estimate the evolutionary distance between two sequences, one first has to find the homologous regions. Andi does that by using anchors for gap-free local alignments. It has to do so for every pair of input sequences, in both directions. Thus, for a dataset of n sequences, n full-text indexes need to be built and $n^2 - n$ comparisons have to be made. So andi inherently scales quadratically with the number of input sequences. To improve this design, I reduced the quadratic part of andi's algorithm. The result of this redesign is the program phylonium, which is much faster than andi, while retaining most of its accuracy. Perhaps surprisingly, phylonium is not only faster than andi, it also uses less memory. The central concepts of phylonium are given in Chapter 3.

With andi sequence comparison is always strictly pairwise, thereby maximizing the local homology between the two sequences. However, as the gene content can differ greatly in bacterial species, distances estimated will be based on different sections of the genome, similar to measuring with different scales. In MSA, this problem is often solved via *complete deletion* (Stecher et al. 2016). If a column contains a gap, that whole column is ignored for phylogeny reconstruction (Section 1.2). As andi only computes approximate pairwise alignments, it cannot carry out complete deletion. Section 3.2 explains how the complete deletion is implemented in phylonium.

Phylogenies usually come with support values (Felsenstein 2004, ch. 20). These support values represent the reliability of a node or split when recovering it on an independent but identically distributed set of characters. In alignment-based methods this can be achieved via resampling the columns of an alignment (Section 1.2). As andi is alignment-free that same method cannot be applied here, and other techniques have to be used to express the uncertainty in phylogeny reconstruction as explored in the following sections.

2.3 Support Values

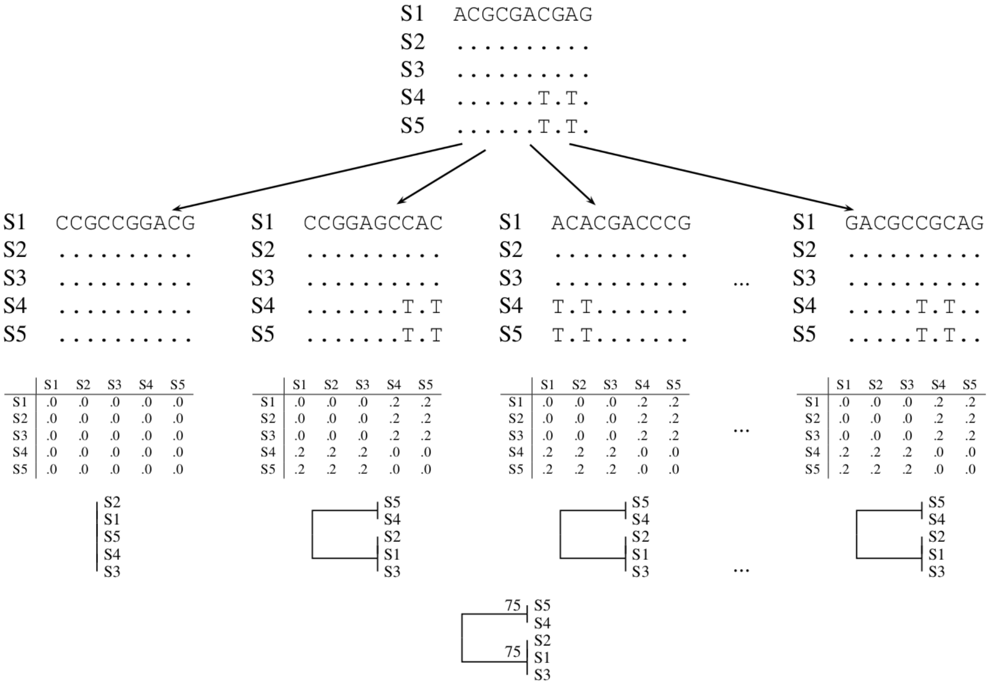
There are many ways to compute a phylogenetic tree with Chapter 1 giving an overview of the most commonly used methods. Maximum likelihood approaches guarantee to recover the most fitting tree given the data and a suitable model of sequence evolution. This confidence, however, only applies to the phylogeny as a whole and not to individual clades. Particularly, individual splits might only be supported by a small number of segregating sites, or worse, artifacts such as sequencing errors (Md Mukarram Hossain et al. 2015). Even the most likely tree might not be a good fit.

Where in classical statistics one would give a confidence interval of an estimated parameter to express the uncertainty, the same is not that easy with phylogenies. Instead, phylogenies are usually annotated with support values. A high support value implies a high confidence in the reconstructed clade or branch. Conversely, a low support value indicates that there exist alternative topologies that could also explain the data. The bootstrap, the most commonly used method for support values, can only be applied to phylogenies based on an MSA (Felsenstein 1985). In the wake of andi we developed the pairwise bootstrap that can be applied to alignment-free methods (Klötzl and Haubold 2016). In the following I compare these two, and two additional methods based on quartets, for computing support values.

2.4 Classical Bootstrap

The *bootstrap* is a method to estimate the distribution of a random variable, X, given a sample of size n from which X can be calculated (Efron 1979). New samples can be generated from the original by redrawing n elements with replacement, followed by recomputing X.

Consider a six-sided die of which one is interested in the expected roll. By way of example, when rolled five times, it produces the sequence $1, 2, 3, 5, 5$ the mean of which is 3.2. Using the bootstrap, instead of rerolling the die, one resamples from the above sequence with replacement, for instance $1, 1, 2, 2, 5$, giving a mean of 2.2. Repeating this many times gives a distribution of means that is centered close to the expected value, 3.5.



	S1	S2	S3	S4	S5
S1	.0	.0	.0	.0	.0
S2	.0	.0	.0	.0	.0
S3	.0	.0	.0	.0	.0
S4	.0	.0	.0	.0	.0
S5	.0	.0	.0	.0	.0

	S1	S2	S3	S4	S5
S1	.0	.0	.0	.2	.2
S2	.0	.0	.0	.2	.2
S3	.0	.0	.0	.2	.2
S4	.2	.2	.2	.0	.0
S5	.2	.2	.2	.0	.0

	S1	S2	S3	S4	S5
S1	.0	.0	.0	.2	.2
S2	.0	.0	.0	.2	.2
S3	.0	.0	.0	.2	.2
S4	.2	.2	.2	.0	.0
S5	.2	.2	.2	.0	.0

...

	S1	S2	S3	S4	S5
S1	.0	.0	.0	.2	.2
S2	.0	.0	.0	.2	.2
S3	.0	.0	.0	.2	.2
S4	.2	.2	.2	.0	.0
S5	.2	.2	.2	.0	.0

Figure 2.3: Bootstrapping a Multiple Sequence Alignment by Resampling the Columns; Taken from Klötzl and Haubold (2016, CC-BY-4.0).

In the context of phylogeny reconstruction, the bootstrap is applied to the columns of a multiple sequence alignment (Felsenstein 1985). Figure 2.3 shows the method in action: From one original alignment, columns are repeatedly resampled with replacement to give new alignments. These are then subjected to the same phylogeny reconstruction algorithm as the original.

For every clade in the original tree it is then checked how many times they also appear in any of the bootstrapped phylogenies. This *support value* is then written next to the node. A support value of 75 % thus means that one quarter of the bootstrapped replicates do not reproduce that particular clade. In contrast to p-values, support values are generally desired to be large. A high support value thus inspires confidence in the phylogeny.

The bootstrap has the important property that if the original sample is representative and many bootstrap samples were generated, the estimated null distribution of the random variable investigated, in our case the phylogeny, will be accurate in many cases (Singh 1981; Bickel and Freedman 1981). Additionally, the bootstrap is computationally simple and fast for small data sets.

However, the bootstrap was introduced into phylogenetics in the mid 1980s when genetic data was scarce and trees were small. To overcome the computational requirements for whole genome phylogenies, faster variations were invented. These make the process of bootstrapping and subsequent tree computation faster, but do not change the underlying methodology (Stamatakis et al. 2008; Hoang et al. 2017).

Table 2.1: Six artificial sequences. The nucleotides are visually split into logical blocks.

			11111	11
	123	456789	01234	56
A	AAA	CAAAAA	CAAAA	AA
B	AAA	ACAAAA	CAAAA	CC
C	AAA	AACAAA	ACACC	CC
D	AAA	AAACAA	ACCAC	AA
E	AAA	AAAACA	ACCAC	AA
O	CCC	AAAAAC	AAAAC	AA

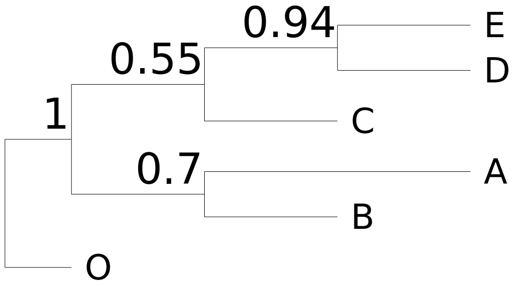


Figure 2.4: A phylogeny with support values computed via bootstrapping.

To explore the bootstrap further, consider six nucleotide sequences, each sixteen residues long (Table 2.1). Sequence O is an outgroup, distantly related to the others. Each sequence has a singleton mutation specific to that sequence (columns 4 through 9) giving a star phylogeny, (A,B,C,D,E,O). Columns 10 to 14 group clades (C,D,E) and (A,B,O) to give the tree (((A,B), O), (C, (D,E))). However, the substitutions in columns 15 and 16 are shared only by B and C. Under the infinite sites model, where each site may only be hit by a mutation once, such homoplasies can arise only by recombination.

Figure 2.4 shows the result of bootstrapping this artificial data set with one hundred replicates. Each split is labeled by the frequency with that it appeared in the replicates; i. e. in 94 replicates, the two sequences D and E were grouped together. In 70 cases A and B came together. In only 55 cases, C was grouped with D and E. This low support value is caused by the homoplasy.

2.5 Pairwise Bootstrap

The classical bootstrap requires an MSA, which is not given in alignment-free sequence comparison. However, in andi we have *pairwise* approximate alignments. One example of such an alignment is shown in the top panel of Figure 2.5. In Klötzl and Haubold (2016) we showed how the bootstrap can be extended to this kind of alignments. A similar

Original

```
S1 TTCAACCCGTAGTCCGAGCCGTCTCATGATCGCTTACCCG
S2 ......................T.................
```

Replicate 1

```
S1 ACATTAGCCGCTTATTGGCATTTCAGCTCCCCCCGGTGAA
S2 ........................................
```

Replicate 2

```
S1 TGTCTCCCCCCCCAAGACCAGCTTAATAACAGACACCGAG
S2 .................T...T..................
```

Replicate 3

```
S1 TCAGCGTGTGCTTGCGGAGCTCCACCCATTACAACACGCG
S2 ...................T....................
```

Figure 2.5: Pairwise Bootstrap. Identical nucleotides are represented by a dot.

method had previously been used for rearrangement data (Y. Lin et al. 2012).

Figure 2.5 depicts a situation as it would commonly arise in andi or phylonium: two anchors are separated by a single mismatch. This approximate alignment can be bootstrapped. Figure 2.5 features three bootstrapped replicates created via resampling the columns.

The original pairwise alignment has 40 matching nucleotides, with one mismatch in between. During the bootstrap procedure, there is a one in 40 chance of drawing the mismatch from all homologous columns. After creating the replicates, one again is only interested in the number of mismatches for each to be used in phylogeny reconstruction. Thus, the pairwise bootstrap simplifies to a Bernoulli process and can be modeled—without explicit drawing and redrawing—by the binomial distribution.

For each pair of sequences one first needs the number of homologous nucleotides h, and the number of mismatches m. In our example $h = 40$ and $m = 1$. The pairwise bootstrap thus simplifies to the binomial distribution $B(h; m/h)$, where the number of samples is h and m/h the success probability. For a new replicate we just draw a new sample from this distribution. In andi and phylonium we implement this using the Gnu Scientific Library (Galassi et al. 2016). This process can also be extended to more complex models of sequence evolution such as Kimura two-parameter with a multinomial distribution (Kimura 1980).

This pairwise bootstrap process is simple and fast. Unfortunately, its usefulness declines with the sequence length n. Let X be a random variable representing the amount of mismatches in a new bootstrap replicate with the distribution $X \sim B(n; p)/n$. Then the variance is $\mathrm{Var}(X) = p(1-p)/n$. So with increasing sequence length the data will be less perturbed and the support values approach 1.

Figure 2.6 depicts the result of pairwise bootstrap on the running example. For this small data set the numbers are very similar to the classical bootstrap. As already noted in Klötzl and Haubold (2016), pairwise support values tend to be lower than those derived from classical bootstrap.

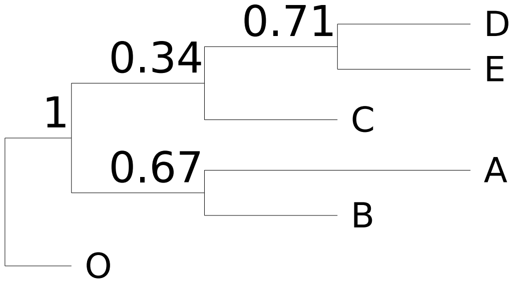


Figure 2.6: A phylogeny with support values computed via pairwise bootstrapping.

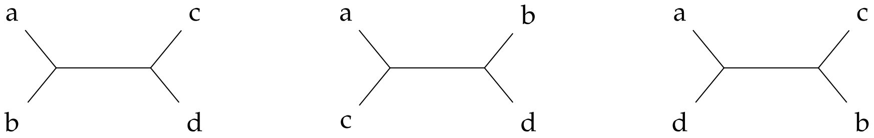


Figure 2.7: For four taxa, there are three possible topologies.

2.6 Quartet Analysis

Alignment-free methods use a distance matrix as an intermediate step in phylogeny reconstruction (Chapter 1). Guénoche and Garreta (2001) invented the *quartet analysis* to detect how well a tree fits the associated distance matrix. A quartet is the topology of four leaves. Figure 2.7 shows the three possible quartets for the leaves a, b, c, d on a tree. Here a tree is defined as a connected, cycle-free graph with nodes of degree one or three, i. e. an unrooted tree.

Given an unrooted tree T and a distance matrix M, of the three topologies for any quartet only one will be embedded in the tree. Consider Figure 2.8, each internal edge divides all leaves into four sets. Picking one leaf per set gives a quartet, i. e. $a \in A, b \in B, c \in C$, and $d \in D$. This quartet *supports* the edge if e is its best configuration.

$$M_{ab} + M_{cd} < \min\{M_{ac} + M_{bd},\ M_{ad} + M_{bc}\}$$

Iterating over all possible quartets one can compute a support value for that particular edge (Guénoche and Garreta 2001). I here use an indicator variable $\mathbb{1}$ that has the value 1 if a quartet shows support for an edge, otherwise 0.

$$R_e = \frac{\sum_{a,b,c,d} \mathbb{1}(a, b, c, d \text{ support } e)}{|A||B||C||D|}$$

We implemented a simple $O(n^5)$ algorithm to compute the quartet support values (Klötzl and Haubold 2016): For each edge of the tree, iterate over all possible induced quartets and count how many of them are supporting. This implementation computes the support values for a tree of the 2681 *E. coli* in 12 minutes. An alternative to enumerating all quartets per edge is to only take a random sample. On the same data set, at a sample

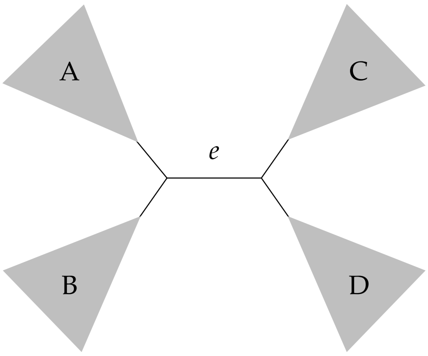


Figure 2.8: Each internal edge splits a tree into four disjunct sets of leaves.

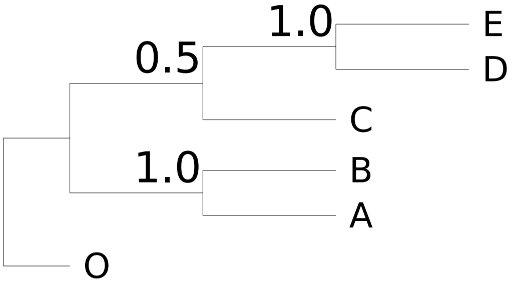


Figure 2.9: A phylogeny with support values computed via quartet analysis.

size of 100, that reduces the runtime to 20 seconds at a negligible change in support values.

Figure 2.9 shows the result of quartet analysis on the running example. It correctly gives low support for the branch separating C, D, E from A, B. As this example contains only six taxa, the value of 0.5 means that two of four quartets support this split. It should be noted, that where bootstrapping can give a support value in quantities of the number of replicates, quartet analysis inherently scales with the number of quartets formed by an edge.

2.7 Quartet Concordance

In the previous section I defined R_e as the fraction of supporting quartets per edge. In a recent paper the following alternative was proposed (Pease et al. 2018): given a tree and an MSA, sample up to $N = 200$ quartets per edge. For each quartet compute the likelihood of all three possible topologies (Figure 2.7). A quartet is said to agree with an edge if that topology has the highest likelihood. Let n_1 be the number of quartets agreeing with an edge and n_2, n_3 the alternative topologies. The *quartet concordance* is

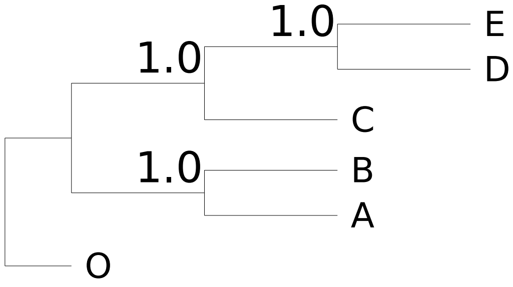


Figure 2.10: A phylogeny with support values computed via quartet concordance.

then defined as

$$QC_e = 1 + \sum_{i \in \{1,2,3\}} \frac{n_i}{N} \log_z \left(\frac{n_i}{N} \right) \, .$$

Here z is the number of non-zero counts out of n_1, n_2, n_3. Additionally, if there is more support for an alternative edge ($n_1 < \max\{n_2, n_3\}$), the sign of QC_e is inverted. Thus, QC_e is a value in the range $[-1, 1]$. A positive value indicates support for the given edge, values around zero prefer none of the three topologies, and negative scores indicate support for an alternative topology. The quartet concordance thus has the unique feature of differentiating between low support and counter evidence.

