Computing informative k-mers for phylogenetic placement

Nikolai Romashchenko

THÈSE POUR OBTENIR LE GRADE DE DOCTEUR DE L'UNIVERSITÉ DE MONTPELLIER

En Informatique

École doctorale I2S - Information, Structures, Systèmes

Unité de recherche UMR 5506 – LIRMM –
Laboratoire d'Informatique, de Robotique et de Micro-électronique de Montpellier

Calcul de k-mers informatifs pour le placement phylogénétique

Présentée par Nikolai ROMASHCHENKO
Le 14 décembre 2021

Sous la direction de Eric RIVALS

Devant le jury composé de

Mme Alessandra CARBONE, Professeur, Sorbonne Université	Rapportrice
M. Burkhard MORGENSTERN, Professeur, Université de Göttingen	Rapporteur
M. Nick GOLDMAN, Directeur de recherche, EMBL Hinxton	Président du jury
M. François COSTE, Chargé de recherche, INRIA Rennes	Examinateur
M. Frédéric MAHÉ, Chargé de recherche, CIRAD Montpellier	Examinateur
M. Fabio PARDI, Chargé de recherche, CNRS LIRMM, Université de Montpellier	Co-encadrant de these
M. Benjamin LINARD, Chargé de recherche, Spygen	Co-encadrant de these
M. Eric RIVALS, Directeur de recherche, CNRS LIRMM, Université de Montpellier	Directeur de these

UNIVERSITÉ DE MONTPELLIER

THESIS TO OBTAIN THE DEGREE OF DOCTOR OF THE UNIVERSITY OF MONTPELLIER

In Computer Science

Doctoral School I2S

Research Unit UMR 5506 - LIRMM

Computing informative k-mers for phylogenetic placement

Presented by Nikolai ROMASHCHENKO

December 14, 2021

Thesis Supervisor: Eric RIVALS

Devant le jury composé de

Alessandra CARBONE, Professor, Sorbonne University	Reporter
Burkhard MORGENSTERN, Professor, University of Göttingen	Reporter
Nick GOLDMAN, Research director, EMBL Hinxton	Jury president
François COSTE, Researcher, INRIA Rennes	Examiner
Frédéric MAHÉ, Researcher, CIRAD Montpellier	Examiner
Fabio PARDI, Researcher, CNRS LIRMM, University of Montpellier	Co-supervisor
Benjamin LINARD, Researcher, Spygen	Co-supervisor
Eric RIVALS, Research director, CNRS LIRMM, University of Montpellier	Supervisor

UNIVERSITÉ DE MONTPELLIER

Résumé

Étant donné un arbre d'évolution des espèces (ou phylogénie) et les séquences de référence qui ont permis de la construire, le placement phylogénétique tente de déterminer la branche d'origine d'une séquence requête dans la phylogénie. L'application principale du placement phylogénétique est l'identification d'espèces, une question essentielle de bioinformatique utilisée en écologie, en agronomie et en médecine. Les méthodes algorithmiques dites « sans alignement » proposent une nouvelle approche capable d'éviter d'aligner la séquence requête avec les séquences de référence, une étape qui limite fortement le passage à l'échelle du placement phylogénétique à l'ère des technologies de séquençage à haut-débit (SHD).

RAPPAS, qui appartient aux méthodes sans alignement, introduit le concept de k-mer informé phylogénétiquement, ou phylo-k-mer pour faire court. Pour un entier k, il s'agit d'une séquence de longueur k (ou k-mer) associée à des probabilités d'observation sur les branches de la phylogénie. Pour un k-mer issue d'une séquence requête, cela permet d'estimer la probabilité qu'il provienne de chaque branche de la phylogénie. RAPPAS pré-traite la phylogénie et les séquences associées pour calculer les phylo-k-mers et les stocker dans un index. Une fois calculé, cet index permet de placer d'énorme quantité de séquences requêtes ; cependant sa construction demeure coûteuse en temps de calcul et en mémoire.

Cette thèse étudie le calcul et l'indexation efficaces des phylo-k-mers. Le chapitre 1 introduit la thématique après un survol historique des notions de biologie et de bioinformatique nécessaires à sa compréhension. Il discute de l'importance de l'identification d'espèces par séquençage et bionformatique, ainsi que du défi causé par le SHD. Enfin, il présente un état de l'art bioinformatique du placement phylogénétique.

Le chapitre 2 décrit et analyse l'algorithme existant de l'étape centrale du calcul des phylo-k-mers : le calcul des phylo-k-mers pour une fenêtre de longueur k de l'alignement de référence. En outre, il propose un nouvel algorithme utilisant une stratégie de « diviser pour régner ». Cette approche surpasse l'algorithme existant tant en théorie qu'en pratique.

Cependant le volume mémoire occupé par l'index de phylo-k-mers et leur nombre peuvent dans certains cas s'avérer gênants. Le chapitre 3 propose de sélectionner les phylo-k-mers les plus informatifs en se basant sur l'information mutuelle. L'algorithme de filtrage proposé permet de réduire considérablement l'espace nécessaire en impactant la précision du placement de façon négligeable. Enfin, il examine la connexion entre RAPPAS et les méthodes d'apprentissage automatique de classification de textes basées sur une approche dite « bayésienne naïve ».

Le chapitre 4 décrit deux nouveaux programmes permettant le calcul et l'utilisation des phylo-k-mers : *XPAS* pour le calcul efficace d'index de phylo-k-mers et son stockage sur disque, et *RAPPAS2* qui réimplante l'algorithme de placement phylogénétique original de RAPPAS. Les résultats expérimentaux démontrent que ces deux programmes, qui combinés remplacent le RAPPAS original, réduisent grandement l'espace mémoire utilisée et améliorent fortement les temps de calcul. Leur efficacité provient de l'implantation en C++ moderne, de leur optimisation, et en fait de programmes d'ores et déjà utilisables.

Finalement, le chapitre de conclusion aborde des pistes de recherche pour l'utilisation des phylo-k-mers, les défis à venir, et les perspectives du placement phylogénétique.

Abstract

Phylogenetic placement determines possible phylogenetic origins of unknown query DNA or protein sequences, given a fixed reference phylogeny. Its main application is species identification, an essential bioinformatics problem with environmental ecology applications, microbial diversity studies, and medicine. Alignment-free methods for phylogenetic placement are a novel group of methods designed to eliminate the need to align query sequences within reference sequences — a current limit to the applicability of phylogenetic placement methods in the next-generation sequencing (NGS) era.

One of such methods is RAPPAS. It introduced the concept of phylogenetically aware k-mers (phylo-k-mers): k-mers paired with relevant probabilistic information about the reference phylogeny. This information determines how probable it is to observe any k-mer in hypothetical sequences arising from different parts of the reference tree. RAPPAS preprocesses the reference phylogenetic tree and alignment, computing phylo-k-mers. This allows fast phylogenetic placement of vast amounts of query sequences; however, the computation of phylo-k-mers is expensive in both running time and memory.

This thesis studies the problem of effective indexing of reference phylogenies with phylo-k-mers. Chapter 1 gently introduces the reader to the problem. Starting with a historical overview of biology and bioinformatics of the last decades, it discusses the importance of sequence identification in modern bioinformatics, overwhelmed with amounts of sequencing data produced by NGS technologies. Then, it overviews existing methods of phylogenetic placement and discusses their limitations.

Chapter 2 describes and analyzes the existing solution for the central algorithmic problem of phylo-k-mer computation: computing phylo-k-mers for one node in a k-sized window of the reference alignment. In addition, it describes a novel algorithm for this problem based on the divide-and-conquer approach. This algorithm improves the existing solution both theoretically and in practice.

Chapter 3 proposes a novel method of filtering phylo-k-mers based on Mutual Information. This method allows reducing memory consumption of phylogenetic placement significantly with a negligible decrease in placement accuracy. It also describes how RAPPAS is connected to well-studied methods of text classification with Naive Bayes.

Finally, Chapter 4 presents two new phylo-k-mer-related tools: XPAS for efficient computation of phylo-k-mers and RAPPAS2, an effective reimplementation of RAPPAS. Experimental results provided show that XPAS and RAPPAS2 outperform RAPPAS both in running speed and memory consumption. Both tools are written

in modern C++, optimized for efficiency, and are ready to use.

The final chapter discusses possible directions of future work on phylo-k-mer-related methods, the challenges that are yet to be overcome, and a discussion on the future of phylogenetic placement.

Acknowledgments

I would like to thank Eric Rivals for his scientific guidance and support in my work. I am sure that this work would not have been possible without Eric and his guidance in all matters related to the scientific career. I was often carried away by not-so-important questions during the thesis, and Eric always courteously directed my attention to the most important ones. Without his experience in supervising, I would have been lost in a vast sea of unresolved scientific questions, and I would hardly have gotten very far in my work. I am grateful to Fabio Pardi for his scientific guidance and genuine interest in my work. I have to admit that most of the results I obtained were an attempt to impress him. I thank him for his ability to be passionate about the questions I come to his office with and for his substantial contribution to the editing of the manuscript. I want to thank Benjamin Linard for his guidance, the dozens of hours he spent in discussions with us, his valuable comments on the manuscript, without which the manuscript would have been much less interesting to read. And, of course, I thank him for all the time we spent outside the lab.

I would like to thank Alessandra Carbone and Burkhard Morgenstern for kindly agreeing to review my thesis and attend to my defense. Special thanks to Burkhard for his interest in our work and for using PEWO for his publication. I would also like to thank the other members of the jury: Nick Goldman, François Coste, and Frédéric Mahé. Special thanks to Frédéric for participating in the individual committee and for his comments on my work.

I wish to thank Bastien Cazaux for his valuable comments about the methods I was working on and his endless cheerfulness. Thanks to Fati Chen just for his existence, to Guillaume Scholz and Julie Ripoll for stoically sharing the office with me all this time, and to all the other colleagues from MAB with whom we spent countless coffee breaks together. And, of course, thanks to the University of Montpellier, without which this work would not have happened, and for the dozens of hours of courses without which I would hardly have spoken any French. Thanks to LIRMM for hosting us and to the team of the IFB cluster for hundreds of hours of computations they made possible.

I am grateful to Anna Shlyaeva, who many years ago inspired me to take up science. I hope that if she ever reads this work, she will be pleased to know that working with her was the turning point when I decided to switch to science. Thanks to Vsevolod Krishchenko for inspiring lectures; unfortunately, this gratitude is many years late, and he can no longer read it. Also, thanks to Ilya Korvigo for his strong passion for science: without having worked with him, I would not even have been able to start this project. Unfortunately, he is no longer with us too.

I am thankful to my family for their support. I can hardly imagine how it would be possible if I were alone. Thanks to Lucie Berquiere, without whom, it would have been challenging for me to settle down in France. Thanks to Vladimir Pimonov for the countless hours we spent together during this time, and thanks to everyone else I forgot to thank.

Contents

1 Introduction **13**

1.1 Background . . . 13
1.1.1 Evolution and DNA . . . 13
1.1.2 Genomics and early sequencing technologies . . . 14
1.1.3 Taxonomy and phylogenetics . . . 15
1.1.4 Diversity of microorganisms . . . 19
1.1.5 The high-throughput sequencing revolution . . . 19
1.2 Approaches to eDNA sequence identification . . . 21
1.2.1 Sequence clustering-based . . . 21
1.2.2 Sequence alignment-based . . . 22
1.2.3 Phylogenetic inference . . . 23
1.2.4 Phylogenetic placement . . . 23
1.3 State-of-the-art of phylogenetic placement . . . 24
1.3.1 Alignment-based methods . . . 24
1.3.2 Alignment-free methods . . . 28
1.3.3 Wrappers using other phylogenetic placement methods . . . 30
1.3.4 Related tools . . . 31
1.3.5 Summary . . . 31
1.4 Evaluating phylogenetic placement . . . 32
1.4.1 Absolute placement accuracy . . . 33
1.4.2 Relative placement accuracy . . . 33
1.4.3 Pruning-based accuracy evaluation . . . 33
1.5 Phylo-k-mers . . . 34
1.5.1 Intuition . . . 34
1.5.2 Computation of phylo-k-mers . . . 35
1.6 RAPPAS: alignment-free phylogenetic placement using phylo-k-mers . . . 38
1.7 Motivation and thesis structure . . . 38
1.8 Notation . . . 39
1.8.1 Notation for strings . . . 39
1.8.2 Notation for matrices . . . 39

2 Algorithms for computing phylo-k-mers **41**

2.1 Computational model . . . 41
2.1.1 Word-RAM assumptions . . . 41
2.1.2 Assumptions about the alphabet . . . 42

2.1.3 Representation of k-mers . . . 42
2.2 Exhaustive Phylo-k-mer Computation . . . 43
2.2.1 Naive algorithm . . . 44
2.2.2 Window-wise and k-mer-wise approaches . . . 45
2.2.3 Common Prefix Algorithm . . . 45
2.2.4 Divide-and-Conquer . . . 46
2.3 Threshold-based Phylo-k-mer Computation . . . 48
2.3.1 Problem statement . . . 48
2.3.2 Branch-and-bound . . . 49
2.3.3 Divide-and-conquer for threshold-based computation . . . 52
2.3.4 Chained Windows Technique . . . 54
2.4 Experimental results . . . 55
2.4.1 Experimental setup . . . 55
2.4.2 Running time . . . 56
2.5 Conclusion . . . 58

3 Informative phylo-k-mers **59**
3.1 Phylogenetic placement as classification . . . 60
3.2 Text Classification with Naive Bayes . . . 61
3.2.1 Bernoulli model . . . 62
3.2.2 Multinomial model . . . 62
3.3 RAPPAS as a Naive Bayesian Classifier . . . 63
3.3.1 Query representation . . . 64
3.3.2 Model parameters . . . 64
3.3.3 Classification . . . 65
3.4 Connecting RAPPAS and Bernoulli Naive Bayes . . . 66
3.4.1 Bernoulli-based phylogenetic placement . . . 66
3.4.2 Query length correction for RAPPAS . . . 67
3.4.3 The link between RAPPAS and Bernoulli phylogenetic placement 67
3.5 Feature selection . . . 69
3.5.1 Dealing with high-dimensional feature spaces . . . 69
3.5.2 Approaches to feature selection . . . 70
3.5.3 Feature selection using Mutual Information . . . 71
3.6 Mutual Information filters for RAPPAS . . . 72
3.6.1 Deriving the Mutual Information Filter . . . 72
3.6.2 Making assumptions about f . . . 74
3.6.3 Filter-based selection of phylo-k-mers . . . 76
3.6.4 Random filter . . . 77
3.7 Experimental evaluation . . . 78
3.7.1 Likelihood-based Accuracy . . . 78
3.7.2 Datasets . . . 79
3.8 Experimental results & discussion . . . 81
3.8.1 Filter performance: LAC experiments . . . 81
3.8.2 Filter performance: PAC experiments . . . 83
3.8.3 Dataset complexity and parameter choice . . . 86

3.9 Conclusion 87

4 XPAS and RAPPAS2 **89**
4.1 XPAS: a standalone tool for creating phylo-k-mer databases 89
4.1.1 Creating a phylo-k-mer database 90
4.1.2 The XPAS Core Library 94
4.2 RAPPAS2: a faster reimplementation of RAPPAS 97
4.3 The phylo-k-mer database computation bug in early versions RAPPAS 97
4.4 Experimental results 98
4.4.1 Time and memory requirements 98
4.4.2 Placement accuracy 102
4.5 Conclusion 105

5 Conclusion and perspectives **107**
5.1 Future work 108
5.1.1 The optimal algorithm for phylo-k-mer computation 108
5.1.2 Alternative score model 108
5.1.3 Bernoulli-based phylogenetic placement 110
5.1.4 Parallelism and distributed memory 111
5.2 Final thoughts 112

A Tables and proofs **115**
A.1 Proof of Lemma 3.6.2 115
A.2 Tables 117

B Figures **119**

Chapter 1

Introduction

1.1 Background

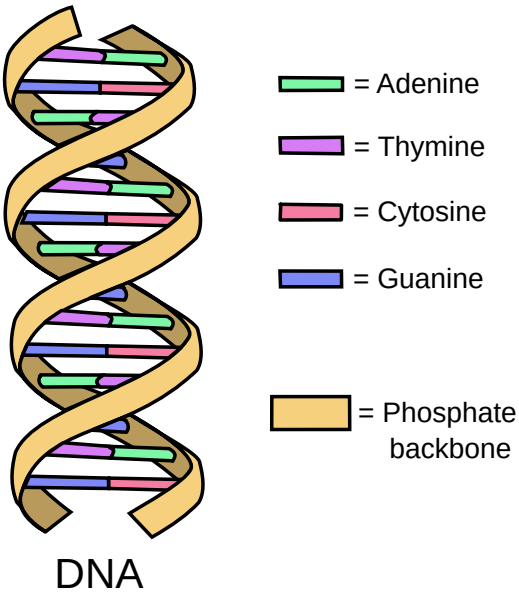
1.1.1 Evolution and DNA

> *It has not escaped our notice that the specific pairing we have postulated immediately suggests a possible copying mechanism for the genetic material.*
> — J. Watson, F. Crick [220]

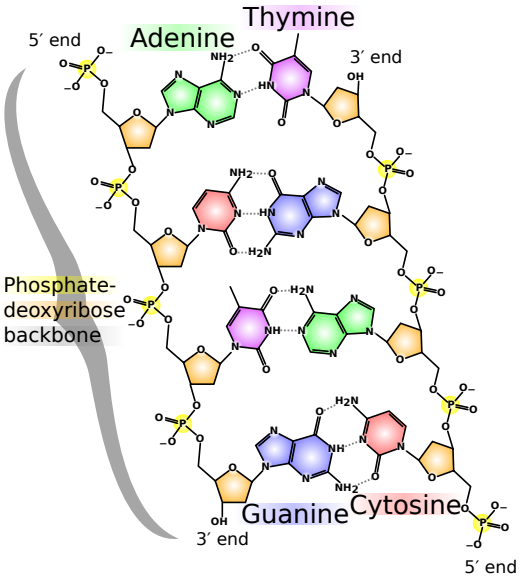
From the very beginning of life on Earth, all living organisms are subject to *natural selection* [46]. It is dictated by the environment in which they find themselves: the most adapted organisms have a greater chance of surviving and reproducing. As they reproduce, they pass on their properties to their offspring in a process called inheritance. Thus, the constantly changing environment favors the reproduction of those organisms that carry characteristics conducive to survival. The endless selection of such properties and the adaptation of living organisms to the changing environment is called *evolution* [49]. The great diversity of all possible organisms, from bacteria, archaea, and viruses to fungi, plants, and animals, although exceptional in scale and complexity, is based on this fundamental principle.

The inheritance of the properties of living organisms is possible thanks to molecular mechanisms of storing information. The prevalent carrier of hereditary information is DNA, a molecule of deoxyribonucleic acid. This molecule consists of two strands connected by hydrogen bonds, with each strand consisting of two parts: the sugar-phosphate backbone and the nucleic bases — namely adenine, cytosine, guanine, thymine — attached to the backbone. Figure 1-1a illustrates the structure of DNA. In most cases, nucleic bases in both strands are arranged in pairs: adenine in one chain is paired with thymine in the other, and cytosine with guanine (see Figure 1-1b). Such pairs are called *base pairs*.

In a single cell, DNA can reach millions of base pairs in length and describes the



(a) A schematic representation of the structure of DNA. Nucleic bases attached to the phosphate backbone form a double-stranded helix. Source and license: [71].



(b) The chemical structure of DNA: nucleic bases attached to the backbone form hydrogene bounds. Source and license: [13].

Figure 1-1 – The structure of DNA.

"recipes" for constructing other biological sequences, namely proteins. It is proteins that are the functional molecules responsible for various sorts of functions in living organisms [136]. Those sections of DNA used by organisms to encode proteins are called *genes*, but they often represent only a tiny fraction of the DNA sequence. One organism's entire hereditary information is called its *genome*, and the discipline that studies genomes is called *genomics*. The importance of the discovery of the hereditary properties of DNA ([9] which was inspired by [79] and confirmed later by [87]), and the structure of the DNA molecule ([220] followed by [221]) can hardly be overestimated: these outstanding achievements determined the further development of biology.

1.1.2 Genomics and early sequencing technologies

The detailed study of the genes of living organisms was made possible by the invention of sequencing technology. Sequencing is the process of retrieving the sequence of base pairs of any DNA strand. The first sequencing method was invented by Allan Maxam and Walter Gilbert [147, 148] in 1977, and another one was suggested by Frederick Sanger [182, 183] in the same year. Sanger sequencing, also called the chain-termination method, proved more successful and had a huge influence over the next 30 years in biology. Initially requiring manual work, this method was eventually automated [197], which led to the emergence of laboratories equipped with a large number of DNA sequencing machines, operated by a large number of personnel in parallel [190]. It allowed sequencing genes and eventually whole genomes to study their structure, function, and evolution. Thus, DNA sequencing gave rise to genomics:

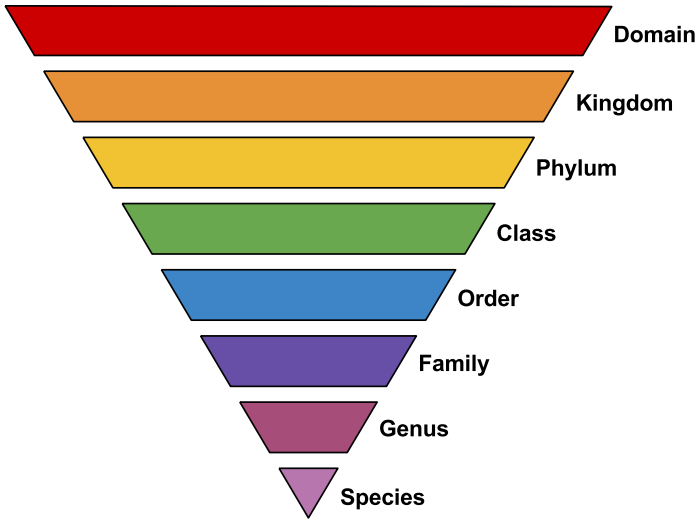


Figure 1-2 – The taxonomic ranks used in modern taxonomy, from general to specific: domain, kingdom, phylum, class, order, family, genus, species. Source and license: [171], which is derived from [27].

the first bacterial genome was published in 1995 [70], followed by the first eukaryotic genome [77], the genome of the nematode *Caenorhabditis elegans* [58], the genome of *Drosophila melanogaster* [3], and the efforts of many scientists completed the largest collaborative genomic project at the time, the Human Genome Project, in 2003 [121, 213, 187, 160]. This is by no means a complete list of the genomes published back then; many others followed, and today sequencing a genome of a living or extinct creature is commonplace.

Sequencing technology led to new disciplines, new scientific questions, and a re-thinking of existing ones. To understand the importance of those changes, let us take a step back and take a historical look at the questions raised in biology before.

1.1.3 Taxonomy and phylogenetics

One of the oldest and most essential biological tasks had been and remained the task of classifying organisms and describing their relationships. In the 18th century, Carl Linnaeus proposed the *binomial nomenclature* [31], naming species of living things based on two Latin words: the generic name, identifying the genus to which the species belongs, and the specific name, which distinguishes the species within the genus[1]. His works greatly advanced the field of *taxonomy* dealing with classifying groups of biological organisms based on shared characteristics. He introduced the standard of *class*, *order*, *genus*, and *species*, a system for ranking living organisms [215, 217, 216]. Such a system, albeit somewhat modified, is still in use today (see Figure 1-2 for an illustration of the modern taxonomic hierarchy of living organisms).

Initially, taxonomy grouped organisms based mostly on similarities and dissimi-

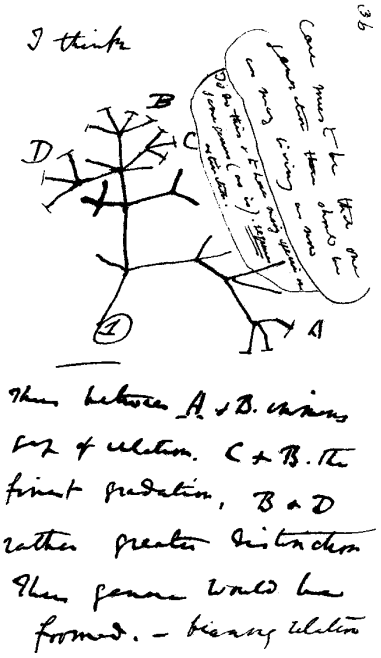
[1] According to this nomenclature, those able to read and understand this text most probably refer to *Homo sapiens*.

larities of their phenotypical traits. This, however, did not reflect on the evolutionary relationships of those organisms; thus, taxonomic trees did not always correctly represent the ancestral relationships of underlying organisms.

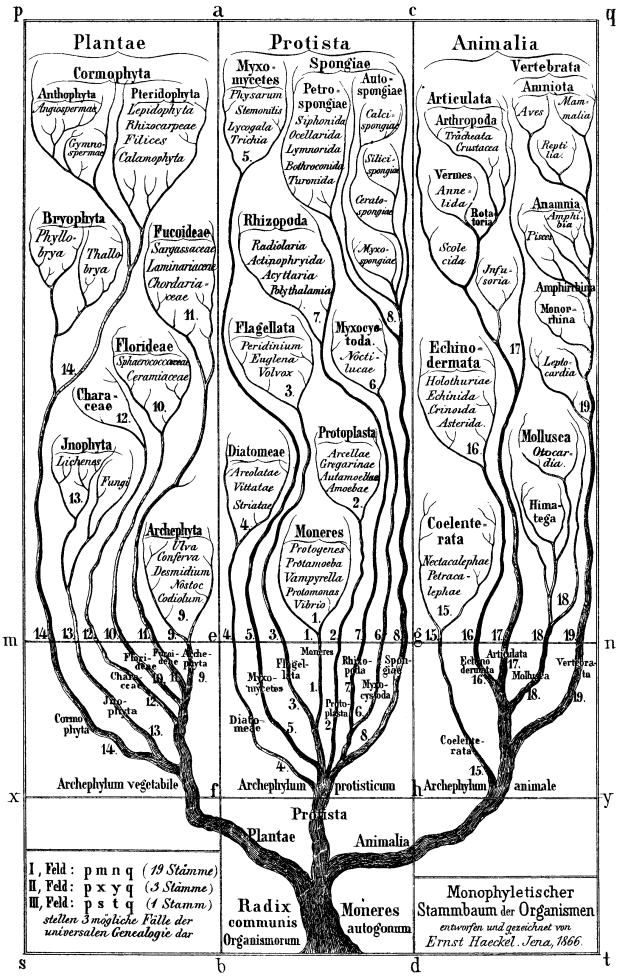
Another major part of biological systematics addressed this problem, that is, the field of *phylogenetics*. While also classifying living creatures, phylogenetics specifically addresses the inference of the evolutionary history among or within groups of organisms. The basic model of phylogenetics is the *phylogenetic tree*, a tree-like diagram representing our understanding of the evolution of the organisms under study. The idea of describing evolutionary relationships with the help of such trees appeared in the 19th century and referred back to Darwin, who popularized the so-called "evolutionary tree". Some early historical examples of such trees were published in works of Darwin and Haeckel (see Figure 1-3).

In modern phylogenetics, trees are usually less artsy than those suggested by Haeckel (Figure 1-3b). One example of a modern phylogenetic tree of life is presented in Figure 1-4. In a phylogenetic tree, the tips, or the *leaves* of the tree represent existing organisms or groups of organisms of different taxonomic levels. The nodes in the tree represent evolutionary events of the past that lead to division of the common ancestor of the node's subtree into different lineages. Phylogenetic trees can be unrooted if the common ancestor of the whole tree can not be identified; otherwise, the root represents such an ancestor. Edges of the tree are called *branches*, and the branch length between two nodes is meant to be proportional to the evolutionary distance between the organisms represented by the nodes connected by this branch. Those evolutionary distances and the topology of the tree can be inferred from observable traits of organisms represented in the tree. In the early days of phylogenetics, phenotypic traits were used as sources of phylogenetic information, but as early as the 1960s, the idea of using molecular sequences appeared [238].

Even before the invention of DNA sequencing, there was increasing evidence that molecular sequences contained reliable phylogenetic signals and should be used to infer phylogenies. Much attention was paid to the idea of using proteins as sources of phylogenetic information [69, 68, 238]; however, protein sequences were scarce. A fundamental breakthrough happened in 1977, although many contemporaries did not notice it. Woese and Fox argued that we should reconstruct phylogenetic relationships in terms of comparable property, common between various organisms yet related to evolutionary history. They suggested comparing microorganisms by short genes coding for ribosomal RNA, 16S rRNA gene for prokaryotes, and 18S rRNA for eukaryotes [226]. Such a comparison immediately suggested the existence of a new kingdom of prokaryotic organisms, archaebacteria (now known as one of three domains of life, *Archaea*, along with *Bacteria* and *Eukarya*). Those findings were yet to be confirmed later [228, 227]. The suggested genes, however, became the standard of phylogenetic markers for decades and are still used nowadays: those genes are highly conserved within living organisms of the same genus and species, but they differ enough between organisms of other genera and species [230].



(a) The first diagram of an evolutionary tree made by Charles Darwin in 1837. Handwriting: "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. [between] C + B the finest gradation. [between] B+D rather greater distinction..." Source and license: [45]. Transcription: [16].



(b) Reproduction of the "genealogical oak" suggested by Haeckel in 1866 [83]. Represents three kingdoms: Plantae (plants), Protista (microorganisms) and Animalia (animals). Source and license: [84].

Figure 1-3 – Early examples of phylogenetic trees.

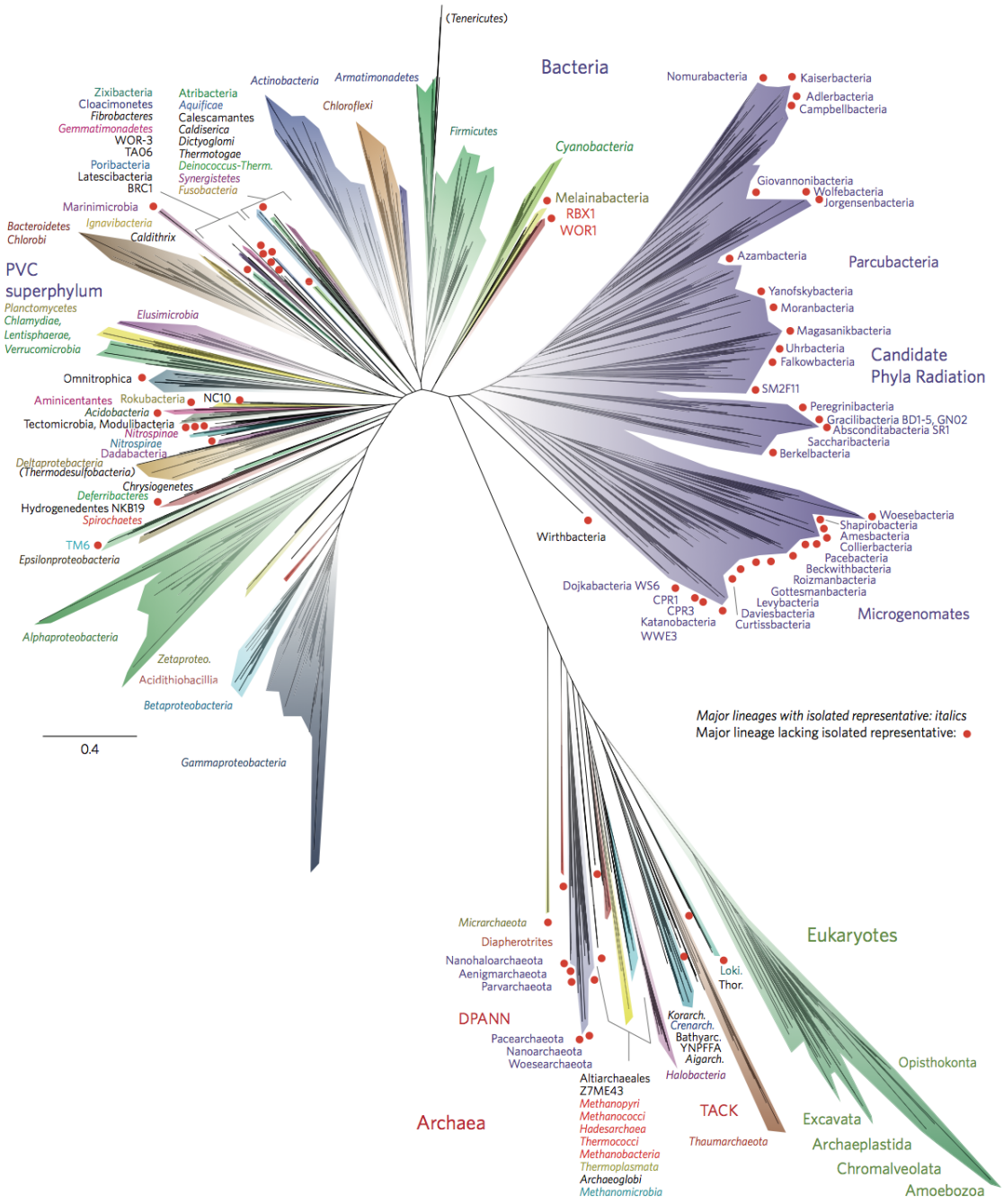


Figure 1-4 – A modern phylogenetic tree of life constructed using sequencing data. Source and license: [92], which was derived from [91].

1.1.4 Diversity of microorganisms

Sequencing technologies and new phylogenetic data brought reinforced suspicions that our understanding of the microcosm was far from complete. In the study of microbes, a common approach was cultivation: microorganisms were extracted from the environment and grown in the laboratory for further research (e.g., sequencing their DNA sequences). Unfortunately, cultivation suffers from a major problem. As early as 1932, the Russian microbiologist Razumov described that the number of bacteria visible under a microscope is much greater than the number cultured under laboratory conditions [167, 135]. It has been confirmed by many researches later: the number of microorganisms that cannot be cultured in the laboratory is estimated to be $> 99\%$ of their total number [7, 175]. Staley and Konopka coined the term "The Great Plate Count Anomaly" for this effect [201], and resolving this "anomaly" became an essential issue in microbiology. Thus emerged the idea of studying microorganisms *in situ*, i.e., extracting their DNA directly from the environment in which they live [28, 93]. Studying such environmental DNA (or eDNA) is now one of the primary ways to study diversity of microorganisms [230]. Examples of eDNA sources include, but are not limited to, feces, mucus, gametes, shed skin, hair, water, soil, air, and others [205, 24, 204, 211, 72, 99, 208].

Thus, microbiology entered the 21st century with the understanding that microorganisms are the primary source of diversity of life, the vast majority of which are unknown [94] and cannot be studied outside their habitat. Amann et al. described this situation as nothing short of a "failure of microbiologists to describe natural diversity" [7]. Against the background of the new successes of eukaryotic genomics of that time, the task of determining the genomes of all microorganisms remained untractable without much hope of solving it any time soon [93].

1.1.5 The high-throughput sequencing revolution

The first signs of an approaching breakthrough in sequencing technology emerged in 2005 with the invention of sequencing-by-synthesis (pyrosequencing) technology by 454 Life Sciences (later acquired by Roche) [143] and Multiplex Polony Sequencing [196]. However, although sequencing throughput achieved an approximately 100-fold increase over Sanger sequencing, it took time for the scientific community to bring these technologies into widespread use [190]. The DNA reads produced by the 454 machines were much shorter (at first 50-150bp [143], later this number reached 400bp, versus about 750bp for Sanger sequencing), and there was a need to handle the large volume of data generated using the new technology. Nevertheless, new technologies kept emerging in the sequencing market — Solexa (became part of Illumina) [18, 19], SOLiD [212], semiconductor chip based sequencing (Life Technologies, Ion Torrent) [178] — technologies called *Next Generation Sequencing* (NGS) as opposed to Sanger sequencing (sometimes called first-generation sequencing). These technologies produce short reads (no more than a few hundred base pairs) of high quality, e.g., with a low percentage of incorrectly sequenced base pairs. Some time later, long-read sequencing technologies such as single-molecule real-time sequencing (Pa-

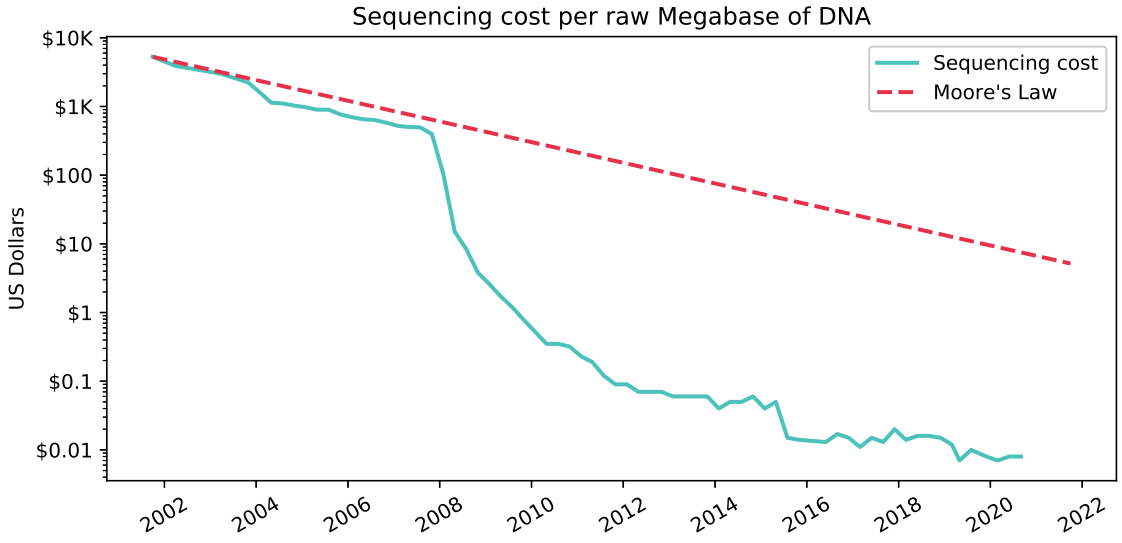


Figure 1-5 – The cost of sequencing per raw megabase of DNA against the hypothetical cost reflecting the Moore's Law ([172] based on [224]). The cost includes labor, administration, management, utilities, reagents, and consumables expenses, as well as amortized costs of sequencing instruments, informatics activities directly related to sequence production, and other indirect costs. For details, see [224].

cific Biosciences) [57, 35] and nanopore sequencing [50] also appeared; they produce much longer sequences at the cost of higher error rates. Today, the DNA sequencing market is dominated by Illumina, which successively replaced Sanger sequencing and won the race against the competitors [193].

Sequencing technologies led to a significant reduction in sequencing costs and a substantial increase in the amount of data obtained. For example, sequencing the first human genome as a haploid reference took nearly ten years, while now, an entire diploid human genome sequence can be accomplished in just a few days [150]. Figure 1-5 demonstrates the drop in sequencing costs over the past decades against Moore's Law (which states the number of transistors in integrated circuits doubles about every two years). Thus, a new era in bioinformatics has dawned: a time when the importance of data interpretation and the very possibility to process available data is critical. At the same time, the cost and time of obtaining data may be considered negligible [17].

NGS allowed answering new scientific questions. For example, it enabled *metagenomics*: sequencing all possible DNA from an environmental sample, including many different organisms (thus implementing the approach of sequencing random parts of DNA, also known as *shotgun sequencing* [152]). In contrast to the approach based on marker genes (e.g., 16S, 18S rRNA) discussed earlier, called *metabarcoding*, metagenomic experiments are not limited to sequencing a single gene and produce full-genome resolution DNA reads. The clinical applications of metagenomics include, but are not limited to, the detection of potential pathogens in a sample for diagnostics [36], detection of antibiotic related genes in human gut bacteria [53], investigation of

outbreaks [137], identification of SARS-CoV-2 infection [164], and prediction of novel emerging coronaviruses [33].

Nevertheless, there is still a need for targeted metagenomics, i.e., metabarcoding. First, metagenomics requires much higher sequencing power to acquire comparable sequencing depth. For example, it may not be possible to capture rare but ecologically important microbial species in large-scale metagenomic experiments [138], which is possible for metabarcoding [195, 104]. Second, metabarcoding is indispensable for biodiversity studies in agriculture and ecology [177, 134]: studying soil and water microbial communities is still challenging due to the lack of genetic and phylogenetic data on underlying organisms. Thus, it makes accurate identification of metagenomic reads challenging since reference genomes of interest may not be known.

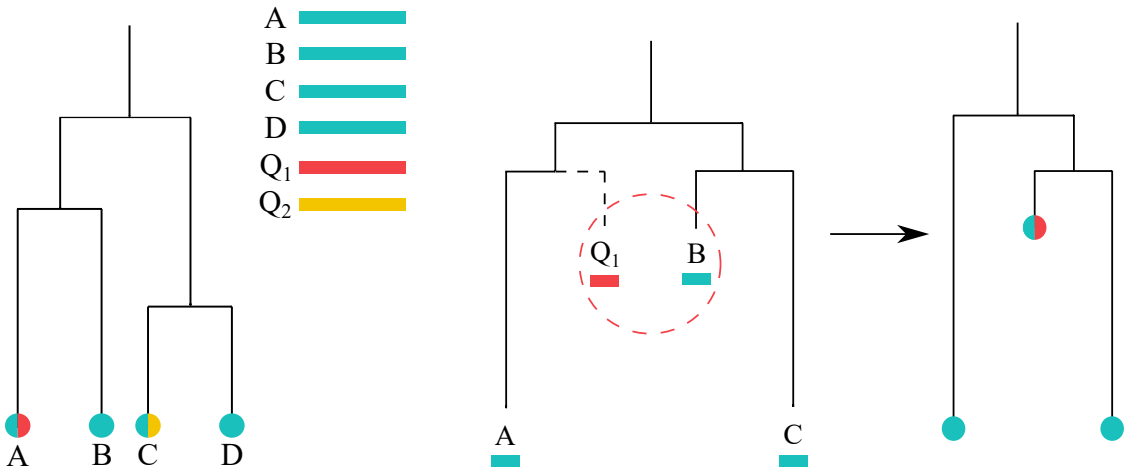
In light of the above, the identification of short eDNA sequences is an essential task of bioinformatics. In this work, I address this problem in the context of metabarcoding. The overview of existing methods addressing this problem will be given in Section 1.2.

1.2 Approaches to eDNA sequence identification

1.2.1 Sequence clustering-based

The study of microbial communities is burdened not only by the lack of genomic data but also by the lack of phylogenetic knowledge. Although much progress has been made in the last decade in determining the phylogeny of microorganisms of different environments [52, 165, 169, 234], it is often insufficient for accurate sequence identification at the genus and species level. In such a case, the sequences are used to cluster into *operational taxonomic units* (OTU), representing organisms of different taxonomic groups. OTUs thus are a "proxy" taxonomical concept used instead of classic taxonomic ranks (such as species, genus, etc.) due to the lack of exact phylogeny of the underlying organisms. Despite this obvious simplification, this approach is widely used in medical and ecological microbiology to analyze the biodiversity of microbial communities [78].

To form OTU clusters from a collection of *query* sequences, they are clustered by sequence similarity (standard thresholds of similarity are 95% and 97%). Popular sequence clustering software are CD-HIT [127, 126, 89, 73], DNACLUST [75], USEARCH [56] and VSEARCH [170], and Swarm [141, 142]. A consensus sequence is taken to represent every OTU. Thus, query sequences are identified with consensus sequences; those can then be used to infer the phylogeny. This is called *de novo* OTU picking, and this approach is used when the reference phylogeny is not available. Otherwise, if the reference phylogeny is given (based on a set of known *reference sequences*), query sequences can be clustered against reference sequences. The reference phylogeny remains unchanged, which is known as *closed-reference* OTU picking. It is substantially faster than *de novo* OTU picking but requires the phylogeny to be sufficiently relevant: query sequences not reaching the required similarity threshold can not be identified and are discarded. In addition, it produces less accurate re-



(a) A hypothetical assignment of query sequences Q_1, Q_2 to a known phylogeny based on reference sequences A, B, C, D. Colored circles represent tree nodes within which sequences are associated. In this example, Q_1 is clustered together with A, and Q_2 with C.

(b) An example of phylogenetically suboptimal clustering based on sequence similarity. Given different evolution rates between branches of the tree, query Q_1 is evolutionary closer to A, but is clustered together with B based on higher sequence similarity.

Figure 1-6 – Sequence identification based on similarity and clustering.

sults than *de novo* OTU picking [223]. Figure 1-6a illustrates closed reference-based identification. Finally, *open-reference* OTU picking is the combination of the two approaches, where closed-reference OTU picking is performed at the beginning, and unclustered sequences are clustered *de novo*.

The popular pipelines implementing those approaches are QIIME [32, 25] and MOTHUR [186, 185]. OTU picking-based assignment is scalable for millions of query sequences, especially for its closed-reference version. However, it lacks phylogenetic resolution for individual sequences: similarity-based sequencing clustering may produce suboptimal results from the evolutionary point of view. Consider Figure 1-6b: the lack of phylogenetic knowledge of underlying organisms may lead to query sequences being clustered together in ways misrepresenting the underlying evolution.

1.2.2 Sequence alignment-based

A similar method is based on alignment. In the process of evolution, DNA sequences mutate: insertions, deletions, and substitutions of base pairs occur. Sequence alignment is a fundamental task of bioinformatics aiming at comparing two or more DNA sequences to determine the degree of their difference in terms of mutations. An alignment algorithm introduces gaps in input sequences, maximizing the number of matching sites of those sequences assuming a substitution model. Such a process can be done for two (*pairwise alignment*) or more (*multiple alignment*) sequences at once.

Many algorithmic approaches have been developed for this task (e.g., [161, 198]);

a popular application is searching for query sequences in sequence databases. Tools like BLAST [6, 103] and VSEARCH [170] are used for this purpose. For the task of sequence identification, the closest sequence found in the database can be considered as the answer. This approach, however, also suffers from the problems we have discussed in Section 1.2.1. First, the lack of reference sequences in databases, which is the case for microbes, leads to the inability to identify queries of novel species. Second, such identification may be phylogenetically irrelevant. The following sections introduce phylogenetically aware methods allowing to overcome these problems.

1.2.3 Phylogenetic inference

Phylogenetic inference involves constructing a phylogenetic tree for a set of sequences. Tree inference is usually cast as an optimization problem under a certain model, with optimization criteria expressing the fit of the tree with the sequence data. These optimization problems have often been shown to be computationally hard. For all types of phylogenetic trees, as the number of taxa grows, the number of possible tree topologies grows super-exponentially [62, 64]. Many different approaches have been suggested. Distance-based methods aim at optimizing a criterion based on pairwise distances between sequences, such as least squares and minimum evolution [199]. Parsimony-based approaches seek to minimize a (discrete) measure of the number of evolutionary events implied by the tree [184, 61]. Maximum likelihood and Bayesian methods are based on probabilistic criteria [63, 90, 173]. Those interested in this challenging problem may be interested in [64] and [119].

Having a set of query sequences and a set of reference sequences, one can compute multiple alignment of all-against-all, assume or estimate an evolutionary model, and infer a phylogenetic tree *de novo*. In addition to sequence identification, this approach also allows to reconstruct evolutionary relationships between query sequences. However, not only phylogenetic inference but also multiple alignment is a hard problem [219, 59]. Unfortunately, it makes it impossible to scale it to the current amounts of query sequences produced by NGS technologies. Even inferring a tree of ten thousand query sequences is challenging, while current microbiome studies deal with millions of sequences. In addition, tree inference methods require longer sequences as the number of taxa increases [159], while the length of metabarcoding reads is limited. Thus, metabarcoding sequences may not contain enough phylogenetic signal to reconstruct the comprehensive phylogeny correctly [20].

1.2.4 Phylogenetic placement

Phylogenetic placement is a group of phylogenetically aware methods aiming at overcoming the problems of full phylogenetic inference discussed above. The input of phylogenetic placement is a reference tree, a set of reference sequences (called together the *reference dataset*), and a set of query sequences. The reference tree represents the evolution leading up to the reference sequences (in a one-to-one correspondence with the tree's leaves), and the reference sequences are often pre-aligned in a multiple sequence alignment given as input. The general concept of placement is that

query sequences are processed one by one independently and are *placed* into the fixed reference phylogeny. For every query, placement evaluates *how likely each branch is as the phylogenetic origin of the query*. The output of phylogenetic placement is the assignment, or *placement* of every query sequence: the list of branches with the highest probabilities for those branches to be the phylogenetic origin of the query. This informs about the possible positional uncertainty of query sequences in the reference tree, providing an informative per-query picture of the possible query's evolution [146]. When placing a query sequence on a branch, some methods also estimate the exact branching point within that branch, and the length of the new pendant branch.

Started as an attempt to overcome the scalability problems of phylogenetic inference, it evolved in a group of different methods. It scales better than phylogenetic inference of reference and query sequences *de novo*: phylogenetic placement can be easily parallelized since queries are placed independently. Moreover, there is evidence that phylogenetic placement of short sequences can be as good or even *more* accurate than *de novo* inference [100]. However, phylogenetic placement does not reconstruct evolutionary relationships between query sequences, only between every query and the set of reference sequences.

The work I present in this thesis started in the context of phylogenetic placement but has proven to have other applications in evolutionary bioinformatics. Phylogenetic placement of unknown query sequences to a known reference phylogeny will remain one of the central problems throughout this work. Section 1.3 gives a review of existing approaches in phylogenetic placement.

1.3 State-of-the-art of phylogenetic placement

Phylogenetic placement methods can be divided into two groups: *alignment-based* and *alignment-free*. Alignment-based methods require aligning query sequences with reference sequences. Alignment-free methods *do not* require it (but they still may require reference sequences to be aligned), and "alignment-free" should be understood in this sense.

1.3.1 Alignment-based methods

The initial requirement for any alignment-based method is to align queries with respect to the reference alignment. This can be done using NAST [51], MAFFT [106], HMMER [66, 55], or other alignment software. Phylogenetically aware alignment algorithms can also be used: PaPaRa showed to improve placement accuracy when used together with phylogenetic placement compared to phylogeny-agnostic tools [21]. Alignment is not considered a part of the placement algorithm but a prerequisite; nevertheless, this step can take considerable time if thousands or hundreds of thousands of queries are placed. This requirement makes alignment-based methods poorly scalable with today's amounts of sequences produced by NGS technologies, which are often in the hundreds of thousands and millions.

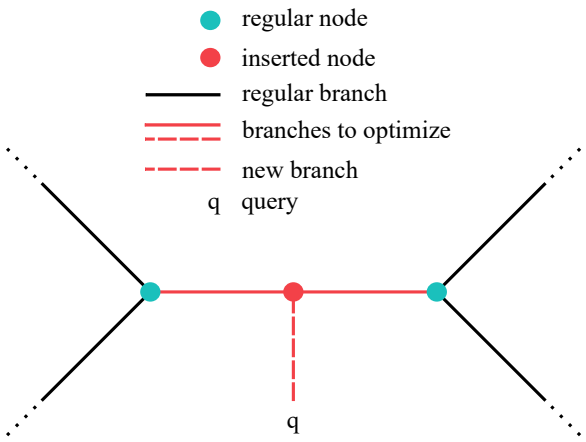


Figure 1-7 – Phylogenetic placement with EPA. For a query sequence q, a branch is split into two branches (shown in red), and a new branch is introduced (shown as dashed), leading to the leaf representing the query. Lengths of the red branches will be reoptimized, and the resulting tree evaluated to give the score for this placement.

Evolutionary Placement Algorithm (EPA)

One of the first phylogenetic placement approaches is the Evolutionary Placement Algorithm (EPA), implemented as a part of RAxML [20, 202, 203]. It belongs to the group of *maximum likelihood* approaches, which are used for inferring phylogenies. Thus, it assumes that the input phylogeny is inferred using maximum likelihood under a particular evolutionary model.

The algorithm iterates over query sequences and places them; I will describe how to place one query sequence. Let T denote the reference tree consisting of n taxa, and q denote the query sequence. For every branch b of T, the query q is placed to the branch b by creating a new node that splits b into two branches of smaller size. Also, a new branch is added from the newly inserted node to a new leaf representing the query. Thus, a new tree of $n + 1$ taxa is created (see Figure 1-7 for an illustration). Now we need to optimize the tree's branch lengths by maximising the tree's likelihood, given the multiple alignment and the evolutionary model used to infer the tree. We need to have the query aligned with the reference sequences at this stage, which makes this approach alignment-based. A naive algorithm would optimize the lengths of all branches; however, it is computationally expensive. Instead, only the lengths of the branches adjacent to the inserted node are optimized. This heuristic provides a good trade-off between speed and accuracy [20]. Then, the likelihood of this tree is calculated, and the calculated value would be the score of placing q to branch b.

EPA showed consistently better placement accuracy than the BLAST-based method (discussed in Section 1.2.2) while providing comparable placement speed. Methods of evaluating placement accuracy will be discussed later in Section 1.4.

PPLACER

Around the same time PPLACER, another program for phylogenetic placement based on maximum likelihood, was published [146]. Its principle is similar to EPA, but it uses a different heuristic to speed up the naive algorithm, called the *baseball heuristic*. The algorithm consists of two stages: The first stage selects the candidate branches expected to lead to placements with high likelihood (*preplacement*). The query q is placed on the center of a branch b in the same way as in Figure 1-7. Then, the likelihood of the resulting extended tree is computed. When the likelihoods for all branches are computed, the branches are sorted by likelihood value. A certain number of best branches (depending on the parameters of the algorithm) are chosen as candidates. In the second stage, for each candidate branch, the lengths of all branches of the tree are optimized (*thorough placement*), and the final likelihood of the placement is computed.

PPLACER's running time is linear in the number of query sequences, reference sequences (and therefore, the reference tree size), and sequence length. However, its memory consumption can be relatively high due to various optimizations designed to faster placement running time [146]. Accuracy-wise it shows comparable or better results than EPA depending on the dataset and algorithm's parameters.

EPA-NG: massively parallel EPA

EPA-NG [15] is the successor of EPA, which improves the previous version in every aspect. It implements four different strategies of placement: exhaustive thorough placement (called above the naive algorithm), the algorithm of EPA, the baseball heuristic (reproduces the results of PPLACER), and a new heuristic modifying the preplacement stage. For this new heuristic, the number of selected branches is dynamic and depends on placement uncertainty among candidate branches. Besides that, it implements the strategy of *masking*, i.e., filtering alignment sites that consist entirely of gaps either in the reference or in the query sequence and ignores leading and trailing gaps in the query.

EPA-NG is excellently optimized. If run in the sequential mode, it is up to 30 times faster than PPLACER and EPA. In addition to that, it runs in parallel in both shared (via OpenMP) and distributed memory (via MPI). Authors present results showing that the parallel efficiency of EPA-NG scales well up to 2048 cores (on 128 machines). This massive parallelization allowed them to place one billion short queries to a reference phylogeny of several thousand taxa in just seven hours. This is an impressive result for a likelihood-based method. Unfortunately, the authors did not report how long it took to *align* this number of queries within the reference alignment.

In light of the above, I see no reason to prefer PPLACER or EPA to this program. Given that EPA-NG can be used in EPA/PPLACER compatibility mode, it will likely replace them soon. However, the maximum likelihood approach can hardly be significantly optimized beyond this. The authors acknowledge that the alignment of queries within the reference alignment remains a significant bottleneck in placement analysis [15]. Another scalability issue of EPA-NG (as well as other maximum

likelihood-based tools) is memory consumption. It is challenging to use it to place on large phylogenies: for example, it might take dozens of gigabytes of RAM if used for trees of several thousand taxa [12].

LSHPlace

LSHPlace [29] implements an interesting approach based on locality-sensitive hashing. First, it determines slowly evolving (i.e., *conservative*) sites of the reference alignment. Then, it finds *ancestral sequences* for every node of the tree using maximum likelihood. Ancestral sequences are hypothetical sequences that most likely existed in ancestral organisms corresponding to nodes of the tree. Ancestral sequence reconstruction is directly relevant to my work and will be discussed in Section 1.5.2. Finally, it splits the alignment into overlapping regions, and for conservative regions of the alignment, it hashes the most probable ancestral sequences into collections of hash maps using locality-sensitive hashing.

For placing a query sequence q, it searches the query in many hashmaps using locality-sensitive search. For every hash map hit, the node corresponding to the ancestral sequence found is considered the local search algorithm's starting point. During this algorithm, neighbor branches are evaluated as possible phylogenetic origins of q. The evaluation of one branch involves estimating evolutionary distances between the ancestral sequences associated with the nodes adjacent to the branch and the query sequence.

This software is up to two orders of magnitude faster compared to pplacer. However, it is much less accurate. In addition to that, it is still alignment-based (despite that the alignment does not seem to be used during the placement stage): LSHPlace uses queries' positions in the alignment to make sure that multiple hash tables cover every region of the alignment where queries appear. I suspect that this requirement could have been bypassed applying methods based on *k-mers* (i.e., short sequences of fixed size k); however, LSHPlace does not seem to be used in practice. Unfortunately, neither the source code nor binaries are available online.

PhyClass

PhyClass [65] is a distance-based method that relies on *minimum evolution*, a well-known optimization criterion for phylogenetic inference [108]. This method uses a matrix of $n \times n$ pairwise distances precomputed between the n reference sequences. Moreover, for each query, PhyClass estimates the $n+1$ distances between the query and the reference sequences. The authors assume that all distances are computed in an alignment-based way. The placement phase relies on the following idea: to evaluate the goodness-of-fit of placing a query on a given branch, the criterion to minimize is the total branch length of the reference tree extended by the placement of the query. This criterion can be computed efficiently using standard least-squares formulas.

Despite showing accuracy comparable to that of maximum likelihood methods, the PhyClass prototype described in the original paper is substantially slower (probably due to the inefficient implementation). Whether it is possible to achieve comparable

placement accuracy using alignment-free sequence distances remains unclear. Unfortunately, neither the source code nor binaries are available online; I could not find any studies which applied this software.

APPLES

APPLES (*Accurate Phylogenetic Placement using LEast Squares*) is a recently published placement method [12, 11] aiming at enabling placement on ultra-large phylogenies. This is a distance-based method, and it requires the branch lengths of the tree to be optimized using a distance-based method (see Section 1.2.3). If another method was used to infer the tree, the topology could be kept, but the branch lengths have to be reoptimized using distance-based methods.

APPLES can be alignment-based or alignment-free. The default usage assumes the reference alignment as a requirement as input, from which pairwise distances between sequences are computed. In either case, the distance matrix should be provided by the user. Given the (sequence) distance matrix, APPLES optimizes placements by minimizing square root differences between the tree distance of two nodes and their sequence distance.

This approach allows APPLES to accurately place queries on large phylogenies (one and two hundred thousand of taxa), which is unachievable with maximum likelihood methods. Nevertheless, it is inferior to other methods in terms of placement accuracy. Recently, a preprint presenting the new version of this program was published [11]. It implements a divide-and-conquer approach to limit the distance computation and phylogenetic placement to the parts of the tree most relevant to the query. This improves APPLES in terms of placement accuracy, speed, and memory consumption: the author shows that the running time grows sublinearly with the number of taxa.

Thus, APPLES successfully overcomes the tree size scalability problem. However, the authors did not devote enough attention to the scalability in terms of the number of query sequences: in the alignment-free mode, it is the user's responsibility to calculate the distances accurately enough. The authors acknowledged that the reliable ways to calculate these distances without aligning queries and references are yet to be figured out [12]. Some progress in this direction has been made by Blanke and Morgenstern [22], which will be discussed later.

1.3.2 Alignment-free methods

RAPPAS

RAPPAS (*Rapid Alignment-free Phylogenetic Placement via Ancestral Sequences*) implements an alignment-free approach based on *phylo-k-mers* [130]. Both the notion of phylo-k-mers and the RAPPAS approach are fundamental to this work. They will be described in detail later in Sections 1.5 and 1.6.

RAPPAS does not require queries to be aligned within the reference alignment, making it scalable for analyzing massive amounts of metabarcoding sequences. The idea is to split the processing of the input data into two stages. During the first stage,

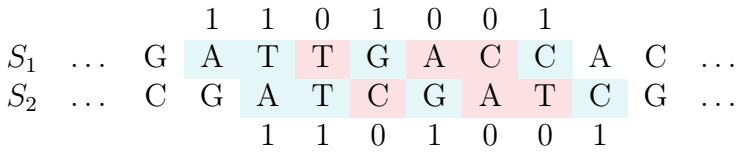


Figure 1-8 – An example of a spaced-word match between two sequences with respect to the pattern 1101001. Ones represent *match* positions (those positions are mandatory to match), zeros represent *don't care* positions (allowed to mismatch).

the reference alignment and the reference tree are preprocessed to calculate phylo-k-mers. Notably, the preprocessing does not rely on any information about query sequences contrary to LSHPlace. Therefore, preprocessing time does not depend on the number of queries, and it needs to be done only once for a given reference dataset. The result of preprocessing is a data structure called phylo-k-mer database, which allows placing queries blazingly fast. Phylo-k-mer databases can be reused many times to place different sets of query sequences.

Accuracy-wise it shows comparable results with maximum likelihood methods and places query sequences faster than EPA-ng (run in the sequential mode) [130]. Moreover, there is evidence that for some types of data it may outperform maximum likelihood-based approaches in placement accuracy [130, 22]. However, the first version published showed consistently worse accuracy for two types of data compared to the maximum likelihood methods. First, it is datasets containing many gaps in the reference alignment. Second, it showed worse performance for a dataset based on a long alignment (about 10k base pairs in length) compared to datasets based on short ones (one or two thousand of base pairs). Besides that, there are other issues related to the usage of RAPPAS that will be discussed later in Section 1.7.

App-SpaM

App-SpaM (*Alignment-free Phylogenetic Placement algorithm based on SPAced word Matches*) is yet another recently published software for alignment-free phylogenetic placement [22]. This is a distance-based method that requires neither query sequences nor reference sequences to be aligned together. The method is based on *filtered spaced-word matches* (*FSWM*, [124]), and consists of three stages. First, it finds spaced-word matches between every query and every reference sequence. Spaced-word matches are substring matches for which character mismatches are allowed at certain positions according to a binary pattern (see Figure 1-8 for an example). Second, for every query, it estimates phylogenetic distances between the query and the reference sequences. Spaced-word matches that correspond to high sequence comparison scores given a substitution model (i.e., filtered spaced-word matches) are used to calculate those distances. Finally, it finds queries' placements in the reference tree using either phylogenetic distances or information about spaced-word matches; one of five different strategies can be used.

Compared to RAPPAS, the competing alignment-free method, App-SpaM shows better performance in terms of placement speed, but lower in placement accuracy. It

targets small trees based on short phylogenetic markers and can place short reads to unassembled reference sequences of long genomes. Besides that, phylogenetic distances calculated by App-SpaM can be used as input for APPLES — called SpaM+APPLES — making it an alignment-free pipeline for placing on large phylogenies. However, few experiments have been done to evaluate this approach; those presented by [22] showed that this pipeline is less accurate than standalone App-SpaM (if the most accurate strategy is used).