For the running example the quartet concordance does not detect the recombination in the data set (Figure 2.10). However, there is another value computed in parallel, the *quartet informativeness*. A quartet is only included in QC_e, if its likelihood is a few orders of magnitude larger than the next best arrangement. The QI_e measures the amount of quartets contributing to the QC_e value. For the edge separating C, D, E and A, B, O this value is 0.5. So half of the quartets agree with the given topology and for the other half the best two options were equally likely and thus considered non-discriminatory.

2.8 Comparison

To further explore the usefulness of the different ways to compute support values, I applied them to a data set of 29 *Escherichia coli/Shigella* genomes. Figure 2.11 shows the phylogeny as computed by andi annotated with the support at three branches. The support values are given in the order of the classical bootstrap, pairwise bootstrap, quartet analysis and quartet concordance, respectively. For the top most given branch, labeled A), only the quartet methods produce a support value less than 1.

The second set of support values, B), splits the cluster of the *K12* strain. These genomes are particularly closely related with the number of substitutions varying from 11 to 130. This is one or two magnitudes lower compared to the rest of the data set. Out of one hundred replicates in the classical bootstrap, none produced an alternative topology. The pairwise bootstrap is less certain and gives a support value of 0.73. The quartet analysis finds no deviating topology. However, the quartet concordance, is very low with just 10^{-5}. This near-zero value can be interpreted such that no topology is favored, all are

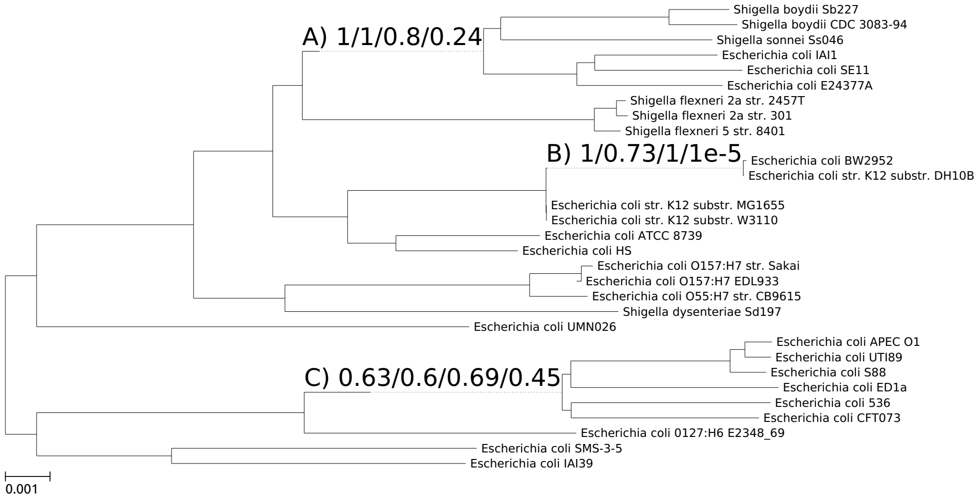

A) 1/1/0.8/0.24
Shigella boydii Sb227
Shigella boydii CDC 3083-94
Shigella sonnei Ss046
Escherichia coli IAI1
Escherichia coli SE11
Escherichia coli E24377A
Shigella flexneri 2a str. 2457T
Shigella flexneri 2a str. 301
Shigella flexneri 5 str. 8401
B) 1/0.73/1/1e-5
Escherichia coli BW2952
Escherichia coli str. K12 substr. DH10B
Escherichia coli str. K12 substr. MG1655
Escherichia coli str. K12 substr. W3110
Escherichia coli ATCC 8739
Escherichia coli HS
Escherichia coli O157:H7 str. Sakai
Escherichia coli O157:H7 EDL933
Escherichia coli O55:H7 str. CB9615
Shigella dysenteriae Sd197
Escherichia coli UMN026
Escherichia coli APEC O1
Escherichia coli UTI89
Escherichia coli S88
C) 0.63/0.6/0.69/0.45
Escherichia coli ED1a
Escherichia coli 536
Escherichia coli CFT073
Escherichia coli 0127:H6 E2348_69
Escherichia coli SMS-3-5
Escherichia coli IAI39
0.001

Figure 2.11: A Phylogeny for 29 *E. coli/Shigella* genomes. Support values are given in the order of classical bootstrap, pairwise bootstrap, quartet analysis and quartet concordance, respectively. Note that the branch lengths have been extended to make room for the labels. Particularly branch B) is very short.

equally likely. This is a reasonable answer, as any topology is supported by only very few substitutions.

For the bottom set of support values, C), all methods produce a mediocre support value. This is correct, as that branch is a known site of recombination. For instance, the two sequences of *E. coli 536* and *E. coli ED1a* are prone to be relocated to different splits on the tree depending on the reconstruction method. Comparing the two sequences using rush I found a strong indicator of recombination, $p < 4.4 \cdot 10^{-120}$ (Section 1.7; Haubold et al. 2013).

After evaluating the four methods for computing support values on simulated and real data, I cannot claim that one is better than the other. There seems to be no generally applicable solution. The classical bootstrap proves to be insensitive on whole genomes (see the K12 branch, above). Quartet concordance produces the most meaningful results. For the K12 strain, it hints that no topology is favorable, showing that the optimum is a star, not a tree. However, it is also the computationally most expensive method. Not only does it require an MSA, for each branch the likelihood for numerous topologies is recomputed. Where the other methods only took a few seconds, the quartet concordance took 45 minutes to produce results for the 29 *E. coli* genomes.

With its initial publication, andi was almost as accurate as alignment-based methods albeit being much faster (Haubold et al. 2015). Both was mainly due to the approximate pairwise alignments based on an efficient implementation of maximal matches (Section 1.6). Since then, the program has been further optimized (Section 2.1) and extended with features such as the pairwise support values (Section 2.3). However, the amount of sequence data continues growing and andi will be considered slow when compared to newly developed, faster but less accurate methods. Thus, in the next chapter I explain our new approach phylonium, which keeps the ideas of anchors, but computes only a single approximate alignment for multiple sequences.

3 Implementing Phylonium

> During the last few years, rapid sequencing of DNA has become feasible, and data on nucleotide sequences of various parts of the genome in diverse organisms have started to accumulate at an accelerated pace.
>
> Matoo Kimura

> When we had no computers, we had no programming problem either. When we had a few computers, we had a mild programming problem. Confronted with machines a million times as powerful, we are faced with a gigantic programming problem.
>
> Edsger Dijkstra

With andi we knew that anchor distances accurately estimate evolutionary distances (Haubold et al. 2015). However, the run time of andi scales inherently quadratic with the number of sequences. For a data set of n sequences, n indices are built and $n(n-1)$ pairwise comparisons performed. A tool that builds only one index and piles all other sequences on this reference would be much faster. That idea is implemented in phylonium (Klötzl and Haubold 2019).

This chapter covers the implementation of phylonium in detail. First, a broad overview of the algorithm is given. The remaining sections then cover more individual aspects and optimizations down to instruction levels.

3.1 An Overview of Phylonium

Figure 3.1 is a flow chart depicting the algorithm of phylonium in seven steps. First, the sequence data is read and stored in memory **(A)**. The data is also normalized as files may contain residues such as an `N`, indicating an unidentified nucleotide as opposed to just the four canonical bases `ACGT`. Characters other than `ACGT` are removed from the data.

The old program, andi, now would create an index for each sequence and continue with pairwise comparison. In contrast, phylonium builds only a single index for one of the sequences in the whole data set **(B)**. This particular input sequence is called the *reference*. If no reference is explicitly set by the user, a sequence of median length is picked by default (Section 4.4). For this reference an ESA is built using the libdivsufsort library (Fischer and Kurpicz 2017). Going from n indices to just one not only makes the program faster, it also provides a common system of coordinates for the pairwise comparison step.

Next, all other sequences are aligned against the reference **(C)**, producing an *anchor alignment* (Figure 3.2). Note that, to save memory, a homologous segment does not retain

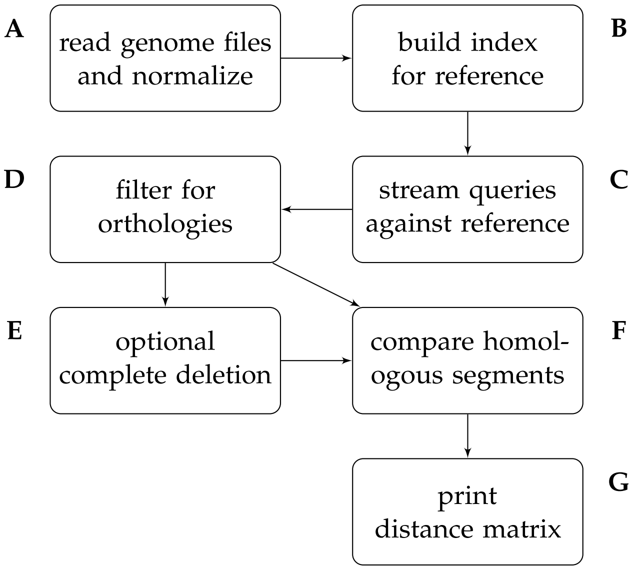

A
read genome files and normalize
B
build index for reference
D
filter for orthologies
C
stream queries against reference
E
optional complete deletion
F
compare homologous segments
G
print distance matrix

Figure 3.1: A flow chart presenting the seven central steps of phylonium.

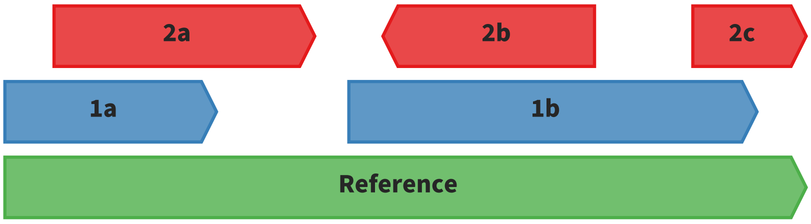


Figure 3.2: An Anchor Alignment. Each homologous segment begins and ends with an anchor, an exact match to the reference sequence.

the information where anchors, and thus, exact matches are located. Hence, the many exact matches contained in anchors are again compared during distance computation (Section 3.3). In contrast, in andi anchors could be skipped and only the gaps in between needed to be compared.

Distance estimation should ideally be based on orthologous regions. The homologies found by phylonium thus need to be filtered for duplications **(D)**: If homologous segments from the same query sequence overlap on the subject, all but one instance are removed (Section 3.2).

At this point all homologous segments from all queries are ordered with respect to the same coordinate system, the reference sequence, as depicted in Figure 3.2. This allows an implementation of complete deletion similar to MSAs, introduced in Section 1.2 **(E)**. Instead of explicitly scanning each column for gaps, with phylonium I can use a sweeping line approach (Section 3.2).

Once the set of homologous segments has been determined, with or without complete deletion, the next step is the pairwise comparison **(F)**. For this, phylonium looks at all

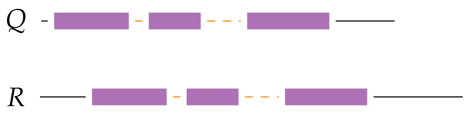


Figure 3.3: An Anchor Alignment. The blocks represent anchors. Spanning from left to right these anchors form a homologous segment.

overlapping segments from two sequences and compares them nucleotide by nucleotide. At this point it needs to decide whether one or even both of the segments match the reference in reverse. Phylonium calls the optimized routines for the comparisons to count the number of mismatches between all pairs of sequences (Sections 3.3 and 3.5).

As a final step, the number of mismatches for each pairwise comparison are transformed into evolutionary distances via the Jukes-Cantor correction (Jukes and Cantor 1969). This method corrects for hidden substitutions (e. g. A $\rightarrow$ T $\rightarrow$ A, Section 1.2). Finally, phylonium prints the distance matrix in Phylip format **(G)**.

3.2 Anchor Alignment

Phylonium is categorized as *alignment-free* as it does not require an MSA as one of its inputs. However, it builds a similar structure internally that can be considered an approximate gap-free MSA, the anchor MSA. Figure 3.3 depicts the reference R and a query sequence Q. Using the ESA, phylonium detects the longest prefix of Q that also appears in R (Section 1.6 and Listing 3.1). If this match is unique and longer than the threshold t, it is an anchor (Section 1.5). The next nucleotide is guaranteed to differ between Q and R and is skipped. If the match starting right after it is another anchor, this right anchor and the previous left anchor form a pair if they are equidistant. Thus, an anchor pair consists of two exact matches with the same amount of intermediate nucleotides on both sequences. Similarly, Figure 3.3 depicts three equidistant anchors, which then form a single local alignment.

The algorithm in Listing 3.1 makes use of the techniques developed in Section 1.6 for quickly finding maximal matches. In particular, it always finds the longest exact match starting from any position q. In line 39 q gets increased by the length of the current match plus one for the substitution at its end. Finding such a match takes time $O(|P||\Sigma|)$ where $|P|$ is the length of the maximal match. In total the run time per query thus is $O(|Q||\Sigma|)$. The memory requirement is small as only a single ESA and the list of homologous segments need to be stored. The ESA computation consumes $O(|R|)$ memory, the homologous segments $O(|Q|)$.

Streaming a query against the reference results in many of these local alignments (Figure 3.2). Of these segments phylonium only stores the beginning and end points on both sequences. Implicitly this also encodes the direction of homology as is necessary for reverse matches such as segment 1c in Figure 3.4. All the segments from one query are then piled onto the reference. The evolutionary model built into our anchors guarantees that the segments are homologous, i. e. they arise from common descent. However, homologies may also arise within a sequence. Figure 3.4 depicts a situation that can easily

Listing 3.1: Matching a query *Q* against a reference *R* in phylonium producing a list of homologous segments *h*. The enhanced suffix array is denoted by *E*.

```
fn anchor_alignment
requires R, E
input Q

let t ← threshold(R)

let last_pos_q ← 0
let last_match ← ⊥
let last_was_right_anchor ← false

let current ← segment(0,0) // pos and length
let q ← 0
while q < |Q| do // Stream the complete query

  // Find the next maximal match
  m ← lucky_match(R, Q, last_match, q) or get_match(E, Q[q...])
  if m.is_unique and m.length ≥ t then
    // m is an anchor

    if q > last_pos_q and q - last_pos_q = m.pos - last_match.pos then
      // Found a pair
      current.length ← current.length + q - last_match.end + m.length
      last_was_right_anchor ← true
    else
      // new anchor but breaks current segment
      if last_was_right_anchor then
        h.append(current)
      end

      current ← m
      last_was_right_anchor ← false
    end

    // Save values for later
    last_pos_q ← q
    last_match ← m
  end

  // Skip the mutation
  q ← q + m.length + 1
end

output h
```

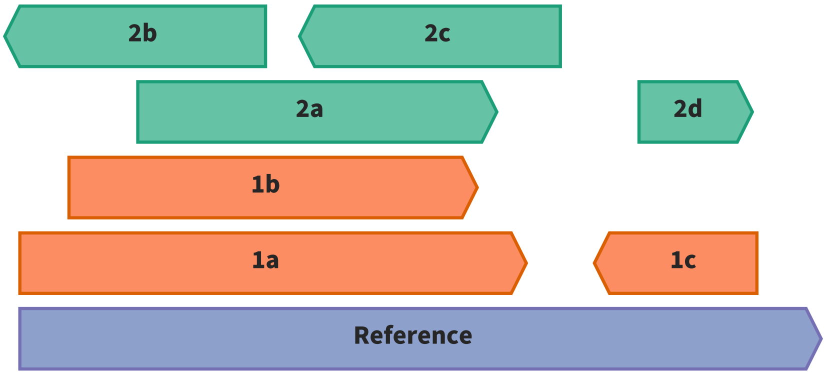


Figure 3.4: Multiple segments of each query align to the same region on the reference leading to ambiguous orthology.

crop up in sequence comparison. The reference contains a segment that appears twice in each query. If one were to compare all five homologous segments, that would be six comparisons: 1a×2a, 1a×2b, 1a×2c, 1b×2a, 1b×2b, and 1b×2c. Making all comparisons would bias the overall estimation. As copy number variations of transposons are common in prokaryotes, the local alignments have to be cleaned.

To remove the repeats and limit analysis to a single ortholog, Gao and Miller (2020) pick the copy with the longest maximal match as the primary ortholog. For phylonium we instead focus on chains of non-overlapping segments. Only segments on the chain with the maximum number of aligned nucleotides are kept. For instance, in Figure 3.4 the chain 1a-1c aligns more nucleotides than 1b-1c. Thus, the former is kept and 1b is removed. The algorithm to compute the best chain is given in Listing 3.2. For each homologous segment a score and a predecessor value is kept. An initial chain is set to contain only the leftmost homology. With the next element, one checks whether it can extend the first one without overlap. If so, the chain is extended and the score increased. Otherwise, a new chain is started. Iteration continues until all homologous segments are processed.

The highest value in `score` now gives the end of the chain with the most aligned nucleotides. Iterating through the `pred` array, one can visit all segments in the chain. The elements not visited are removed from the list of homologous segments for this query. This algorithm is related to the algorithm for determining the heaviest increasing subsequence and runs in time $O(m^2)$ where m is the number of homologous fragments (Ohlebusch 2013, Section 8.3). Even though $O(m \log m)$ algorithms exist, as m is usually much smaller than the sequence length, a quadratic algorithm is acceptable. As a result, all remaining homologous segments of a single query are now free of overlaps. Figure 3.5 depicts the situation after chaining.

Listing 3.3 gives the algorithm for complete deletion used in phylonium. The input is H, a list of homologous segments, and H_i is the list for sequence i. Each homologous segment has a start and an end point on the reference. First, they are sorted by the starting point. Then for each sequence one counter f_i is initialized to zero. These counters represent the segments to be checked for overlap and initially, this front is the leftmost

Listing 3.2: Chaining of homologous segments

```
input h // a sorted list of homologous segments

m ← size(h)
for i = -1 to m:
  pred[i] ← -1
  score[i] ← 0
  vis[i] ← false

score[0] ← h[0].length

for i = 1 to m:
  j ← $argmin_i$ {score[i] | h[j].end < h[i].start }
  pred[i] ← j
  score[i] ← score[j] + h[i].length

s ← $argmax_i$(score[i])
while s >= 0:
  vis[s] ← true
  s ← pred[s]

output [h[i] if vis[i]]
```

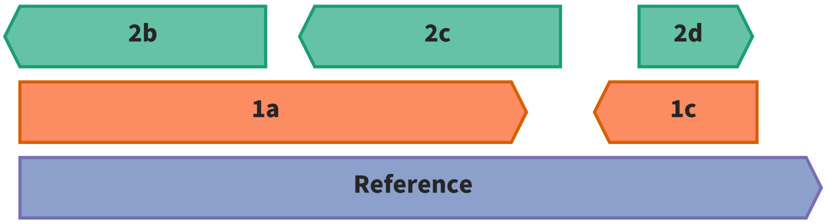


Figure 3.5: An anchor alignment with resolved duplications after chaining.

Listing 3.3: Complete deletion on sets of homologous segments

```
input $H_i$

for i = 0 to $n$:
  sort ($H_i$)
  $f_i$ ← 0

while each $f_i$ < size($H_i$) do
  common_start ← max$_i$($H_i$[$f_i$].start)
  $e$ ← argmin$_i$($H_i$[$f_i$].end)
  common_end ← $H_e$[$f_e$].end

  if start < end:
    for i = 0 to $n$:
      $D_i$.append( trim($H_i$[$f_i$], common_start, common_end))

  $f_e$ ← $f_e$ + 1
done

output $D$
```

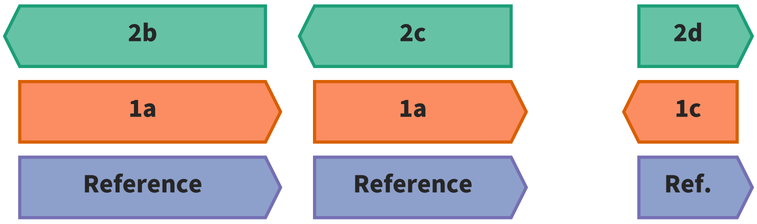


Figure 3.6: An anchor alignment after complete deletion.

segment per sequence. For instance, in Figure 3.2 the front consists of 1a, 2b and the Reference.

Among the segments of the front, the algorithm looks for the ones with the right-most start and the leftmost end. These values give the overlap of all segments in the front. If there is an overlap, a trimmed copy of all homologous segments is appended to the output alignment D_i.

The element e with the leftmost end limits the front to the right. It thus gets removed from the front and is replaced by its right neighbor (2b by 2c). This new front is then in turn checked for an overlap. Iteration continues until all segments of one sequence are processed.

The result of applying complete deletion to the example is given in Figure 3.6. Given n sequences and at most m homologous segments per sequence, the sorting step takes time $O(nm \log m)$. In each iteration of the while-loop, the maximum and minimum of n elements are found. Further, if there is an overlap, all n homologous segments need to be trimmed. Thus, each loop iteration is $O(n)$. In the worst case, there is one overlap per iteration. As there are at most mn homologous segments to consider, the total run time of this algorithm is $O(n^2 m)$, assuming already sorted homologous segments. In practice,

Listing 3.4: Comparing two sequences

```
size_t mismatches(const char *seq1, const char *seq2, size_t length)
{
 size_t num_mismatches = 0;
 for (size_t k = 0; k < length; k++) {
  if (seq1[k] != seq2[k]) {
   num_mismatches++;
  }
 }
 return num_mismatches;
}
```

the algorithm is very fast and does not impact the run time of phylonium.

3.3 Fast Sequence Comparison

As explained in Section 3.2, phylonium, unlike andi, does not record which parts of a homologous segment are made up of exact matches. This saves memory, but as a consequence, phylonium counts the mismatches in much longer regions for all $O(n^2)$ comparisons where n is the number of sequences. To ensure performance, I optimized the mismatch counting.

A number of libraries exist for computing a pairwise alignment, or for calculating the edit-distance between two sequences including mismatches and indels. Among these are seqan (Döring et al. 2008), biopp (Dutheil et al. 2006), and edlib (Šošić and Šikić 2017). However, for anchor pairs we assume they are already aligned and contain no indels. Thus, for phylonium, we need a quick method to count only the mismatches. Listing 3.4 shows such a function written in C (adapted from Seemann 2018).

The code compares the two sequences `seq1` and `seq2` byte by byte and increases a counter whenever two nucleotides mismatch. This code is simple but comes with a few performance problems. Firstly, byte-by-byte comparison is suboptimal, and secondly, branch misprediction can lead to an additional slowdown. Both issues can be overcome at the cost of higher code complexity.

A modern central processing unit (CPU) has so-called single instruction multiple data (SIMD) capabilities, allowing it to process multiple input data simultaneously. Compilers with *auto-vectorization* leverage the SIMD instructions for improved throughput. For the code from Listing 3.4, an optimizing compiler creates a vectorized version, which greatly improves the performance with no additional work required by the programmer. However, manual intervention still pays off: using compiler intrinsics, one can increase the throughput significantly (Fiori et al. 2017). Listing B.2 shows code two times faster than the machine code produced by the compiler for Listing 3.4. The new code loads 16 bytes from both sequences and then compares them for equality. This results in another 16 byte value containing a byte with all bits set to one for equal and all zeros for unequal nucleotides. Leveraging the `_mm_movemask_epi8` function from the Intel SSE2 instruction set, this long value is reduced by copying just the highest bit from each byte into a 16 bit value. The result is a mask containing a `1` for each equal nucleotide and a `0` for a mismatch.

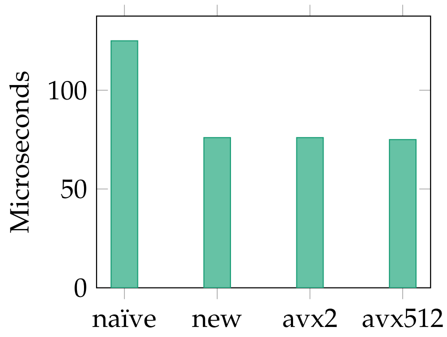


Figure 3.7: Run times for Comparing two Sequences of a Million Nucleotides. Each number is the average of a few thousand runs measured using the Google Benchmark library. The new method uses SSE2 instructions on 16 bytes at once. AVX2 and AVX512 operate on 32 and 64 bytes, respectively.

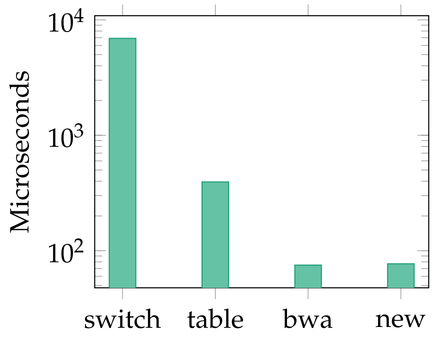


Figure 3.8: Run times for Computing the Reverse Complement of a Million Nucleotides. Each number is the average of a few thousand runs measured using the Google Benchmark library.

Inverting this using a bitwise not (~) and then counting the `1`s (`popcount`) computes the number of mismatches in the 16 byte chunks. All of these counts get accumulated and finally, for sequence lengths not divisible by 16, any remaining bytes are checked.

Depending on the CPU, its supported instruction set, and the length of the sequences, the new code is up to two times faster than the auto-vectorized version (Figure 3.7). Further improvements can be made by loop unrolling or using more advanced instruction set extensions such as AVX2 (Intel Corporation 2018). Even wider registers using AVX512 instructions can further reduce the number of loop iterations but also reduce the clock speed of the CPU. Thus, using AVX instructions slows down the regular code. Hence, they only make sense for workloads with large compute-to-memory ratios and have small advantage for sequence comparison (Tiwari et al. 2018). With this new method, two sequences from the same strand can be compared at 13.1 GB/s, 60 % faster than the 8.0 GB/s without these optimizations.

3.4 Computing the Reverse Complement

DNA is a double helix consisting of two strands. In a sequencing experiments reads may come from either of them. Thus, we also have to consider the opposite strand when looking for homologous regions (remember the homologous segment `1c` in Figure 3.4).

To compute the reverse complement, some programs iterate the given sequence in reverse order and complement individual bases using a `switch` statement, for instance mash and mummer (Ondov et al. 2016; Kurtz et al. 2004). This approach is simple and

Listing 3.5: Computing the reverse complement

```
void revcomp(const char *sequence, size_t length, char *dest)
{
  for (size_t i = 0; i < length; i++) {
    char nucl = sequence[length - 1 - i];
    int sum = nucl & 2 ? ’G’ + ’C’ : ’A’ + ’T’;
    dest[i] = sum - nucl;
  }
}
```

works on nucleotides in the common IUPAC notation (NC-IUB 1985). Another method uses an explicit data table for complementing (e. g. Altschul et al. 1990), which is approximately ten times faster (Figure 3.8). An even faster way to compute the reverse complement was created for BWA (Li and Durbin 2009): At first nucleotides are mapped to small integers.

$$\mathtt{A} \mapsto 0 \qquad \mathtt{C} \mapsto 1 \qquad \mathtt{G} \mapsto 2 \qquad \mathtt{T} \mapsto 3$$

These values are chosen so that the simple function $c(x) = 3 - x$ complements the nucleotides. BWA uses a value of 4 to represent non-canonical bases. While this approach is very fast, one first has to bring the sequences into this encoded format, which interferes with debugging as the strings are no longer human-readable. Thus, I created a new method for computing the reverse complement that works directly on unencoded sequences and is almost as fast as BWA.

The new method works by distinguishing between two cases of complementation: A $\leftrightarrow$ T and C $\leftrightarrow$ G. For each case the subtraction approach from BWA can be used. For instance, $f(x) = 149 - x$ complements between A and T with ASCII values. Assuming the input data contains only canonical bases, the second lowest bit can be used to load the correct constant for subtraction (Table 3.1). If the bit is set, load 158, which complements C and G, otherwise load 149.

Table 3.1: The Ascii Representation of Nucleotides

Nucl.	Bits
	8765 4321
A	0100 0**00**1
C	0100 0**01**1
G	0100 0**11**1
T	0101 0**10**0

Listing 3.5 shows the code employing this new idea. It is both simple and SIMD-friendly. Furthermore, it is almost as fast as the method from BWA, but more convenient.

To complement a sequence of a million nucleotides the switch-based method took 6057 µs (Figure 3.8). The table method is much faster at just 534 µs. This difference is due to every fourth instruction being a branch in the switch-method of which every fifth is mispredicted. The constant rollbacks stall the CPU, which reduces the executed number of instructions per cycle to 0.45. For comparison, the table based method manages a great 4.57 instructions per cycle.

The new method described here requires only 75 µs to reverse-complement a million nucleotides. Even though it only executes 1.06 instructions per cycle, each instruction handles multiple bytes. The numbers for the BWA approach are very similar. So both are about ninety times faster than the switch-based method at the expense of generality.

3.5 Comparing with the Reverse Complement

In Section 3.3 I explained how special CPU instructions can be leveraged to quickly count the number of substitutions between two sequences. However, that technique requires both sequences to be in the same orientation, either both forward or both backward. If one is forward and the other backward, more care needs to be taken. For instance, consider the homologous segments 2b and 1a from Figure 3.6. To compare those two, one strategy would be to first compute the reverse complement of either sequence and then get the substitution rate using the procedures from the previous sections. Unfortunately, this requires temporary memory for the reverse complement. While that could be avoided using a static fixed-size buffer, a fast method exists that compares a sequence directly to a complementary one.

Table 3.1 shows the bit representation of nucleotides. Note how for complementary nucleotides the second bit is equal, but the third bit has to differ. Thus, XORing the two nucleotides and selecting only bits two and three, enables checking for the complement with just one comparison.

```
int is_complement(char x, char y)
{
  int value = x ^ y;
  return (value & 6) == 4;
}
```

Equipped with this branchless way of testing for the complement one can quickly count the number of substitution for two sequences on opposite strands (Listing B.3). The two sequences are again processed with SIMD instructions: A chunk from the beginning of one sequence and one from the end of the other are loaded. The second one is then reversed using a special shuffling instruction. Next, the idea from is_complement is applied: The two chunks are XORed, bit-masked, and compared. Analogously to Section 3.3, I leverage a popcount to quickly count substitutions. Any remaining bytes are processed via simple scalar code.

Comparing across different strands should ideally be as fast as comparing on the same strand. Thus, as a target I here use the code from previous sections (Listing B.2). As a baseline, I use a composition of the fast reverse complement and the fast counting method (Figure 3.9). Again, the simulated sequences are one megabase long and every one hundredth position is mutated.

As in the previous sections, simply comparing two sequences with a million nucleotides takes 71 µs; reversing and then comparing 128 µs. Directly comparing the forward with

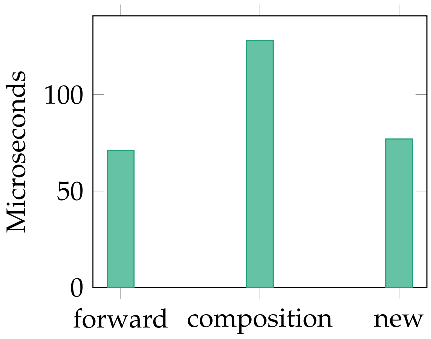


Figure 3.9: Run times for Comparing a Million Nucleotides on Opposite Strands. Each number is the average of a few thousand runs measured using the Google Benchmark library. The forward comparison was explained in Section 3.3. The second method is a reverse complement followed by forward comparison. The new method combines the two steps.

the reverse strand as explained takes 77 µs per run. Thus, it is at par with the direct comparison.

With the new functions, two sequences from the same strand can be compared at 13.1 GB/s, from different strands with 12.9 GB/s. At this speed, two whole human genomes can be compared in less than half a second. For comparison, the maximum memory bandwidth of the used CPU is 85.3 GB/s. It thus might be of interest to find out if phylonium is bound by that limit when running on multiple cores.

A Library of Bioinformatics Routines

In the previous sections I covered three routines central to bioinformatics: mismatch counting, the reverse complement, and counting mismatches across strands. Further, I outlined three implementations with superior performance compared to common approaches. As these could also benefit third-party software apart from phylonium, I created a library containing described routines and more (libdna in Chapter A).

Usually, when a program is compiled, it is optimized for the system it was compiled on. With libdna multiple versions of the same function are provided. That way the optimal implementation can be chosen at run time depending on the instruction set provided by the CPU. The same method is used in phylonium.

3.6 File Parsing

The input to andi and phylonium are assembled DNA sequences. These come in the text-based Fasta format (Pearson 1990).

```
>AE005174 Escherichia coli O157:H7 EDL933, complete genome
AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAAGAGTCTCTGACAGCAGC
TTCTGAACTGGTTACCTGCCGTGAGTAAATTAAAATTTTATTGACTTAGGTCACTAAATACTTTAACCAA
TATAGGCATAGCGCACAGACAGATAAAAATTACAGAGTACACAACATCCATGAAACGCATTAGCACCACC
```

A Fasta file contains multiple sequences each starting with a header line. The header is denoted by a '>' (greater than) followed by the sequence name and an optional comment. All the following lines, not starting with a '>', are the sequence. Back, when sequences were short, it became custom to insert a line break after 70 characters to improve legibility. However, the complete genome of *Escherichia coli* O157:H7 EDL933 is 5.5 million nucleotides long, interspersed by almost 79 000 line breaks. For proper processing in phylonium, sequences need to be concatenated with all line breaks removed.

While parsing a Fasta file is easy, doing so efficiently is not. Many bioinformatics programs use kseq.h, a parser that also handles sequence read data (Li 2012). Unfortunately, it does not validate the input properly. For instance, every non-empty Fasta file should start with a '>'. However, kseq.h just reads the file until it finds the first '>', thereby accepting malformed files and producing garbage. For phylonium I wanted to provide a better user experience. Thus, I wrote pfasta, a new parser that specifically checks this condition and produces an appropriate error message if necessary. Extensive error checking implies a small run time cost, but at the same time improves the security. The latest version of kseq.h can parse data at 2.16 GB/s. My own parser runs at 2.05 GB/s or about 1.7 cycles per input byte. This small slowdown is due to the additional checks required for error handling. I validated my parser using a static analyzer, as well as fuzzing in conjunction with the address sanitizer (Stallman and DeveloperCommunity 2009).

In this chapter, I introduced anchor distances in general and my new implementation, phylonium, in particular. I gave an overview of the used algorithms and data structures. I explained how we estimate the evolutionary distance and mentioned some techniques used to make phylonium fast. In the following chapter, I explore phylonium with respect to accuracy and computational performance.

4 Applying Phylonium

> The second thing is this 'gut feel' that true things are simple when you understand them really (in my view) excludes a massive amount of biology where it's basically 'all about the details'. There are some big sweeping truths in biology but . . . so much baroque insanity.
>
> Ewan Birney

> The easiest way to be at the top of your field is to choose a very small field.
>
> Simone Giertz

In Chapter 3 I introduced a new tool for alignment-free distance estimation, phylonium. In the following I evaluate it under a variety of different conditions with respect to accuracy and computational performance. For that I use a number of simulated and real data sets similar to those used in the AFproject, a recent thorough assessment of alignment-free phylogeny reconstruction methods (Zielezinski et al. 2019). The AFproject covered 74 tools, of which I include in my analysis the five most accurate programs already covered in Section 1.4: cophylog was one of the first *k*-mer based tools to give accurate distance estimates; fswm, an acronym for *filtered spaced word matches*, is very accurate up to high substitution rates (but slower than cophylog); mash is very fast by hashing sequences to a fixed size sketch; andi and phylonium have been covered extensively in Chapters 2 and 3, respectively. Additionally, I use the MSA program mugsy for computing alignment-based reference phylogenies.

4.1 Accuracy

One essential test for alignment-free tools is the estimation of distances on simulated data: For two sequences of the same length, separated by a specific number of substitutions, a good tool should accurately recover the number of substitutions. This task is simple in the sense that the sequences contain no gaps and are free of structural rearrangements such as duplications or transversions.