1.3.3 Wrappers using other phylogenetic placement methods

SEPP

SEPP (*SATé-Enabled Phylogenetic Placement*) is a wrapper method that utilizes other placement software, pplacer, to perform phylogenetic placement [155]. Its algorithm is as follows: First, it splits the set of reference tree taxa into disjoint subsets, *insertion-taxon-subsets*. Second, every subset is split into smaller subsets — *alignment-subsets* — aligned independently using HMMER [66]. Third, it searches for the alignment-subset best matching q; the query is aligned within this subset. Finally, it uses the alignment to place q to the corresponding insertion-taxon-subset tree and inserts the placement to the original tree.

Thus, SEPP limits the maximum tree size used for placement (insertion-taxon-subset size) and the maximum number of sequences to align (alignment-subset size). In some instances, it significantly reduces RAM usage (compared to HMMER and pplacer run on the whole dataset) and even improves placement accuracy.

pplacerDC and pplacerXR

pplacerDC is a wrapper method that runs pplacer for subtrees of the reference tree [112]. Given a query, it splits the reference tree in a divide-and-conquer approach until it obtains a collection of subtrees $T_1, T_2, \ldots, T_i$ on disjoint subsets of no more than a certain number leaves (authors suggest 500). Then, for every subtree, it runs pplacer to place the query to every subtree in parallel. Every subtree is modified by inserting the query as reported by placements, and subtrees' likelihoods are evaluated with RAxML. The placement of the tree obtaining the highest likelihood is reported as the final result. pplacerDC allows placing on large phylogenies (up to 100k taxa) without significant loss in placement accuracy compared to pplacer. For large phylogenies, it improves the results of APPLES while being slower.

pplacerXR [222] is also a wrapper over pplacer (in addition, authors created a wrapper over EPA-ng implementing the same algorithm), and it operates as follows. Given the query q, pplacerXR finds the closest reference sequence to q using alignment-based Hamming distances. Then, it runs a breadth-first search from the corresponding to this sequence leaf l until it finds a certain number (authors suggest 2000) of leaves, closest to l in terms of total branch length. Finally, the subtree containing those leaves is given to pplacer to place the query. The best placement reported is the answer for the reference tree. pplacerXR showed better accuracy

than APPLES on huge trees, but is significantly slower. However, authors reported accuracy results only for full-length queries, the accuracy for short sequences (comparable to modern NGS read sizes) was not measured.

Both pplacerDC and pplacerXR seem to be more of a proof-of-concept than ready-to-use tools since they are poorly optimized. In addition, I foresee all wrapper methods discussed to be inappropriate for placing queries that are evolutionarily distant from the reference sequences. For such queries, correct placements will be branches close to the root, which may not be present in the subtrees processed by these algorithms.

1.3.4 Related tools

There are several other programs and libraries relevant to phylogenetic placement, and I will mention them here for the sake of completeness. HmmUFOtu is a pipeline for phylogenetically aware taxonomic assignment of 16S to OTUs [237]. HmmUFOtu applies phylogenetic placement at one stage of the processing of 16S amplicon sequences. The placement algorithm is similar to those of pplacer and EPA, but it is a part of the pipeline and can not be easily used as a standalone tool.

Genesis, gappa [42], and guppy are tools allowing for post-processing, analyzing, visualizing, and manipulating phylogenetic placement results. Such results are usually stored in .jplace files; .jplace is the standard JSON-like format for phylogenetic placement outputs [145].

PhyloSift allows phylogenetic analysis of genomes and metagenomes and utilizes phylogenetic placement [44]. PhyloMagnet allows screening large metagenomic and metatranscriptomic datasets to find genes of interest with phylogenetic placement [191]. SCRAPP applies phylogenetic placement to quantify the diversity of environmental samples [14, 105, 158].

A recently published pipeline for evaluating phylogenetic placement is PEWO [129] (*Placement Evaluation WOrkflows*). It allows running different phylogenetic placement tools to evaluate and compare their placement accuracy, running time, and memory consumption. I had a chance to participate in the development of this pipeline; some details about this work will be discussed in Sections 1.4, 3.7.1. I encourage all researchers related to the development of phylogenetic placement methods to use PEWO — the field needs a standardized framework for comparing different placement methods.

1.3.5 Summary

This concludes the overview of the state-of-the-art of phylogenetic placement. As far as I am aware, this is a complete list of existing methods for phylogenetic placement; the list of related tools may be incomplete.

Three main characteristics can be used to evaluate phylogenetic placement tools: placement accuracy, the scalability in terms of the number of queries, and the scalability in terms of the number of taxa of the reference tree. Placement accuracy can be measured in different ways and will be discussed later. Overall, the most accurate

approaches are maximum likelihood-based (EPA, PPLACER, EPA-NG), but they are problematic to scale with respect to the number of queries or the number of taxa. Wrapper methods can deal with the latter, but they have limitations discussed before. APPLES solves the tree size problem, but it is less accurate than competitors.

I believe that the problem of scalability in the number of queries is more important: the amount of data produced by NGS technologies is *already* challenging, and metagenomic research is limited methodologically, not technically. This problem will become even more relevant in the future with the further development of sequencing technologies. Meanwhile, the tree size scalability is only relevant for ultra-large accurate phylogenies that are not yet common.

Two phylogenetic placement tools — RAPPAS and APP-SPAM — target the query number scalability problem by applying alignment-free approaches. RAPPAS is more accurate, while APP-SPAM is faster. Table 1.1 summarizes the overview of phylogenetic placement tools. The "accuracy" column is a subjective measure based on literature, experimental results, and my experience using these tools; since there are different accuracy measures and different kinds of data, the results of accuracy comparisons are controversial. Thus, the reader should be aware that this evaluation may be debatable. The "scalability" columns are more objective and easier to justify. Additional information about these tools and placement-related tools and libraries can be found in Appendix (Tables A.1, A.2).

Table 1.1 – A comparison of existing standalone phylogenetic placement software. Abbreviations: not available (NA), maximum-likelihod (ML), alignment (Aln.), ancestral reconstruction (AR). Two last columns indicate the main scalability issue (e.g., the requirement to align the queries against the references or RAM consumption).

Software	Year	Ref	Method	Accuracy	Scalability # of queries	Scalability # of taxa
PPlacer	2010	[146]	Aln+ML	best (overall)	Aln-based	RAM
EPA	2011	[20]	Aln+ML	best (overall)	Aln-based	RAM
LSHPlace	2013	[29]	Aln+AR	poor	potentially[1]	?
Phyclass	2015	[65]	Aln+Dist.	good	potentially[2]	?
EPA-ng	2018	[15]	Aln+ML	best (overall)	Aln-based	RAM
RAPPAS	2019	[130]	AR	best (short q)[3]	yes	RAM
APPLES	2021	[11]	Aln+Dist.	OK	potentially[2]	yes
App-SpaM	2021	[22]	Distances	good	yes	?

[1] Methodological improvements are required.
[2] Using alignment-free methods to calculate distances, e.g., APP-SPAM.
[3] Only for short queries placed on phylogenies of short phylogenetic markers.

1.4 Evaluating phylogenetic placement

In this work, we will need to evaluate the accuracy of the phylogenetic placement made by different methods. There are two ways to estimate accuracy: absolute

measures and relative measures. Absolute placement accuracy is needed when we need to understand *how accurately a method places the query.* Relative accuracy is needed when we need to *compare two different methods placing the same query.* In this section, I will briefly introduce the most used measures.

1.4.1 Absolute placement accuracy

Absolute accuracy can be measured in *node distance* (ND, also known as *topological error*), which is calculated as follows. First, consider a reference tree, a query, and a branch y, which is the placement of the query on the tree. Then, suppose that we know the true placement, i.e., the branch y', which is the phylogenetic origin of the query. Then, node distance is measured as the number of tree nodes on the path from y to y'.

This method has certain disadvantages. The obvious one is that we need to know the true placements to measure accuracy; we will discuss workarounds to this in Section 1.4.3. Another disadvantage is that it does not account for branch lengths, so it may be poorly adapted to trees of organisms that evolve at different rates. However, it is the most popular way to measure accuracy of phylogenetic placement methods (used in [146, 20, 29, 130, 22]).

An alternative could be the total length of the branches between the placement node and the true placement node on y'; I use node distance in my work as it is the standard measure of accuracy.

1.4.2 Relative placement accuracy

The relative accuracy of placement can be measured using the *Kantorovich-Rubinstein metric* (KR-metric). The result of phylogenetic placement can be thought of as a distribution on the reference tree. Placement software reports likelihoods of placement for different branches, and normalizing those gives us the "placement distribution" (also known as *Likelihood Weight Ratio*). Thus, placements made by two different methods can be compared as two distributions on the tree, e.g., with the KR-metric[2]. In short, this is a measure of the "work" needed to transform one placement distribution into the other one.

The KR-metric allows us to determine how different the two placements are, but it does not determine which one is better. For this reason, I used another measure of relative accuracy in my work, differences in *likelihood-based accuracy.* I implemented a module for measuring it in PEWO [129] for this study. The measure will be described in Section 3.7.1.

1.4.3 Pruning-based accuracy evaluation

Pruning-based evaluation can be used to measure absolute placement accuracy if no true placements of queries are known. The method is as follows.

[2]Readers familiar with *weighted UniFrac* should probably experience a sense of recognition right now. The rest will find [60, 139] an informative read.

1. Given a reference tree T, split it into two subtrees, T_q and T_r, by cutting it along a branch y'.

2. Reoptimize branch lengths of T_r.

3. Use reference sequences corresponding to the leaves of T_q to generate a set of queries Q.

4. Place those queries to T_r.

5. The node of y' that belongs to T_r points to the true placements for queries from T_q. Use it to calculate the measure of interest.

If T_q has only one leaf, it is known as the *leave-one-out* method, otherwise, it is called *leave-many-out*. This approach has been applied in several phylogenetic placement studies [146, 15, 130, 12, 11, 22]. PEWO evaluates the accuracy of phylogenetic placement tools by repeating this procedure many times for random branches of the tree. We call one T_q a *pruning*; mean accuracy per pruning is the main accuracy measure used in this work.

The disadvantage of this approach is that it is computationally expensive. For alignment-based methods, it requires aligning generated queries within sequences corresponding to the leaves of T_r.

1.5 Phylo-k-mers

Here we come to the central subject of this thesis, namely, phylo-k-mers. Understanding the notion of phylo-k-mers is essential for further reading; without understanding it, nothing discussed after would make any sense. I will first give an intuition of what are phylo-k-mers. A detailed description of how to obtain them will follow after (this description is exactly how RAPPAS computes phylo-k-mers). Finally, I will describe how RAPPAS uses phylo-k-mers for phylogenetic placement.

1.5.1 Intuition

Phylo-k-mers are phylogenetically informed k-mers. Thus, phylo-k-mers can only be defined in the presence of a fixed reference tree. We also require a reference sequence alignment for reasons that will become clear later. Phylo-k-mers can be understood as k-mers accompanied by probabilistic information about the reference phylogeny and the reference alignment.

A phylo-k-mer consists of a k-mer w and a *score* $S_y(w)$ of this k-mer for a particular branch y of the reference tree. Thus, we can define a phylo-k-mer as a triple

$$\big(w, y, S_y(w)\big). \tag{1.1}$$

The score $S_y(w)$ approximates the probability that the k-mer w is present in a sequence that diverged from the reference tree somewhere along branch y. In practice,

observing the k-mer within a sequence provides evidence about the phylogenetic origin of that sequence; the score quantifies how likely y is to be the phylogenetic origin of that k-mer.

1.5.2 Computation of phylo-k-mers

The input data for the computation of phylo-k-mers are a reference alignment A_0, an evolutionary substitution model, a rooted reference phylogenetic tree T_0, the length of k-mers k, and a score threshold value ε. The reference alignment consists of aligned sequences over the alphabet Σ of size σ (e.g., $\Sigma = \{A, C, G, T\}$ for DNA). The leaves of T_0 have to be in one-to-one correspondence with the sequences of A_0, The lengths of branches have to be previously fitted based on A_0 and the substitution model. Computation of phylo-k-mers requires the following steps: alignment filtering, ghost node injection, ancestral sequence reconstruction for ghost nodes, and finally the actual computation of phylo-k-mer scores. The following paragraphs describe each step of this process in more detail.

Reference alignment filtering

First, we remove those columns of A_0 that contain more than a certain percentage of gaps "–" (in RAPPAS, the default percentage is 0.99). The rationale for this step is that ancestral reconstruction techniques — used by one of the later steps — do not account for gaps. Furthermore, columns with high percentage of gaps have been observed to be deleterious for computation of phylo-k-mers. We call the result of this step the *reduced reference alignment*, and refer to it as A.

Ghost node injection

Subsequently, we extend the reference phylogenetic tree by introducing new nodes and branches. For every branch in T_0 we inject a certain number of *ghost nodes* associated with this branch. Here we shortly describe the way it is suggested by RAPPAS, although one can vary the number of ghost nodes per branch, as well as the way they are introduced to the tree.

For every branch y in T_0, two ghost nodes u_y and v_y are introduced: a node u_y at the midpoint of y and a new leaf v_y. A new branch (u_y, v_y) is also added to the tree. The length of this new branch is set to the average length among all paths from u_y to the leaves of T_0 that descend from u_y. The modified tree T thus obtained is called an *extended tree*.

From now on, we are interested in the set of G_y of ghost nodes ($G_y = \{u_y, v_y\}$ in this case), associated to the branch y. Variations of the ghost node injection described above could imply adding more ghost nodes, placed over the new branch (u_y, v_y) in different ways. For those variations, we assume that G_y consists of more than two elements.

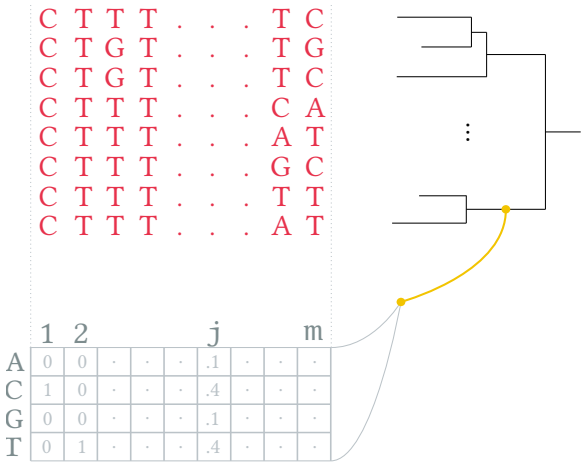


Figure 1-9 – Illustration of ghost node injection and ancestral reconstruction for DNA sequences. A ghost branch and two ghost nodes (yellow) are added to the reference (black) tree. For every ghost node, a probabilistic profile of ancestral sequences of the node is computed (represented by the matrix in gray for one of the ghost nodes). Column j of the matrix gives probabilities of observing different nucleotides at position j in such sequences.

Ancestral sequence reconstruction

During this step, we aim to model probabilities of observing different states (i.e., nucleotides for DNA) at the ghost nodes of the extended reference tree. Given an evolutionary model describing the process of nucleotide/amino acid substitutions, we estimate probabilities $p_j^u(a)$:

$$p_j^u(a) = \mathbb{P}[\text{observing state } a \in \Sigma \text{ at site } j \text{ of the sequence at node } u \mid A] \qquad (1.2)$$

The probabilities from 1.2 depend on the alignment, the tree, and the evolutionary substitution model assumed. They can be estimated by applying standard ancestral reconstruction techniques. Implementations of those techniques are provided by PhyML [80], PAML [232], and RAxML-NG [115]. The result of this step is a $\sigma \times m$ matrix P^u for every ghost node u, where m is the length of A, and values of P^u are $P^u_{i,j} = p_j^u(a_i)$, $a_i \in \Sigma$. Figure 1-9 illustrates the ghost node injection and ancestral reconstruction steps.

Score computation

For both ancestral reconstruction and computation of phylo-k-mer scores, we assume independent evolution at different sites of the alignment, which is standard in phylogenetics. Having estimated probabilities of different states at every site of the alignment for every ghost node, we can calculate scores of phylo-k-mers as follows. Let $w = w_1 w_2 \dots w_k$ be a k-mer over Σ. Then the score $P_j^u(w)$ of w at position j for the ghost node u can be simply found by taking the product of scores of corresponding

k-mer states at consecutive sites, starting from j:

$$P_j^u(w) = p_j^u(w_1) \cdot p_{j+1}^u(w_2) \cdot \ldots \cdot p_{j+k-1}^u(w_k) \tag{1.3}$$

$P_j^u(w)$ is the probability of observing the k-mer w at position j for the ghost node u (should not be confused with $p_j^u(a)$, which is the probability of observing the state a at position j for the same ghost node). Those are used to compute the *phylo-k-mer score* $S^u(w)$ of w that does not depend on the position, but estimates the probability of observing w at any position for u. RAPPAS suggests taking the maximum score over all positions for a k-mer and a predetermined *threshold value* ε:

$$S^u(w) = \max\left\{ \max_{j=1}^{m-k+1} P_j^u(w), \varepsilon \right\} \tag{1.4}$$

The threshold value plays an essential role in Formula 1.4. First, the naive approach involves calculating *all* phylo-k-mers for *all* ghost nodes of the tree. The number of obtained phylo-k-mers grows exponentially with k. By introducing the threshold value, we eliminate the need to store many of them: we just need to store the phylo-k-mers with scores higher than ε, assuming the score of ε for the rest. This allows decreasing the final number of phylo-k-mers significantly. Second, the threshold guarantees that no k-mer is excluded from consideration, even those for which the score is zero.

The threshold value is assumed to be very small (in RAPPAS, the value for DNA is $(\omega/4)^k$, where ω is a parameter of the method with the default value of 1.5). By changing this parameter, we can control how many phylo-k-mers with low scores we want to estimate precisely and how much memory we will need to store the result.

Having calculated scores of w for all ghost nodes $u \in G_y$ associated to the branch y, we can finally compute the score of w for this branch. This is just the maximum score across the ghost nodes associated to the branch:

$$S_y(w) = \max_{u \in G_y} S^u(w) \tag{1.5}$$

Phylo-k-mer scores $S_y(w)$ should be calculated for every branch y in T_0 and every k-mer $w \in \Sigma^k$. Since score values can reach extremely small numbers, log-values of $S_y(w)$ can be used instead. Calculated log-score values are then stored in an associative array for later use. The keys in the associative array are binary representations of k-mers (will be discussed later in Section 2.1.3), and the values are lists of pairs $\{(y_{i_1}, \log S_{y_{i_1}}(w)), (y_{i_2}, \log S_{y_{i_2}}(w)), \ldots\}$. Thus, in practice, we do not store phylo-k-mers as triplets to minimize memory consumption. The phylo-k-mers with score less or equal to the threshold score are also not stored. Note that the latter means that lists associated with different k-mers may vary in length (and some k-mers may not be present in the associative array at all).

RAPPAS uses hash maps to store phylo-k-mers. The hash map containing phylo-k-mers for every branch of the reference tree is called *a phylo-k-mer database* [3]. Once

[3]My software-engineering-self finds this term rather unfortunate. Collections of phylo-k-mers have nothing to do with neither relational nor non-relational databases. Calling it a *phylo-k-mer*

constructed, a database of phylo-k-mers is saved on disk for later use.

1.6 RAPPAS: alignment-free phylogenetic placement using phylo-k-mers

RAPPAS preprocesses the reference dataset as described above, which we call the *database computation stage* or *reference dataset preprocessing*. This takes place before any queries are even observed. Afterward, the stage of query *placement* can take place many times, reusing the same phylo-k-mer database. This stage is as follows. Its input is a phylo-k-mer database and a set of query sequences Q. Every query sequence $q \in Q$ is placed independently; for a query sequence q, each branch y receives the *placement score*:

$$\sum_j \log S_y(w_j) \tag{1.6}$$

Here $w_1, w_2 \dots w_{|q|-k+1}$ is the list of k-mers composing the query. To retrieve values $S_y(w_j)$, RAPPAS loads the database into memory and stores all phylo-k-mers in an associative array. Then, every k-mer w of the query is searched in the associative array. If k-mer w is found, the list of pairs $\{(y, \log S_y(w))\}$ is retrieved. For the branches that are not present in the list, the logarithm of ε is assumed. If the k-mer is not found, the logarithm of ε is also assumed for all branches. This process is repeated for every k-mer in the query sequence, and every branch accumulates retrieved log-score values. Finally, the branches are ranked by their total accumulated log-score in Equation 1.6, which is the result of the phylogenetic placement. The highest-scoring branches are interpreted as the most likely placements for query q.

The associative array used to store phylo-k-mers provides a constant-time lookup. Then, the placement algorithm takes $\mathcal{O}(|q| \cdot \bar{\ell}(q))$, where $\bar{\ell}(q)$ is the average length of the list associated to a k-mer in q. The placement algorithm and its complexity are discussed in detail in the supplementary materials of [130].

1.7 Motivation and thesis structure

The reference dataset preprocessing technique allows RAPPAS to index the phylogeny with phylo-k-mers. Since the content of phylo-k-mer databases does not depend on queries, it solves the scalability problem of phylogenetic placement in terms of the number of query sequences. However, the preprocessing itself may be challenging since it can take considerable time and memory to preprocess the reference dataset. For example, phylo-k-mer databases can reach gigabytes in size even for small and medium-sized trees (under 2000 taxa for the default value of $k = 10$) based on short phylogenetic markers such as the 16S rRNA gene. This limits the application of the method for both larger trees and trees based on longer reference sequences.

index would be much better; however, I use the established terminology.

Thus arises the question of *how to effectively index phylogenies with phylo-k-mers*, which is the central question of the thesis. First, I address this question from the algorithmic point of view in Chapter 2 (how to compute phylo-k-mers as fast as possible?), aiming at reducing the computation time of the phylo-k-mer database creation. Second, I address the question from the information-theoretic point of view in Chapter 3 (which phylo-k-mers are informative and which are not? Can we exclude some phylo-k-mers from consideration?), aiming at reducing sizes of phylo-k-mer databases, which reduces RAM consumption during phylogenetic placement. Third, in Chapter 4, I present XPAS and RAPPAS2, which are effective reimplementations of RAPPAS. They outperform RAPPAS in both running time and memory consumption. Finally, Chapter 5 summarizes the results of all chapters and discusses future work directions on methods for phylo-k-mers.

1.8 Notation

1.8.1 Notation for strings

Let Σ be an ordered finite alphabet of cardinality σ: $a_1 < a_2 < \cdots < a_\sigma$, $a_i \in \Sigma$. I consider strings (or sequences) over alphabet Σ. Let k be a positive integer. Let Σ^k denote the set of all possible strings of length k over Σ. Given a string s of k characters, the length of s is denoted by $|s| = k$. For any integer $1 \leq i \leq k$, s_i denotes i^{th} character of s, and s is written down as $s_1 s_2 \dots s_k$.

Given the alphabet Σ, the set of possible strings over Σ is denoted by Σ^*. Given a set of strings X, X^* denotes the Kleene closure over X.

1.8.2 Notation for matrices

I use two notations for matrix elements that are used interchangeably. I consider matrices whose elements are indexed by integers in a standard way: rows are indexed with $1 \leq i \leq \sigma$, and columns are indexed as the positions of a multiple alignment, $1 \leq j \leq m$. Besides that, rows are indexed by symbols of the alphabet Σ according to the order of Σ. A column stores the probability of occurrences of each possible symbol (a state in phylogenetic terms) at that position. Hence, we term such matrices *probability matrices* since the sum of the values of a column sum to one.

For a $\sigma \times m$ probability matrix P, $p_j(a)$ denotes the element on line indexed by a, $a \in \Sigma$, and column j of P, $1 \leq j \leq m$. Equally, the same element is denoted by $P_{i,j}$, where i is the rank of a in the order Σ. For two integers i, j such that $1 \leq i \leq j \leq m$, $P[i : j]$ denotes the matrix P restricted to columns from i to j included.

Chapter 2

Algorithms for computing phylo-k-mers

This chapter discusses the central algorithmic problem of RAPPAS' reference dataset preprocessing: phylo-k-mer computation. We aim at answering two questions: How to compute phylo-k-mers effectively? What are different algorithms of phylo-k-mer computation, and how performant are they? Before addressing the problem in its classic formulation — that is, compute those phylo-k-mer scores that surpass a given threshold — I first study a simpler problem of calculating *all* phylo-k-mer scores. Thus, we will discuss two statements of the problem of phylo-k-mer computation and algorithms for solving both. Besides existing solutions, I describe novel ones and give an experimental comparison of their performance.

Given phylo-k-mer scores of a k-mer for different ghost nodes, phylo-k-mer scores for branches are easily obtained as described in the previous chapter. Therefore, here we focus on the problem of calculating phylo-k-mers for one ghost node u given a matrix P^u, that is, calculating $S^u(w)$ not $S_y(w)$. I do not address the question of how exactly to calculate the P^u matrix. This question is beyond the scope of this chapter; instead, I only discuss how to compute phylo-k-mer scores when P^u is given. Computing phylo-k-mer scores of all k-mers for all ghost nodes is the most computationally heavy part of the algorithm described in Section 1.5.2. In the worst-case it implies calculating scores for all possible k-mers $w \in \Sigma^k$.

For this chapter only, I use a simplified notation for brevity: P^u becomes P, and $p^u_j(a)$, $P^u_{i,j}$, $P^u_j(w)$, $S^u(w)$ become $p_j(a)$, $P_{i,j}$, $P_j(w)$, $S(w)$, respectively. It implies that algorithms described here are applied for all ghost nodes u of the extended tree.

2.1 Computational model

2.1.1 Word-RAM assumptions

Later in this chapter, I will give an analysis of algorithms' performance. All algorithms of this chapter are analyzed under the assumptions of the word-RAM model [85]. This model assumes operating on words of size b and performing arithmetic and bitwise

operations in constant time. We also assume that any k-mer can be represented with a constant number of machine words, which yields $b = \Theta(\log \sigma^k)$, where σ is the size of the alphabet. Those are standard assumptions made in the analysis of algorithms; however, I deliberately draw the reader's attention to those assumptions. They allow us to effectively operate on k-mers represented as unsigned integer numbers, leading to improved theoretical bounds of algorithms described below.

2.1.2 Assumptions about the alphabet

Although the algorithms described in this chapter can be applied to compute phylo-k-mers over an arbitrary alphabet, in practice, phylo-k-mers are applied only to analyze DNA or protein sequences. Quite obviously, the alphabet does not change within one phylo-k-mer-based application (e.g., the phylogenetic placement of DNA sequences). Therefore, I assume that σ is constant.

2.1.3 Representation of k-mers

Using word-RAM assumptions allows us to gain certain benefits from representing strings as integer numbers. It is standard practice in bioinformatics, but I describe it here for the sake of completeness.

Definition 2.1.1 (Alphabet symbol codes)**.** Let Σ be a totally ordered set of size σ, $a_1 < a_2 < \cdots < a_\sigma$, $a_i \in \Sigma$. Then $ord : \Sigma \to [0, 1, \ldots, \sigma - 1]$ defines *codes* of symbols of Σ as follows:

$$\mathrm{ord}(a_i) = i - 1$$

We will be interested in both $\mathrm{ord}(a_i)$ values and their binary representations of fixed length. Let c be the smallest number of bits allowing representation of all symbols in Σ, that is $2^{c-1} < \sigma \leq 2^c$. Then we use c-long binary representations of symbol codes.

Example 2.1.1. For the alphabet $\Sigma = \{A, C, G, T\}$, we can encode every symbol of Σ with two bits as in Table 2.1.

index	1-mer	code
0	A	00
1	C	01
2	G	10
3	T	11

Table 2.1 – 1-mer codes for DNA.

To express k-mers over Σ in codes, we concatenate binary representations of all k-mer symbols. The advantage of this encoding is that one can calculate k-mer codes from codes of k-mer symbols straightforwardly.

Definition 2.1.2 (k-mer codes)**.** For a totally ordered set Σ and a symbol code function *ord*, defined as in Def. 2.1.1, we can extend the definition of *ord* to every k-mer $w \in \Sigma^k$:

$$\mathrm{ord}(w) := \sum_{i=1}^{k} 2^{c(k-i)} \cdot \mathrm{ord}(w_i) \tag{2.1}$$

Example 2.1.2. For the same alphabet and the same ordering function as in Example 2.1.1, Table 2.2 gives codes of all 2-mers.

index	2-mer	code	index	2-mer	code
0	AA	0000	8	GA	1000
1	AC	0001	9	GC	1001
2	AG	0010	10	GG	1010
3	AT	0011	11	GT	1011
4	CA	0100	12	TA	1100
5	CC	0101	13	TC	1101
6	CG	0110	14	TG	1110
7	CT	0111	15	TT	1111

Table 2.2 – 2-mer codes for DNA.

Now that we have established this correspondence, k-mers and their codes are interchangeable. Moreover, string concatenation can also be expressed in terms of codes: for two strings p and s of sizes l and r the code of $p \cdot s$ is calculated in the following manner:

$$\mathrm{ord}(p \cdot s) := \mathrm{ord}(p) \cdot 2^{cr} + \mathrm{ord}(s) \tag{2.2}$$

The following sections introduce algorithms for computing phylo-k-mers. I will use assumptions of the word-RAM model, paired with this representation of k-mers, in the analyses of those algorithms.

2.2 Exhaustive Phylo-k-mer Computation

This section focuses on *exhaustive*, or *exact* phylo-k-mer computation: given a probability matrix P for a ghost node, how to calculate scores for *all* possible phylo-k-mers of this node? We should clarify at once that it is impractical to calculate all phylo-k-mer scores. There are two reasons for this. First, the running time of an algorithm solving this problem is $\Omega(\sigma^k)$. Second, most of this time is spent calculating scores of k-mers that are highly improbable; that is, most of them are very unlikely to be used in phylo-k-mer-based applications. Instead, in practice, we are interested in solving a similar problem, the problem of threshold-based phylo-k-mer computation (TPKC), covered in later sections. However, we study the exhaustive formulation first: it is

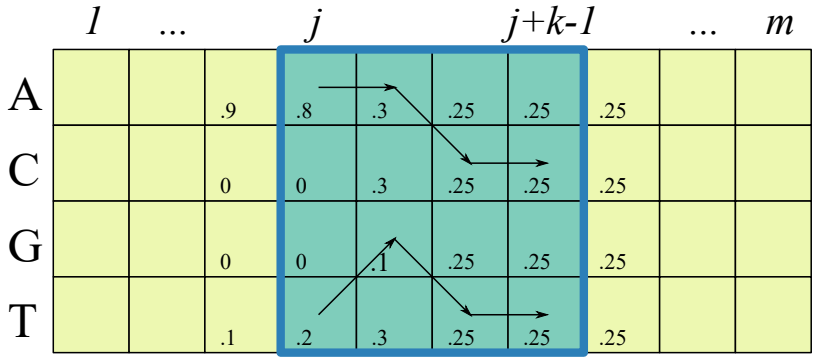


Figure 2-1 – An example of input matrix P for the DNA alphabet $\{A, C, G, T\}$, and $k = 4$. In the highlighted window of size k, the arrows traverse elements corresponding to the k-mers $AACC$ and $TGTT$. The product of the traversed elements gives the scores of corresponding k-mers for position j.

simpler but still has much in common with TPKC. Solving it will help us address the real problem of interest later on.

Problem 1 (Exhaustive Phylo-k-mer Computation (EPKC))**.**

Input: An integer $k > 1$; a $\sigma \times m$ probability matrix P.