To verify that all tools pass this basic test, I simulated a 200 kb sequence with uniform nucleotide composition. A second sequence was derived from the first one by introducing a fixed number of substitutions. A hundred of these pairs were compared per tool for one data point in Figure 4.1. It is apparent that all programs accurately estimate the evolutionary distance. Mash shows some variation at both ends of the spectrum. For distances above 0.3 substitutions per nucleotide, estimation becomes more difficult for

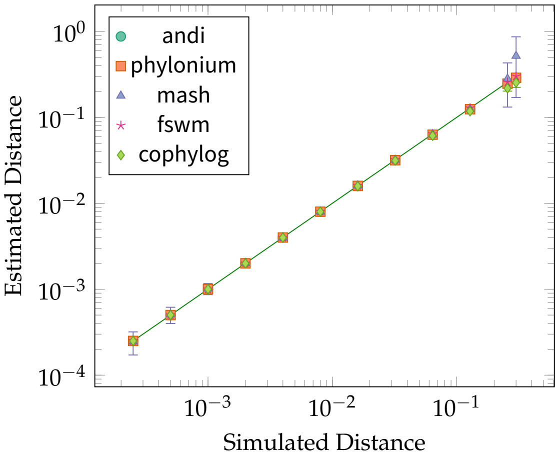


Figure 4.1: Estimation of substitution rates on *perfect* data. Shown are the mean and standard deviation of 100 runs.

all programs and their results start to fluctuate. This is understandable, as every third or fourth residue is mutated and their homology begins to fade.

4.2 Limits

With an increasing substitution rate, it becomes more difficult to accurately estimate the evolutionary distance for unaligned genomes. Figure 4.2 shows the accuracy of mash and phylonium at the high end of substitution rates. The estimates of phylonium stay very close to the ideal, represented by the dotted line, up to a substitution rate of 0.5. Mash, on the other hand, tends to underestimate distances starting at 0.3. Beyond these two points, 0.5 and 0.3, the estimates of both tools start to vary significantly. Furthermore, a comparison can fail if mash detects no overlap between the sketches, or if phylonium finds no anchor pairs. The dashed lines in Figure 4.2 represent the fraction of successful comparisons. At small substitution rates all, comparisons are successful, but from 0.25 on for mash and from 0.55 for phylonium the numbers drop rapidly.

For substitution rates above 0.3, phylonium underestimates the evolutionary distance, though not as strongly as mash. Furthermore, the fraction of the sequence contributing to the distance estimates, called the coverage, becomes less and less (Figure 4.3). Already at a substitution rate of 0.3 the coverage is almost down to half the sequence. At even higher rates, the estimated distance is based on only a tiny fragment of the sequences. To alert the user when this happens, phylonium prints a warning once the coverage drops below 20 %.

The minimum length, t, for an anchor is chosen such that at most 2.5 % of random matches pass this threshold. As the threshold is an integer, the actual number of random matches passing is noticeably lower. For instance, for two 100 kb sequences only 1.2 % of random matches are longer than the threshold, $t = 12$, regardless of the substitution

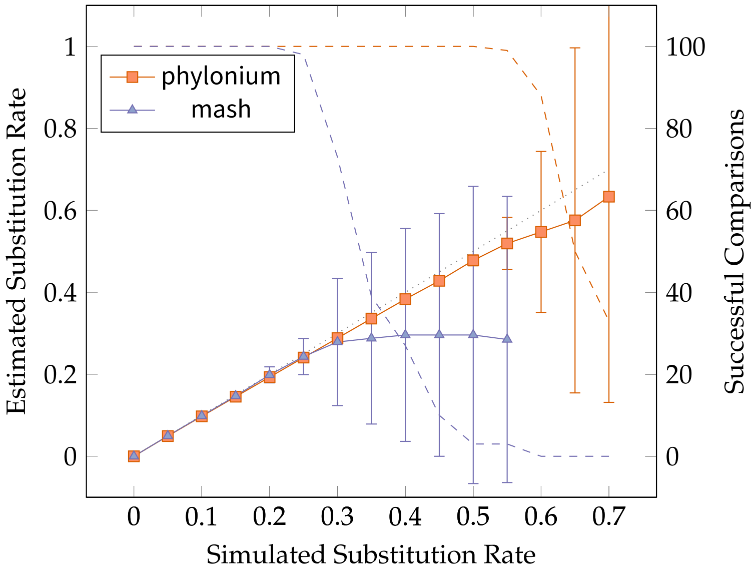


Figure 4.2: The distances estimated from highly divergent sequences. Simulated were a hundred 100 kb sequence pairs. The dashed lines show how many successful comparisons contribute to each data point.

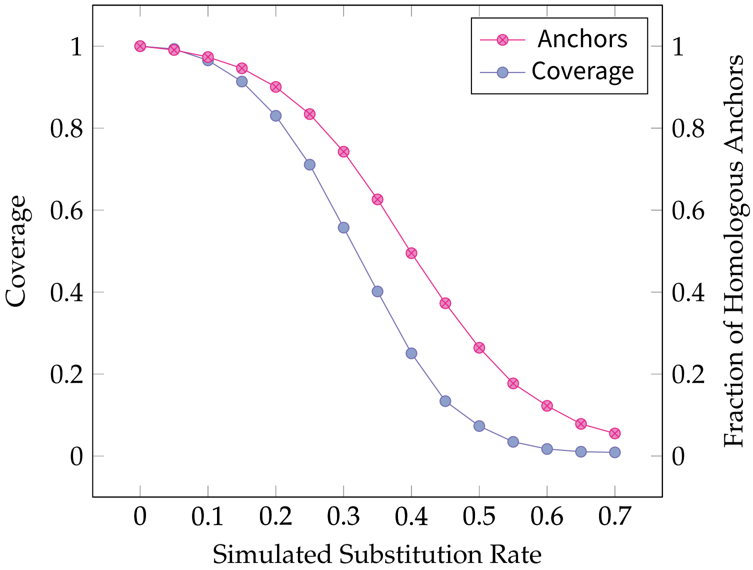


Figure 4.3: With increasing divergence, the estimates from phylonium are based on ever smaller regions. The main reason for this is that matches become shorter and harder to distinguish from non-homologous matches.

rate. So the number of non-homologous background anchors is the same for all pairwise comparisons. However, as the substitution rate increases, the proper, homologous anchors become shorter (Section 2.1). Thus, as the number of matches passing the threshold decreases, the proportion of homologous anchors also decreases (Figure 4.3). So with an increasing substitution rate the probability of finding an anchor pair decreases until estimation becomes impossible.

On simulated sequences, substitutions are typically distributed uniformly across the length of the sequences for simplicity. However, the same is not true for real sequences, as genomes are subject to highly variable functional constraints. Introns mutate faster than exons, and intergenic regions faster than regulatory regions. Some regions, like the genes encoding ribosomal RNA, are so conserved, they can be aligned across the whole tree of life (Woese and Fox 1977). All this leads to local changes in the substitution rate (Section 1.7). This is a significant challenge in alignment-free phylogeny reconstruction. For instance, Röhling et al. (2020) found that distances computed by mash started to diverge when a sequence was surrounded by unrelated random sequence. Phylonium does not exhibit this behavior and instead limits its analysis to the homologous regions (Klötzl and Haubold 2019, Fig. S2). This is an effect of the coverage as shown in Figure 4.3: as the divergence in a region increases, it will contribute less to the estimated distance. Thus, for mosaic genomes the more closely related regions are covered better and the overall substitution rate is underestimated. So on real data, as sequences become more divergent, phylonium has the tendency of limiting its analysis to conserved regions. We thus recommend to mainly use phylonium on closely related sequences, for example sequences from a bacterial outbreak.

4.3 Benchmark Data

In the AFproject, Zielezinski et al. (2019) compared 74 methods for alignment-free phylogeny reconstruction. To this end, they used various real and simulated data sets. They ran each tool on every data set, computed a phylogeny, and compared that to a reference tree. For the tree comparison they used the Robinson-Foulds distance (RF), a measure of topological similarity (Robinson and Foulds 1981).

It was a design goal to make phylonium accurately estimate evolutionary distances, so that a resulting tree not only had the correct topology, but also correct branch lengths. RF distances are based solely on tree topology. However, phylonium produces a distance matrix. The relationship between tree and distance matrix is not straightforward. To illustrate this, Figure 4.4 depicts three rooted trees of five taxa, $T_1, \ldots, T_5$. Assume tree **A** reflects their true phylogeny. For tree **B** the taxa T_2 and T_3 were swapped. This leads to the two clades $\{T_1, T_2\}$ and $\{T_3, T_4\}$ that appear in **A** but not **B**. In the other direction it is $\{T_1, T_3\}$ and $\{T_2, T_4\}$. These four clades exclusive to either tree correspond to a rooted RF-distance of 4 between **A** and **B**.

Between **A** and **C** the taxa T_2 and T_5 are swapped. This leads to two clades unique to either tree: $\{T_1, T_2\}$ and $\{T_1, T_5\}$. However, as the outgroup changed, that again corresponds to a rooted RF-distance of 4. So **C**, just like **B**, has an RF-distance to **A** of 4, but **C** represents a much more drastic departure from **A** than **B**.

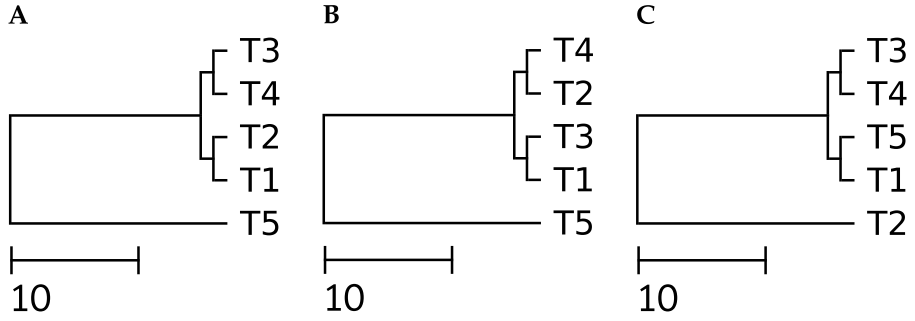


Figure 4.4: Three trees. Analog to Klötzl and Haubold (2019).

A

	T_1	T_2	T_3	T_4	T_5
T_1	0	2	4	4	34
T_2	2	0	4	4	34
T_3	4	4	0	2	34
T_4	4	4	2	0	34
T_5	34	34	34	34	0

B

	T_1	T_2	T_3	T_4	T_5
T_1	0	4	2	4	34
T_2	4	0	4	2	34
T_3	2	4	0	4	34
T_4	4	2	4	0	34
T_5	34	34	34	34	0

C

	T_1	T_2	T_3	T_4	T_5
T_1	0	34	4	4	2
T_2	34	0	34	34	34
T_3	4	34	0	2	4
T_4	4	34	2	0	4
T_5	2	34	4	4	0

Figure 4.5: Three matrices.

Figure 4.5 shows the distance matrices from which the trees in Figure 4.4 were computed. In Klötzl and Haubold (2019) we used a variant of the Hausdorff-metric to compare two distance matrices D and d:

$$\Delta(D,d) = \max\{|D_{ij} - d_{ij}| : 1 \leq i,j \leq n\} \, .$$

For the running example $\Delta(\mathbf{A},\mathbf{B}) = 2$ and $\Delta(\mathbf{A},\mathbf{C}) = 32$. The new measure thus captures the difference between the matrices, and in extension, the trees, where RF sees none. We consider the Δ-score a complementary measure to the topological RF-distances.

The Δ-score can also be used to detect differences that are lost in the tree representation. Consider the two distance matrices in Figure 4.6. The entries differ by up to 0.125, the Δ-score. However, using neighbor-joining, they both produce the same phylogeny. Not only is the topology identical (RF = 0), but so are the branch lengths. So there can be a significant difference between two matrices that is lost after tree reconstruction.

	A	B	C	D
A	0	3.1	4	4
B	3.1	0	3	3.5
C	4	3	0	3
D	4	3.5	3	0

	A	B	C	D
A	0	3.1	3.875	4.125
B	3.1	0	3.125	3.375
C	3.875	3.125	0	3
D	4.125	3.375	3	0

Figure 4.6: Two different matrices implying the same phylogeny.

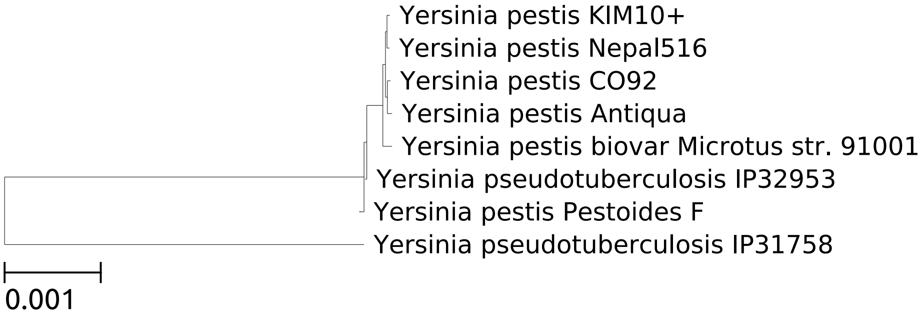

Yersinia pestis KIM10+
Yersinia pestis Nepal516
Yersinia pestis CO92
Yersinia pestis Antiqua
Yersinia pestis biovar Microtus str. 91001
Yersinia pseudotuberculosis IP32953
Yersinia pestis Pestoides F
Yersinia pseudotuberculosis IP31758
0.001

Figure 4.7: An alignment-based phylogeny of eight *Yersinia* species.

Yersinia

One of the data sets in the AFproject contains eight whole *Yersinia* genomes, a bacterial genus including the cause of the plague. Their length varies from 4.5 to 4.7 million nucleotides, amounting to a total size of 36 Mb. Aligning these with mugsy takes about 6 minutes. From that I computed a reference phylogeny using Jukes-Cantor distance estimation and neighbor-joining (Figure 4.7).

Phylogeny reconstruction on this particular data set is very sensitive to small changes in the estimated distances. Except for the outgroup, all other sequences are very closely related. Thus, a small estimation error can lead to a different topology and hence to an increase in the RF-distance. Table 4.1 shows the accuracy of results produced by different tools when compared to the reference.

On this data set, mash produces a surprisingly bad tree, topology wise (RF $=$ 12). Phylonium is almost as accurate on the *Yersinia* data as andi but much better than mash (Table 4.1). While fswm and cophylog have a small RF-value of 2, both their Δ-values are even bigger than that of mash.

In the following I used seq-gen to simulate sequences along the true *Yersinia* phylogeny with varying degrees of divergence (Rambaut and Grass 1997). Initially, the scale is the same as in Figure 4.7. On these 200 kb sequences mash is much better and gives an RF-distance of only 2 (Figure 4.8A). Its estimates vary just a bit up to the point of 0.03, beyond which some comparisons fail and no correct distance is estimated. Phylonium, like andi, stays very accurate until 0.04.

Table 4.1: Reconstruction accuracy on the *Yersinia* tree.

Tool	RF	Δ-score
andi	6	0.0011913
phylonium	8	0.0014488
mash	12	0.0023069
fswm	2	0.0023171
cophylog	2	0.0033996

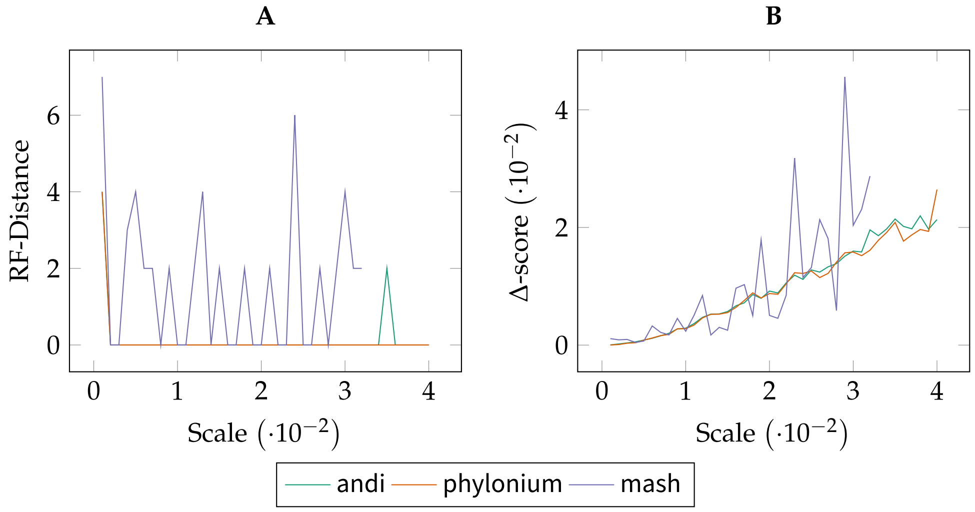


Figure 4.8: Reconstruction accuracy on simulated *Yersinia* sequences. The leftmost point represents the original scale of the tree. **A** The RF-value of the estimated tree to the simulated tree. **B** The Δ-value between the simulated sequences and the distance matrix estimated by the different tools. As noted in Section 4.6, cophylog and fswm are rather slow, so they were excluded from this analysis.

Escherichia coli and *Shigella*

Another data set commonly used in the benchmarking of alignment-free distance estimation consists of 29 whole *Escherichia coli* and *Shigella* genomes (Yi and Jin 2013; Haubold et al. 2015). These are more distantly related than the *Yersinia* with an average substitution rate of 0.02 compared to 0.002 for the *Yersinia*. Figure 4.9 shows their phylogeny computed using mugsy.

Again, I ran the same set of tools on this data set (Table 4.2). Phylonium, with a Δ-score very close to andi, produces a slightly different topology. Mash is worse for both the RF and the Δ-score, but can be improved by using ten times bigger sketches: RF = 6 and Δ = 0.007394 for mash10k which comes with a 20 % increase in run time. Fswm and cophylog produce mediocre results despite taking more than a minute to compare the sequences. As the other tools are much faster, these two are henceforth excluded from the analysis (see the run times in Section 4.6).

As with the previous data set, I simulated sequences along the original tree. However, instead of stretching the tree, it was compressed so that the distances between sequences are reduced. At the original scale all tested tools produce a small RF and Δ-score (Figure 4.10). As the branch lengths shrink, the estimated distances start to vary. For all tools, as the number of mutations decreased, the reconstructed trees became less accurate, and the RF-distance grows (Figure 4.10A). Phylonium is on a par with andi. Mash with default parameters is less accurate, but improves by increasing the sketch size. This, however, does not hold for the Δ-score where both mash variants are less accurate than andi and phylonium (Figure 4.10B).