Output: An array Q containing the scores of k-mers. For every k-mer $w \in \Sigma^k$, $Q[ord(w)]$ stores $S(w)$ with

$$S(w) := \max_{j=1}^{m-k+1} P_j(w)$$

where $P_j(w)$ is the score of w at position j:

$$P_j(w) := \prod_{l=1}^{k} p_{j+l-1}(w_l)$$

Note that the definitions of $S(w)$ and $P_j(w)$ above coincide with those given in Chapter 1 — see Equations 1.3 and 1.4 — for the threshold $\varepsilon = 0$.

2.2.1 Naive algorithm

A straightforward solution to Problem 1 is the following. Let us consider a single window W of k consecutive columns in P, starting from position j. For this window, all values $P_j(w)$ can be calculated in $\mathcal{O}(k \cdot \sigma^k)$: for every $w \in \Sigma^k$, we take a product of k values. Figure 2-1 gives an illustration of this process for one window. To obtain $S(w)$, one has to perform this process for all windows W of P, preserving the maximum score of every k-mer among different windows.

There are $m - k + 1$ windows, which yields time complexity of $\mathcal{O}(mk \cdot \sigma^k)$. The memory complexity of the algorithm is $\mathcal{O}(\sigma^k)$, which is given by the size of the output. Note that here we rely on the assumptions of the word-RAM model, described in the previous section: we assume that one value of $S(w)$ takes $\mathcal{O}(1)$ memory to store.

Since the memory complexity is theoretically optimal, we are interested in improving time complexity; thus, all improvements over the naive algorithm lie in improving time complexity.

2.2.2 Window-wise and k-mer-wise approaches

Notice that I suggested approaching the problem, iterating window-by-window of size k. Another naive algorithm could iterate k-mer-by-k-mer, calculating scores of one k-mer among all positions before going for the next k-mer. We call those approaches window-wise and k-mer-wise, respectively; notice that both yield the same theoretical complexities. A few other algorithms described below utilize the window-wise approach as well. However, they differ in computing $P_j(w)$ values for a fixed window. For those algorithms, Algorithm 1 describes the nature of the process in general.

Algorithm 1: Window-wise EPKC

```
   Input  : Alphabet Σ, k, a posterior probability matrix P
   Output: Array S of phylo-k-mer scores
1  Function WindowWiseEPKC P, k, Σ
2      S ← array of size σ^k
3      foreach window W of size k of P do
4          S^W ← compute phylo-k-mers for the window W
5          for i ← 0..|S^W| − 1 do
6              S[i] ← max(S[i], S^W[i])
7      return S
```

2.2.3 Common Prefix Algorithm

We can improve the naive algorithm as follows. Let us again consider one window W of k consecutive columns of P in the same manner as above. For a fixed window, we can save on calculating scores of phylo-k-mers with common prefixes. Having calculated the score of a prefix $w_1 w_2 \dots w_j$, we reuse its score to calculate the scores of all k-mers that start with $w_1 w_2 \dots w_j$. We calculate scores for k-mers in lexicographic order. See Algorithm 2 for a detailed listing.

Under assumptions of the word-RAM model, line 7 takes $\mathcal{O}(1)$ time. Let $h = k - j$ be the number of columns of W that left to process, and $T(h)$ be the running time of CommonPrefix called with $j = k - h$. Then line 12 takes $T(h-1)$ time, and overall time complexity of CommonPrefix can be expressed by the following recurrence:

$$T(h) = \begin{cases} \Theta(1) & \text{if } h = 0 \\ \sigma \cdot T(h-1) & \text{if } h \geq 1 \end{cases} \tag{2.3}$$

Theorem 2.2.1. The recurrence 2.3 implies $T(h) = \Theta(\sigma^h)$.

Algorithm 2: Common Prefix Algorithm (CP) for EPKC

Input : An integer $k > 0$ and a $\sigma \times k$ probability matrix W
Output: A σ^k long array S containing the scores of all phylo-k-mers

```
1  Function ComputePhyloKmers(W):
2      S ← 0-based array of size σ^k
3      return CommonPrefix(W, S, 0, 0, 0, 1)
4  Function CommonPrefix(W, S, i, j, code, score):
5      if i, j > 0 then
6          score ← score · W_{i,j}
7          code ← 2^⌈log_2 σ⌉ · code + i − 1
8      if j = k then
9          S[code] ← score
10     else
11         for i' ← 1 ... σ do
12             S ← CommonPrefix(W, S, i', j + 1, code, score)
13     return S
```

Proof.

Base case: $T(0) = \Theta(1) = \sigma^0$.

Induction step: Let $T(h-1) = \Theta(\sigma^{h-1})$. Then $T(h) = \sigma \cdot T(h-1) = \Theta(\sigma^h)$.

□

Thus, CommonPrefix achieves asymptotically optimal running time, yielding $\Theta(\sigma^k)$ time to process a window of k consecutive columns. As for memory consumption, it takes $\Theta(k)$ for the call stack and $\Theta(\sigma^k)$ to store the output. The latter yields the memory complexity of $\Theta(\sigma^k)$.

2.2.4 Divide-and-Conquer

While CommonPrefix is optimal, we are interested in algorithms demonstrating better running time in practice. Here I describe another algorithm for computing phylo-k-mers in a window of size k that outperforms CommonPrefix. This algorithm applies the classic divide-and-conquer idea to calculate scores for prefixes and suffixes of phylo-k-mers. It aims to split each window into two windows of equal or almost equal size l and r. For l-mers and r-mers — *prefixes* and *suffixes* — scores are calculated recursively in corresponding subwindows. To calculate scores of k-mers, we iterate over all possible prefix-suffix combinations and multiply their scores (see Algorithm 3).

Algorithm 3: Divide-and-Conquer (DC) for EPKC

Input : A $\sigma \times k$ probability matrix W
Output: A σ^k long array S containing the scores of all phylo-k-mers

```
1  Function ComputePhyloKmers(W):
2      return DivideAndConquer(W)
3  Function DivideAndConquer(W):
4      h ← number of columns in W
5      if h = 1 then
6          return W
7      else
8          l ← ⌊h/2⌋; r ← ⌈h/2⌉
9          L ← DivideAndConquer(W[1 : l])
10         R ← DivideAndConquer(W[r : h])
11         S ← array of size σ^h of zeros
12         for i ← 0 ... σ^l − 1 do
13             for j ← 0 ... σ^r − 1 do
14                 code ← i · 2^⌈log2 σ⌉r + j;
15                 S[code] ← L[i] · R[j];
16         return S
```

Complexity analysis

Let $T(h)$ represent the running time of DivideAndConquer for the window of size k. Here I break down the running time complexity of this procedure line by line.

For the case $h = 1$ we return the single column of the window as the answer (lines 5—6), which takes $\Theta(\sigma) = \Theta(1)$ time. For the other case: line 8 takes constant time by the word-RAM assumptions. Recursive calls on lines 9—10 take $2 \cdot T(h/2)$, and line 11 takes $\Theta(\sigma^h)$ time.

Let us have precomputed powers of two up to 2^k. Then lines 14—15 take constant time under assumptions of word-RAM; this gives us $\Theta(\sigma^l \cdot \sigma^r) = \Theta(\sigma^h)$ for the block 12—15. Therefore, the overall time complexity of DivideAndConquer is given by the following recurrence:

$$T(h) = \begin{cases} \Theta(1) & \text{if } h = 1 \\ 2 \cdot T(h/2) + \sigma^h & \text{otherwise} \end{cases} \tag{2.4}$$

Theorem 2.2.2. The recurrence 2.4 implies $T(h) = \Theta(\sigma^h)$.

Proof. Let us use the master theorem [38] to solve the recurrence: $a = 2, b = 2, f(h) = \sigma^h$. It corresponds to the case of the theorem, i.e., the work needed to recombine the results of the recursive calls dominates the work needed for subproblems.

$$
\begin{aligned}
h^{\log_b a+\epsilon} &= h^2 && \text{for } \epsilon = 1 \\
&\leq 2^h && \text{for } h \geq 4 \\
&\leq \sigma^h && \text{for } \sigma \geq 2 \\
&= f(h)
\end{aligned}
$$

Therefore, $f(h) = \Omega(h^{\log_b a+\epsilon})$. Let us now show that the regularity condition is satisfied: $af(h/b) \leq cf(h)$ for $c < 1$ and some $h > h_0$.

$$
\begin{aligned}
af(h/b) &= 2 \cdot \sigma^{h/2} \\
&\leq \sigma^{1+h/2} && \text{for } \sigma \geq 2 \\
&\leq \sigma^{h-1} && \text{for } h > h_0 = 3 \\
&= 1/\sigma \cdot \sigma^h \\
&= cf(h) && \text{for } c = 1/\sigma,\ c < 1
\end{aligned}
$$

$T(h) = \Theta(\sigma^h)$ follows immediately.

□

Thus, the algorithm's running time is $\Theta(\sigma^k)$, making this algorithm an optimal solution for EPKC. It takes $\Theta(\sigma^{h/2})$ in memory to store prefixes and suffixes (lines 9—10), apart from $\Theta(\sigma^h)$ taken by the result. This gives us the same recurrence for the memory consumption, which yields the memory complexity of $\Theta(\sigma^k)$.

2.3 Threshold-based Phylo-k-mer Computation

This brings us to the central question of this chapter: given a probability matrix P for a ghost node, how to compute phylo-k-mers, whose scores are greater than a particular threshold value? Of course, we could first solve EPKC and then choose from the answer phylo-k-mers with suitable scores. This is indeed suboptimal since we are interested in computing *only* the phylo-k-mers whose scores surpass the threshold. This sections overviews algorithms for solving this problem.

2.3.1 Problem statement

In this problem reformulation, we approximate low-score phylo-k-mers with a single score threshold value ε. The motivation for such an approximation is as follows. The number of k-mers grows exponentially with the number k; in practice, however, many of these k-mers have scores close to zero. By approximating their scores by ε, we avoid storing their actual scores, almost without losing precision. That allows

us to store substantially fewer phylo-k-mers, partly overcoming the combinatorial explosion problem in the number of phylo-k-mers.

To formalize this, let $\mathcal{Z}_\varepsilon$ be the set of k-mers whose scores are higher than the threshold value ε, i.e.,

$$\mathcal{Z}_\varepsilon = \{w \in \Sigma^k : S(w) > \varepsilon\} \tag{2.5}$$

Let us store only the phylo-k-mers of $\mathcal{Z}_\varepsilon$, and assume the score of ε for the rest. A formal definition of the problem is given by Problem 2.

Problem 2 (Threshold-based Phylo-k-mer Computation (TPKC))**.**
Input: k, P as in EPKC (Problem 1); a threshold value $\varepsilon \in [0, 1)$
Output: Associative array A storing the pairs $\{(w, S(w)) \mid w \in \mathcal{Z}_\varepsilon\}$, where $S(w)$ is defined as in Problem 1.

Notice that the size of the output of the problem is only $\Theta(|\mathcal{Z}_\varepsilon|)$, not $\Theta(\sigma^k)$ as for the previous one. We can decrease ε to include more k-mers in the answer — which makes phylo-k-mer-based applications more accurate — or increase it to save on memory. From now on, we assume that $|\mathcal{Z}_\varepsilon| \ll \sigma^k$.

Window-wise algorithms

Let us now look at just one window W that starts at position j of the alignment. Then we can define $\mathcal{Z}_\varepsilon^W$ as follows:

$$\mathcal{Z}_\varepsilon^W = \{w \in \Sigma^k : P_j(w) > \varepsilon\} \tag{2.6}$$

That is, $\mathcal{Z}_\varepsilon^W$ includes k-mers that achieve score greater than ε in the window W. Those sets for all windows straightforwardly combine into $\mathcal{Z}_\varepsilon$:

$$\mathcal{Z}_\varepsilon = \bigcup_{W \in \substack{\text{all} \\ \text{windows}}} \mathcal{Z}_\varepsilon^W \tag{2.7}$$

For phylo-k-mers of one window we can also assume the choice of ε for which $|\mathcal{Z}_\varepsilon^W| \ll \sigma^k$.

2.3.2 Branch-and-bound

Here I describe the first algorithm for solving Problem 2, as it has been introduced for the first time in [130]. It has not been described in detail yet; little attention has been paid to its theoretical properties as well. This algorithm is based on the CommonPrefix algorithm described above.

Low-score prefixes

We can modify CommonPrefix to solve Problem 2 with one simple observation. That is, if the score of a prefix is lower than ε, no k-mer that starts with this prefix can be in $\mathcal{Z}_\varepsilon^W$.

Theorem 2.3.1. Let $p = w_1 \dots w_j$ be a string over Σ^j, $j < k$, and W be a k-sized window of P. If $P^W(p) \leq \varepsilon$, then $\forall w \in \Sigma^k$ such that p is a prefix of w, it is true that $w \notin \mathcal{Z}_\varepsilon^W$.

Proof. All elements of P are less than 1: $\forall i, j\ P_{i,j} \leq 1$. Then for any k-mer w that starts with p it is true that $P^W(w) \leq P^W(p) \leq \varepsilon$. The statement follows immediately. □

Thus, we can give up calculating scores for all k-mers that start with a prefix whose score is lower than ε.

Sorting matrix columns

There is another possible improvement to this algorithm. Suppose that the elements of each column in the input matrix have been previously sorted in descending order. That is, we consider a probability matrix $\widetilde{P}$ with $\widetilde{P}_{1,j} \geq \widetilde{P}_{2,j} \geq \ldots \geq \widetilde{P}_{\sigma,j}$ for every j.

Let us now start calculating prefix scores in the same manner as in CommonPrefix, but over $\widetilde{W}$ not W. If a prefix w is such that $P^W(w) \leq \varepsilon$, then all prefixes w' obtained from w by replacing its last symbol with a lower-probability symbol will also be such that $P^W(w') \leq \varepsilon$. As soon as we reach a prefix w with $P^W(w) \leq \varepsilon$, we can abort computation not just for w, but also for potentially other prefixes that would follow w in $\widetilde{P}$. Figure 2-2 illustrates this idea.

Putting it all together

Since the probability matrix $\widetilde{W}$ is permuted, we can not rely on it to calculate k-mer codes. Let us store the sorting permutation for each column of the window W in a $\sigma \times k$ matrix R. That is, R_{*j} is a 0-based permutation of elements of j-th column of W that would sort this column in descending order. The codes of phylo-k-mers computed now can be obtained with values of R_{ij}; see Algorithm 4 for the full listing.

Notice that this algorithm looks very similar to CommonPrefix. The first difference is that on line 7, R_{ij} is used to calculate the correct prefix code. The second and the most important one is lines 12—13 that stop recursion for prefixes with low scores. The latter is what has given the name of this algorithm: it allows to cut branches of computation for sets of k-mers sharing the same low-score prefix.

Worst-case performance

Despite that the output size of Algorithm 4 is $\Theta(|\mathcal{Z}_\varepsilon^W|)$, this algorithm has a worst-case performance of $\mathcal{O}(\sigma^k)$ in time. We can show this with the following example. Consider a binary alphabet ($\sigma = 2$) and suppose that all probabilities in W equal 1/2. Thus, for any k-mer w its score $P^W(w) = (1/2)^k$. For $\varepsilon = ((1/2)^{k-1} + (1/2)^k)/2$, all possible $(k-1)$-mers will be calculated with the score of $(1/2)^{k-1} > \varepsilon$. However, no k-mer will be added to S: because $(1/2)^k < \varepsilon$, the final set of phylo-k-mers $|\mathcal{Z}_\varepsilon^W|$ is empty. Thus, the time complexity of Algorithm 4 is $T(k) = \mathcal{O}(\sigma^{k-1}) = \mathcal{O}(\sigma^k)$ if the alphabet Σ is fixed.

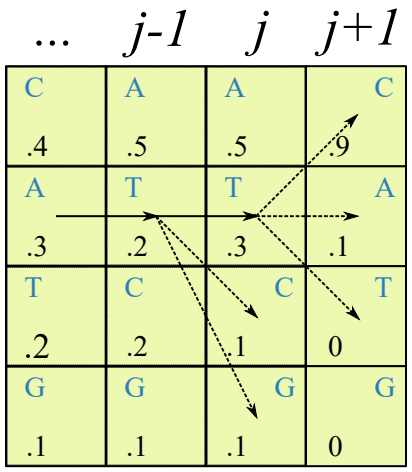


Figure 2-2 – An example of Branch-and-bound computation: if $P^W(ATT) \leq \varepsilon$, scores of k-mers starting $ATT*$ are not computed due to low prefix score. Scores of k-mers starting with $ATC*$, $ATG*$ are also not computed since scores of ATC, ATG are lower than the score of ATT.

Algorithm 4: Branch-and-bound (BB) for TPKC

Input : An integer $k > 0$, a $\sigma \times k$ probability matrix $\widetilde{W}$ with sorted columns, a $\sigma \times k$ matrix R of indices, a threshold ε

Output: An associative array S storing phylo-k-mer scores

Function ComputePhyloKmers($\widetilde{W}$, R):
  $S \leftarrow$ empty associative array
  return BranchAndBound($\widetilde{W}, R, S, 0, 0, 0, 1$)

Function BranchAndBound($\widetilde{W}, R, S, i, j, code, score$):
  if $i, j > 0$ **then**
    $score \leftarrow score \cdot \widetilde{W}_{i,j}$
    $code \leftarrow 2^{\lceil \log_2 \sigma \rceil} \cdot code + R_{ij} - 1$
  if $j = k$ **then**
    $S[code] \leftarrow score$
  else
    for $i' \leftarrow 1 \dots \sigma$ **do**
      if $score \cdot \widetilde{W}_{i',j+1} \leq \varepsilon$ **then**
        break
      $S \leftarrow$ BranchAndBound($\widetilde{W}, R, S, i', j+1, code, score$)
  return S

As for memory complexity, it takes $\mathcal{O}(k)$ for the call stack and $\Theta(|\mathcal{Z}_\varepsilon^W|)$ to store the result, which yields $\mathcal{O}(k + |\mathcal{Z}_\varepsilon^W|)$.

2.3.3 Divide-and-conquer for threshold-based computation

Here I present a novel algorithm for solving the problem of threshold-based phylo-k-mer computation. This algorithm is a modification of Algorithm 3 (DivideAndConquer). It is also based on the divide-and-conquer idea, splitting the window in two parts, calculating scores of prefixes and suffixes recursively, and combining prefix-suffix combination to compute full k-mers. However, a few additional steps are added to avoid computation of scores that are less than ε.

First, once the scores of prefixes and suffixes are computed, we sort suffixes by decreasing score. Second, we change the way prefixes are combined with suffixes: for each prefix l of L, we iterate over suffixes in the score descending order until prefix score is less or equal to ε/(score of l). See Algorithm 5 for the complete listing.

Algorithm 5: Divide-and-conquer (DC-ε) for TPKC

Input : A $\sigma \times k$ probability matrix W, and a threshold ε
Output: An associative array S storing phylo-k-mer scores

```
1  Function ComputePhyloKmers(W):
2      return DivideAndConquerThr(W)
3  Function DivideAndConquerThr(W):
4      S ← empty associative array
5      h ← the number of columns in W
6      if h = 1 then
7          for i ← 1 ... σ do
8              if W_{i,1} > ε then
9                  S[i − 1] ← W_{i,1}
10         return S
11     else
12         L ← DivideAndConquerThr(W[1 : ⌊h/2⌋])
13         R ← DivideAndConquerThr(W[⌈h/2⌉ : h])
14         R′ ← array of pairs ((h/2)-mer, score) of R
15         Sort R′ by score
16         foreach l ∈ L do
17             j ← the last i : R′[i] > ε/L[l]
18             R_l ← R′[1 : j]
19             foreach r ∈ R_l do
20                 code ← l · 2^{c⌈h/2⌉} + r
21                 S[code] ← L[l] · R[r]
22         return S
```

Complexity analysis

Because we do not iterate over the entire list of suffixes for every prefix, this algorithm achieves running time complexity better than $\mathcal{O}(\sigma^k)$. Let again $T(k)$ be the running time of DivideAndConquerThr for the window of size k. I will analyze the running time complexity of this algorithm step by step.

In the analysis of this algorithm, the size of $\mathcal{Z}_\varepsilon^W$ will play an important role. Remember that this is a set of h-mers that have score higher than ε, where h is the number of columns in W. DivideAndConquerThr is recursive by its nature, and each level of recursion has its own subproblem size h and its own set $\mathcal{Z}_\varepsilon^W$.

For the case $h = 1$ (lines 6—10) we return only h-mer-score pairs if corresponding scores surpass the threshold; again as in DivideAndConquer, it takes $\Theta(1)$ time if Σ is fixed. Let us now take a closer look at the case of $h > 1$. Lines 12—13 take $2 \cdot T(h/2)$ time, lines 14—15 take $\Theta(|R| \log |R|)$ time. The outer loop performs $|L|$ iterations, each of which is split in two parts: line 17 taking $\Theta(\log |R|)$ and the rest taking $\Theta(|R_l|)$ time.

Thus, the outer loop starting on line 16 takes $\Theta(\sum_{l \in L}(\log |R| + |R_l|)) = \Theta(|L| \log |R| + |\mathcal{Z}_\varepsilon^W|)$. This gives us the total time of Algorithm 5:

$$T(h) = \begin{cases} \Theta(1) & \text{if } h = 1 \\ 2 \cdot T(h/2) + (|L| + |R|) \cdot \log |R| + |\mathcal{Z}_\varepsilon^W| & \text{otherwise} \end{cases} \tag{2.8}$$

To find a closed-form bound for $T(h)$, let us consider the following recurrence first:

$$T'(h) = \begin{cases} \Theta(1) & \text{if } h = 1 \\ 2 \cdot T'(h/2) + 2 \cdot \sigma^{h/2} \log \sigma^{h/2} + |\mathcal{Z}_\varepsilon^W| & \text{otherwise} \end{cases} \tag{2.9}$$

Theorem 2.3.2. The recurrence $T'(h) = 2 \cdot T'(h/2) + 2 \cdot \sigma^{h/2} \log \sigma^{h/2} + |\mathcal{Z}_\varepsilon^W| = 2 \cdot T'(h/2) + h \cdot \sigma^{h/2} + |\mathcal{Z}_\varepsilon^W|$ implies $T'(h) = \Theta(h \cdot \sigma^{h/2} + |\mathcal{Z}_\varepsilon^W|)$.

Proof. Applying the master theorem, we get $a = 2, b = 2, f(h) = h \cdot \sigma^{h/2} + |\mathcal{Z}_\varepsilon^W|$.

$$\begin{aligned} h^{\log_b a + \epsilon} &= h^2 && \text{for } \epsilon = 1 \\ &\leq h \cdot \sigma^{h/2} && \text{holds for } h \geq 4, \sigma \geq 2 \\ &\leq h \cdot \sigma^{h/2} + |\mathcal{Z}_\varepsilon^W| \\ &= f(h) \end{aligned}$$

Therefore, $f(h) = \Omega(h^{\log_b a + \epsilon})$. Now we need to show that the regularity condition is met: $af(h/b) \leq cf(h)$ for some $c < 1$ and some $h > h_0$. To do this, we should first clarify what is $f(h/b)$: $f(h/b) = h/b \cdot \sigma^{h/2b} + |\mathcal{Z}_\varepsilon^{W/b}|$. Here $|\mathcal{Z}_\varepsilon^{W/b}|$ is the number of phylo-(h/b)-mers whose scores surpass ε in the subwindow of size h/b. Notice that $|\mathcal{Z}_\varepsilon^{W/b}| \leq \sigma^{h/b}$.

	CP	DC	BB	DC-ε
Time	$\Theta(\sigma^k)$	$\Theta(\sigma^k)$	$\mathcal{O}(\sigma^k)$	$\mathcal{O}(k \cdot \sigma^{k/2} + \lvert\mathcal{Z}_\varepsilon^W\rvert)$
Memory	$\Theta(\sigma^k)$	$\Theta(\sigma^k)$	$\mathcal{O}(k + \lvert\mathcal{Z}_\varepsilon^W\rvert)$	$\mathcal{O}(\sigma^{k/2} + \lvert\mathcal{Z}_\varepsilon^W\rvert)$

Table 2.3 – Time and memory complexities of Algorithms 2 to 5 for one window of P.

$$
\begin{aligned}
a \cdot f(h/b) &= 2 \cdot (h/2 \cdot \sigma^{h/4} + |\mathcal{Z}_\varepsilon^{W/2}|) && \text{applying } |Z_\varepsilon^{W/2}| \leq \sigma^{h/2} \\
&\leq h \cdot \sigma^{h/4} + 2 \cdot \sigma^{h/2} \\
&\leq h \cdot \sigma^{h/2-1} && \text{for } h \geq 8, \sigma \geq 2 \\
&\leq h \cdot \sigma^{h/2-1} + 1/\sigma \cdot |\mathcal{Z}_\varepsilon^W| \\
&= 1/\sigma \cdot (h \cdot \sigma^{h/2} + |\mathcal{Z}_\varepsilon^W|) \\
&= c \cdot f(h) && \text{for } c = 1/\sigma
\end{aligned}
$$

Therefore, $af(h/b) \leq cf(h)$ for $c = 1/\sigma$ and $h \geq 8$. The latter completes the proof, implying $T'(h) = \Theta(h \cdot \sigma^{h/2} + |\mathcal{Z}_\varepsilon^W|)$. □

Theorem 2.3.3. The recurrence 2.8 implies $T(h) = \mathcal{O}(h \cdot \sigma^{h/2} + |\mathcal{Z}_\varepsilon^W|)$.

Proof.

Since $|L| \leq \sigma^{h/2}$ and $|R| \leq \sigma^{h/2}$, we can conclude that $T(h) = \mathcal{O}(T'(h))$. The statement follows immediately after Theorem 2.3.2.

□

As for memory complexity, it takes $\mathcal{O}(\sigma^{h/2})$ memory to store L and R, $\Theta(\log h)$ for the call stack, and $\mathcal{O}(|\mathcal{Z}_\varepsilon^W|)$ to store the result. Thus, the overall memory complexity is $\mathcal{O}(\sigma^{h/2} + |\mathcal{Z}_\varepsilon^W|)$. The time and memory complexities of all considered algorithms are summarized in Table 2.3.

2.3.4 Chained Windows Technique

Original problems are defined on a matrix of size $\sigma \times m$, and in practice $m >> k$. Algorithms described above solve them for each window separately. However, DC and DC-ε can be naturally improved by exploiting the following observation: suffixes of one window are prefixes of another window. More precisely, the set of phylo-n-mers R constructed for a window $W_j = P[j : j + k - 1]$ can be used as the set L for the window $W_{j+k/2} = P[j + k/2 : j + k/2 + k - 1]$. Thus, calculation of prefixes can be omitted. To enable this optimization, we can iterate over windows of P straightforwardly while keeping sets of suffixes R in memory for the last $k/2$ windows. This will take $\mathcal{O}(k \cdot |\mathcal{Z}_\varepsilon^{W/2}|)$ in memory apart from memory taken by the result. If k is even, we can improve it to $\mathcal{O}(|\mathcal{Z}_\varepsilon^{W/2}|)$ by iterating over windows with the step of $k/2$, applying the technique I call *chained windows* (see Figure 2-3).

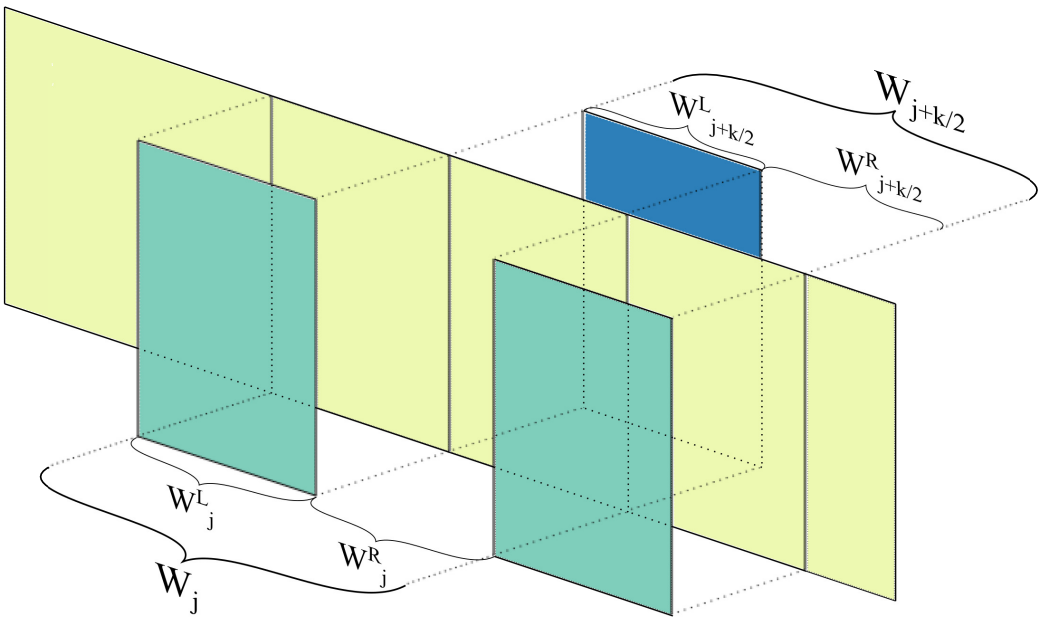


Figure 2-3 – Chaining window technique. The right subwindow W_j^R of window W_j is the left subwindow of $W_{j+k/2}$. We reuse suffix scores calculated in W_j as prefix scores in $W_{j+k/2}$.

This technique does not change the algorithmic complexity of the algorithm; nevertheless, as we will see later, it improves the algorithm's running time in practice.

2.4 Experimental results

This section describes the experimental evaluation of the algorithms for computing threshold-based phylo-k-mers, i.e. solving TPKC. Although the exhaustive version of the problem is indeed interesting, calculating all phylo-k-mers is impractical. In this regard, comparing different algorithms to solve EPKC is beyond the scope of our consideration.

2.4.1 Experimental setup

Implementation

All the algorithms compared were developed during the development of XPAS, a phylo-k-mer database building application. The algorithms are implemented in C++. Phylo-k-mer computation is only one step of the process of constructing a phylo-k-mer database, and I do not go into detail about implementing the whole pipeline. A more detailed description of the implementation details is given in Chapter 1 (section 1.5) and Chapter 4.

Experiment design

For every dataset, two ghost nodes per branch were introduced for all branches of the reference tree. For every ghost node, phylo-k-mers that surpass the threshold value of $\varepsilon = (\omega/\sigma)^k$ were computed. For every dataset, experiments include computing phylo-k-mers of different lengths k; thereby a different value of ε is set for each experiment. Notice that for a fixed dataset, values of parameters k and ω determine the sizes of $\mathcal{Z}_\varepsilon$ for every ghost node — that is, for the same dataset and the same ghost node u, the sets of phylo-k-mers computed $\mathcal{Z}_\varepsilon$ are different for different values of k and σ. Total time required to compute phylo-k-mers for every ghost node was measured with a steady monotonic timer (`std::chrono::steady_clock`).

Since different datasets have different number of reference sequences, and therefore different number of ghost nodes introduced, one should not compare time measurements between two different datasets; only a comparison of the performance of different algorithms for the same dataset is valid. The same is true for different parameter values: one can only compare relative performance of algorithms for a fixed set of parameters of the same dataset. No results retrieved with different parameter values should be compared.

Hardware and data

Experiments were run on a single core of a computer equipped with Intel(R) Xeon(R) W-2133 CPU @ 3.60GHz / 8.25 MB Cache / 62 GB RAM. Three datasets of DNA sequences — D218 (bacterial 16S rRNA from [20]), D500 (chloroplast rbcL gene sequences from [20]), NEOTROP (eukaryotic 18S rRNA sequences from [15]) — and one dataset of amino acid sequences — D140 (whole-genome sequences of Papillomaviridae from [20]) — were used. All these datasets were used previously for comparing accuracy of phylogenetic placement tools.

2.4.2 Running time

Total time measured to compute phylo-k-mers for different datasets and parameter values is given by Figures 2-4 and 2-5. Computation time is measured in seconds and depicted in a log-scale. BRANCHANDBOUND (referred to as BB in the legend) shows relatively comparable results with DIVIDEANDCONQUERTHR (referred to as DC-ε if no chained windows technique was applied and as DCCW-ε otherwise). Despite a poorer theoretical bound on running time in the worst case scenario, BRANCHANDBOUND demonstrates the same level of growth with k for DNA experiments (both NEOTROP and D218, $\omega = 1.0, 2.0$) as DIVIDEANDCONQUERTHR. Although BB is faster for the protein dataset and $\omega = 10$ up to $k = 10$, the running times of DC-ε and DCCW-ε increase more slowly with k, suggesting that they may perform better than BB for larger values of k.

From the data presented, I can suggest that BRANCHANDBOUND is faster than DIVIDEANDCONQUERTHR in cases of small numbers of phylo-k-mers (low values of k and/or high values of ω). DIVIDEANDCONQUERTHR is preferable for both increasing the k-mer length and the number of calculated phylo-k-mers (decreasing ω).

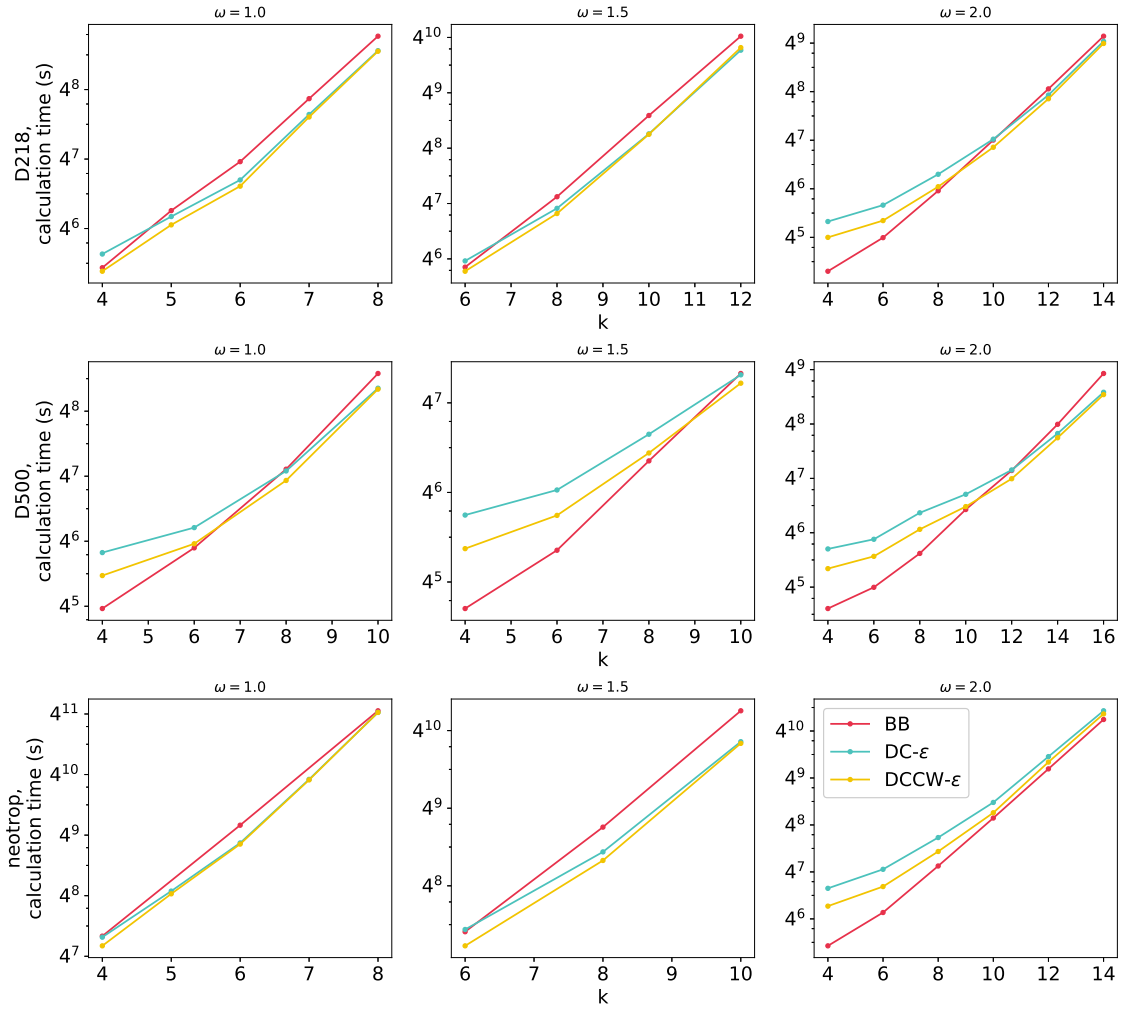


Figure 2-4 – Total running time of phylo-k-mer computation for three DNA datasets (alphabet size $\sigma = 4$) and the threshold value of $\varepsilon = (\omega/\sigma)^k$. Different columns show running times for $\omega = 1$, 1.5, and 2, which corresponds to solving TPKC with different thresholds.

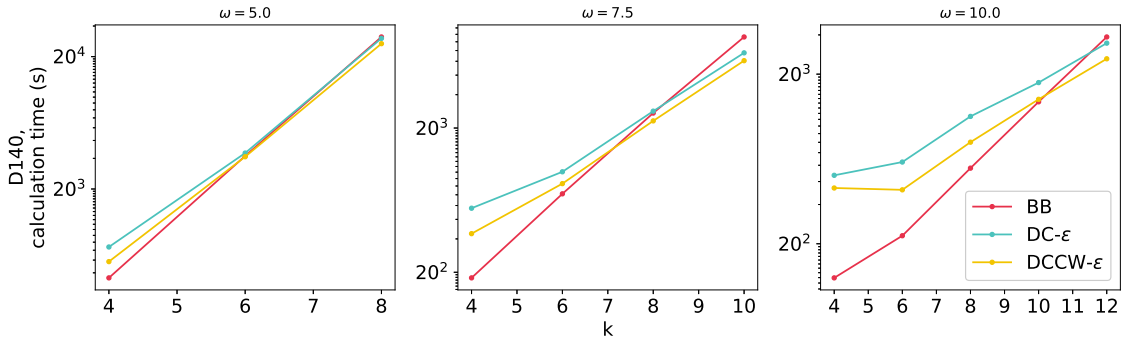


Figure 2-5 – Total running time of phylo-k-mer computation for the protein dataset (alphabet size $\sigma = 20$) and the threshold value of $\varepsilon = (\omega/\sigma)^k$.

2.5 Conclusion

In this chapter, I described the problem of computing phylo-k-mers for one ghost node. This is the central algorithmic problem of RAPPAS, as the running time for reference dataset preprocessing consists mainly of solving this problem. First, I considered the problem of exhausting phylo-k-mer computation. Even though this problem is easier, it provides us insights for solving the real problem of interest, threshold-based phylo-k-mer computation.

It was not described before how RAPPAS solves this problem. I described this algorithm, BranchAndBound, and showed that it is suboptimal in the worst case. Besides that, I suggested a new algorithm based on the divide-and-conquer principle. Theoretical analysis shows it to be better than BranchAndBound in the worst-case; however, it is not clear what happens with the latter on average. Both algorithms are implemented as a part of XPAS, a reimplementation of RAPPAS, which will be covered in Chapter 4.

Apart from this, I suggested an interesting optimization of the DivideAndConquerThr applied for consecutive windows of the alignment, the chained windows technique. While it does not improve the algorithm's complexity, it improves its running time for small values of k and ε. The underlying idea may be interesting for algorithm researchers and may be helpful in other algorithmic problems.

Finally, I held experiments for DNA and protein datasets, measuring the total running time of phylo-k-mer computation with different sets of input parameters. Experiments show contradictory results depending on the dataset and input parameters: BranchAndBound showed better performance for low values of k, as expected; DivideAndConquerThr outperforms it for higher values of k. For the default parameter values of XPAS, $k = 10$ and $\omega = 1.5$, the new algorithm showed better performance for all DNA datasets.

Chapter 3

Informative phylo-k-mers

In the previous chapter, we reformulated the exhaustive phylo-k-mer computation problem as a threshold-based problem. That allows to compute and store significantly fewer phylo-k-mers, possibly with a little loss of accuracy of phylo-k-mer based applications. However, phylo-k-mer databases can still reach gigabytes in size, which limits their application. In this chapter, I suggest another approach to trade-off accuracy for smaller phylo-k-mer database size in the context of phylogenetic placement. In addition to approximating phylo-k-mer scores with the threshold values, we can exclude some phylo-k-mers from the database. To do this, we need to answer the following questions: Are all phylo-k-mers necessarily *informative* for phylogenetic placement? If not, which phylo-k-mers are informative and which are not? To what extent are they informative? How is it possible to find the most informative phylo-k-mers?

Those questions arise from the following intuition. Consider two k-mers that are very close in terms of edit distance. We can assume that they are more likely to have closer scores than a pair of highly distant k-mers. In other words, vectors of scores for two k-mers can be correlated, and phylo-k-mers as features (in the machine learning sense of this word) for phylogenetic placement can be redundant. From this we can foresee that some phylo-k-mers can be safely excluded from consideration with little effect on the accuracy of phylogenetic placement. This is not only an intuition: experiments confirm the validity of those assumptions. Later in this chapter, I will show that even if we randomly exclude gradually increasing numbers of phylo-k-mers from the database, the placement accuracy decays very slowly. Of course, I propose better methods of finding informative phylo-k-mers than just random selection. This chapter is devoted to the presentation and discussion of such methods.

We will start approaching the problems of this chapter by reformulating phylogenetic placement as a classification problem. First, I discuss how the method of RAPPAS is related to text classification, namely, Naive Bayes text classification. Afterward, I look into existing feature selection methods and discuss how these methods can be applied for finding informative phylo-k-mers. Finally, I give an experimental evaluation of such applications: we will see that the methods I suggest allow excluding a significant part of phylo-k-mers with a negligible effect on the accuracy of phylogenetic placement.

3.1 Phylogenetic placement as classification

Classification is a classic machine learning task that arises in many areas of science and engineering. It consists of assigning data points to one of a set of pre-defined classes based on the set of attributes or *features* of the data points [113, 153]. Classification is performed by a classifier that can be defined as an algorithm computing a mapping function $\hat{y}$ from input variables X to a discrete class variable Y:

Definition 3.1.1 (Classifier)**.** A classifier is an algorithm that computes a function $\hat{y} : X \to Y$, where the elements of X are called *instances* or *queries*, and Y is a finite set, whose elements are called *classes*.

Let us consider phylogenetic placement as a classification problem. Again, consider a reference phylogenetic tree T_0 and a set A_0 of reference (possibly aligned) sequences over alphabet Σ in one-to-one correspondence with leaves of T_0. For a large collection of *query sequences* Q over Σ, phylogenetic placement aims at inserting every query sequence $q \in Q$ into T_0. Phylogenetic placement outputs an assignment of q to a branch of T_0 or multiple assignments if the placement is uncertain. For simplicity, let us assume that the result of phylogenetic placement for every query q is just a single branch of the tree representing the best placement for this query according to the algorithm. Then, a phylogenetic placement algorithm is a classifier that assigns query sequences to branches of the reference phylogeny, i.e., $X = \Sigma^*$ and $Y = E(T_0)$, where $E(T_0)$ denotes the set of branches of the reference tree.

Problem 3 (Phylogenetic placement as classification)**.**
Input: A set of possibly aligned reference sequences; a reference tree T_0 whose leaves correspond to the reference sequences and whose branches define the set of classes $Y = \{y_1, y_2, \ldots, y_N\}$; a set Q of query sequences.
Output: $\hat{y}(q) \in Y$, the predicted class of q for every $q \in Q$.

In this chapter, I only consider phylogenetic placement as performed by RAPPAS. The formulation above will allow us to find correspondence between the RAPPAS approach and the well-studied problem of text classification. However, there is an essential difference between phylogenetic placement classification and the common understanding of classification in machine learning. The latter is a *supervised learning* method: the classifier is constructed on the basis of a *training set*, i.e. the set of classified examples $\{(x, y) : x \in X, y \in Y\}$. Classification algorithms use the training set to reconstruct the intrinsic relationship between X and Y, and the learning methods are domain-agnostic (for example, Support Vector Machines can be applied to image classification, text classification, and many other classification tasks). Contrary to that, modern phylogenetic placement methods do not perform any supervised learning. There is no training set in the traditional sense: no examples of query sequences placed to the reference phylogeny are given to the classifier. Instead, it gets *only* the reference phylogeny and the set of reference (possibly aligned) sequences as input. To place unknown sequences to the reference tree, the classifier exploits domain knowledge of phylogenetics to reconstruct the relationship between X and Y.

For example, RAPPAS infers the classifier from the reference data as described in Sections 1.5 and 1.6.

Having this difference in mind, let us look at the similarities between RAPPAS and text classification methods. Understanding those similarities will allow us to answer the questions of the chapter.

3.2 Text Classification with Naive Bayes

The task of classifying texts has become very important in recent decades due to the development of the Internet and the spread of computers in general. The task is to label texts of arbitrary length with pre-determined classes. A considerable amount of literature has been published on this topic, and describing existing methods is beyond the scope of this thesis. An interested reader may find [4], [192] and [114] useful as detailed overviews of those methods. Here, I briefly introduce only one method for text classification, that is, Naive Bayes text classification. While being one of the simplest classification methods, Naive Bayes shows superb classification performance despite its rather unrealistic assumptions, showing excellent performance for text classification [235, 149, 180, 168, 209].

Being a supervised learning method, Naive Bayes (NB) Classification involves two stages: learning on a training set and classification itself. Let V be a vocabulary, a set of words of the language of choice. A sequence of elements of V of arbitrary length is called a document. Classification of documents can be formalized in the following manner.

Problem 4 (Supervised Learning for Text Classification)**.**
Input: A set of document classes $Y = \{ y_1, y_2, \ldots, y_N \}$; a training set of classified documents $\{ (x_1, y_{j_1}), (x_2, y_{j_2}), \ldots, (x_M, y_{j_M}) \}$, $x_i \in V^*$.
Output: a classifier $V^* \to Y$.

The vocabulary V can be the set of all words encountered in training documents or an arbitrary superset of such a set. The classifier ranks available classes according to the amount of evidence that a query document $q \in V^*$ belongs to each class; alternatively, it selects the highest-rank class as a single answer [192]. *Probabilistic classifiers* implement this idea by estimating conditional probabilities $\mathbb{P}(y \mid q)$ for any class $y \in Y$ and label q with the class that obtained the maximal probability:

$$\hat{y}(q) = \arg\max_{y \in Y} \mathbb{P}(y \mid q) \tag{3.1}$$

According to Bayes' theorem, the posterior probability $\mathbb{P}(y \mid q)$ can be expressed as:

$$\mathbb{P}(y \mid q) = \frac{\mathbb{P}(q \mid y)\mathbb{P}(y)}{\mathbb{P}(q)} \propto \mathbb{P}(q \mid y)\mathbb{P}(y) \tag{3.2}$$

Values of $\mathbb{P}(q)$ from Equation 3.2 do not play a role in classification since they are class-independent. $\mathbb{P}(y)$ can be estimated from training data. However, values

of $\mathbb{P}(q \mid y)$ are not straightforward to estimate. Here, the basic assumption of Naive Bayes comes into play: we represent the query document as a vector of *features* (e.g., document word counts) and assume that they are statistically independent, given the class y. This assumption is clearly violated for texts in natural languages; nevertheless, it does not make the method unreliable in practice. The assumption of independence allows us to express $\mathbb{P}(q \mid y)$ as the product of the posterior probability of each feature given y. Next, we will look into the two most popular methods of representing a text document as a vector of features and show how to derive $\mathbb{P}(q \mid y)$ in practice.

3.2.1 Bernoulli model

Under this model, q is represented as a vector of binary values (i.e., zeros or ones) indicating whether a word occurs in the document or not:

$$q = (b_{w_1}, b_{w_2}, \ldots, b_{w_{|V|}}) \tag{3.3}$$

$$b_w = \mathbb{1}\{\exists j : W_j = w\} \tag{3.4}$$

Here W_j is the random variable denoting the jth word of the input document. The Bernoulli model assumes that the document is a result of $|V|$ independent Bernoulli trials, one for every word of the vocabulary V. The method is to estimate the parameters of those Bernoulli distributions for every class. To formalize this, let us denote by

$$S_y^B(w) := \mathbb{P}[\exists i : W_i = w \mid y] \tag{3.5}$$

the probability of the word w to be present in the document of class y. Note that for any word w the probability $S_y^B(w) \in [0, 1]$. The assumption of independence of the features given the class y allows us to express the probability of the document given its class as follows:

$$\mathbb{P}(q \mid y) = \prod_{w \in V} \underbrace{S_y^B(w)^{b_w}}_{\substack{\text{present} \\ \text{words}}} \underbrace{(1 - S_y^B(w))^{(1-b_w)}}_{\substack{\text{absent} \\ \text{words}}} = \prod_{w:b_w=1} S_y^B(w) \prod_{w:b_w=0} (1 - S_y^B(w)) \tag{3.6}$$

Thus, the calculation of $\mathbb{P}(q \mid y)$ is reduced to the calculation of $S_y^B(w)$. Those can be estimated from the training set straightforwardly as the frequency of the word w in documents of class y; different regularizations such as Laplace smoothing can be applied to avoid estimated probabilities be zero [149].

3.2.2 Multinomial model

The Multinomial model captures not only the presence of the word in the document but also the word frequency in the document. Under this model, we represent the

document q as a vector of word occurrences:

$$q = (n_{w_1}, n_{w_2}, \ldots, n_{w_{|V|}}) \tag{3.7}$$

$$n_w = \text{the number of times } w \text{ occurs in } q. \tag{3.8}$$

In this representation, n_w can be greater than 1, unlike the values of b_w in the Bernoulli model. The model assumes that q is a sequence of independent word events drawn with replacement from a class-dependent multinomial distribution over all vocabulary words. Such distributions are defined by values of

$$S_y^M(w) := \mathbb{P}[W_j = w \mid y]. \tag{3.9}$$

Here $S_y^M(w)$ is the probability to sample w for the class y in one sampling event. This implies not only $S_y(w) \in [0, 1]$ as for the previous model, but as well the fact that $S_y^M(w)$ of all words for the same class sum to one:

$$\sum_{w \in V} S_y^M(w) = 1. \tag{3.10}$$

Finally, the probability of the document q given its class y is:

$$\mathbb{P}(q \mid y) = \frac{(\sum_{w \in V} n_w)!}{\prod_{w \in V} n_w!} \prod_{w \in V} \left(S_y^M(w)\right)^{n_w}. \tag{3.11}$$

Note that $\frac{(\sum_{w \in V} n_w)!}{\prod_{w \in V} n_w!}$ does not depend on the class and contributes equally to each y, which gives us:

$$\mathbb{P}(q \mid y) \propto \prod_{w \in V} \left(S_y^M(w)\right)^{n_w} = \prod_{w : n_w > 0} \left(S_y^M(w)\right)^{n_w} \tag{3.12}$$

Again, $S_y^M(w)$ can be estimated from training data. See [149] for a detailed discussion about both models and their performance for text classification.

3.3 RAPPAS as a Naive Bayesian Classifier

In Section 3.1, we have seen that phylogenetic placement can be treated as a classification task. For the rest of this chapter, I will consider the method of RAPPAS (described in detail in Section 1.5 and 1.6) from this point of view. While not being straightforwardly the same, RAPPAS classification has a lot in common with the Naive Bayes text classification described above. To make it easier to relate the method of RAPPAS to those for text classification, let us break it down in the same way as we did for Naive Bayes classification methods.

3.3.1 Query representation

A query document q — i.e., a DNA sequence or a protein sequence — is represented by the multiset of all substrings of q of size k. Note that if we ignore all k-mers with zero counts, this representation is the same as the one used in the Multinomial Naive Bayes text classifier. Let us consider an example for clarity.

Example 3.3.1.
Query: AACTGACT, $k = 3$.
Representation: { AAC: 1, ACT: 2, CTG: 1, TGA: 1, GAC: 1 }.

Let the vocabulary V be the set of all possible k-mers over the alphabet Σ. Then, we can define the representation of q in the same terms as in Equation 3.7 and 3.8:

$$q = (n_{w_1}, n_{w_2}, \ldots, n_{w_{\sigma^k}}) \tag{3.13}$$

$$n_w = \text{the number of times } w \text{ occurs in } q. \tag{3.14}$$

However, we can assume that RAPPAS uses values of k that guarantee that no k-mer is found more than once in a query sequence, i.e., $n_w \in [0, 1]$. It can be justified as follows: k-mers must be long enough that the presence or absence of a k-mer is informative about its phylogenetic origin. If a k-mer occurs at multiple positions in the query, then that k-mer does not point at a unique placement in the phylogeny, and the value of k is too small relative to the query size. Thus, in practice, queries are almost always represented in the same way as in the Bernoulli model.

3.3.2 Model parameters

While the query representation in RAPPAS matches the one used in the Multinomial model, the meaning of parameters $S_y(w)$ of word distributions is different. As will be explained more thoroughly in Section 5.1.2, RAPPAS aims at estimating conditional probabilities

$$\mathbb{P}\left(q \text{ contains } w \mid |q| = m; q \text{ originates from } y\right) \tag{3.15}$$

for every word w given the class y, where m is the length of the reduced reference alignment (see Section 1.5.2). The estimations of those probabilities are the phylo-k-mer scores $S_y(w)$ discussed in Chapter 1. Efficient algorithms for computing these values are discussed in Chapter 2. Here I summarize the whole computation of $S_y(w)$ in one formula for the sake of completeness (please refer to previous sections for detail):

$$S_y(w) = \max_{u \in G_y} \max \left\{ \max_{j=1}^{m-k+1} \prod_{l=1}^{k} p_{j+l-1}(w_l), \varepsilon \right\} \tag{3.16}$$

Here k is the length of k-mers used; G_y is the set of ghost nodes injected for y; ε is the score threshold value, and $p_j(a)$ are elements of P^u matrix calculated in the process of ancestral reconstruction for the ghost node u (see Section 1.5.2).

If we assume that the query sequences are results of independent multinomial sample events, it is not straightforward to understand whether this formula gives accurate estimates of the parameters of the underlying model or not[1]. I assume the latter: one may speculate that Equation 3.16 does not produce Bayes-optimal estimates of required parameters. However, it has proven to be a good solution for estimating phylo-k-mer scores that are accurate enough for phylogenetic placement [130]. Whether there are better ways to estimate phylo-k-mer scores lies out of the scope of this chapter; I take this formula as-is since it is the only one that has been implemented in RAPPAS so far. Refer to Section 5.1.2 for a discussion about an alternative formula for $S_y(w)$.

As in both the Bernoulli and Multinomial models, every value $S_y(w)$ belongs to the range $[0, 1]$. However, unlike the Multinomial model, RAPPAS does not require the scores to sum to one, and in general $\big(\sum_w S_y(w)\big) \in [0, \sigma^k]$.

3.3.3 Classification

As both the Bernoulli and the Multinomial Naive Bayes, RAPPAS makes the *naive Bayesian assumption* about the statistical independence of features given the class:

$$\begin{aligned}\mathbb{P}(q \text{ is composed of } w_1, w_2, \ldots, w_{m-k+1} \;\big|\; |q| = m; y) \propto\; & \mathbb{P}\big(w_1 \text{ in } q \;\big|\; |q| = m; y\big) \times \\ & \mathbb{P}\big(w_2 \text{ in } q \;\big|\; |q| = m; y\big) \times \\ & \times \ldots \times \\ & \mathbb{P}\big(w_{m-k+1} \text{ in } q \;\big|\; |q| = m; y\big)\end{aligned}$$

Note that the probabilities on the right-hand side of the equation above are just short forms of the probability in Equation 3.15. Because RAPPAS uses $S_y(w)$ as an estimate of $\mathbb{P}\big(w \text{ in } q \;\big|\; |q| = m; y\big)$, the equation above is computed by RAPPAS as:

$$\prod_{w:n_w>0} S_y(w)^{n_w}. \tag{3.17}$$

Seeking the class y that maximizes (3.17) is equivalent to computing

$$\hat{y}(q) = \arg\max_{y \in E(T_0)} \prod_{w:n_w>0} S_y(w)^{n_w} = \arg\max_{y \in E(T_0)} \sum_{w:n_w>0} n_w \log S_y(w) \tag{3.18}$$

(equivalent to Equation 1.6 on page 38). Note that, with the exception of $S_y(w)$ replacing $S_y^M(w)$, the criterion above coincides with the formula for $\mathbb{P}(q \mid y)$ of the Multinomial model (Equation 3.12).

In conclusion, RAPPAS uses the same query representation as a Multinomial NB

[1]To be honest, just one look at this formula immediately made me question the validity of RAPPAS' approach. Only substantial empirical evidence of RAPPAS placement accuracy made me believe that this formula estimates the underlying probabilities accurately enough.

classifier, and its formula for classification is also related to that of a Multinomial NB classifier. However, the meaning of the model parameters $S_y(w)$ is different and corresponds to the meaning of the Bernoulli model parameters. Moreover, there is a way to show a strong link between RAPPAS and the Bernoulli Naive Bayes classification, which will follow in the next section.

3.4 Connecting RAPPAS and Bernoulli Naive Bayes

3.4.1 Bernoulli-based phylogenetic placement

To understand how RAPPAS is connected to the Bernoulli model, let us introduce a Bernoulli-based phylogenetic placement classifier. How would it be possible to adapt the text classification approach to phylogenetic placement? Let us now look at all three parts of the model: query representation, estimating model parameters, and the classification itself. The first is given by Equations 3.3 and 3.4 on page 62, which I will quote here again for convenience:

$$q = (b_{w_1}, b_{w_2}, \ldots, b_{w_{\sigma^k}}) \tag{3.19}$$

$$b_w = \mathbb{1}\{\exists i : W_i = w\}. \tag{3.20}$$

It matches the query representation of RAPPAS under the assumption that k is large enough to guarantee that $n_w \in [0, 1]$ for all w, which yields $b_w = n_w$. The classification is given by Equations 3.1, 3.2 and 3.6, which are equivalent to:

$$\hat{y}(q) = \arg\max_{y \in Y} \mathbb{P}(q \mid y)\mathbb{P}(y) \tag{3.21}$$

$$\mathbb{P}(q \mid y) = \prod_{w:b_w=1} S_y^B(w) \prod_{w:b_w=0} (1 - S_y^B(w)). \tag{3.22}$$

We can assume the uniform prior $\mathbb{P}(y)$ over branches of the tree, meaning that $\mathbb{P}(y) = 1/N$ for all classes y, which yields:

$$\hat{y}(q) = \arg\max_{y \in Y} \Big(\prod_{w:b_w=1} S_y^B(w) \prod_{w:b_w=0} (1 - S_y^B(w)) \Big) \tag{3.23}$$

The most challenging question is how to estimate the model's parameters, i.e., how to calculate values of $S_y^B(w)$. There could be two ways of doing this. The first is to obtain many classified query sequences for each class, similar to how it is done in text classification; this approach is mainly theoretical since we do not have such data in the framework of phylogenetic placement. The second way is to estimate the parameters from the reference phylogeny, as RAPPAS does by calculating $S_y(w)$. However, we can not simply take $S_y(w)$ as parameters instead of $S_y^B(w)$: $S_y(w)$ is an approximation of the probability of observing a k-mer w in a query q *of the size of the alignment*, given that q originates from the branch y (Equation 3.15). On the

other hand, $S_y^B(w)$ should approximate the probability of observing a k-mer w in a query q *of any size*, given that q originates from y. In other words, we can assume that $S_y^B(w) = S_y(w)$ only if placed queries are as long as the reference alignment, which often is not the case. However, even for shorter queries, we can derive $S_y^B(w)$ from $S_y(w)$ knowing the query size. Let us assume that q is generated by randomly sampling k-mers without replacement out of another sequence q' with $|q'| = m$. Then, we can introduce a query length correction parameter denoted by f:

$$f = \frac{|q| - k + 1}{m - k + 1} \tag{3.24}$$

$$\begin{aligned} S_y^B(w) &= \mathbb{P}\left(q \text{ contains } w \mid |q| \leq m; q \text{ originates from } y\right) \\ &= f \cdot \mathbb{P}\left(q' \text{ contains } w \mid |q'| = m; q' \text{ originates from } y\right) \\ &\approx f \cdot S_y(w). \end{aligned} \tag{3.25}$$

Estimating the parameters in this way allows us to rewrite Equation 3.23 as follows:

$$\hat{y}(q) = \arg\max_{y \in Y} \Big(\prod_{w:b_w=1} f S_y(w) \prod_{w:b_w=0} (1 - f S_y(w)) \Big) \tag{3.26}$$

which completes the definition of the Bernoulli-based phylogenetic placement.

3.4.2 Query length correction for RAPPAS

The careful reader may have noticed that the parameter f did not appear in the formula of RAPPAS classification (Equation 3.18). However, the same reasoning about the query length can be applied if we place queries of shorter size than m by RAPPAS. I believe that RAPPAS should use $fS_y(w)$ as parameters of classification instead of $S_y(w)$. However, this is not happening, and the reason for this is very simple. Since the parameter f is class-independent, the query length correction does not contribute to the classification:

$$\prod_{w:n_w>0} \Big(f S_y(w) \Big)^{n_w} \propto \prod_{w:n_w>0} \Big(S_y(w) \Big)^{n_w}. \tag{3.27}$$

However, as we will see later in this chapter, f does play a role in the process of finding informative k-mers. But before we get to that, I will describe a formal link between RAPPAS and the Bernoulli-based phylogenetic placement.

3.4.3 The link between RAPPAS and Bernoulli phylogenetic placement

Let us now look at a slightly modified Bernoulli placement classifier in which the parameter f can take any value in the range $[0, 1]$ regardless of the query length. I will call it *f-weighted Bernoulli* placement. For the value $f = (|q|-k+1)/(m-k+1)$ it

corresponds to the Bernoulli-based phylogenetic placement as introduced above. I will now show that, interestingly, when $f \to 0$, this classifier is equivalent to RAPPAS.

Let us denote by $\hat{y}_f^B(q)$ the prediction of the f-weighted Bernoulli classifier (Equation 3.26 with an arbitrary value of f) for a particular query q, and by $\hat{y}^R(q)$ the prediction of RAPPAS (Equation 3.18) for the same query.

Theorem 3.4.1 (The connection theorem)**.** If $n_w = b_w$ for any k-mer w in a query q, then $\exists f_0 > 0$ such that $\forall f : 0 < f < f_0$, $\hat{y}_f^B(q) = \hat{y}^R(q)$, that is, f-weighted Bernoulli is equivalent to RAPPAS for f sufficiently small.