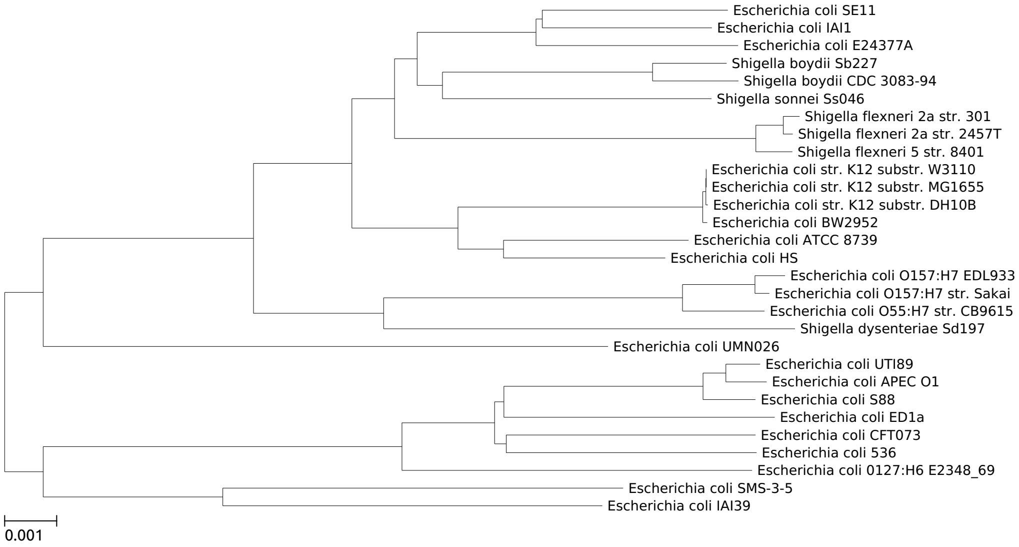

Escherichia coli SE11
Escherichia coli IAI1
Escherichia coli E24377A
Shigella boydii Sb227
Shigella boydii CDC 3083-94
Shigella sonnei Ss046
Shigella flexneri 2a str. 301
Shigella flexneri 2a str. 2457T
Shigella flexneri 5 str. 8401
Escherichia coli str. K12 substr. W3110
Escherichia coli str. K12 substr. MG1655
Escherichia coli str. K12 substr. DH10B
Escherichia coli BW2952
Escherichia coli ATCC 8739
Escherichia coli HS
Escherichia coli O157:H7 EDL933
Escherichia coli O157:H7 str. Sakai
Escherichia coli O55:H7 str. CB9615
Shigella dysenteriae Sd197
Escherichia coli UMN026
Escherichia coli UTI89
Escherichia coli APEC O1
Escherichia coli S88
Escherichia coli ED1a
Escherichia coli CFT073
Escherichia coli 536
Escherichia coli O127:H6 E2348_69
Escherichia coli SMS-3-5
Escherichia coli IAI39
0.001

Figure 4.9: A phylogeny of 29 *E. coli/Shigella* genomes computed using an MSA.

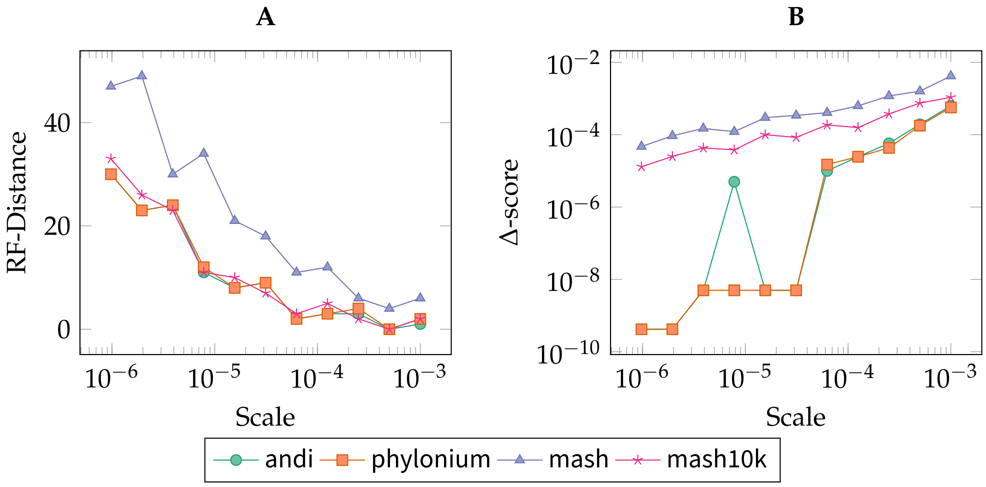


Figure 4.10: Reconstruction accuracy on simulated *E. coli/Shigella* sequences. The rightmost point represents the original scale of the tree. **A** The RF-value of the estimated tree to the simulated tree. **B** The Δ-value between the simulated sequences and the distance matrix estimated by the different tools.

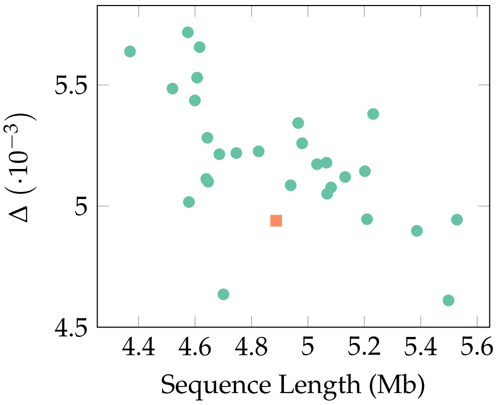


Figure 4.11: The distance estimation accuracy for each possible reference as a function of reference length.

4.4 Choice of Reference Sequence

In Section 3.1 I explained that phylonium picks a single reference sequence from the data set. For this reference an index is built and all other sequences are aligned against it. The choice of reference thus influences the accuracy of the estimation.

For the data set of 29 *E. coli/Shigella*, I ran phylonium once with each sequence set as reference. Figure 4.11 summarizes the results of this experiment. The Δ values fluctuate between 0.0046 and 0.0057. Thus, the best reference is at par with andi, while the worst reference is still better than mash ($\Delta = 0.008$, Table 4.2).

While the plot suggests that there is a negative correlation between sequence length and reconstruction accuracy, this is a property of this particular data set. (For instance on the *Yersinia* the correlation is slightly positive.) In the absence of a user-supplied reference, we choose a sequence of median length as the reference. This is more robust than picking one of the extremes as they might be misassemblies or outgroups (Section 4.6). Instead, our analysis is based on a common representative of the sample. The automatically determined reference is highlighted in Figure 4.11 and is indeed among the better choices.

Table 4.2: Reconstruction accuracy on the Eco29 tree.

Tool	RF	Δ-score
andi	2	0.004674
phylonium	6	0.004940
mash	8	0.008284
mash10k	6	0.007394
fswm	6	0.006398
cophylog	4	0.012042

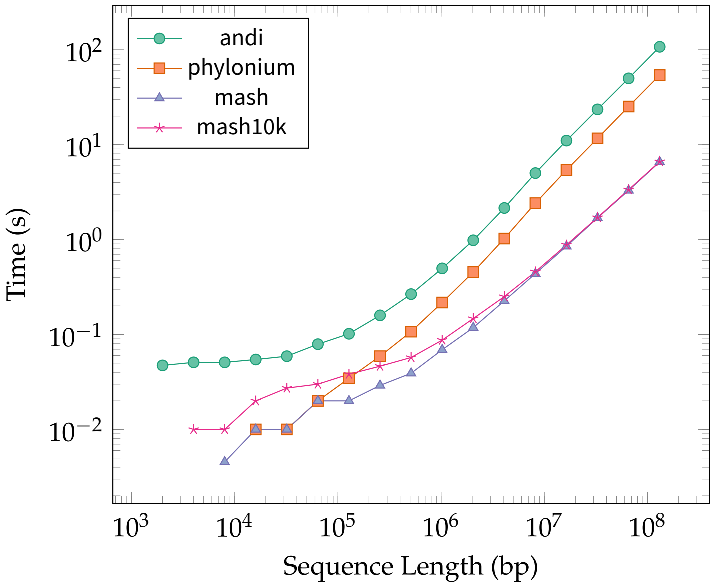


Figure 4.12: Run times for sequences of varying length.

4.5 Complete Deletion

Internally, phylonium aligns all sequences against the reference. However, this anchor alignment is approximate in that it does not contain gaps at nucleotide resolution but on a larger scale. Due to variation at sequence level and gene content, the aligned segments do not begin and end at the same sites for all queries. Specifically, some parts of the reference may only be covered by a subset of sequences. In order to restrict distance estimation to segments common to all sequences, phylonium supports complete deletion (Section 3.2).

Given the data set of 29 *E. coli/Shigella*, I ran phylonium and a reference MSA both with complete deletion. Compared to previously, phylonium only has an RF-distance of 1 to the new reference phylogeny (6 to the old reference). The Δ-score however increased to 0.0061, thereby exceeding the range of values from the previous section (0.0046 – 0.0057).

4.6 Resource Consumption

Alignment-free distance estimation should ideally be as accurate as alignment-based methods, only faster. To explore to what extent this is true for phylonium, I first explore the time and memory consumption on simulated data, particularly the length of sequences. To this end I simulated two sequences of varying lengths with a fixed 0.01 substitutions per site. Figure 4.12 shows the run time for 2 kb sequences up to 1.6 Mb sequences. The tools exhibit diverse behavior for short sequences, but all grow linearly for long sequences. As andi does calculated distances in both directions, it is two times slower than phylonium.

Table 4.3: Run time and memory consumption on the Eco29 data set.

Tool	Run time (s)	Memory (MB)	CPU utilization
mugsy	6647.24	2877	101 %
andi	15.17	1374	668 %
cophylog	67.14	1876	550 %
fswm	196.24	2576	790 %
mash	0.70	86	722 %
mash10k	0.85	104	708 %
phylonium	2.21	357	250 %

In previous sections we saw that accurately estimating distances on real data is much harder than on simulated data due to the presence of duplications, indels etc. One might expect that the additional complications of real data also slow down the analysis. For instance, the string algorithms employed in phylonium are sensitive to long repeats. Older suffix array construction methods have been particularly sensitive to repeats (Manzini and Ferragina 2004). To test this hypothesis, I a) simulated sequences, and b) took a prefix of an *E. coli* genome and mutated it with the same substitution rate. Comparing the run times of tools on these two kinds of data gave no significant difference even at varying sequence lengths (p-value $= 0.9801$, two-sample Kolmogorov-Smirnov test).

One other variable at play is the number of sequences. As already noted at the beginning of this thesis, studies can contain thousands of bacterial isolates (Tang et al. 2017). Since the output of all alignment-free tools is a distance matrix, the computational demand grows quadratically in the size of the data set. It thus is crucial to reduce the run time for each individual comparison to a minimum. In the following I explore the efficiency of different tools on real data sets of varying size.

Eco29

At first I measured the resource consumption on the set of 29 *E. coli* and *Shigella* already explored in Section 4.3. All tests were executed under real-world conditions: Tools could use eight of the twelve virtual cores of an Intel Xeon W-2133 CPU clocked at 3.6 GHz. The data set itself is 145 Mb in size but was stored in RAM to avoid disk latency.

As a baseline, computing an MSA for this data set using mugsy takes almost 2 hours and 2.9 GB RAM (Table 4.3). Andi is much faster, taking 15 s to compare all sequences, requiring 1.4 GB of RAM. Mash is again much faster with just 0.7 s for the comparison. Since mash converts sequences to sketches as they are read, it never has to hold the whole data set in memory, thus using only 86 MB of RAM. Phylonium is in between with 2.2 s and 360 MB of RAM. However, as this is such a small data set and the computation of the ESA for the reference cannot be parallelized, on average it utilizes only 2.5 CPUs of the eight available.

Table 4.4: Run time and memory consumption on 297 *E. coli*.

Tool	Run time (s)	Memory (MB)	CPU utilization
andi	771.52	2841	755 %
mash	6.22	114	790 %
mash10k	7.81	150	789 %
phylonium	28.63	1924	355 %

Table 4.5: Run time and memory consumption on 2681 *E. coli*.

Tool	Run time (s)	Memory (MB)	CPU utilization
andi	59 256.04	15 545	722 %
mash	68.36	153	801 %
mash10k	145.95	442	799 %
phylonium	794.52	14 358	462 %

297 Ensembl *E. coli*

Next, we analyzed larger data sets that are unfeasible to align. As in Klötzl and Haubold (2019), I downloaded all sequences labeled *Escherichia coli* from the Ensembl database release 45 (Yates et al. 2016). Of the whole data set I sampled 297 sequences equally across the length spectrum, thereby getting a uniformly distributed sample including the outliers. These amount to 1.5 Gb of sequence data, roughly ten times bigger than the previous data set. As the number of sequences grows tenfold, the number of comparisons grows a hundredfold. This superlinear increase is most apparent in andi (Table 4.4). On this data it takes 771 s, which is a 51-fold increase to the previous data set. The mash run time is dominated by the work spent hashing each individual sequence. Thus, with 6.2 s it is about nine times slower than previously. Coming from andi, with phylonium I tried to reduce the quadratic component. Its run time increased twelvefold to 27 s. Thus, it is slower than mash but has much better scalability than andi.

While the run times increase dramatically, memory does not. Andi requires 2.8 GB, mash 114 MB and phylonium 1.7 GB. Essentially, mash needs only a bit more memory than previously, phylonium needs space for the bigger data set, whereas andi's data usage doubles. However, all tools stay well below the maximum of 31 GB available on the test machine.

2681 Ensembl E. coli

As a final challenge I ran the tools on all 2681 *E. coli* sequences from Ensembl. The number of sequences grows ninefold compared to the subsample above, as does the data size, which now is 14 GB. Mash took only 68 s and 442 MB for the comparison (Table 4.5). It thus requires much less resources than phylonium with 795 s and 14 GB. While this seems slow, phylonium is still very fast on such a big data set. Particularly it is much faster than andi (59 256 s $\approx$ 16 hours).

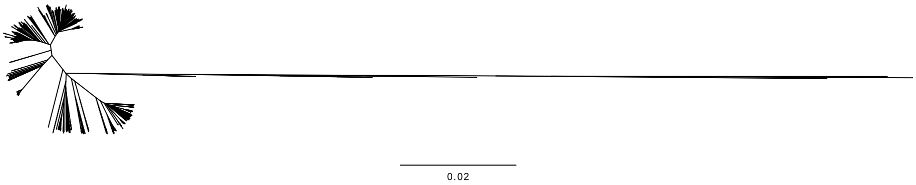


Figure 4.13: A phylogeny of all 2681 sequences in Ensembl labeled *E. coli*.

From these numbers it is apparent that andi scales strictly quadratically with the number of input sequences. With mash each pairwise comparison is very fast and thus it scales roughly linearly. Phylonium also grows quadratically, but with much better coefficients than andi.

Figure 4.13 shows the phylogeny of all *E. coli* as computed by phylonium. Most sequences cluster nicely into groups but some taxa stick out as they are far more distantly related to the rest. As it turns out, these are not *E. coli* at all but simply mislabeled in the database: Blasting parts of them against NCBI they are identified as *Enterobacter hormaechei*, *Klebsiella pneumoniae* and *Citrobacter freundii* (Boratyn et al. 2013).

In this chapter, I evaluated phylonium with respect to phylogeny reconstruction accuracy as well as run time performance. Accuracy was measured using the common RF distance as well as the new Δ-value. In most settings, phylonium proved to be as accurate as andi while almost as fast as mash. This makes it useful for even the largest data sets.

5 Discussion and Future Directions

> So for us there is no longer a fundamental mystery about life. It is all the process of extraordinary eruptions of information. And it is information that gives us this fantastically rich, complex world in which we live. But at the same time that we have discovered that, we are destroying it at a rate that has no precedent in history. Unless, you go back to the point that we were hit by an asteroid.
>
> Douglas Adams

Phylogenies are so widely used, it might seem self-evident that the relationships between organisms are best depicted by a tree. Perhaps, the most famous evolutionary tree was drawn by Darwin in a sketchbook, captioned "I think" (Figure 5.1). Later, Haeckel also used the tree metaphor to depict his Pedigree of Man (Figure 1.1) and trees are still the central metaphor in evolutionary biology (Ragan 2009). Trees are also widely used in genomic epidemiology to visualize evolutionary relationships (Tang et al. 2017). In population genetics, random trees, or coalescents, are used to infer population statistics (Wakeley 2009). Trees also serve as a tool in computational biology. For instance, Kelleher et al. (2019) use trees to succinctly represent data on genetic variation. Their tree representation not only leads to compression, it also simplifies subsequent calculations.

The high level of lateral gene transfer in microbials breaks the tree metaphor (Dagan and Martin 2009). No single phylogeny can capture the changing evolutionary relationships along genomes caused by horizontal gene transfer, and they would be better represented by a network. However, trees remain popular as they are by and large correct and easy to interpret. They can even be annotated with support values to reflect uncertainty about their topology.

These examples illustrate that trees remain a central organizing principle in biology. However, computing phylogenies has become difficult for the large genomic data sets produced by high-throughput sequencing. Which brings me to the topic of this thesis, fast estimation of distances for phylogeny reconstruction.

5.1 Accuracy and Limits

In this thesis, I introduced phylonium, a fast and accurate program for phylogeny reconstruction from genomic data. It is classified as an *alignment-free* method because it does not require an MSA as one of its inputs. Instead, phylonium maps parts of each sequence onto a common reference. These homologous segments can then be compared between sequences to estimate the substitution rate (Chapter 3).

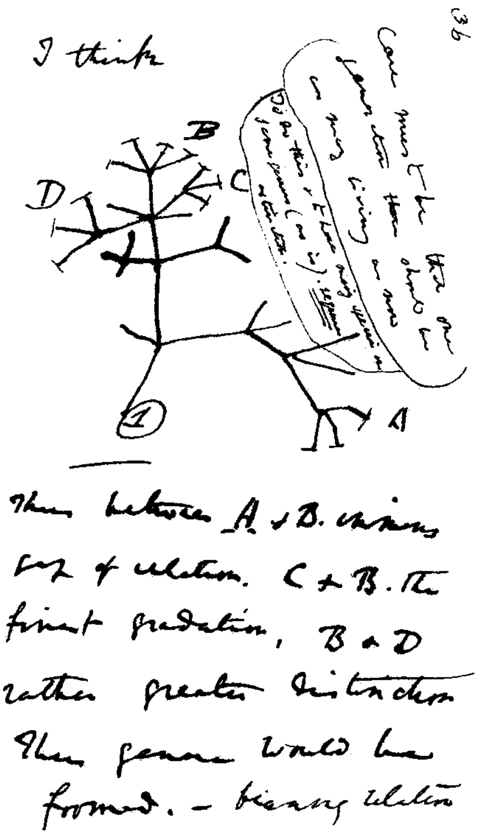

36
I think
case must be that one generation should have as many living as now
To do this & to have many species in same genus (as is). requires extinction.
B
C
D
1
A
Thus between A & B. immense gap of relation. C & B. the finest gradation, B & D rather greater distinction Thus genera would be formed. — bearing relation

Figure 5.1: The sketch of an evolutionary tree by Darwin (1837). The text states: I think case must be that one generation should have as many living as now. To do this and to have as many species in same genus (as is) requires extinction. Thus, between A + B the immense gap of relation. C + B the finest gradation. B + D rather greater distinction. Thus, genera would be formed. Bearing relation [next page] to ancient types with several extinct forms.