Proof. By Equation 3.27, the prediction of RAPPAS does not change if we change the scoring of classes from

$$\prod_{w:n_w>0} \Big(S_y(w)\Big)^{n_w}$$

to

$$\prod_{w:n_w>0} \Big(fS_y(w)\Big)^{n_w}$$

and since $n_w = b_w$, then

$$\prod_{w:n_w>0} \Big(fS_y(w)\Big)^{n_w} = \prod_{w:b_w=1} fS_y(w).$$

Let us consider a class y and the ratio of scores given to this class for the query q by the two classifiers:

$$\lim_{f\to 0} \frac{\prod_{w:b_w=1} fS_y(w) \prod_{w:b_w=0}(1 - fS_y(w))}{\prod_{w:n_w>0} \Big(fS_y(w)\Big)^{n_w}} =$$

$$\lim_{f\to 0} \left(\frac{\prod_{w:b_w=1} fS_y(w)}{\prod_{w:b_w=1} fS_y(w)} \prod_{w:b_w=0} (1 - fS_y(w)) \right) =$$

$$\lim_{f\to 0} \prod_{w:b_w=0} (1 - fS_y(w)) = 1.$$

The statement follows if we apply the reasoning above to all classes $y \in E(T_0)$. □

Thus, we can see that RAPPAS is a Naive Bayes classifier and is equivalent to a specific version of Bernoulli Naive Bayes classification under certain conditions. I describe this connection for the first time here, and this connection will help us find informative phylo-k-mers. The reader may have guessed that the task of finding informative phylo-k-mers is virtually feature selection. The connection between the classifiers described in the sections above suggests that feature selection methods applied for Naive Bayes text classification can be applied to RAPPAS. The following section will introduce these methods and explain their place in the rich world of feature selection.

3.5 Feature selection

3.5.1 Dealing with high-dimensional feature spaces

Many supervised and semi-supervised learning tasks have to deal with massive sets of features where the number of features is in the tens of thousands or more. It may seem easier for algorithms to learn from data if they confront more detail about every data point (i.e., learn in a space of higher dimension), but at a certain size of the feature space, algorithms start to suffer from *the curse of dimensionality* [133]. This phenomenon happens if the number of features is too large relative to the number of training data points: in those high-dimensional spaces, data points tend to become equidistant, and learning algorithms tend to overfit the training set. Apart from that, learning from datasets that are rich in features may be challenging for other reasons. Irrelevant and redundant features can mislead learning algorithms, making it harder to spot important relationships in the training data [123, 122, 111, 166]. Thus, eliminating irrelevant and redundant features can improve classification performance [110, 37, 8]. A good set of features contains only features that are highly correlated to the class and independent from each other [86]. Ideally, the feature space size should correspond to the intrinsic dimensionality of the training data, which, however, may be difficult to guess in advance.

There are multiple ways of dealing with excessive dimensionality. The first way involves examining the existing features and their interdependencies and building fewer new features from them. Such features must preserve necessary information to predict the class variable; of course, there must be a way to translate unclassified data samples into the new feature space. Those are methods of *dimensionality reduction* and *feature extraction*, which include Principal Component Analysis and its extensions [163, 229, 2, 188], Linear Discriminant Analysis and its extensions [67, 154, 233, 54], techniques of multidimensional scaling (MDS, e.g., Principle Coordinate Analysis, metric and nonmetric MDS [116, 117, 118, 40, 26]), and others, such as Independent Component Analysis [98].

Another way of tackling this problem is thoughtful and accurate manual feature engineering. Mainly being an ad hoc practice, it requires a deep understanding of the data and the meaning of every feature. Features can be transformed and eliminated by experienced data scientists based on their domain knowledge. While it can be beneficial for our goals, the scalability of manual feature engineering is limited: processing every new dataset requires manual interventions. Those who are interested in feature engineering should be referred to [236]. In this work, I only consider fully automated methods since in phylo-k-mer based applications, the number of features easily achieves hundreds of thousands, and manual feature engineering seems to be very challenging in this case.

Finally, the feature space can be reduced via *feature selection*, a family of automatized methods for picking important and meaningful features. It is also referred to as *semantics-preserving dimensionality reduction* [101] since it does not change the meaning of the features, leaving them easy to interpret. There are two key aspects of this process: feature evaluation and search strategies. Feature evaluation determines

what features are selected; it will be covered below. The selected subset of features can be evaluated to determine the quality of this subset according to different criteria. Search strategies answer how to progress with selecting the next subset of features from the previous one. It is not always possible to evaluate all possible subsets of features since an exhaustive evaluation takes exponential time; instead, we can apply different heuristic algorithms to visit the feature subset space such as genetic algorithms [88, 10, 207, 231, 162], simulated annealing [109, 128, 151, 1], greedy algorithms (such as forward and backward elimination) and others [81, 225, 144]. Feature selection has been intensively studied for at least two decades: classic overviews of those methods are [81, 156, 23, 48]; [133] gives a comprehensive overview of feature selection, including methods for text classification and its applications in bioinformatics. A broad overview of feature selection methods in bioinformatics can also be found in [179], while [218] is another excellent survey on applications of feature selection methods. The most common classification of feature selection methods splits them into three categories: wrappers, filters, and embedded methods. In this work, I use this classification, which will be covered in the next section. Other classifications also exist: [218] categorizes feature selection methods into exhaustive, heuristic, and hybrid methods. [125] classifies feature selection methods based on the principles applied to data processing: similarity-based, information theory-based, sparse learning-based, and statistical-based methods. The method for selecting informative phylo-k-mers described in this work belongs to the category of information theory-based methods.

In this work, I only consider feature selection methods and not feature extraction or manual engineering. They are straightforward yet powerful, and, in addition, they do not modify the feature space, only eliminating specific features. It is not the case for other methods. The next section will introduce the most popular classification of feature selection methods.

3.5.2 Approaches to feature selection

Wrappers and embedded methods

Wrappers use predictive models as a black-box to evaluate predictive power of a subset of features [81, 34, 110]. Using wrappers require three stages. At the beginning, a candidate feature subset is generated. Then, the predictive performance of the candidate subset is estimated using cross-validation on the training set. Finally, another candidate feature subset is generated from the current one according to a search strategy. Many search strategies have been suggested in literature: forward selection, backward elimination, best-first, branch-and-bound and others [81, 110]. Different stopping criteria are used depending on the search strategy. Thus, wrappers are black-box models: they do not rely on the information about features but only on the predictive performance of every subset evaluated during each iteration of the feature subset space search. It makes them universal, model-agnostic and simple; they generally provide better results then filters but are computationally expensive, especially applied to large feature spaces [181, 47, 206, 81, 107].

Embedded methods aim at reducing the computational cost of wrappers by per-

forming feature selection during the learning of the model [34, 81, 120]. Embedded feature selection usually exploits the information about the predictive model, e.g., neural networks weights [194], feature weights in SVM [82], and decision tree splitting criteria in [132]. While being more efficient than wrappers, they are still generally slower than filter approaches [214].

Filters

Filters are another feature selection method family that does not utilize the learning algorithm to perform the selection. Instead, the features are evaluated in various ways and filtered out according to the evaluation. While they are often perceived to be model-agnostic [81], it is argued that filters, like wrappers, need to take into account the classification algorithm to be effective [210]. Filters are generally faster than wrappers because they do not require any learning; wrappers are considered to perform better because they can fit the feature subset to the learning algorithm (see [47] for a discussion on this topic).

In this work, I consider simple filtering of phylo-k-mers based on Maximal Mutual Information a.k.a. Information Gain. This approach has been successfully applied in many areas of machine learning, and it will be described later in this chapter. Many other filtering methods have been suggested, including more advanced ones; this work is the first attempt to apply feature selection to phylo-k-mers, and application of more advanced techniques goes beyond the scope of this work. For a survey on other methods of information theory-based feature selection, I refer the reader to [30] and its references.

3.5.3 Feature selection using Mutual Information

Early mentions of applying mutual information to select features for text classification are [102, 149, 41]. The method selects features that achieve highest values of individual mutual information between the feature and the document class variable. This section introduces definitions that are necessary to describe this method.

Definition 3.5.1. The entropy $H(X)$ of a discrete random variable X with possible outcomes $\mathcal{X}$ is defined by:

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x) \tag{3.28}$$

where $p(x)$ denotes the probability mass function of X.

Definition 3.5.2. Consider two random variables X and Y with a joint probability mass function $p(x, y)$ and marginal probability mass functions $p(x)$ and $p(y)$. Let $\mathcal{X}$ and $\mathcal{Y}$ denote the sets of their possible outcomes. The conditional entropy $H(Y \mid X)$ is defined as

$$H(Y \mid X) = -\sum_{x \in \mathcal{X}} p(x) \sum_{y \in \mathcal{Y}} p(y \mid x) \log p(y \mid x). \tag{3.29}$$

Definition 3.5.3. Consider again X and Y as in Definition 3.5.2. The mutual information $I(X;Y)$ is the relative entropy between the joint distribution and the product distribution $p(x)p(y)$:

$$I(X;Y) = \sum_x \sum_y p(x,y) \log \frac{p(x,y)}{p(x)p(y)}. \tag{3.30}$$

As shown in [39], there is a connection between mutual information and conditional entropy:

$$I(X;Y) = H(X) - H(X \mid Y). \tag{3.31}$$

The definitions above can be now applied to define the mutual information filter for Bernoulli Naive Bayes text classification in the following way. Let Y be the class variable. Let B_w be the random variable that takes 0 or 1 indicating if the word w is present in the document, and let b_w denote its realization. Then, mutual information between the feature and the class variable can be expressed as:

$$I(Y;B_w) = \sum_{y \in \mathcal{Y}} \sum_{b_w \in \{0,1\}} \mathbb{P}(y, b_w) \log \frac{\mathbb{P}(y, b_w)}{\mathbb{P}(y)\mathbb{P}(b_w)}. \tag{3.32}$$

The probabilities in Equation 3.32 can be estimated from the training set in the following manner. $\mathbb{P}(y, b_w)$ is estimated by the frequency of documents of class y that contain w. $\mathbb{P}(y)$ is estimated by the frequency of documents of class y in the training set. $\mathbb{P}(b_w)$ is estimated by the frequency of the documents containing w in the training set.

In conclusion, the filter ranks the features by the value $I(Y;B_w)$ in descending order, and takes a fixed number of features with the highest values of $I(Y;B_w)$. The next section will demonstrate how to adapt this approach to select k-mers in the framework of RAPPAS.

3.6 Mutual Information filters for RAPPAS

3.6.1 Deriving the Mutual Information Filter

Let us now express the mutual information between the class variable Y and the variable indicating the presence of a k-mer B_w. According to Equation 3.31,

$$\begin{aligned} I(Y;B_w) &= H(Y) - H(Y \mid B_w) \\ &= H(Y) - \Big(\mathbb{P}(B_w = 1)H(Y \mid B_w = 1) + \mathbb{P}(B_w = 0)H(Y \mid B_w = 0)\Big). \end{aligned} \tag{3.33}$$

Let N again denote the number of branches of T_0. We assume a uniform prior $\mathbb{P}(y)$ over all branches of T_0, which yields that $H(Y) = -\sum_{y \in E(T_0)} \mathbb{P}(y) \log \mathbb{P}(y) =$

$-N(1/N \log 1/N) = -\log 1/N$ is a constant value, which contributes equally to the ranking of all features. Before we move on, one important point needs to be made. $\mathbb{P}(B_w = 1)$ mentioned in Equation 3.33 means the probability of observing w in the query q of arbitrary size. Those can be derived from conditional probabilities of observing w given the class y, i.e. $\mathbb{P}(B_w = 1 \mid y)$. In Sections 3.4.1 and 3.4.2 we discussed that if placed queries are shorter than the reference alignment, then $\mathbb{P}(B_w = 1 \mid y)$ is estimated correctly by $fS_y(w)$ not just $S_y(w)$. Here the parameter f appears again, and it will play an important role in the expansion of Equation 3.33.

Let us now break down every member of Equation 3.33 and write them down in terms of $S_y(w)$ and $S_w = \sum_y S_y(w)$. First, let us define the probabilities $\mathbb{P}(B_w = 1)$, $\mathbb{P}(B_w = 0)$:

$$\mathbb{P}(B_w = 1) = \sum_y \mathbb{P}(y)\mathbb{P}(B_w = 1 \mid y) = \sum_y \frac{fS_y(w)}{N} = \frac{f}{N}\sum_y S_y(w) = \frac{fS_w}{N} \tag{3.34}$$

$$\mathbb{P}(B_w = 0) = 1 - \mathbb{P}(B_w = 1) = 1 - \frac{fS_w}{N}. \tag{3.35}$$

Now let us express the probabilities in $H(Y \mid B_w = 1)$, $H(Y \mid B_w = 0)$:

$$\mathbb{P}(y \mid B_w = 1) = \frac{\mathbb{P}(B_w = 1 \mid y)\mathbb{P}(y)}{\mathbb{P}(B_w = 1)} = \frac{fS_y(w) \cdot 1/N}{1/N \cdot fS_w} = \frac{S_y(w)}{S_w} \tag{3.36}$$

$$\mathbb{P}(y \mid B_w = 0) = \frac{\mathbb{P}(B_w = 0 \mid y)\mathbb{P}(y)}{\mathbb{P}(B_w = 0)} = \frac{(1 - fS_y(w)) \cdot 1/N}{1 - \frac{fS_w}{N}} = \frac{1 - fS_y(w)}{N - fS_w}. \tag{3.37}$$

Finally, we can write down the conditional entropies of Y:

$$H(Y \mid B_w = 1) = -\sum_y \mathbb{P}(y \mid B_w = 1) \log \mathbb{P}(y \mid B_w = 1) = -\sum_y \frac{S_y(w)}{S_w} \log \frac{S_y(w)}{S_w} \tag{3.38}$$

$$\begin{aligned} H(Y \mid B_w = 0) &= -\sum_y \mathbb{P}(y \mid B_w = 0) \log \mathbb{P}(y \mid B_w = 0) \\ &= -\sum_y \frac{1 - fS_y(w)}{N - fS_w} \log \frac{1 - fS_y(w)}{N - fS_w}. \end{aligned}$$

Therefore, maximising the mutual information

$$I(Y; B_w) \to \max_w \tag{3.39}$$

is equivalent to:

$$\underbrace{\frac{fS_w}{N}}_{\mathbb{P}(B_w=1)} \underbrace{\sum_y \frac{S_y(w)}{S_w} \log \frac{S_y(w)}{S_w}}_{-H(Y|B_w=1)} + \underbrace{\left(1 - \frac{fS_w}{N}\right)}_{\mathbb{P}(B_w=0)} \underbrace{\sum_y \frac{1 - fS_y(w)}{N - fS_w} \log \frac{1 - fS_y(w)}{N - fS_w}}_{-H(Y|B_w=0)} \to \max_w \tag{3.40}$$

Now we can see that it is impossible to proceed without having guessed the value of f. However, the filtering of phylo-k-mers is meant to happen during the first stage of the RAPPAS algorithm, that is, phylo-k-mer database computation (see Section 1.5). During this stage, query sizes *are unknown*. Moreover, we would like to select phylo-k-mers that will be informative to place *queries of different sizes*. How would it be possible?

One way could be to guess different values of f in the range $[0, 1]$, apply Equation 3.40 multiple times and produce multiple filtered databases. Depending on the query size, the user would need to use the phylo-k-mer database that was produced with the value of f that was the closest to $(|q|-k+1)/(m-k+1)$. This seems impractical, because the rationale for filtering is to reduce the number of phylo-k-mers, and therefore to reduce the final size of produced phylo-k-mer databases. From this point of view, producing many databases could be counterproductive. Instead, I suggest a simpler route: assume only one fixed value of f and evaluate the performance of the Mutual Information filter for this fixed value for queries of different sizes. Query sizes are in the range $[k, m]$, therefore $f \in (0, 1]$. Let us make two different assumptions about the value of f.

3.6.2 Making assumptions about f

The first assumption will be $f = 1$, which corresponds to full-length queries ($|q| = m$). This is a common scenario in metabarcoding that RAPPAS targets. Assuming $f = 1$, it is straightforward to derive the first filter from Equation 3.40:

$$\frac{S_w}{N} \sum_y \frac{S_y(w)}{S_w} \log \frac{S_y(w)}{S_w} + \left(1 - \frac{S_w}{N}\right) \sum_y \frac{1 - S_y(w)}{N - S_w} \log \frac{1 - S_y(w)}{N - S_w} \to \max_w \tag{3.41}$$

This is the definition of the first filter I suggest. Let us call it $MI_{f=1}$. The value of f suggests that this filter should produce phylo-k-mers that place long queries better than short queries. What happens in practice will be covered later in this chapter.

Another assumption will be $f \to 0$, for which RAPPAS is obtained from Bernoulli-based placement. Our objective is to derive a closed-form expression of Equation 3.40 that does not contain f. To do this, let us consider both members of the formula: f does not appear in $H(Y \mid B_w = 1)$, only in $H(Y \mid B_w = 0)$. Let us take a closer look at its value when $f \to 0$.

Theorem 3.6.1. $H(Y \mid B_w = 0) = \log N + \mathcal{O}(f^2)$.

Proof. Let us look at the Taylor series of $H(Y \mid B_w = 0)$ at $f = 0$, where

$$H(Y \mid B_w = 0) = -\sum_y \frac{1 - fS_y(w)}{N - fS_w} \log \frac{1 - fS_y(w)}{N - fS_w}$$

Let us substitute $g(f) = H(Y \mid B_w = 0)$. Then, $g(f) = g(0) + g'(0) + \mathcal{O}(f^2)$. The first member of the series is:

$$g(0) = -\sum_y \frac{1}{N} \log \frac{1}{N} = -\log 1/N = \log N$$

To express the second member, we need to take the derivative of g.

Lemma 3.6.2. For

$$g(f) = -\sum_y \frac{1 - fS_y(w)}{N - fS_w} \log \frac{1 - fS_y(w)}{N - fS_w}$$

where log is the a base logarithm, its derivative is

$$\frac{d}{df} g(f) = -\sum_y \frac{1}{\ln a} \frac{NS_y(w) - S_w}{(N - fS_w)^2} \Big(\ln \frac{1 - fS_y(w)}{N - fS_w} + 1 \Big).$$

Proof. See Appendix A.1. □

Let us now calculate the value of g' at $f = 0$:

$$\begin{aligned}
g'(0) &= -\sum_y \frac{1}{\ln a} \frac{NS_y(w) - S_w}{N^2} \Big(\ln \frac{1}{N} + 1 \Big) \\
&= \frac{\ln N - 1}{\ln a} \sum_y \frac{NS_y(w) - S_w}{N^2} \\
&= \frac{\ln N - 1}{\ln a} \sum_y \frac{S_y(w)}{N} - \frac{NS_w}{N^2} \\
&= \frac{\ln N - 1}{\ln a} \Big(\frac{S_w}{N} - \frac{S_w}{N} \Big) \\
&= 0.
\end{aligned}$$

The statement of Theorem 3.6.1 follows immediately. □

By substituting the result above into Equation 3.40, we then have that $I(Y; B_w) \to \max_w$ is equivalent to:

$$\frac{fS_w}{N} \sum_y \frac{S_y(w)}{S_w} \log \frac{S_y(w)}{S_w} - (1 - \frac{fS_w}{N})(\log N + \mathcal{O}(f^2)) \to \max_w$$

$$-\log N + \frac{fS_w}{N}\left(\sum_y \frac{S_y(w)}{S_w}\log\frac{S_y(w)}{S_w} + \log N\right) + \mathcal{O}(f^2) + \mathcal{O}(f^3) \to \max_w$$

If we ignore $-\log N$ (a constant) and only keep the $O(f)$ term which dominates over the higher-degree infinitesimals for $f \to 0$, we get:

$$\frac{fS_w}{N}\left(\sum_y \frac{S_y(w)}{S_w}\log\frac{S_y(w)}{S_w} + \log N\right) \to \max_w$$

which finally allows us to drop f/N since it contributes to all classes equally. Thus, Equation 3.40 with $f \to 0$ is equivalent to:

$$S_w\Big(\log N + \sum_y \frac{S_y(w)}{S_w}\log\frac{S_y(w)}{S_w}\Big) \to \max_w \tag{3.42}$$

or, equivalently:

$$S_w\Big(H(Y) - H(Y|B_w = 1)\Big) \to \max_w \tag{3.43}$$

Equations 3.42 and 3.43 are definitions of the second filter suggested in this work. Let $MI_{f\to 0}$ denote this filtering function. Equation 3.43 has an interesting interpretation: on the one hand, it favors k-mers with high S_w, i.e., *probable ones*. On the other hand, it favors the ones that produce higher reductions in the entropy of the class variable when the k-mer is in the query. In other words, the filter based on $MI_{f\to 0}$ selects k-mers that reduce entropy the most if observed.

$MI_{f=1}$ (defined in Equation 3.41) and $MI_{f\to 0}$ are used to rank features for filtering, and the ranking determines in what order features should be selected. There is still an essential detail of the implementation of feature selection we have to cover: how many features should we select?

3.6.3 Filter-based selection of phylo-k-mers

Before I describe the selection process, let me again clarify the meaning of k-mers and phylo-k-mers to avoid confusion between them. k-mers are substrings of size k observed in the query sequence, i.e., *features* of classification. Phylo-k-mers are pairs $\{(branch, score)\}$ associated with a particular k-mer, i.e., *feature values*. Filters select informative k-mers based on their phylo-k-mer scores.

Filters lack mechanisms for finding how many features should be selected contrary to wrappers: they do not evaluate the selected subset of features. Instead, users suggest a fixed number of features to select. However, for RAPPAS, selecting a fixed *number of k-mers* is inappropriate for the following reason. The way phylo-k-mers are stored implies that they take different amounts of space depending on the k-mer: the number of scores stored for a particular k-mer is the number of branches whose score surpasses the threshold ε (as described at page 37). Selection of a fixed number of

k-mers	phylo-k-mer scores			
AA	$(y_1, 0.9)$	$(y_4, 0.3)$		
AC	$(y_1, 0.99)$	$(y_2, 0.1)$	$(y_3, 0.01)$	$(y_4, 0.01)$
AT	$(y_4, 0.7)$			
...	...			
TA	$(y_1, 0.01)$	$(y_2, 0.01)$	$(y_3, 0.01)$	$(y_4, 0.01)$

Figure 3-1 – A toy example of a database of phylo-k-mers. Selection of a fixed number of k-mers results in a different numbers of phylo-k-mers in the filtered database, depending on what k-mers are selected. For example, selecting a single k-mer would result storing only one phylo-k-mer if AT is selected and four phylo-k-mers for AC.

features can favor filters that give high ranks to k-mers with a larger number of phylo-k-mers explicitly stored. As a result, filtered databases can differ in final size, which makes it unfair to compare the performance of filters using this approach (Figure 3-1 illustrates the problem).

Instead, I suggest another way. Let $\mathcal{D}$ denote the entire database of phylo-k-mers, and $|\mathcal{D}|$ denote the total number of phylo-k-mers stored in $\mathcal{D}$. Let μ denote the relative size of the selected subset of features compared to the full one, $\mu \in (0, 1]$. For example, for the value of $\mu = 0.5$, the filtered database should be no larger than $|\mathcal{D}|/2$ in the number of phylo-k-mers; $\mu = 0.1$ means that the filtered database should be no larger than $|\mathcal{D}|/10$. $\mu = 1.0$ corresponds to the entire original database with no filtering applied. Filtering happens as follows: having all k-mers ranked by the filtering function, we iteratively include k-mers one by one to the filtered database until it reaches $\mu \cdot |\mathcal{D}|$ in the number of phylo-k-mers. Since each k-mer w is associated with a different number of scores $S_y(w)$ stored explicitly, the final number of k-mers included may vary for different filters even if μ is the same.

3.6.4 Random filter

In practice, filters do not assume evaluation of their performance, which makes them faster compared to other feature selection methods. However, we still need to evaluate the performance of suggested filters to answer these questions:

1. How could we understand which filter is better?

2. How effective are filters on different datasets compared to a meaningful baseline?

3. How does filtering influence placement accuracy?

To answer those questions, we need to choose a baseline filtering method to compare against and an evaluation procedure. For the baseline, I suggest ranking all k-mers randomly and selecting them according to their ranks until the database reaches $\mu \cdot |\mathcal{D}|$ in the number of phylo-k-mers. I will call this process Random filtering. The process of evaluation will be described in Section 3.7.

3.7 Experimental evaluation

This section presents experimental evaluation of suggested filtering methods. All experiments were carried out using *Placement Evaluation WOrkflows* (PEWO) [129], a recently published framework for comparing phylogenetic placement tools performance. Effectiveness of filters is evaluated in terms of relative *placement accuracy* achieved with filtered databases compared to non-filtered ones. PEWO suggests two procedures to measure placement accuracy: *pruning-based accuracy* (PAC) and *likelihood-based accuracy* (LAC). The main experiments used the latter method, which is a technique introduced in [129]. This choice is based on that PAC is computationally much more heavy; however, some results obtained using PAC are also presented below.

I implemented all filtering methods in XPAS, a new tool for computing databases of phylo-k-mers. Computed databases were used by RAPPAS2, a new tool for phylogenetic placement that I developed in this work. A detailed description of those programs will follow in Chapter 4; XPAS and RAPPAS2 were used in conjunction with PEWO for LAC and PAC experiments.

3.7.1 Likelihood-based Accuracy

LAC is a placement evaluation procedure I implemented while working on this project. It is one of two evaluation procedures available in PEWO at the moment, and it assesses the *relative accuracy* of phylogenetic placement. The procedure is as follows.

1. Align a query q within the reference alignment A, obtaining an alignment A_q. In PEWO, HMMER is used as the standard alignment tool.

2. Given a reference tree T, a placement method with a fixed set of parameters, place the query q to T. Let $\hat{y}(q)$ be the branch that is returned as the most likely placement for q.

3. Create an extended tree T_q by modifying T as follows: create a new node in T_q by splitting $\hat{y}(q)$ in two branches. Attach to this new node a new pendant branch leading to a leaf labelled by q.

4. Reoptimize the branch lengths of T_q under the same model used to infer T and calculate the log-likelihood (referred to as $\log \mathcal{L}(q)$) of T_q. PEWO uses RAxML-NG for this purpose:

```
$ raxml-ng --evaluate --msa Aq --tree Tq \
           --model MODEL
```

We also call $\log \mathcal{L}(q)$ *LAC values* to differentiate them from the log-likelihoods of the different possible placements for one query reported by phylogenetic placement software. LAC values obtained for different placement methods and the same query sequence can be compared. However, in this chapter, I do not compare different placement software. Instead, I compare the performance of RAPPAS2 applied with

filtered databases against its performance with non-filtered databases. The relative performance of placement obtained with different filtering methods is also a subject of interest.

For every dataset consisting of a reference alignment and a reference tree, the following procedure was applied. In order to generate query sequences, a large subtree was pruned out of the original tree. Then, sequences that corresponded to the leaves of the subtree were excluded from the alignment. Those sequences were used to randomly sample query subsequences of a specific size. The branch lengths in the rest of the original tree were reoptimized and the phylo-k-mer database was computed on the basis of the resulting tree, as follows. For a fixed combination of parameters k, ε, a database of phylo-k-mers was computed and filtered with gradually decreasing values of μ : 1.0, 0.5, 0.25, 0.125, 0.0625, 0.03125, 0.015625. This corresponded to filtering out 0%, 50%, 75%, 87.5%, $\approx$ 94%, $\approx$ 97%, $\approx$ 98.5% of phylo-k-mers from the entire database. Queries generated from the pruned subtree were placed using every filtered database. For every query placed with a filtered database obtained for a fixed value of μ, the LAC value ($\log \mathcal{L}^{\mu}(q)$) was compared to the LAC value obtained by placing the same query with a non-filtered database ($\log \mathcal{L}^{1.0}(q)$):

$$\Delta \log \mathcal{L}(q) = \log \mathcal{L}^{\mu}(q) - \log \mathcal{L}^{1.0}(q) \tag{3.44}$$

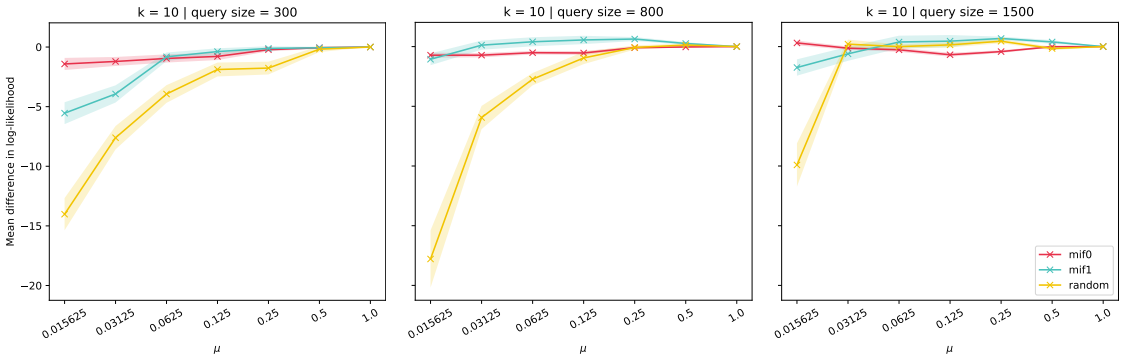
Figures 3-2, 3-3, 3-4, and 3-5 present the results of those experiments for different datasets. A detailed analysis of these results is presented below. Every figure of these plots presents the means of $\Delta \log \mathcal{L}$ averaged over all queries placed. Values of zero mean that LAC values have not changed on average due to filtering; negative values of LAC difference mean worse accuracy on average obtained for filtered databases compared to non-filtered ones.

3.7.2 Datasets

The first dataset, D652, consists of 652 bacterial 16S rRNA sequences and a reference tree, taken as-is from [130]. It was derived from another dataset used in literature, namely *bv* from [200] used in other works on phylogenetic placement [15, 43, 22]. The LAC procedure was applied twice for this dataset by taking subtrees of 386 taxa and 235 taxa which were used to compute phylo-k-mer databases for $k = 10$; the rest of the tree was used to generate queries of different size. The second dataset — D500 — consists of 500 Rbcl gene sequences [20]. The procedure was applied for one subtree of 192 taxa but databases were built for $k = 10$ and $k = 12$. Finally, D155 from [130] consists of 155 whole-genome sequences (9.5k base pairs) of the hepatitis C virus, from which a subtree of 65 taxa was taken as reference. See Table 3.1 for parameter values used in all experiments.

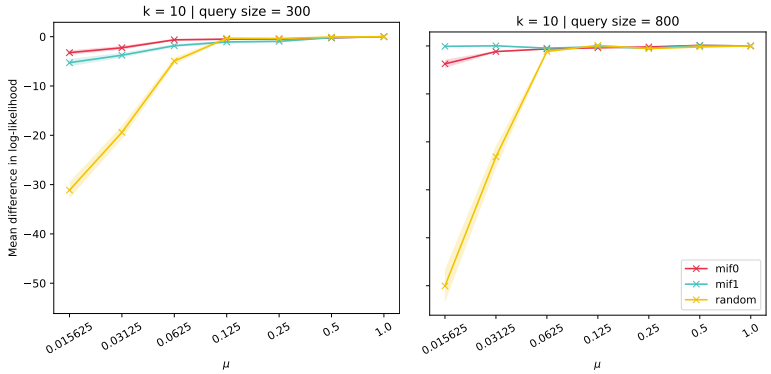
Dataset	Figure	# taxa	m	k	ε	query size	#queries
D652	3-2a	386 / 652	1.7Kbp	10	$(1.5/4)^k$	300	100
D652	3-2b	386 / 652	1.7Kbp	10	$(1.5/4)^k$	800	100
D652	3-2c	386 / 652	1.7Kbp	10	$(1.5/4)^k$	1500	100
D652	3-3a	235 / 652	1.7Kbp	10	$(1.5/4)^k$	300	200
D652	3-3b	235 / 652	1.7Kbp	10	$(1.5/4)^k$	800	200
D500	3-4a	192 / 500	1.4Kbp	10	$(1.5/4)^k$	300	100
D500	3-4b	192 / 500	1.4Kbp	10	$(1.5/4)^k$	700	100
D500	3-4c	192 / 500	1.4Kbp	10	$(1.5/4)^k$	1300	100
D500	3-4d	192 / 500	1.4Kbp	12	$(1.5/4)^k$	300	100
D500	3-4e	192 / 500	1.4Kbp	12	$(1.5/4)^k$	700	100
D500	3-4f	192 / 500	1.4Kbp	12	$(1.5/4)^k$	1300	100
D155	3-5a	65 / 155	9.5Kbp	10	$(1.5/4)^k$	300	50
D155	3-5b	65 / 155	9.5Kbp	10	$(1.5/4)^k$	4800	50
D155	3-5c	65 / 155	9.5Kbp	12	$(1.5/4)^k$	300	50
D155	3-5d	65 / 155	9.5Kbp	12	$(1.5/4)^k$	4800	50

Table 3.1 – The parameters of LAC experiments on k-mer filtering.



(a) (b) (c)

Figure 3-2 – Means and standard errors of LAC differences obtained for the first experiment on D652 for three filtering methods: $MI_{f\to 0}$ (red), $MI_{f=1}$ (cyan), and Random filtering (yellow). Higher values are better. Positive values indicate that filtering had improved placement on average compared to placement using the entire database of phylo-k-mers. Every dot represents a mean of 100 LAC value differences for queries placed with a filtered phylo-k-mer database obtained with a fixed set of parameters k, ε, and μ. (a), (b), (c) present results for queries of size 300, 800, and 1500 base pairs, respectively.



(a) (b)

Figure 3-3 – Means and standard errors of LAC differences obtained for the second experiment on D652 for three filtering methods, 200 queries per database. This figure should be interpreted in the same way as Figure 3-2. (a) Query size of 300 bp. (b) Query size of 800 bp.

3.8 Experimental results & discussion

3.8.1 Filter performance: LAC experiments

D652

Figure 3-2a shows that all filters perform very well for $\mu = 0.5$ (mean change in likelihood is close to zero), suggesting that filtering out a half of the phylo-k-mers did not decrease the accuracy of the placement in this case. For even smaller values of μ the placement accuracy drops at different rates for different filters. For $\mu < 0.5$, $MI_{f\to 0}$ (red) performs better on average than $MI_{f=1}$ (cyan), except for $\mu = 0.125$. Both $MI_{f\to 0}$ and $MI_{f=1}$ perform much better than Random filtering (yellow).

Results are similar for longer queries (Figures 3-2b, 3-2c): until μ reaches the value of 0.25 for queries of 800 bp and 0.03125 for queries of 1500 bp, both mutual information-based filters show the performance close to the baseline; they perform much better for very small values of μ. $MI_{f=1}$ shows a better performance in the mid-range of μ values than $MI_{f\to 0}$, achieving positive values of mean LAC difference. This suggests that filtering may improve the accuracy of placement for this case, and some phylo-k-mers in non-filtered databases may be harmful for phylogenetic placement; however, the statistical significance of this effect was not confirmed. $MI_{f\to 0}$ shows more stable performance for all values of μ tested: for example, 3-2c suggests that for longer queries, taking only 1.5% of phylo-k-mers ($\mu = 0.015625$, red line) is as good as taking 50% ($\mu = 0.5$) or 100% ($\mu = 1.0$). The latter is a strong indication that the assumption of phylo-k-mer database redundancy is correct, at least for some combination of input parameters of the method.

An interesting effect can be noticed if we compare the performance of the Random filter for different query sizes. The baseline performance degrades slower with decreasing μ for longer queries: on Figure 3-2a, $\mu = 0.5$ is as good as $\mu = 1.0$; on

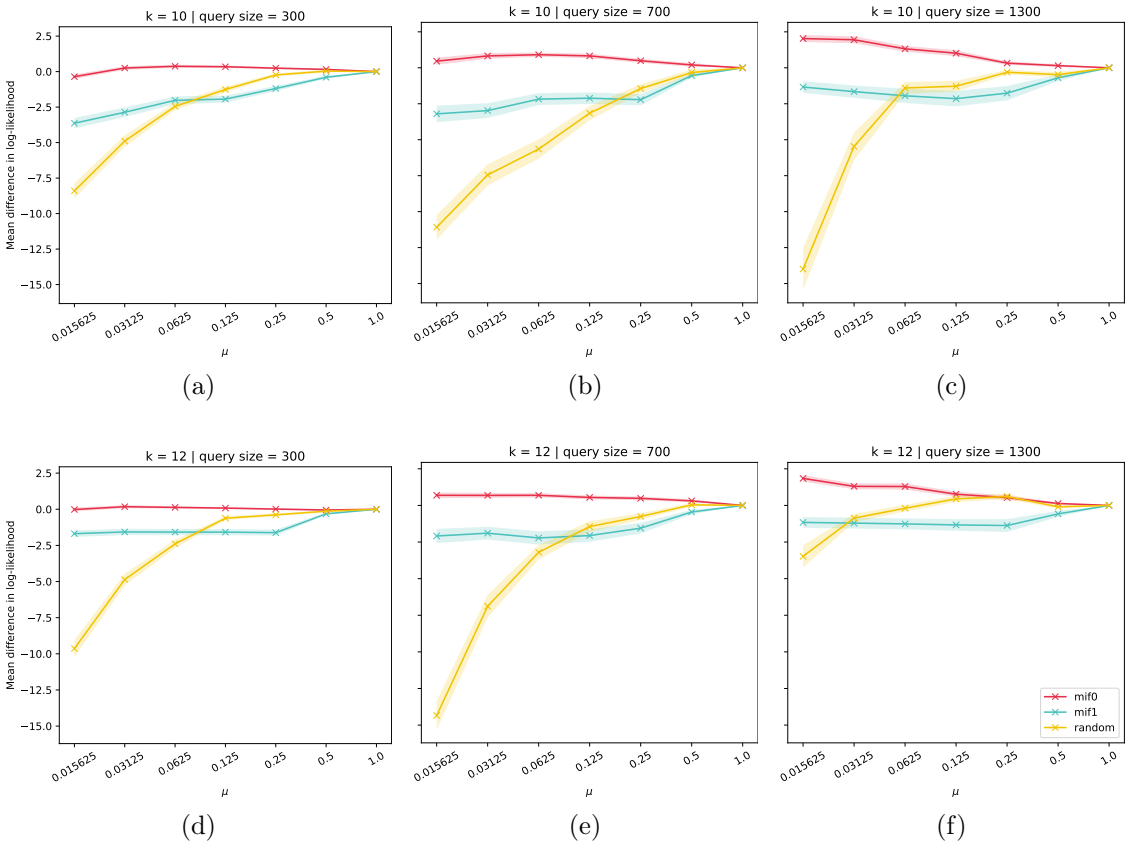


Figure 3-4 – Means and standard errors of LAC differences obtained for the experiment on D500 for three filtering methods, 100 queries per database. (a), (b), (c) present results obtained for $k = 10$ and query sizes of 300 bp, 700 bp, 1300 bp, respectively; (d), (e), (f) for $k = 12$ and the same query sizes.

Figure 3-2b, $\mu = 0.25$ is enough for accurate placement, and on Figure 3-2c, even $\mu = 0.03125$ is as good as $\mu = 1.0$. This suggests that longer queries are easier to place: they contain more of a phylogenetic signal that helps to determine the query's exact placement. This result is consistent with the results of [130], where RAPPAS and other phylogenetic placement tools such as EPA-NG and PPLACER showed higher placement accuracy for longer queries.

D500

Figure 3-4 present results for D500, obtained for different query sizes (300, 700, and 1500) and two values of k: $k = 10$ (3-4a, 3-4b, 3-4c) and $k = 12$ (3-4d, 3-4e, 3-4f). In all cases $MI_{f\to 0}$ performs better than $MI_{f=1}$; both filters achieve high accuracy even for very small values of μ, outperforming random filtering. Notice that $MI_{f\to 0}$ slightly improves the accuracy of placement for long queries (Figures 3-4c, 3-4f), while $MI_{f=1}$ does not, contrary to the case of D652. The significance of the improvement is to be established.

D155

For this dataset, the picture is more complicated, and it is different depending on the value of k. For $k = 10$ (Figures 3-5a, 3-5b) both mutual information-based filters show good performance only for $\mu \geq 0.25$. For $\mu < 0.25$, they do not perform better than random filtering. The reasons of poor performance for smaller values of μ are unclear. One reason could be that the reference alignment of D155 is much longer ($\approx$ 9.6k base pairs against $\approx$ 1.7k and $\approx$ 1.4k base pairs for the previous datasets), and the size of k-mers may be relatively too small. Experiments of [130] also suggested that this dataset is challenging for RAPPAS: it showed poor performance compared to other phylogenetic placement tools. However, increasing the size of the k-mer ($k = 12$, Figures 3-5c, 3-5d) improves the results: proposed filters show better performance than the baseline. This could mean that computing a phylo-k-mer database with $k = 10$ and low μ values ($\mu < 0.25$) is insufficient to represent the phylogeny as a collection of phylo-k-mers, and such values do not allow the database to describe the high amount of phylogenetic information of this dataset. I will comment on this problem later in Section 3.8.3.

3.8.2 Filter performance: PAC experiments

In addition to the LAC experiments, I conducted more classic PAC experiments with two datasets (D500 and D652), on which the filtering showed promising results in the previous section. These experiments are much more computationally expensive since they require computing many phylo-k-mer databases (see Section 1.4 for details of this procedure). These experiments aimed at determining the absolute placement accuracy of filtered databases (that is, how far in terms of node distance filtered databases can place query sequences). Because of the computational complexity of these experiments, the values of k were reduced to 8 and 10. For every dataset and a set of parameters k, μ, 30 pruning experiments were done; for every pruning of the reference tree, a database of phylo-k-mers was calculated and used to place query sequences of size 300 bp generated from the pruned subtree. Then, the node distance of every query was calculated, and the mean node distance for the pruning is shown as a single dot on the resulting figures.

Figure 3-6 presents the results of those experiments obtained for D652. In all figures of PAC experiments, the lower the values, the better. Any cloud of points can be compared with the cloud obtained for $\mu = 1$ of the same plot to understand the effect of filtering on accuracy compared to placement without filtering. For $k = 8$, $MI_{f\to 0}$ and $MI_{f=1}$ (Figures 3-6a, 3-6b) perform as well as random filtering (Figure 3-6c) for $\mu \geq 0.125$ and show a better performance for lower values of μ. Apart from the lowest value of $\mu = 0.015625$, it is difficult to determine which of the suggested MI-based filters works better. Finally, for $k = 10$, $MI_{f\to 0}$ (Figure 3-6d) outperforms $MI_{f=1}$ (Figure 3-6e), especially for the lowest values of μ. All those results are consistent with the results obtained with LAC, where $MI_{f\to 0}$ consistently outperformed $MI_{f=1}$ for the same query size of 300bp.

Figure 3-7 presents the results for D500. For $k = 8$, Figure 3-7a shows that $MI_{f\to 0}$

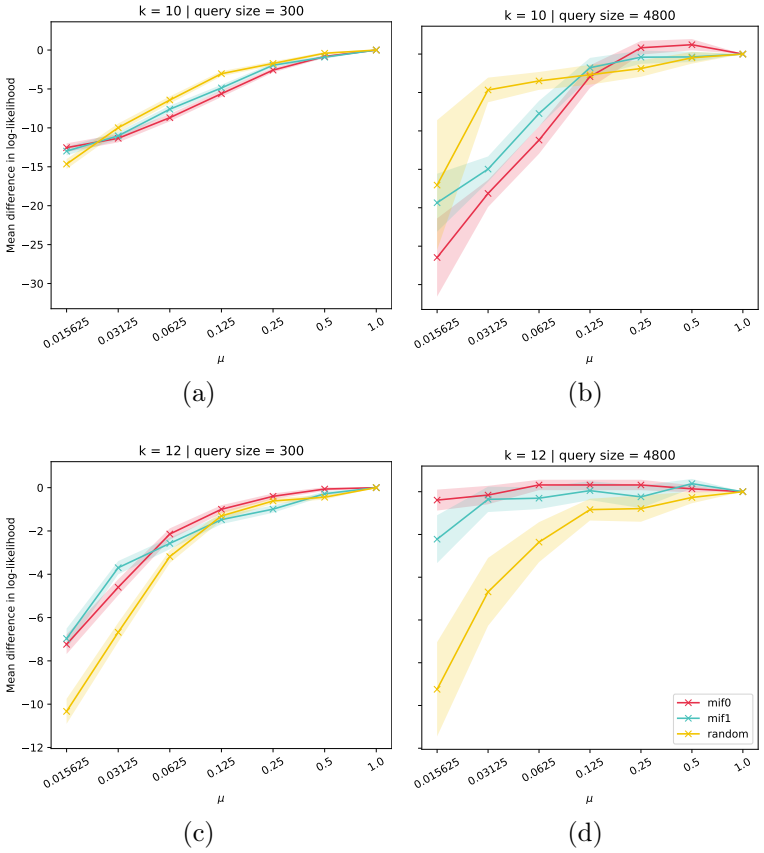


Figure 3-5 – Means and standard errors of LAC differences obtained for the experiment on D155, 50 queries per database. (a), (b) show results for $k = 10$ and query sizes of 300 bp, 4800 bp, respectively; (c, d) show results for $k = 12$ and the same query sizes.

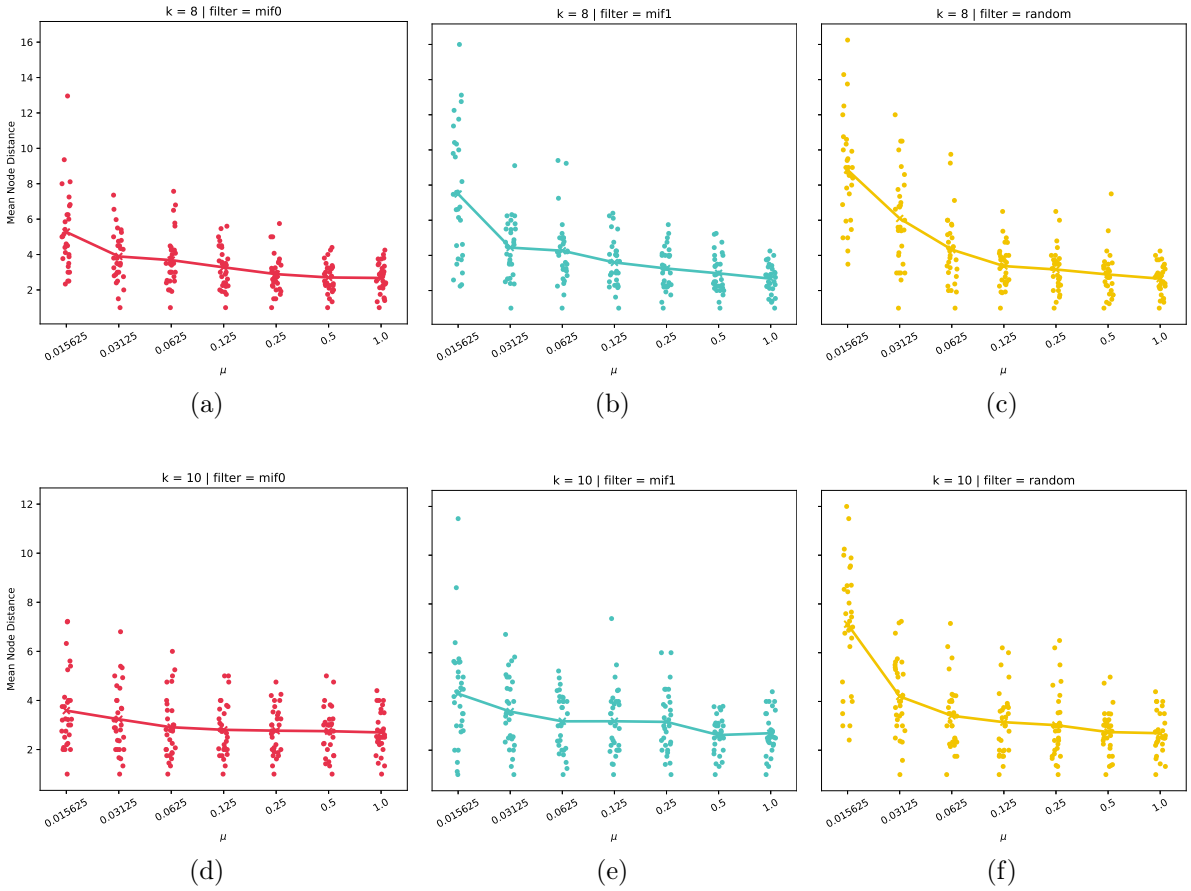


Figure 3-6 – Mean node distance values obtained during PAC experiments on D652 for three filters: $MI_{f\to 0}$ (red), $MI_{f=1}$ (cyan), and Random filtering (yellow). (a, b, c) correspond to the results obtained with $k = 8$, (d, e, f) correspond to the results obtained with $k = 10$. Every dot represents one pruning of the original tree used to produce a filtered database and a set of queries; node distances of placed queries are used to calculate the mean value of mean node distances. Lower values are better. The lines represent the mean values of the distributions.

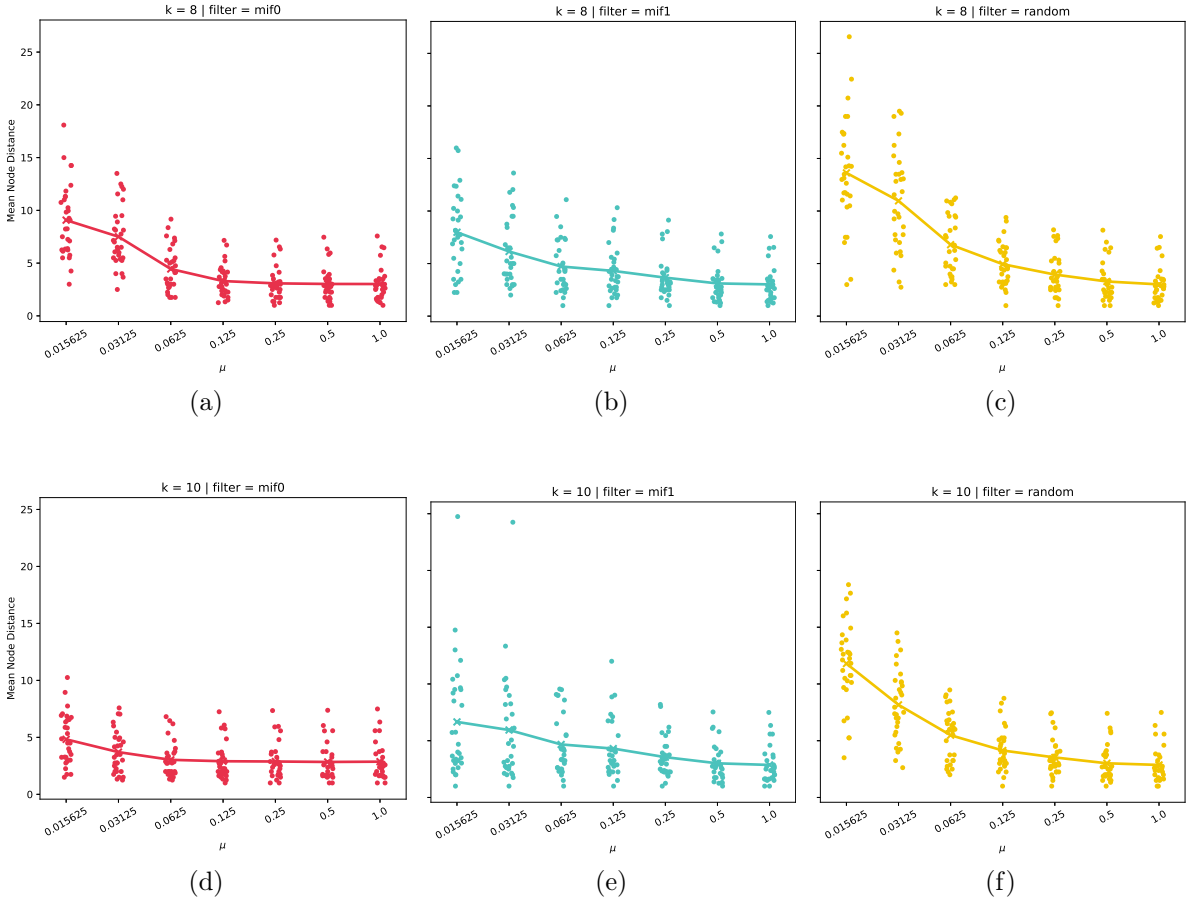


Figure 3-7 – Mean node distance values obtained during PAC experiments on D500. This figure should be interpreted in the same way as Figure 3-6.

allows us to lower μ to 0.125 without reducing the placement accuracy, outperforming the baseline (Figure 3-7c). $MI_{f=1}$ (Figure 3-7b) shows similar but slightly worse results on average, still performing better than random filtering, especially for low values of μ. For $k = 10$, the difference between two mutual information-based filters is higher: $MI_{f=1}$ (Figure 3-7e) suffers from outliers while still performing better than the baseline (Figure 3-7f) on average. $MI_{f\to 0}$ (Figure 3-7d) shows the best results, allowing to lower μ to 0.0625 with no loss of the placement accuracy. The improvement in accuracy produced by filtering for long sequences (1300bp) obtained in LAC experiments for this dataset is yet to be reproduced.

3.8.3 Dataset complexity and parameter choice

The difference in filtering performance for different datasets shows that not all datasets are equally easy to deal with. I assume that different datasets have different intrinsic *complexity*, which we can think of as the amount of information needed to describe the data. For example, long and variable reference alignments have more information than short and conservative ones. I assume that higher values of k are needed to

represent phylogenies inferred from longer sequences in phylo-k-mers. Similarly, for "easy" datasets, more phylo-k-mers can be filtered out, which we observe in the LAC experiments on D652 and D500 compared to D155. In light of this, important practical questions arise: How to estimate the complexity of a dataset? What parameters should be used for each specific dataset? So far, there is no direct answer to these questions, but some considerations can help us sort this out.

First, we should note that increasing k causes us to keep more data, and decreasing μ causes us to keep less. For phylogenetic placement of targeted metabarcoding reads (such as 16S rRNA, 18S rRNA) with a length of no more than a couple of thousand base pairs, RAPPAS2 shows good accuracy in the cases of $k = 10, 12$. Most likely, $\mu \in [0.1, 0.5]$ can be used in these cases without a significant loss of accuracy. For placing reads on phylogenies inferred from longer alignments (more than several thousand base pairs), it is most likely worth using $k = 12$ (or more) and more conservative values of $\mu \in [0.5, 1.0]$. In any case, more experiments with this type of data are needed to make recommendations on parameter values for such cases.

Of course, we would like to keep μ as low as possible as long as it does not significantly decrease placement accuracy: reducing μ leads to a decrease in memory consumption during both phylo-k-mer database computation and placement. So far, we cannot easily figure out what value of μ is optimal in this sense. It is an open question whether or not it is possible to determine the optimal value from dataset characteristics alone. I believe that, for now, the only realistic way to find the optimal μ is to use wrappers or embedded methods. However, as discussed before, such a solution will significantly increase the time needed to build the database of phylo-k-mers. Until we know fast and reliable ways to estimate optimal values of μ, it makes sense to use the maximum amount of RAM available on the user's machine.

The problem of finding the optimal value for k also exists. We want to keep k as low as possible without hurting placement accuracy. However, the question of how to choose the optimal value of k remains unanswered. Intuitively, we should expect to obtain higher placement accuracy when increasing values of k. Values of k higher than 10 did not seem practical before: phylo-k-mer computation was too long, and the resulting databases were too large. Further improvements of phylo-k-mer computation algorithms, combined with the filtering algorithms presented here, could move this boundary. This may extend the scope of RAPPAS2 in the future, making phylogenetic placement for long alignments with RAPPAS2 more accurate.

3.9 Conclusion

In this chapter, we addressed the question of the informativeness of phylo-k-mers. I described the method of RAPPAS as classification and showed that RAPPAS is a Naive Bayes classifier. I showed that RAPPAS is equivalent to phylogenetic placement based on the Bernoulli Naive Bayes classification obtained with $f \to 0$. This condition may be interpreted as follows: placements of RAPPAS and the Bernoulli Naive Bayes are identical when queries are very short relative to the reference align-

ment. In addition to giving us a more accurate understanding of the nature of RAPPAS, it also gives rise to a discussion about its optimality. First, from a Bayesian point of view, the parameter estimate (i.e., the formula by which phylo-k-mer scores are computed) may be refined. So far, there are no precise estimates of how inaccurate the scores of the phylo-k-mers calculated by RAPPAS are. I will discuss it in Section 5.1.2. Second, while Bernoulli-based placement is identical to RAPPAS for short queries and long alignments, it is yet unknown how this classifier behaves for other cases. It may turn out that the accuracy of RAPPAS can be improved with the Bernoulli classifier based on phylo-k-mers. I will discuss it in Section 5.1.3.

Furthermore, the connections to Naive Bayes classifiers pushed me to look into feature selection methods that are commonly used for these classifiers. As a result, I suggested two new methods of selecting informative phylo-k-mers based on Mutual Information — $MI_{f\to 0}$ and $MI_{f=1}$ — and conducted experiments measuring placement accuracy obtained with filtered databases. The experimental results suggest that both methods perform better than the baseline (i.e., the random selection of phylo-k-mers). As for comparing $MI_{f\to 0}$ and $MI_{f=1}$ with each other, the results are inconsistent, but $MI_{f\to 0}$ more often shows higher accuracy than $MI_{f=1}$ and should be the default filtering method for phylo-k-mers.

Both methods allow to significantly reduce database sizes with a little or no loss in placement accuracy. PAC and LAC experiments with short phylogenetic markers (16S rRNA for D652 and Rbcl gene for D500) and short queries (query size of 300 bp) showed that even though database sizes were only 6% and 12% ($k = 10$ for D500 and D652, respectively) compared to unfiltered databases, they still provided enough phylogenetic information to place queries just as accurately as with unfiltered databases. LAC experiments showed that database sizes can be reduced to as little as 3% for longer queries without loss of placement accuracy. Experiments with a dataset based on a longer alignment (D155, complete HCV sequences) have shown that databases can be halved without loss of accuracy for placing short sequences (query size of 300 bp) and be reduced to no more than 6% in size for placing long sequences ($k = 12$, query size of 4.8 Kbp). This suggests that the appropriate degree of filtering may depend on how long sequences are to be placed, how long is the reference alignment, and what is the size of k.

The different nature of the data (bacteria in D652, plants in D500, viruses in D155) shows the versatility of the phylo-k-mer filtering approach in general, which works for data from different kingdoms. Therefore, we can confidently answer the chapter's original question: not all phylo-k-mers are necessary for accurate placement. Moreover, even selecting a random small part of phylo-k-mers is often enough for precise placement. In addition to that, some experimental results appear to show that filtering may improve phylogenetic placement, suggesting that some phylo-k-mers may be not just non-informative but harmful and misleading for the classification algorithm. Those results, however, are yet to be reproduced and confirmed in different experimental setups.

Chapter 4

XPAS and RAPPAS2

This chapter describes two practical results of my work: XPAS and RAPPAS2, two new phylo-k-mer related tools, which entirely reimplement RAPPAS. The functionality of RAPPAS is split into two parts: XPAS implements the phylo-k-mer database computation, and RAPPAS2 implements the phylogenetic placement. XPAS includes the new algorithm for computing phylo-k-mers (see Chapter 2) and the phylo-k-mer filtering discussed in Chapter 3. I implemented those tools having high efficiency in mind. As a result, the presented tools are one to two orders of magnitude faster than RAPPAS.

4.1 XPAS: a standalone tool for creating phylo-k-mer databases

XPAS (a recursive acronym for XPAS: Phylo-k-mers of Ancestral Sequences) is a reimplementation of RAPPAS that preprocesses the reference dataset and computes phylo-k-mer databases. The reason XPAS has become an independent tool is simple: phylo-k-mers are now used for other tasks than phylogenetic placement. One such application is SHERPAS, an application for alignment-free detection of recombination in viral genomes [189]. Another application, CLAPPAS, uses phylo-k-mers to classify protein gene families (Linard et al., unpublished results. This software is under development at the time of this writing). Thus, three different programs need to use phylo-k-mers databases. A standalone solution for computing such databases is now XPAS, which is used in all these cases.

There are two parts of XPAS: the database computation tool and the static library. Once a database of phylo-k-mers is created, it can be accessed by the XPAS Core Library, which provides an API for loading databases into memory and searching for k-mers. This separation allows reusing the XPAS code to create new phylo-k-mer based applications.

I implemented XPAS in C++ for performance reasons. An efficient C++ implementation and the new phylo-k-mer computation algorithm allows XPAS to compute databases much faster than RAPPAS, which is written in Java. The comparison of their performance will follow in Section 4.4.1. The source code of XPAS is freely

available at `https://github.com/phylo42/xpas`.

4.1.1 Creating a phylo-k-mer database

Creating a phylo-k-mer database is similar to the procedure used in RAPPAS, described in Section 1.5.2. However, it includes one additional stage: the optional phylo-k-mer filtering. Thus, the computation of a phylo-k-mer database can be done as follows:

1. For a given input, compute the whole database of phylo-k-mers as described in Chapter 1.
2. (Optional) Rank all k-mers according to a filtering criterion.
3. (Optional) Select a subset of highest-ranked k-mers, forming the resulting database of a smaller size.

Although filtering allows us to process larger datasets, the approach described above implemented naively negates this advantage of filtering in terms of the required memory consumption: the naive implementation assumes that the *entire* database of phylo-k-mers is in memory at a certain point.[1] To address this problem, I proposed a modified algorithm for constructing databases of phylo-k-mers. It consists of three steps. First, we compute phylo-k-mers branch by branch, writing results for every branch to disk. This avoids storing the entire phylo-k-mer database in memory. Second, we filter k-mers in *batches*, keeping in memory only one batch of k-mers at a time. Finally, we combine the batch filtering results into a filtered phylo-k-mer database. Now I will describe this process in more detail, which corresponds to Algorithms 6 and 7.

Reference alignment filtering (Algorithm 6, line 3), ghost node injection (line 5), and the process of ancestral reconstruction (lines 8—10) remain the same as in RAPPAS (refer to Section 1.5.2 for detail about these steps). XPAS supports two external tools to reconstruct ancestral states: PhyML [80] and RAxML-NG [115]. Next, we compute phylo-k-mers; the phylo-k-mer computation procedure (lines 13—24) is different compared to RAPPAS, and it is as follows. We split the whole range of possible k-mers into β batches, i.e., subranges $b_1, b_2, \ldots, b_\beta$ of the same size σ^k/β (Figure 4-1 illustrates the idea). We compute phylo-k-mers separately for each branch; I will describe this process for one branch y_i. For every ghost node $g \in G_{y_i}$ we compute phylo-k-mers using DivideAndConquerThr or BranchAndBound (line 17, for the description of the algorithms see Section 2.3). The resulting collection of phylo-k-mers is a *branch database* $\mathcal{D}_i$. We split $\mathcal{D}_i$ into β temporary databases $B_{i1}, B_{i2}, \ldots, B_{i\beta}$: every temporary database stores phylo-k-mers of the corresponding batch of k-mers for branch y_i. Finally, we write every temporary database on disk to a separate file and unload $\mathcal{D}_i$ from memory. We repeat this process for all branches;

[1] Here we discuss the memory consumption of XPAS, i.e., computation of phylo-k-mer databases. Regardless of how it is implemented, filtering does reduce memory consumption during placement done by RAPPAS2.