In Chapter 4, I explored the accuracy of phylonium in many different applications. First, I checked whether it could correctly estimate the substitution rate on simulated sequences. Phylonium is accurate from virtually no substitutions up to 0.5 substitutions per nucleotide. This test is simple in that the simulated data is free of noise. There is no recombination, no insertions or deletions, no rearrangements of regions. Correctly passing this basic test is necessary, though not sufficient, for accurately estimating substitution rates from real data. A number of tools referenced in the Introduction were excluded from my comparison simply because they were inaccurate (e. g. d^2). Others were excluded because they were too slow, for instance missmax (Pizzi 2016). Being accurate up to 0.5 substitutions per base is good enough to compare sequences in the family of great apes (Lynch 2007, p. 85). Note, however, that phylonium was designed for closely related prokaryotes where an average nucleotide identity of 95 % has been proposed as the border between species (M. Kim et al. 2014). Thus, in outbreak data, which consists of many clones from one strain, the substitution rate should be well below 0.05.

When only two simulated sequences are compared, it is easy to check whether the estimated distance is the same as the simulated distance. If more sequences are analyzed, it has been common practice to infer a tree from the estimated distances and to compare it against a reference tree, which is often based on an MSA. Of the many methods to quantify the similarity of two trees, the Robinson-Foulds distance is the most popular. For our work on phylonium we additionally used the largest difference between two distance matrices, the Δ-score (Klötzl and Haubold 2019). It is useful for two reasons. First, it is simple. The resulting values are on the same scale as the estimated distances and thus, can be easily interpreted. Second, directly comparing the distance matrices avoids any noise introduced by the tree reconstruction method (Section 4.3). One major disadvantage of the Δ-score is that the current definition only includes a single difference, the maximum, into the result. The Mantel test is a classic way to assess the association between two distance matrices (Mantel 1967). However, it only quantifies a correlation, which in our case is always given. Further research is needed to find a way of directly comparing distance matrices that incorporates all values and calculates useful scores.

To compare two trees, I used the Robinson-Foulds distance (Robinson and Foulds 1981). As explained in Section 4.3, it is a measure of topological distance. The values taken by this measure, however, are difficult to interpret, because a change at a short branch contributes just as much as a change at a long branch. An alternative, the *branch score*, incorporates branch lengths (Kuhner and Felsenstein 1994). However, despite using floating-point values, it is discontinuous in that for two trees T_1 and T_2 with a distance of, say, 0.1 there might be no intermediate tree T_2' that has a smaller distance to T_1 (Kupczok 2010). There is a more recent topological distance, which can be interpreted as a lower bound on the number of necessary reticulation events, such as lateral gene transfer, to transform one tree into the other (Whidden et al. 2013). However, it is computationally expensive and in order to keep my results comparable (particularly to the AFproject), I settled for the RF-score and Δ-value to compare phylogenies and distance matrices, respectively.

Equipped with these two distance measures, I ran a number of experiments on phylonium and other tools. Overall, phylonium is very precise. For the *Yersinia* and *E. coli/ Shigella* data sets its RF-value is between those of andi and mash. The Δ-value of phylonium is usually much smaller than that of mash. This behavior is most apparent on the simulated data sets. Consider Figure 4.10, where increasing the sketch size of mash helps to reconstruct the correct tree topology. However, the Δ-values of phylonium are a few order of magnitudes better, particularly for very closely related sequences (Section 4.1).

Unfortunately, there is a limit to how far sequences can be separated so that phylonium can still estimate a distance. In Section 4.2 starting from 0.3 substitutions per site, phylonium slightly underestimates distances. This is due to the coverage dropping to about 60 %. At substitution rates above 0.55 the number of anchors has dropped so low, that often no estimation is possible. Hence phylonium is biased towards anchors in regions of low substitution rates. In real sequences, the substitution rate varies along the genome. Thus, phylonium is biased towards the conserved parts. Röhling et al. (2020) drove this to the extreme by adding unrelated regions to two otherwise homologous sequences. Where the mash distance started diverging with the fraction of unrelated sequence, phylonium limits its analysis to only the homologous part, and thus remains accurate. A warning is printed if for a pair of sequences the homologous part covers less than 20 % of either sequence. This warning can also help to find contaminations, for instance, the mislabeled sequences in the Ensembl database (Section 4.6). The only alignment-free tool to accurately estimate distances beyond 0.5 substitutions per position is fswm as it utilizes inexact matching with many *don't care* positions (Leimeister et al. 2017).

5.2 Advancing Phylonium

Alignment-free phylogeny reconstruction methods aim to be as accurate as methods based on an MSA albeit being much faster. I made the direct comparison between alignment-free approaches and MSA based ones on the set of 29 *E. coli/Shigella* genomes (Section 4.6). Table 4.3 summarizes the resource consumption of the six tools investigated. Mash is the fastest tool and also uses the least memory. Phylonium is only a bit behind in both categories. To achieve this performance I have optimized the performance-critical code in both tools down to the instruction level. Section 3.3 covers the advanced sequence comparison techniques used in phylonium. For mash, I have suggested the optimization of the reverse complement (Section 3.4) as well as contributed changes that reduce the run time by a factor of 3.5 (Chapter A): Without these changes mash would take 2.5 s on this data set, compared to 2.2 s for phylonium.

The biggest data set explored in this thesis consists of 2681 *E. coli* genomes from the Ensembl database. It demonstrates the usefulness of alignment-free methods on huge amounts of data. Assuming that the run time of mugsy scales strictly quadratically, it would take 1.9 years to compute an alignment. Phylonium and mash are much faster with just 795 s and 68 s, respectively. Remember, that my comparisons were done on a desktop-grade computer rather than on a dedicated computation server as they are common in research institutions.

By design, phylonium stores the whole data set in memory, which requires one byte per nucleotide. For larger data sets this is problematic as memory is a hard limit. Either no more memory can be allocated, or the computer starts swapping and thereby significantly reduces throughput. Phylonium stores all sequences in RAM, because they have to be cleaned before their comparison (Figure 3.1A). For instance, to aid readability, files containing genetic data are commonly split into lines of sixty or seventy characters. My Fasta parser removes these as the sequences are read into memory (Section 3.6). If, instead of parsing the files upfront, they are loaded raw when needed (e. g. via direct memory mapping) these newlines prove a serious challenge. For instance, the comparison functions become significantly more complex. Instead, new formats for storing sequences need to be explored. For instance, the recent Nucleotide Archival Format (NAF) is a stricter file format for storing genetic sequences that resolves problems such as the newlines. Further, it compresses sequences by a factor of four (Kryukov et al. 2019). This saves disk space, but sacrifices performance. NAF can be decompressed at about 650 MB/s where my Fasta parser runs at 2 GB/s (Section 3.6). Thus, there is a clear disk-space v. run time trade-off.

The run time of phylonium for small data sets is dominated by the index creation for the reference sequence (Figure 3.1B). Phylonium already uses the fastest available library for suffix array creation, libdivsufsort. Unfortunately, the parallel version of libdivsufsort provides only a minor performance improvement of 28 % even when using four threads (Klötzl 2015, p. 43). A parallel implementation that leverages both multi-threading and SIMD could significantly speed up index creation (Labeit et al. 2016). As the size of the data set grows, looking up matches in the index becomes more important (Figure 3.1C). For instance, for the 297 *E. coli* about 40 % of the run time is spent finding matches. As explained in Section 1.6, the enhanced suffix array, an assortment of big tables, needs to be accessed to find a corresponding match. These memory accesses occur in a seemingly unpredictable order. As the ESA is too big to fit into a CPU cache, a lot of these accesses go to the main memory. With each cache miss, the program stalls execution, waiting for the data. This extra time can be reduced in different ways. One, which I already implemented in phylonium, is to reduce the amount of lookups (Sections 1.6 and 2.1). A potential way to further reduce the number of lookups is to use a suffix tree, instead of a suffix array.

Suffix arrays were introduced as an alternative to suffix trees but with reduced memory requirements (Manber and Myers 1993; Abouelhoda et al. 2004). Initial findings also suggested that suffix arrays were faster. However, computer architecture has changed significantly in recent years. While CPUs have become increasingly faster, memory has been lagging behind and thus, a whole hierarchy of caches has been added. Each node from a suffix tree corresponds to many entries across multiple tables in the ESA. So where a cache miss in a suffix tree will add one memory round-trip latency, depending on the state of the caches, a suffix array may add many such waiting times. Thus, it might be beneficial to develop a new suffix tree implementation that is tailored to low run times at the expense of memory. This is the opposite of succinct data structures such as the FM-index that reduces the size of the index at the cost of searching time (Ferragina and Manzini 2000). As phylonium builds only one index, but makes a lot of searches, it is on the opposite end of the space-time trade-off.

A more experimental way to improve the run time is to reduce the impact of each cache miss. Consider an expression such as `array[i]`. If `i` is unpredictable, the CPU has to stall the execution of the program while the requested data is loaded from memory. CPU manufacturers have addressed this problem by introducing simultaneous multi threading (branded as hyperthreading). It allows a single CPU to execute multiple programs at once so that it continues working even while one or several programs are stalled. There have been proposals to implement this behavior in software, which could also be adapted to suffix arrays (Jonathan et al. 2018). The expression `array[i]` loads from a memory address and blocks until that data is available. Unfortunately, there is no asynchronous equivalent in the sense that it would load the data, but keep the program running and notify it once the load has been completed. There is, however, `prefetch(&array[i])`, upon which the CPU will load data into its cache, but not stall execution. When the program then asks for the data later on, it is available without extra latency. Replacing the lookups into the suffix array with prefetches thus, would allow interleaving the matching of multiple queries at once. For each query issue a prefetch and then, as the data is loaded in the background, process other queries using regular loads. A critical parameter would be the number of interleaved queries. For too few queries the additional programming overhead would negate any performance benefits. For too many queries, data would be flushed from the caches again, before getting processed. Further, this method would require a large refactoring to handle multiple simultaneous queries.

A simpler way of speeding up phylonium would be to reduce the number of compared nucleotides (Figure 3.1F). As explained in Section 3.1, phylonium does not store the start and end points of anchors but only more general homologous segments. Remember, an anchor is an exact match with respect to the reference (Section 1.5). So if two queries have overlapping anchors, the overlap is also a mutual exact match and hence, can be skipped when counting mismatches. This, however, requires additional memory and additional memory management to store the positions of anchors. Further, the code to compare sequences will become more complex as anchors now have to be incorporated. To investigate whether such an optimization would be promising, I measured how many nucleotides in a homologous segment come from the anchors within. Figure 5.2 shows the fraction of nucleotides stored in anchors as a function of the substitution rate. For closely related sequences anchors are long and gaps are very short (Section 2.1) and thus, most nucleotides are known to be identical to the reference. For instance, at a substitution rate of 0.025, about 92 % of nucleotides are within anchors. Hence, for two queries, nucleotides at the same site with respect to the reference have an 85 % chance of being identical as they are both in anchors. At a substitution rate of 0.1 this drops to 25 %. For comparison, the largest substitution rate in the set of 29 *E. coli/Shigella* genomes is 0.027 (Section 4.3). Thus, potentially a large amount of nucleotide comparisons could be avoided in real data sets, making this optimization feasible.

Code complexity is already an issue for phylonium. While I tried to avoid premature optimization, the implemented string algorithms are inherently complex to guarantee adequate performance (Knuth 1974). Particularly, the indexing and its usage are heavily optimized and no longer resemble their textbook counterparts, even containing a *harmful* `goto` (Dijkstra 1968). This complexity is necessary to achieve nucleotide precision; *k*-mer methods tend to be simpler. For instance, the most complex data structure in mash is a heap. This simplicity attracts others to build upon it. For example, sourmash is an application of mash to metagenomes (Brown et al. 2020). Metagenomes arise when cells cannot be cultured individually so that a sequencing experiment contains a mixture of various species. It thus becomes difficult to assign a read to a single species.

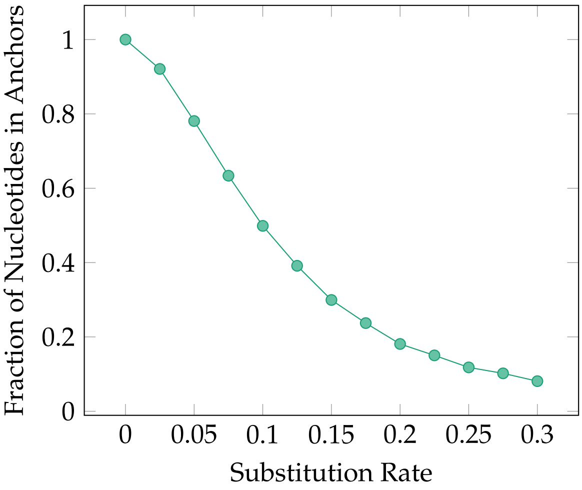


Figure 5.2: Simulated were two 100 kb sequences at a fixed substitution rate. Each homologous segment consists of anchors and gaps. As the substitution rate grows, anchors shorten and gaps widen.

With the advent of high-throughput-sequencing, individual sequencing reads became as short as 150 nucleotides. Working on these unassembled reads is fine for k-mer methods if their k is smaller than the read length. However, string-based methods, such as phylonium, require long contigs for homology detection. While this is currently a disadvantage, it might disappear in the future. Upcoming long-read-sequencing technology yields reads up to a hundred kilobases long from a single DNA molecule. With this technology, both the assembly and the discrimination of reads from different species in metagenomes may become less critical in the future as it will be easier for assemblers to produce long contigs (Ruan and Li 2020).

Phylogenies are usually annotated with support values, traditionally computed via bootstrapping. However, as the classical bootstrap requires an MSA, it cannot be applied to alignment-free methods. However, phylonium forms an approximated MSA, thus, it can be extended to calculate classical bootstrap values. In Chapter 2 I described three alternative ways to compute support values even in the absence of an explicit alignment: pairwise bootstrapping, quartet analysis, and the quartet concordance. The pairwise bootstrap is specific to andi and phylonium, as they approximate pairwise sequence alignments. Two other methods focus on quartet analysis. Of these uncommon methods, the pairwise bootstrap as implemented in andi has already been used in a couple of analyses, for instance in a comparison of wheat pathogens by McDonald et al. (2018). As the popularity of alignment-free methods and mash in particular is rising, it will be interesting to see how the computation of support values will change in the future.

Despite falling into the category of *alignment-free* genome comparison tools, phylonium internally builds approximated alignments. These alignments are not as precise as those produced by proper MSA methods, but good enough for distance estimation. They might also be good enough for usage outside of phylogeny reconstruction. For instance, phylonium could be adapted for population genetics: A column containing a polymorphism is called a segregating site. The number of segregating sites, S, is proportional to the mutation parameter θ (Watterson 1975).

$$\theta = \frac{S}{\sum_{i=1}^{n-1} \frac{1}{i}}$$

Here, n is the number of sequenced samples, equivalent to the number of rows in an alignment. Already, phylonium counts the number of mismatches between two sequences $m(S,Q)$. As there are $\binom{n}{2}$ pairs in a data set, the average is

$$\pi = \frac{2}{n(n-1)} \sum_{S,Q} m(S,Q) \ .$$

Under a neutral model of evolution, at constant population size, these two values should be equal, $\theta = \pi$ (Tajima 1989). Tajima suggested the test statistic D to check for significant deviation from the null model,

$$D = \frac{\pi - \theta}{\sqrt{\mathrm{Var}(\pi - \theta)}} \ .$$

Under neutrality $D \approx 0$. Deviations of D can be attributed to different biological forces. For instance, a positive D can be the result of a recent population expansion or a selective sweep. Conversely, a negative D hints at balancing selection or population contraction. Tajima's D is widely used in population genetics (Hartl and Clark 1997, pp. 303–304). With a future version of phylonium, it would be possible to calculate this statistic quickly for large samples of genomes.

This dissertation introduces phylonium, a new alignment-free tool that estimates evolutionary distances for phylogeny reconstruction. I explained its algorithm, the optimizations in the ESA structure, as well as the advances made to speed up sequence comparison. In Chapter 4 I noted that phylonium is as fast and accurate as competing alignment-free approaches even when applied to large data sets. Further, I showed how phylogenies reconstructed by phylonium can be annotated with support values, which was previously restricted to alignment-based methods. Phylonium demonstrates that in the case of closely related genomes, an approximated alignment is good enough to be useful and can be quickly computed. This approach might influence other alignment-free tools in the future.

A List of Software Contributions

> It takes a minimum of two to three years for a piece of scientific software to become mature enough to publicize.
>
> Titus Brown

While the bulk of this thesis describes phylonium, a lot of auxiliary software had to be created in its wake. Further, I worked with a lot of 3rd party code fixing bugs and improving performance. In the following I outline my contributions to different projects.

Afra Section 2.3 describes different ways to compute support values for phylogenies. As a replacement for the classical bootstrap in the case of alignment-free distances we developed afra (Klötzl and Haubold 2016). While Bernhard and I developed the concept together, the code is mostly mine.
GNU Public License `https://github.com/evolbioinf/afra`

Andi Anchor distances and their implementation andi have been the subject of my Master's thesis. Since then, I added some new features (`file-of-filenames`-parameter, zsh autocompletion), fixed bugs (wrong anchor threshold) and sped up computation (lucky anchors, see Section 2.1).
GNU Public License `https://github.com/evolbioinf/andi`

Biozsh With the 2019 release of macOS Catalania the default shell changed from bash to zsh. The latter features improved autocompletion, among other things. My project biozsh bundles bioinformatics related functionality in one place. It already has attracted attention from the community, including other users providing completion scripts.
MIT License `https://github.com/kloetzl/biozsh`

Cophylog As one of the first alignment-free tools cophylog produced accurate evolutionary distance. It was sufficiently fast on small data sets but the performance on large data sets suffered as it was purely single-threaded. Born out of the need to experiment with it, I created a multi-threaded version of cophylog. The change has been merged upstream.
GNU Public License `https://github.com/yhg926/co-phylog`

GeoMeTree One of the most commonly used tools in bioinformatics is Phylip (Felsenstein 2005). It contains routines to compute the symmetric distance between two phylogenies among many other methods. Unfortunately, it is by design Unix unfriendly and does not work in pipes. GeoMeTree on the other hand is a Python program that can also compute various tree distances (Kupczok et al. 2008). I took the source and slightly

changed its command line interface to fit my needs. The original authors were happy that an updated version is now on GitHub.
GNU Public License `https://github.com/kloetzl/geometree`

Hotspot In a collaboration with two colleagues we created the hotspot project. It bundles four applications for the analysis of recombination hotspots. I helped with packaging the project across different operating systems.
GNU Public License `https://github.com/EvolBioInf/hotspot`

kseq.h The original kseq.h could parse Fasta files at 930 MB/s. While comparing its performance with my own parser, I created a simple patch that boosted throughput the throughput of kseq.h to 1.9 GB/s; the patch has since been merged upstream. Unfortunately, clients tend to copy the file kseq.h into their own code base. Thus, many projects use old versions and do not benefit from the patch.
MIT License `https://github.com/lh3/seqtk/commit/ba10dad8a5cc983c6d26fb4bb47251defe286d24#diff-1f9bb017a5df245ad6173ad820932566`

Libdna During the development of phylonium I wrote a number of high-throughput string processing functions (Sections 3.3 and following). Particularly, I spent some time engineering the code so that at run time the fastest available implementation depending on the instruction set would be chosen. As other tools could also benefit from these implementations, I bundled them in a project called libdna. It contains a number of highly-specialized and generic functions including documentation.
GNU Public License `https://github.com/kloetzl/geometree`

Maf2dist In order to compute a distance matrix from the multiple sequence alignments produced by mugsy I wrote maf2dist. It contains a particularly fast way to count substitutions in the presence of gaps. Further, it also features complete deletion.
ISC License `https://github.com/kloetzl/maf2dist`

Mash Mash is probably phylonium's closest competition in both accuracy and speed (Section 4.3). While looking at the code, trying to figure out how it works, I realized that parts of it were unnecessarily slow. I proposed a few changes which made mash faster by a factor of 3.5.
3-Clause BSD License `https://github.com/marbl/mash`

Mat Tools The output of most alignment-free tools is a distance matrix. To compare them, for instance with our new Δ-value, I created the mat tools. It is an assortment of tools to manipulate and compare distance matrices.
GNU Public License `https://github.com/EvolBioInf/mattools`

MUMmer Maximal matches, which appear only once in the subject and the query sequence are called maximal unique matches, MUMs for short. MUMmer is a suite of programs for pairwise sequence alignments making use of MUMs as alignment anchors. Between the anchors an alignment is formed using a dynamic programming scheme. To this end it created a matrix of nodes, however each node was also made up of three scores, each requiring additional memory for padding. Rearranging the data inside a node reduces the memory usage by a third while also making the program 20 % faster due to better caching behavior.
Artistic License 2.0 `https://github.com/mummer4/mummer/pull/30`

Pfasta The most used parser for Fasta files, kseq.h, had insufficient error handling for my taste. So I created pfasta which is slightly slower by gives much better error messages (see Section 3.6). The parser now comes with a number of convenient tools for the manipulation of sequence files.
ISC License `https://github.com/kloetzl/pfasta`

Phylonium The subject of this thesis.
GNU Public License `https://github.com/evolbioinf/phylonium`

Snp-Dists This tool also counts substitutions but only for already aligned sequences and gives the absolute counts instead of a substitution rate. I fixed a few bugs and made the code cleaner.
GNU Public License `https://github.com/tseemann/snp-dists`

B Source Code

B.1 Anchor Distance

This algorithm computes the uncorrected asymmetric anchor distance of Q with respect to the subject S as implemented in andi. Adapted from (Klötzl 2015).

```
fn dist_anchor
requires S
input Q

let E ← ESA(S)
let L ← threshold(S)
let mismatches ← 0
let homo_nucl ← 0
let last_pos_q ← 0
let last_match ← ⊥
let last_was_right_anchor ← false
let q ← 0
while q < |Q| do // Stream the complete query

  // Find the next match
  m ← lucky_match(S, Q, last_match, q) or get_match(E, Q[q...])
  if m.is_unique and m.length ≥ L then
    // m is an anchor

    if q - last_pos_q = m.pos - last_match.pos then
      // Found a pair
      mismatches ← mismatches + count_diff(Q[last_pos_q...q], S[last_match.pos ...m.pos])
      homo_nucl ← homo_nucl + q - last_pos_q
      last_was_right_anchor ← true
    else
      // Correctly count the nucleotides from right anchors
      if last_was_right_anchor then
        homo_nucl ← homo_nucl + last_match.length
      end

      last_was_right_anchor ← false
    end

    // Cache values for later
    last_pos_q ← q
    last_match ← m
  end

  // Skip the mutation
  q ← q + m.length + 1
end
output mismatches/homo_nucl
```

B.2 Fast Sequence Comparison

A fast function for counting the number of mismatches.

```
size_t mismatches(const char *seq1, const char *seq2, size_t length)
{
  size_t num_mismatches = 0;

  typedef __m128i vec_type;
  size_t num_bytes = sizeof(vec_type);
  size_t vec_offset = 0;
  size_t vec_length = length / num_bytes;

  for (; vec_offset < vec_length; vec_offset++) {
    vec_type seq1_vec;
    memcpy(&seq1_vec, seq1 + vec_offset * num_bytes, num_bytes);
    vec_type seq2_vec;
    memcpy(&seq2_vec, seq2 + vec_offset * num_bytes, num_bytes);

    vec_type comp = _mm_cmpeq_epi8(seq1_vec, seq2_vec);

    unsigned int vmask = _mm_movemask_epi8(comp);
    unsigned int lower_bits = (1 << num_bytes) - 1;
    num_mismatches += __builtin_popcount(~vmask & lower_bits);
  }

  size_t offset = vec_offset * num_bytes;
  for (; offset < length; offset++) {
    if (seq1[offset] != seq2[offset]) {
      num_mismatches++;
    }
  }

  return num_mismatches;
}
```

B.3 Comparing with the Reverse Complement

A function for counting the number of mismatches quickly when two sequences come from opposite strands.

```
size_t
cmprevcomp(
  const char *seq1,
  const char *seq2,
  size_t length)
{
  size_t substitutions = 0;
  size_t num_bytes = sizeof(__m128i);
  size_t vec_offset = 0;
  size_t vec_length = length / num_bytes;

  for (; vec_offset < vec_length; vec_offset++) {
    __m128i b;
    memcpy(&b, seq1 + vec_offset * num_bytes, num_bytes);
    __m128i o;
    size_t pos = length - (vec_offset + 1) * num_bytes;
    memcpy(&o, seq2 + pos, num_bytes);

    __m128i mask = _mm_set_epi8(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
         14, 15);
    __m128i reversed = _mm_shuffle_epi8(o, mask);

    __m128i v1 = _mm_xor_si128(b, reversed);
    __m128i mask6 = _mm_set1_epi8(6);
    __m128i v2 = _mm_and_si128(v1, mask6);
    __m128i mask4 = _mm_set1_epi8(4);
    __m128i v3 = _mm_cmpeq_epi8(v2, mask4);

    unsigned int vmask = _mm_movemask_epi8(v3);
    unsigned int lower_bits = (1 << num_bytes) - 1;
    substitutions += __builtin_popcount(~vmask & lower_bits);
  }

  size_t offset = vec_offset * num_bytes;

  for (; offset < length; offset++) {
    if (!is_complement(seq1[offset], seq2[length - 1 - offset])) {
      substitutions++;
    }
  }

  return substitutions;
}
```

Bibliography

Abouelhoda, M. I., S. Kurtz, and E. Ohlebusch (2002). “The Enhanced Suffix Array and Its Applications to Genome Analysis”. In: *Algorithms in Bioinformatics*. Vol. 2452. Lecture Notes in Computer Science, pp. 449–463.

— (2004). “Replacing Suffix Trees with Enhanced Suffix Arrays”. *Journal of Discrete Algorithms* 2.1, pp. 53–86.

Adams, M. D. et al. (2000). “The Genome Sequence of Drosophila melanogaster”. *Science* 287.5461, pp. 2185–2195.

Aho, A. V. and M. J. Corasick (1975). “Efficient String Matching: An Aid to Bibliographic Search”. *Communications of the ACM* 18.6, pp. 333–340.

Alm, R. A. et al. (1999). “Genomic-sequence comparison of two unrelated isolates of the human gastric pathogen *Helicobacter pylori*”. *Nature* 397 (6715), pp. 176–180.

Altschul, S. F., W. Gish, W. Miller, E. W. Myers, and D. J. Lipman (1990). “Basic local alignment search tool”. *Journal of Molecular Biology* 215.3, pp. 403–410.

Angiuoli, S. V. and S. L. Salzberg (2011). “Mugsy: fast multiple alignment of closely related whole genomes”. *Bioinformatics* 27.3, pp. 334–342.

Apostolico, A., C. Guerra, G. M. Landau, and C. Pizzi (2016). “Sequence similarity measures based on bounded hamming distance”. *Theoretical Computer Science* 638. Pattern Matching, Text Data Structures and Compression, pp. 76–90.

Baker, D. N. and B. Langmead (2019). “Dashing: Fast and Accurate Genomic Distances with HyperLogLog”. *Genome Biology* 20 (1).

Bickel, P. J. and D. A. Freedman (1981). “Some Asymptotic Theory for the Bootstrap”. *Annals of Statistics* 9.6, pp. 1196–1217.

Boratyn, G. M. et al. (2013). “BLAST: a more efficient report with usability improvements”. *Nucleic Acids Research* 41.W1, W29–W33.

Broder, A. Z. (1997). “On the resemblance and containment of documents”. *Compression and Complexity of Sequences*, pp. 21–29.

Brown, C. T. et al. (2020). *dib-lab/sourmash*. Version v3.2.2. `https://doi.org/10.5281/zenodo.3660163`.

Chiaromonte, F., V. B. Yap, and W. Miller (2001). “Scoring Pairwise Genomic Sequence Alignments”. In: *Biocomputing 2002*, pp. 115–126.

Dagan, T. and W. Martin (2009). “Getting a better picture of microbial evolution en route to a network of genomes”. *Philosophical Transactions of the Royal Society B: Biological Sciences* 364.1527, pp. 2187–2196.

Darling, A. E., B. Mau, and N. T. Perna (2010). “progressiveMauve: Multiple Genome Alignment with Gene Gain, Loss and Rearrangement”. *PLOS ONE* 5.6, pp. 1–17.

Delcher, A. L., S. Kasif, R. D. Fleischmann, J. Peterson, O. White, and S. L. Salzberg (1999). “Alignment of whole genomes”. *Nucleic Acids Research* 27.11, pp. 2369–2376.

Devillers, H. and S. Schbath (2012). "Separating significant matches from spurious matches in DNA sequences". *Journal of Computational Biology* 19.1, pp. 1–12.

Dijkstra, E. W. (1968). "Go To Statement Considered Harmful". *Communications of the ACM* 11.3, pp. 147–148.

— (1972). "The Humble Programmer". *Communications of the ACM* 15.10, pp. 859–866.

Domazet-Lošo, M. and B. Haubold (2009). "Efficient estimation of pairwise distances between genomes". *Bioinformatics* 25.24, pp. 3221–3227.

— (2011). "Alignment-free detection of local similarity among viral and bacterial genomes". *Bioinformatics* 27 (11), pp. 1466–72.

Döring, A., D. Weese, T. Rausch, and K. Reinert (2008). "SeqAn An efficient, generic C++ library for sequence analysis". *BMC Bioinformatics* 9.1.

Dutheil, J., S. Gaillard, E. Bazin, S. Glémin, V. Ranwez, N. Galtier, and K. Belkhir (2006). "Bio++: a set of C++ libraries for sequence analysis, phylogenetics, molecular evolution and population genetics". *BMC Bioinformatics* 7.1.

Earl, D. et al. (2014). "Alignathon: a competitive assessment of whole-genome alignment methods". *Genome Research* 24.12, pp. 2077–2089.

Efron, B. (1979). "Bootstrap Methods: Another Look at the Jackknife". *The Annals of Statistics* 7.1, pp. 1–26.

Felsenstein, J. (1981). *Journal of Molecular Evolution* 17 (6), pp. 368–376.

— (1985). "Confidence Limits on Phylogenies: An Approach using the Bootstrap". *Evolution* 39.4, pp. 783–791.

— (2004). *Inferring Phylogenies*. Sinauer Associates, Inc.

— (2005). *PHYLIP (Phylogeny Inference Package)*. Distributed by the author. Version version 3.6. Department of Genome Sciences, University of Washington.

Ferragina, P. and G. Manzini (2000). "Opportunistic data structures with applications". *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pp. 390–398.

Fiori, F. J., W. Pakalén, and J. Tarhio (2017). "Counting Mismatches with SIMD". *Proceedings of the Prague Stringology Conference 2017*. Ed. by J. Holub and J. Žd'árek. Czech Technical University in Prague, Czech Republic, pp. 51–61.

Fischer, J. and F. Kurpicz (2017). "Dismantling DivSufSort". *Proceedings of the Prague Stringology Conference 2017*. Ed. by J. Holub and J. Žd'árek. Czech Technical University in Prague, Czech Republic, pp. 62–76.

Fitch, W. M. (1970). "Distinguishing homologous from analogous proteins". *Systematic Zoology* 19 (2), pp. 99–113.

Fitch, W. M. and E. Margoliash (1967). "Construction of Phylogenetic Trees". *Science* 155.3760, pp. 279–284.

Flajolet, P., É. Fusy, O. Gandouet, and F. Meunier (2007). "Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm". *Proceedings of the 2007 International Conference on Analysis of Algorithms*.

Fleischmann, R. D. et al. (1995). "Whole-genome random sequencing and assembly of Haemophilus influenzae Rd". *Science* 269.5223, pp. 496–512.

Frith, M. C. and A. M. S. Shrestha (2018). "A Simplified Description of Child Tables for Sequence Similarity Search". *IEEE/ACM Transactions on Compututational Biology and Bioinformatics* 15.6, pp. 2067–2073.

Galassi, M., J. Davies, J. Theiler, B. Gough, G. Jungman, P. Alken, M. Booth, F. Rossi, and R. Ulerich (2016). *GNU Scientific Library Reference Manual*. 2.3.

Gao, K. and J. Miller (2020). "Primary orthologs from local sequence context". *BMC Bioinformatics* 21 (1).

Goffeau, A. et al. (1996). "Life with 6000 Genes". *Science* 274.5287, pp. 546–567.

Goodwin, S., J. D. McPherson, and W. R. McCombie (2016). "Coming of age: ten years of next-generation sequencing technologies". *Nature Reviews Genetics* 17 (6), pp. 333–351.

Guénoche, A. and H. Garreta (2001). "Can We Have Confidence in a Tree Representation?" *Computational Biology*. Ed. by O. Gascuel and M.-F. Sagot. Springer Berlin Heidelberg, pp. 45–56.

Gusfield, D. (1997). *Algorithms on Strings, Trees, and Sequences. Computer Science and Computational Biology*. Cambridge University Press.

Haeckel, E. (1897). *The Evolution of Man: A Popular Exposition of the Principal Points of Human Ontogeny and Phylogeny*. New York: Appleton & Co.

Hartl, D. L. and A. G. Clark (1997). *Principles of Population Genetics*. 3rd ed. Sinauer Associates, Inc.

Haubold, B. (2014). "Alignment-free phylogenetics and population genetics". *Briefings in Bioinformatics* 15.3, pp. 407–418.

Haubold, B. and F. Klötzl (2020). "Fast Phylogeny Reconstruction from Genomes of Closely Related Microbes". In: *Bacterial Pangenomics: Methods and Protocols*. Ed. by M. Fondi, A. Mengoni, and G. Bacci. 2nd ed. Springer. Forthcoming.

Haubold, B., F. Klötzl, and P. Pfaffelhuber (2015). "andi: Fast and accurate estimation of evolutionary distances between closely related genomes". *Bioinformatics* 31.8, pp. 1169–1175.

Haubold, B., L. Krause, T. Horn, and P. Pfaffelhuber (2013). "An alignment-free test for recombination". *Bioinformatics* 29.24, pp. 3121–3127.

Haubold, B. and P. Pfaffelhuber (2012). "Alignment-Free Population Genomics: An Efficient Estimator of Sequence Diversity". *G3: Genes, Genomes, Genetics* 2.8, pp. 883–889.

Haubold, B., P. Pfaffelhuber, M. Domazet-Lošo, and T. Wiehe (2009). "Estimating Mutation Distances from Unaligned Genomes". *Journal of Computational Biology* 16.10, pp. 1487–1500.

Haubold, B., N. Pierstorff, F. Möller, and T. Wiehe (2005). "Genome comparison without alignment using shortest unique substrings". *BMC Bioinformatics* 6 (1).

Haubold, B. and T. Wiehe (2006a). "How repetitive are genomes?" *BMC Bioinformatics* 7 (1).

— (2006b). *Introduction to Computational Biology. An Evolutionary Approach*. Birkhäuser.

Hide, W., J. Burke, and D. B. Davison (1994). "Biological Evaluation of d2, an Algorithm for High-Performance Sequence Comparison". *Journal of Computational Biology* 1.3, pp. 199–215.

Hillis, D. and J. Bull (1993). "An Empirical Test of Bootstrapping as a Method for Assessing Confidence in Phylogenetic Analysis". *Systematic Biology* 42, pp. 182–192.

Hoang, D. T., O. Chernomor, A. von Haeseler, B. Q. Minh, and L. S. Vinh (2017). "UFBoot2: Improving the Ultrafast Bootstrap Approximation". *Molecular Biology and Evolution* 35.2, pp. 518–522.

Holt, K. E. et al. (2018). "Frequent transmission of the Mycobacterium tuberculosis Beijing lineage and positive selection for the EsxW Beijing variant in Vietnam". *Nature Genetics* 50, pp. 849–856.

Huelsenbeck, J. P. and B. Rannala (1997). "Phylogenetic Methods Come of Age: Testing Hypotheses in an Evolutionary Context". *Science* 276.5310, pp. 227–232.

Intel Corporation (2018). *Intel 64 and IA-32 Architectures Software Developer's Manual*.

International Human Genome Sequencing Consortium (2001). *Nature* 409 (6822), pp. 860–921.

Jonathan, C., U. F. Minhas, J. Hunter, J. Levandoski, and G. Nishanov (2018). "Exploiting Coroutines to Attack the 'Killer Nanoseconds'". *Proceedings of the VLDB Endowment* 11.11, pp. 1702–1714.

Jukes, T. H. and C. R. Cantor (1969). "Evolution of protein molecules". *Mammalian protein metabolism* 3, pp. 21–132.

Kärkkäinen, J. and P. Sanders (2003). "Simple Linear Work Suffix Array Construction". *Automata, Languages and Programming*. Ed. by J. C. M. Baeten, J. K. Lenstra, J. Parrow, and G. J. Woeginger. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 943–955.

Kärkkäinen, J., P. Sanders, and S. Burkhardt (2006). "Linear Work Suffix Array Construction". *Journal of the ACM* 53.6, pp. 918–936.

Karlin, S. and S. F. Altschul (1990). "Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes". *Proceedings of the National Academy of Sciences* 87.6, pp. 2264–2268.

Kasai, T., G. Lee, H. Arimura, S. Arikawa, and K. Park (2001). "Linear-time longest-common-prefix computation in suffix arrays and its applications". *Combinatorial Pattern Matching*. Springer, pp. 181–192.

Kelleher, J., Y. Wong, A. W. Wohns, C. Fadil, P. K. Albers, and G. McVean (2019). "Inferring whole-genome histories in large population datasets". *Nature Genetics* 51.9, pp. 1330–1338.

Kim, D. K., M. Kim, and H. Park (2008). "Linearized Suffix Tree: an Efficient Index Data Structure with the Capabilities of Suffix Trees and Suffix Arrays". *Algorithmica* 52 (3), pp. 350–377.

Kim, M., H.-S. Oh, S.-C. Park, and J. Chun (2014). "Towards a taxonomic coherence between average nucleotide identity and 16S rRNA gene sequence similarity for species demarcation of prokaryotes". *International Journal of Systematic and Evolutionary Microbiology* 64.2, pp. 346–351.

Kimura, M. (1980). "A Simple Method for Estimating Evolutionary Rates of Base Substitutions Through Comparative Studies of Nucleotide Sequences". *Journal of Molecular Evolution* 16, pp. 111–120.

Klötzl, F. (2015). "Efficient Estimation of Evolutionary Distances". Masters's Thesis. University of Lübeck.

Klötzl, F. and B. Haubold (2016). "Support Values for Genome Phylogenies". *Life* 6.1.

— (2019). "Phylonium: Fast Estimation of Evolutionary Distances from Large Samples of Similar Genomes". *Bioinformatics* 36.7, pp. 2040–2046.

Knuth, D. E. (1974). "Structured Programming with Go to Statements". *ACM Computing Surveys* 6.4, pp. 261–301.

— (2013a). *The Art of Computer Programming 3: Sorting and Searching*. 2nd. Addison-Wesley Professional.

— (2013b). *The Art of Computer Programming 4A: Combinatorial Algorithms, Part 1*. 2nd. Addison-Wesley Professional.

Knuth, D. E., J. J. Morris, and V. Pratt (1977). "Fast Pattern Matching in Strings". *SIAM Journal of Computing* 6.2.

Ko, P. and S. Aluru (2003). "Space efficient linear time construction of suffix arrays". *Journal of Discrete Algorithms*. Springer, pp. 200–210.

Kryukov, K., M. T. Ueda, S. Nakagawa, and T. Imanishi (2019). "Nucleotide Archival Format (NAF) enables efficient lossless reference-free compression of DNA sequences". *Bioinformatics* 35.19, pp. 3826–3828.

Kuhner, M. K. and J. Felsenstein (1994). "A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates." *Molecular Biology and Evolution* 11.3, pp. 459–468.

Kupczok, A. (2010). "Postprocessing Phylogenies: Tree Distances and Supertrees". Dr. rer. nat. Universität Wien.

Kupczok, A., A. von Haeseler, and S. Klaere (2008). "An Exact Algorithm for the Geodesic Distance between Phylogenetic Trees". *Journal of Computational Biology* 15.6, pp. 577–591.

Kurtz, S., A. Phillippy, A. L. Delcher, M. Smoot, M. Shumway, C. Antonescu, and S. L. Salzberg (2004). "Versatile and open software for comparing large genomes". *Genome Biology* 5 (2).

Labeit, J., J. Shun, and G. E. Blelloch (2016). "Parallel Lightweight Wavelet Tree, Suffix Array and FM-Index Construction". *Data Compression Conference*, pp. 33–42.

Leimeister, C.-A., M. Boden, S. Horwege, S. Lindner, and B. Morgenstern (2014). "Fast alignment-free sequence comparison using spaced-word frequencies". *Bioinformatics* 30.14, pp. 1991–1999.

Leimeister, C.-A. and B. Morgenstern (2014). "kmacs: the k-mismatch average common substring approach to alignment-free sequence comparison". *Bioinformatics* 30.14, pp. 2000–2008.

Leimeister, C.-A., S. Sohrabi-Jahromi, and B. Morgenstern (2017). "Fast and accurate phylogeny reconstruction using filtered spaced-word matches". *Bioinformatics* 33.7, pp. 971–979.

Li, H. (2012). *seqtk Toolkit for processing sequences in FASTA/Q formats*. `https://github.com/lh3/seqtk`.

Li, H. and R. Durbin (2009). "Fast and accurate short read alignment with Burrows–Wheeler transform". *Bioinformatics* 25.14, pp. 1754–1760.

Lin, J. (1991). "Divergence measures based on the Shannon entropy". *IEEE Transactions on Information Theory* 37.1, pp. 145–151.

Lin, Y., V. Rajan, and B. M. Moret (2012). "Bootstrapping phylogenies inferred from rearrangement data". *Algorithms for Molecular Biology* 7.1.

Liò, P. and N. Goldman (1998). "Models of Molecular Evolution and Phylogeny". *Genome Research* 8.12, pp. 1233–1244.

Lynch, M. (2007). *The Origins of Genome Architecture*. Sinauer Associates, Inc.

Manber, U. and G. Myers (1993). "Suffix-Arrays: A New Method for On-Line String Searches". *SIAM Journal on Computing* 22.5, pp. 935–948.

Mantel, N. (1967). "The Detection of Disease Clustering and a Generalized Regression Approach". *Cancer Research* 27.2 Part 1, pp. 209–220.

Manzini, G. and P. Ferragina (2004). "Engineering a Lightweight Suffix Array Construction Algorithm". *Algorithmica* 40 (1), pp. 33–50.

McDonald, M. C., D. Ahren, S. Simpfendorfer, A. Milgate, and P. S. Solomon (2018). "The discovery of the virulence gene ToxA in the wheat and barley pathogen Bipolaris sorokiniana". *Molecular Plant Pathology* 19.2, pp. 432–439.

Md Mukarram Hossain, A. S., B. P. Blackburne, A. Shah, and S. Whelan (2015). "Evidence of Statistical Inconsistency of Phylogenetic Methods in the Presence of Multiple Sequence Alignment Uncertainty". *Genome Biology and Evolution* 7 (8), pp. 2102–2116.

Möller, S. et al. (2017). "Robust Cross-Platform Workflows: How Technical and Scientific Communities Collaborate to Develop, Test and Share Best Practices for Data Analysis". *Data Science and Engineering* 2.3, pp. 232–244.

Morgenstern, B., B. Zhu, S. Horwege, and C.-A. Leimeister (2014). "Estimating Evolutionary Distances from Spaced-Word Matches". In: *Algorithms in Bioinformatics*. Lecture Notes in Computer Science, pp. 161–173.

Myers, G. (2014). "Efficient Local Alignment Discovery amongst Noisy Long Reads". *Algorithms in Bioinformatics*. Ed. by D. Brown and B. Morgenstern. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 52–67.

Nomenclature Committee of the International Union of Biochemistry (1985). "Nomenclature for incompletely specified bases in nucleic acid sequences". *European Journal of Biochemistry* 150.

Odenthal-Hesse, L., J. Y. Dutheil, F. Klötzl, and B. Haubold (2016). "hotspot: software to support sperm-typing for investigating recombination hotspots". *Bioinformatics* 32.16, pp. 2554–2555.

Ohlebusch, E. (2013). *Bioinformatics Algorithms*. 1st. Oldenbusch Verlag.

Ondov, B. D., G. J. Starrett, A. Sappington, A. Kostic, S. Koren, C. B. Buck, and A. M. Phillippy (2019). "Mash Screen: high-throughput sequence containment estimation for genome discovery". *Genome Biology* 20.

Ondov, B. D., T. J. Treangen, P. Melsted, A. B. Mallonee, N. H. Bergman, S. Koren, and A. M. Phillippy (2016). "Mash: fast genome and metagenome distance estimation using MinHash". *Genome Biology* 17.1, p. 132.

Pearson, W. R. (1990). "Rapid and sensitive sequence comparison with FASTP and FASTA". *Methods in Enzymology* 183, pp. 63–98.

Pease, J. B., J. W. Brown, J. F. Walker, C. E. Hinchliff, and S. A. Smith (2018). "Quartet Sampling distinguishes lack of support from conflicting support in the green plant tree of life". *American Journal of Botany* 105.3, pp. 385–403.

Pirogov, A., P. Pfaffelhuber, A. Börsch-Haubold, and B. Haubold (2019). "High-complexity regions in mammalian genomes are enriched for developmental genes". *Bioinformatics* 35 (11), pp. 1813–1819.

Pizzi, C. (2016). "MissMax: alignment-free sequence comparison with mismatches through filtering and heuristics". *Algorithms for Molecular Biology* 11.1.

Ragan, M. A. (2009). “Trees and networks before and after Darwin”. *Biology Direct* 4 (1).

Rambaut, A. and N. C. Grass (1997). “Seq-Gen: an application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees”. *Bioinformatics* 13.3, pp. 235–238.

Reinert, G., S. Schbath, and M. S. Waterman (2000). “Probabilistic and Statistical Properties of Words: An Overview”. *Journal of Computational Biology* 7.1-2, pp. 1–46.

Rice, E. S. and R. E. Green (2019). “New Approaches for Genome Assembly and Scaffolding”. *Annual Review of Animal Biosciences* 7.1, pp. 17–40.

Ritchie, D. M. and K. Thompson (1974). “The UNIX Time-Sharing System”. *Communications of the ACM* 17.7, pp. 365–375.

Robinson, D. and L. Foulds (1981). “Comparison of phylogenetic trees”. *Mathematical Biosciences* 53.1–2, pp. 131–147.

Röhling, S., A. Linne, J. Schellhorn, M. Hosseini, T. Dencker, and B. Morgenstern (2020). “The number of k-mer matches between two DNA sequences as a function of k and applications to estimate phylogenetic distances”. *PLOS ONE* 15.2, pp. 1–18.

Ruan, J. and H. Li (2020). “Fast and accurate long-read assembly with wtdbg2”. *Nature Methods* 17 (2), pp. 155–158.

Saitou, N. and M. Nei (1987). “The neighbor-joining method: a new method for reconstructing phylogenetic trees.” *Molecular Biology and Evolution* 4.4, pp. 406–425.

Sanger, F., G. M. Air, B. G. Barell, A. R. Coulson, J. C. Fiddes, C. A. Hutchison, P. M. Slocombe, and M. Smith (1977). “Nucleotide sequence of bacteriophage ΦX174 DNA”. *Nature* 265 (5596), pp. 687–695.

Sanger, F., A. Coulson, G. Hong, D. Hill, and G. Petersen (1982). “Nucleotide sequence of bacteriophage λ DNA”. *Journal of Molecular Biology* 162.4, pp. 729–773.

Scally, A. et al. (2012). “Insights into hominid evolution from the gorilla genome sequence”. *Nature* 483 (7388), pp. 169–175.

Seemann, T. (2018). *Source code for snp-dists software*. Version 0.6.2. `https://doi.org/10.5281/zenodo.1411986`.

Seidel, A. (2017). “Effiziente Berechnung von Genomdistanzen basierend auf aktuellen seeding, seed-Filter und Alignment Methoden”. MA thesis. Universität Hamburg.

Shimodaira, H. and M. Hasegawa (1999). “Multiple Comparisons of Log-Likelihoods with Applications to Phylogenetic Inference”. *Molecular Biology and Evolution* 16.8, pp. 1114–1116.

Singh, K. (1981). “On the Asymptotic Accuracy of Efron’s Bootstrap”. *The Annals of Statistics* 9.6, pp. 1187–1195.

Sinha, R., S. Puglisi, A. Moffat, and A. Turpin (2008). “Improving Suffix Array Locality for Fast Pattern Matching on Disk”. *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’08. Vancouver, Canada: Association for Computing Machinery, pp. 661–672.

Sohrabi-Jahromi, S., C.-A. Leimeister, and B. Morgenstern (2017). “Fast and accurate phylogeny reconstruction using filtered spaced-word matches”. *Bioinformatics* 33.7, pp. 971–979.

Šošić, M. and M. Šikić (2017). “Edlib: a C/C++ library for fast, exact sequence alignment using edit distance”. *Bioinformatics* 33.9, pp. 1394–1395.

Stallman, R. M. and G. DeveloperCommunity (2009). *Using The Gnu Compiler Collection: A Gnu Manual For Gcc Version 4.3.3*. Paramount, CA: CreateSpace.

Stamatakis, A. (2014). "RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies". *Bioinformatics* 30.9, pp. 1312–1313.

Stamatakis, A., P. Hoover, and J. Rougemont (2008). "A Rapid Bootstrap Algorithm for the RAxML Web Servers". *Systematic Biology* 57.5, pp. 758–771.

Stecher, G., S. Kumar, and K. Tamura (2016). "MEGA7: Molecular Evolutionary Genetics Analysis Version 7.0 for Bigger Datasets". *Molecular Biology and Evolution* 33.7, pp. 1870–1874.

Tajima, F. (1989). "Statistical method for testing the neutral mutation hypothesis by DNA polymorphism." *Genetics* 123.3, pp. 585–595.

Tang, P., M. A. Croxen, M. R. Hasan, W. W. Hsiao, and L. M. Hoang (2017). "Infection control in the new age of genomic epidemiology". *American Journal of Infection Control* 45.2, pp. 170–179.

The Arabidopsis Genome Initiative (2000). "Analysis of the genome sequence of the flowering plant Arabidopsis thaliana". *Nature* 408 (6814), pp. 796–815.

The *C. elegans* Sequencing Consortium (1998). "Genome Sequence of the Nematode C. elegans: A Platform for Investigating Biology". *Science* 282.5396, pp. 2012–2018.

Tiwari, P. K., V. V. Menon, J. Murugan, J. Chandrasekaran, G. S. Akisetty, P. Ramachandran, S. K. Venkata, C. A. Bird, and K. Cone (2018). *Accelerating x265 with Intel® Advanced Vector Extensions 512*. Tech. rep.

Tomb, J.-F. et al. (1997). "The complete genome sequence of the gastric pathogen *Helicobacter pylori*". *Nature* 388 (6642), pp. 539–547.

Tria, F. D. K., G. Landan, and T. Dagan (2017). "Phylogenetic rooting using minimal ancestor deviation". *Nature Ecology & Evolution* 1 (1), p. 0193.

Ukkonen, E. (1995). "On-line construction of suffix trees". *Algorithmica* 14.3, pp. 249–260.

Venter, J. C. et al. (2001). "The Sequence of the Human Genome". *Science* 291.5507, pp. 1304–1351.

Vinga, S. and J. Almeida (2002). "Alignment-free sequence comparison—a review". *Bioinformatics* 19.4, pp. 512–523.

Wakeley, J. (2009). *Coalescent Theory. An Introduction*. Greenwood Village, Colorade: Roberts & Company Publishers.

Wang, L. and T. Jiang (1994). "On the complexity of multiple sequence alignment". *Journal of Computational Biology* 1.4, pp. 337–348.

Watterson, G. (1975). "On the number of segregating sites in genetical models without recombination". *Theoretical Population Biology* 7.2, pp. 256–276.

Weiner, P. (1973). "Linear pattern matching algorithms". *14th Annual Symposium on Switching and Automata Theory (swat 1973)*, pp. 1–11.

Wetterstrand, K. (2020). *DNA Sequencing Costs: Data from the NHGRI Genome Sequencing Program (GSP)*. URL: `https://www.genome.gov/sequencingcostsdata`.

Whidden, C., R. G. Beiko, and N. Zeh (2013). "Fixed-Parameter Algorithms for Maximum Agreement Forests". *SIAM Journal on Computing* 42.4, pp. 1431–1466.

Woese, C. R. and G. E. Fox (1977). "Phylogenetic structure of the prokaryotic domain: The primary kingdoms". *Proceedings of the National Academy of Sciences* 74.11, pp. 5088–5090.

Wu, T. D. (2016). "Bitpacking techniques for indexing genomes: II. Enhanced suffix arrays". *Algorithms for Molecular Biology* 11.1.

Yang, Z. and B. Rannala (2012). "Molecular phylogenetics: Principles and practice". *Nature Reviews Genetics* 13, pp. 303–14.

Yates, A. et al. (2016). "Ensembl 2016". *Nucleic Acids Research* 44.D1, pp. D710–D716.

Yi, H. and L. Jin (2013). "Co-phylog: an assembly-free phylogenomic approach for closely related organisms". *Nucleic Acids Research* 41.7.

Zhao, X. (2018). "BinDash, software for fast genome distance estimation on a typical personal laptop". *Bioinformatics* 35.4, pp. 671–673.

Zielezinski, A., S. Vinga, J. Almeida, and W. M. Karlowski (2017). "Alignment-free sequence comparison: benefits, applications, and tools". *Genome Biology* 18.1, p. 186.

Zielezinski, A. et al. (2019). "Benchmarking of alignment-free sequence comparison methods". *Genome Biology* 20 (1).

Zuckerkandl, E. and L. Pauling (1965). "Evolutionary Divergence and Convergence in Proteins". *Evolving Genes and Proteins*. Academic Press, pp. 97–166.

Zuo, G., Q. Li, and B. Hao (2014). "On K-peptide length in composition vector phylogeny of prokaryotes". *Computational Biology and Chemistry* 53. Complexity in Genomes, pp. 166–173.