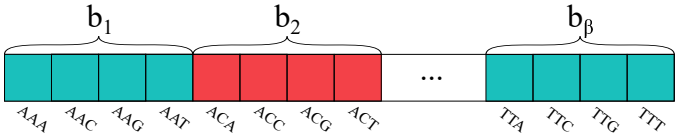


Figure 4-1 – All possible k-mers are split in β batches of the same size.

the result of these steps is a collection of B_{ij} databases on disk. Figure 4-3 illustrates the process and what happens next.

When all temporary databases are computed and written on disk, we filter k-mers in the process called *batch filtering*. This is a three-stage procedure which I also call *merge-filter-merge*; Algorithm 7 describes this procedure, and it is as follows. First, we load $B_{1j}, B_{2j}, \dots B_{Nj}$ back into memory and merge them to form a *batch database* B_j (the *first merge*, which corresponds to lines 7—8 of Algorithm 7). Note that there is a difference between branch databases $\mathcal{D}_i = \bigcup_{j=1}^{\beta} B_{ij}$ split on the previous step and batch databases $B_j = \bigcup_{i=1}^{N} B_{ij}$ merged here (see Figure 4-2 for an illustration). Then, for every k-mer w of the corresponding batch b_j, we calculate the filter value (lines 10—11) using the corresponding filtering function $\mathcal{F}$. This is the value of Mutual Information for $MI_{f \to 0}$ and $MI_{f=1}$ (Equations 3.42 and 3.41) or a random value in the range $[0, 1]$ for random filtering. Afterward, we sort filter values of k-mers of every batch database B_j in descending order (line 13).

Let $|D_i|$ denote the total number of phylo-k-mers in D_i. Having k-mers of every batch sorted by filter value, we need to select k-mers from those batches until we obtain $\mu \cdot |\bigcup_{i=1}^{N} D_i|$ phylo-k-mers associated to those k-mers. This is similar to the merge procedure of the merge sort, except that we have β batches to "merge". We use a simple β-way merge algorithm (the *second merge*, lines 15—25 of Algorithm 7). We add the maximum filter values of every batch to a max-heap. Then, the maximum element of the heap contains the maximum value of the filtering function achieved for a k-mer w. We mark w as filtered and pop out its filter value from the heap. If w belongs to the batch b_j, then we push the next filter value (if any) of the batch to the heap. Then, the next maximum value of the heap is considered; having obtained the necessary number of phylo-k-mers, we load all batches one by one to finally merge the corresponding phylo-k-mers into the final database (lines 27—32). Finally, we compress the final database and write it on disk.

The naive implementation of filtering would require $\mathcal{O}(\sum_{j=1}^{\beta} |B_j| + \sigma^k)$ of additional memory consumption (not counting the size of the final database). Batch filtering allows us to reduce the memory consumption to $\mathcal{O}(\max_{j=1}^{\beta} |B_j| + \sigma^k)$. From this bound, we can see that the minimal additional memory consumption is achieved by choosing β such that $\max_{j=1}^{\beta} |B_j| < \sigma^k$. In the worst-case, the maximum batch database can reach $\sigma^k / \beta \cdot N$ in the number of phylo-k-mers, which gives us a hint for a good value of β:

$$\max_{j=1}^{\beta} |B_j| < \sigma^k \tag{4.1}$$

Algorithm 6: Creating of a phylo-k-mer database

Input : A reference alignment A_0, a reference tree T_0, k, a threshold parameter ω, a reduction ratio r, a filter $\mathcal{F}$, μ

Output: $\mathcal{D}$, the database of phylo-k-mers

Function CreatePhyloKmerDatabase $A_0, T_0, k, \omega, r, \mathcal{F}, \mu$

- /* Reference alignment filtering */
- $A \leftarrow$ remove columns of A_0 in which the fraction of gaps exceeds r
- /* Tree extension */
- $T \leftarrow$ inject ghost nodes for every branch of T_0
- $N \leftarrow$ the number of branches of T_0
- /* Ancestral reconstruction */
- **foreach** *branch* y *of* $E(T_0)$ **do**
  - **foreach** *ghost node* $u \in G_y$ **do**
    - $P^u \leftarrow$ estimate probabilities of every state at every site of A, based on T and A
- $\varepsilon \leftarrow (\omega/|\Sigma|)^k$, the score threshold value
- /* Computation of phylo-k-mers */
- **for** $i \leftarrow 1 \dots N$ **do**
  - $y \leftarrow i$-th branch of T_0
  - $\mathcal{D}_i \leftarrow$ empty database
  - **foreach** *ghost node* $u \in G_y$ **do**
    - $\mathcal{D}_i \leftarrow$ compute phylo-k-mers using P^u
  - $B_i \leftarrow$ empty array
  - **for** $j \leftarrow 1 \dots \beta$ **do**
    - $B_{ij} \leftarrow$ phylo-k-mers of $\mathcal{D}_i$ that are in the batch b_j
    - Write B_{ij} on disk
    - B_i.add(filename of the serialized batch)
- $B \leftarrow [B_1, B_2, \dots, B_N]$
- $\mathcal{D} \leftarrow$ MergeFilterMerge(B, $\mathcal{F}$, μ)
- **return** $\mathcal{D}$

Algorithm 7: Batch filtering

Input : a $N \times \beta$ table B of temporary database files, a filter $\mathcal{F}$, μ
Output: $\mathcal{D}$, a filtered database of phylo-k-mers

Function MergeFilterMerge $B, \mathcal{F}, \mu$
  /* Matrix of filter values of all k-mers */
  $F \leftarrow$ empty $(\sigma^k/\beta) \times \beta$ matrix
  for $j \leftarrow 1 \dots \beta$ **do**
    /* Load all databases of the batch and merge them */
    $B_j \leftarrow$ empty database
    forall $i \leftarrow 1 \dots N$ **do**
      $B_j \leftarrow \text{Merge}(B_j, \text{Load}(B_{ij}))$
    /* Calculate filter values for all k-mers of the batch */
    forall $i \leftarrow 1 \dots |b_j|$ **do**
      if $w_i \in B_j$ **then** $F_{ji} \leftarrow \mathcal{F}$(phylo-$k$-mers of w_i) ;
      else $F_{ji} \leftarrow -\infty$;
    Sort the j-th row of F in descending order
  /* β-way merge algorithm for $F_{1*}, F_{2*}, \dots, F_{\beta,*}$ */
  $H \leftarrow$ empty max-heap
  for $j \leftarrow 1 \dots \beta$ **do**
    H.push(F_{j0})
  $size \leftarrow$ the total number of pairs (k-mer, score) in all batches
  while $S < \mu \cdot size$ **do**
    $w \leftarrow$ the k-mer on the top of H
    $j \leftarrow$ the batch of w
    Mark w as filtered
    H.pop()
    H.push(the next k-mer of batch b_j after w, if its filter value $\neq -\infty$)
    $S = S +$ size of all phylo-k-mers of w
  /* Load batches one by one and collect filtered phylo-k-mers */
  $\mathcal{D} \leftarrow$ empty database
  forall $j \leftarrow 1 \dots \beta$ **do**
    $B_j \leftarrow \text{Load}$(Batch database j)
    forall $w \in b_j$ **do**
      if w *is marked as filtered* **then**
        $\mathcal{D}$.add(phylo-k-mers of w from B_j)
  return $\mathcal{D}$

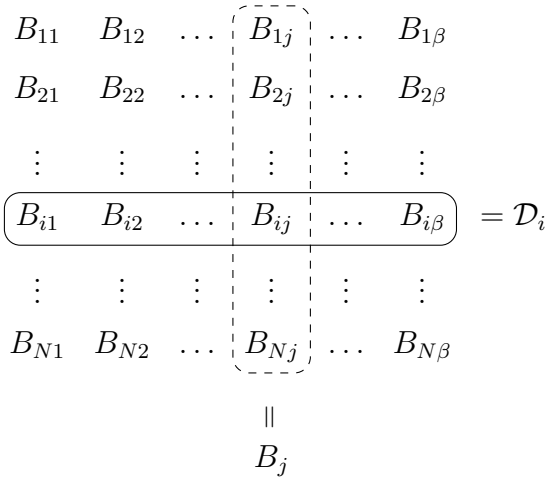


Figure 4-2 – Unions of databases in one row are branch databases $\mathcal{D}_i$, i.e., collections of all phylo-k-mers for one branch. Unions of databases in one column are batch databases B_j, i.e. collections of phylo-k-mers of the batch b_j for all branches.

$$\sigma^k/\beta \cdot N < \sigma^k \tag{4.2}$$

$$\beta > N \tag{4.3}$$

However, this value may be too extreme: batch filtering produces $\beta \cdot N$ temporary files on disk. For the case of $\beta > N$ and large trees, $\Omega(N^2)$ of files may be unwanted if the disk is slow. The machine's filesystem can also be a potential problem since it limits the number of files that can be created. Thus, the β value choice is a trade-off between disk and random-access memory usage. The best values of β are still unclear, and few experiments have been done on this. For DNA, I propose using β of low powers of two such as $8, 16, 32$. At the moment, $\beta = 16$ is the default value in XPAS.

4.1.2 The XPAS Core Library

The XPAS Core Library (XCL) is a library that provides functionality for serializing and deserializing phylo-k-mer databases. XPAS uses it to serialize database results. It is also used by phylo-k-mer-based applications, such as RAPPAS2 and SHERPAS, to load databases and quickly search for k-mers in the database. Figure 4-4 illustrates the dependencies between these software modules and the data. The library encapsulates the details about the serialization format, the compression algorithm and the way k-mers are searched. The advantage of this is that applications do not depend on the details of the library implementation. Instead, they use it transparently to load the database and make k-mer search queries to it. Thus, it makes it easier to develop new applications in the future and improve the implementation of XPAS. XCL uses the Robin-Map library for associative arrays [76], which

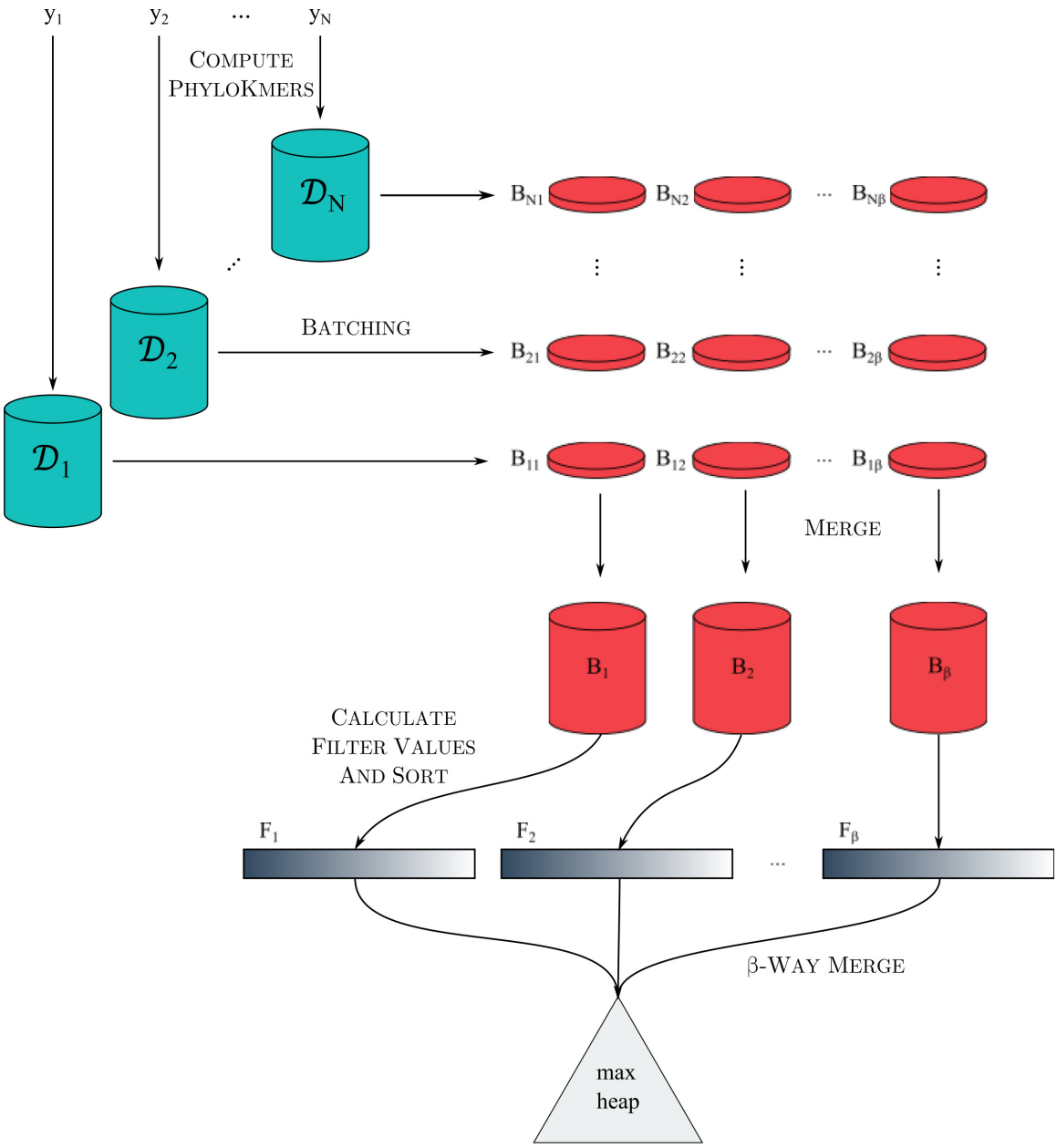


Figure 4-3 – An illustration to the batch filtering procedure (Algorithms 6, 7). Every branch database (in cyan) is split into temporary batch databases; those of different branches are merged together to form a batch database (in red). Filter values are calculated for every batch database and sorted. Finally, filtered k-mers are selected based on their filter values using a β-way merge algorithm until the database reaches $\mu \cdot |\bigcup_{i=1}^{N} D_i|$ in the number of phylo-k-mers.

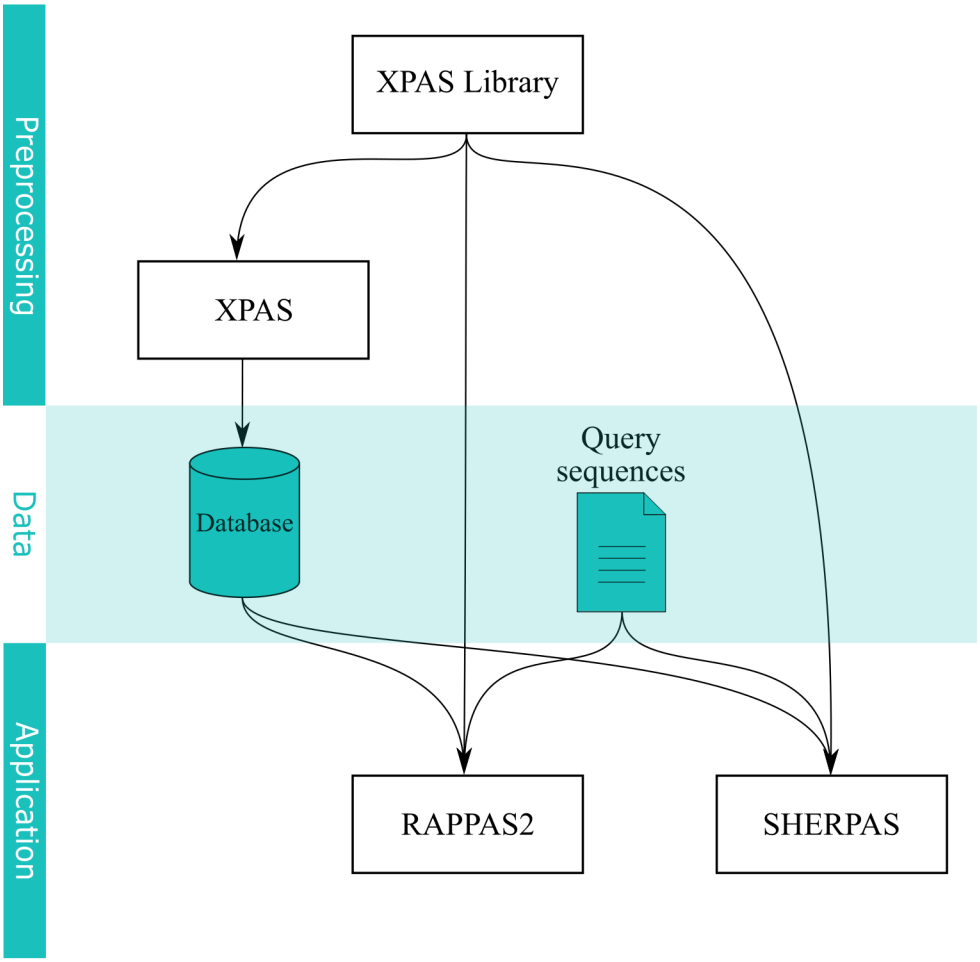


Figure 4-4 – Dependency graph for XPAS, phylo-k-mer-based applications, and data. XPAS, RAPPAS2 and SHERPAS use the library to serialize, deserialize databases, and search for phylo-k-mers.

implements robin-hood hashmaps providing fast search of k-mers. For compression, XCL uses the ZLIB library [74].

XCL can be wrapped to load phylo-k-mer databases and use them in programs in other languages. XCL includes such a wrapper for the python language. It is currently under development, and although it does not provide full functionality, it does make it possible to load phylo-k-mer databases and search for k-mers in the database. Wrappers like this can be helpful for quickly prototyping in scripting languages like python.

4.2 RAPPAS2: a faster reimplementation of RAPPAS

RAPPAS2 is a reimplementation of RAPPAS' placement algorithm in C++. The placement algorithm is identical to the one of RAPPAS; however, thanks to the efficient implementation, RAPPAS2 allows to place queries much faster than RAPPAS (we will compare their performance later in Section 4.4.1). The input of RAPPAS2 is a phylo-k-mer database and a set of query sequences.

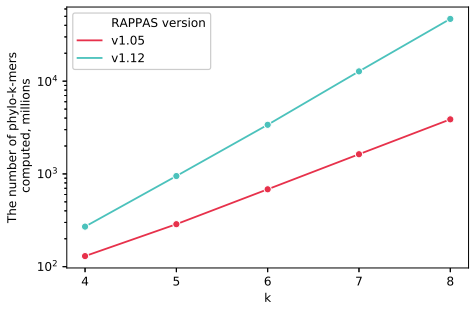
First, RAPPAS2 uses XCL to load the phylo-k-mers into RAM and create an associative array: every w is mapped to the list of pairs $\{(y, S_y(w))\}$. Then, it places queries one by one. To place a query, it splits the query into k-mers, searches for the k-mers in the hashmap, and places the query in the same way as RAPPAS (see Section 1.6, Equation 1.6).

4.3 The phylo-k-mer database computation bug in early versions RAPPAS

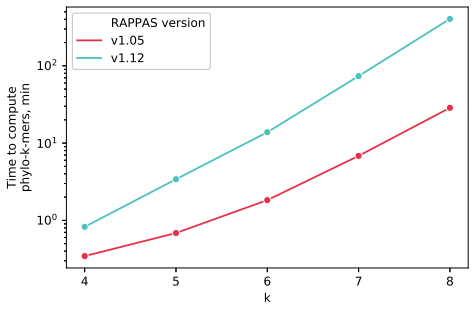
While implementing XPAS, I found a serious bug in the phylo-k-mer computation procedure of RAPPAS. The algorithm for computing phylo-k-mers stopped much earlier than expected, which resulted in many phylo-k-mers with scores above ε not being computed. For all these phylo-k-mers, the value of the ε was thus assumed according to Equation 1.4. This bug was fixed in RAPPAS v.1.12, but it was still in the code when the original article was published. There is reason to believe that incomplete phylo-k-mer databases caused the placement accuracy of RAPPAS to be underestimated in that article [130]. For example, for datasets D218 and D652, which contained a high percentage of gaps (more than 30% of the reference alignment positions), the average RAPPAS placement accuracy worsened with increasing k-mer lengths. The authors believed that the gaps were the reason for this, but it could have been related to the algorithm's omission of the phylo-k-mers due to the bug.

Correcting the bug led to two results. First, the computation time of phylo-k-mers increased considerably since many more phylo-k-mers should be computed. Figure 4-5a demonstrates the difference in the number of phylo-k-mers being computed if the bug is fixed, and it almost reaches one order of magnitude for $k = 8$, which was the default value of k for DNA data. Consequently, the time to calculate phylo-k-mers grows pro-rata (Figure 4-5b), and the fixed version is about one order of magnitude slower at constructing phylo-k-mer databases.

The second consequence is improved placement accuracy. Figures 4-6, 4-7 show the results of PAC experiments for two datasets, D652 and D218. For D652, the mean node distance of the phylogenetic placement performed by RAPPAS v1.21 (currently its latest version, in which the bug is not present) is much lower than that of RAPPAS v1.05 (which contained the bug) for $k = 8$ and $k = 10$. The mean node distance obtained by XPAS+RAPPAS2, run with the same parameters, is also presented for comparison and is consistent with the results obtained by RAPPAS v1.21. For



(a) The total number of phylo-k-mers computed in millions, log scale.



(b) Time needed to compute phylo-k-mers in minutes, log scale.

Figure 4-5 – Comparison of phylo-k-mer database computation for the D652 dataset ($\omega = 1.0$, gap jumps disabled) performed by two versions of RAPPAS: with and without the bug, v1.05 and v1.12 respectively.

D218, the effect of the bug on accuracy is less noticeable but is also present. Again, the results obtained by XPAS+RAPPAS2 match those of RAPPAS v1.21, which improves the results of RAPPAS v1.05 containing the bug.

4.4 Experimental results

4.4.1 Time and memory requirements

I carried out experiments measuring placement time for EPA-ng, App-SpaM, RAPPAS and RAPPAS2 for two datasets, neotrop ([140], 18S rRNA environmental sequences) and D155 ([130], complete genome sequences of the Hepatitis C virus). For neotrop, I used one hundred thousand queries used in [15]; for D155, I used one million simulated sequences from [130]. Placement time does not include the time needed to preprocess the dataset, i.e., query alignment for EPA-ng and the phylo-k-mer database computation for RAPPAS and XPAS+RAPPAS2. I run all tools with the default values of parameters; I also run alignment-free tools with some non-default parameters. See Table 4.1 for the parameter values and their interpretation.

Experiments were carried out one by one on a single core of a computer equipped with Intel(R) Xeon(R) W-2133 CPU @ 3.60GHz / 8.25 MB Cache / 62 GB RAM. All experiments of RAPPAS took two threads to run (because of the additional thread reserved by the virtual machine). Running time and memory consumption was measured with `/usr/bin/time`, which reports the total memory consumption of the process (i.e., for RAPPAS, all memory implicitly reserved by the Java machine but not used by RAPPAS is counted). For the running time, the total wall clock time is considered. For the RAM consumption, the peak maximum resident set size (RSS) is considered.

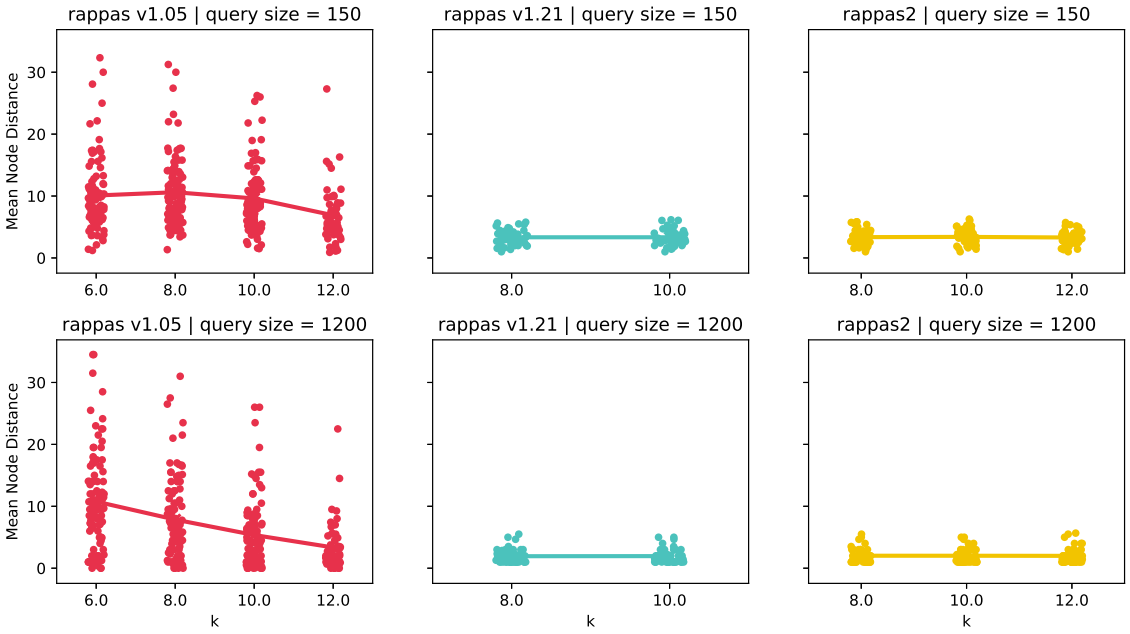


Figure 4-6 – Mean node distance for D652 obtained with RAPPAS v1.05 (includes the bug), the fixed version of RAPPAS, and XPAS+RAPPAS2. The threshold value is $\varepsilon = (1.5/4)^k$.

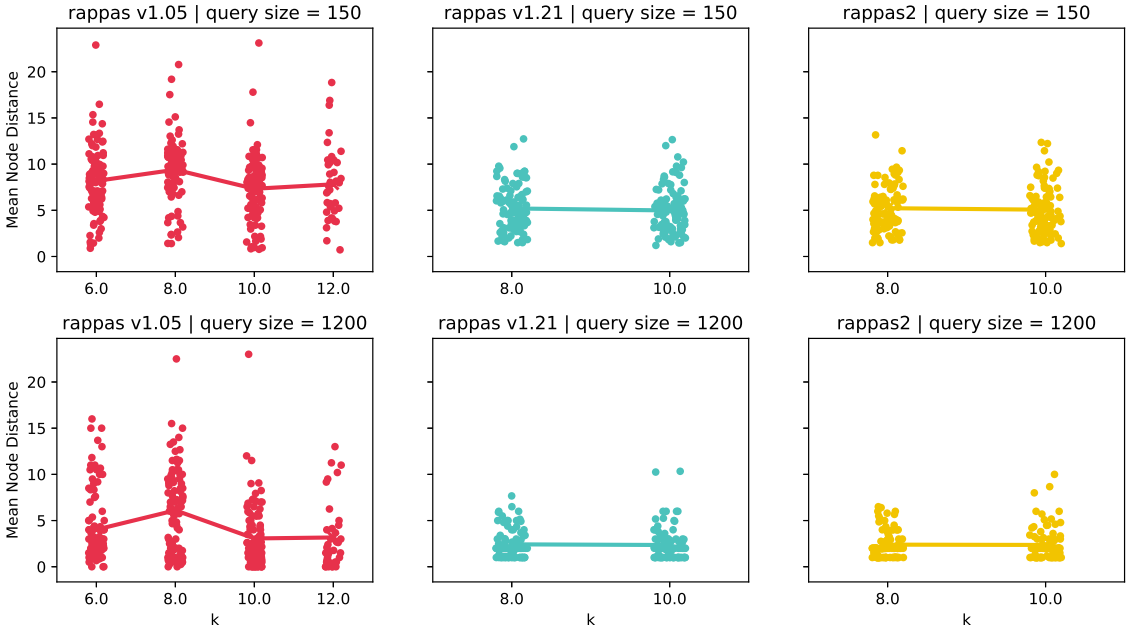


Figure 4-7 – Mean node distance for D218 obtained with RAPPAS v1.05, RAPPAS v1.21, and XPAS+RAPPAS2 for the threshold value of $\varepsilon = (1.5/4)^k$.

Software	Parameters
RAPPAS	$k = 7$, $\omega = 1.5^*$
RAPPAS	$k = 8^*$, $\omega = 1.5^*$
XPAS+RAPPAS2	$k = 8$, $\omega = 1.5^*$
XPAS+RAPPAS2	$k = 10^*$, $\omega = 1.5^*$
App-SpaM	$w = 8$, $p = 1^*$, $m = \text{LCACOUNT}^*$
App-SpaM	$w = 12^*$, $p = 1^*$, $m = \text{LCACOUNT}^*$
EPA-ng	--dyn-heur*, $g = 0.99999^*$

* Default parameters and options.

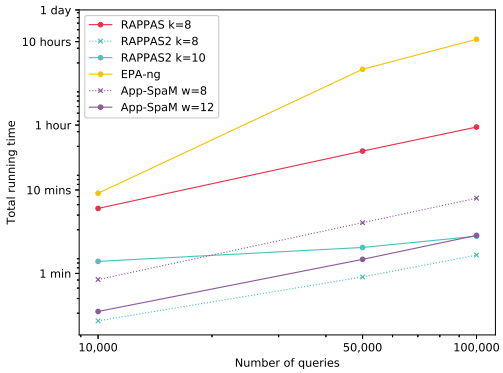
Table 4.1 – The parameters of tested software for placement time experiments. For RAPPAS and XPAS+RAPPAS2, lower values of k generally lead to faster placement with lower accuracy. For App-SpaM, higher values of w lead to faster placement with lower accuracy.

Running time

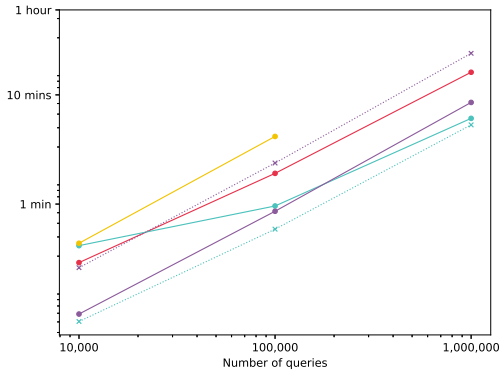
The neotrop experiment (see Figure 4-8a) shows that RAPPAS2 improves RAPPAS by more than an order of magnitude ($k = 8$): to place 100,000 queries, RAPPAS takes almost an hour, while RAPPAS2 finishes under two minutes (34-fold change). The improvement for the D155 experiment (Figure 4-8b) is less: RAPPAS places a million queries in 16 minutes, while RAPPAS2 does it in five minutes. Both experiments show that App-SpaM outperforms RAPPAS, which is consistent with [22]. However, RAPPAS2 scales better than App-SpaM with the increasing number of queries regardless of the parameter values. Recall that this comparison considers only the running time of placement (but not that of phylo-k-mer database computation). Also, note that for $k = 10$, the running time of RAPPAS2 placing tens and hundred thousand queries is dominated by the constant time needed to first load the phylo-k-mer database into RAM, which explains the slow growth for RAPPAS2 in Figure 4-8a.

I also compared RAPPAS and XPAS in terms of the running time to compute a database of phylo-k-mers for the same datasets and different values of k. Figure 4-9 shows the results averaged over three runs. These measurements only include time to compute phylo-k-mers and write resulting databases on disk, which is the most time-consuming part of the database creation. (Another time-consuming part, ancestral reconstruction, is identical for both tools since they use external software for this.) For neotrop, the improvement of XPAS over RAPPAS is 35-fold for $k = 7$ (30 seconds against 18 minutes), and is 100-fold for $k = 10$ (11 minutes against 18 hours). For D155, XPAS is two orders of magnitude faster than RAPPAS: 30 seconds against one hour for $k = 8$, and four minutes against 15 hours for $k = 10$.

The default k-mer length in RAPPAS is 8. However, the faster running time of XPAS allows for changing the default value of k to 10. Experiments on placement accuracy (will be discussed later) suggest that using phylo-k-mer databases with $k = 10$ often improves placement accuracy compared to databases with $k = 8$. Using

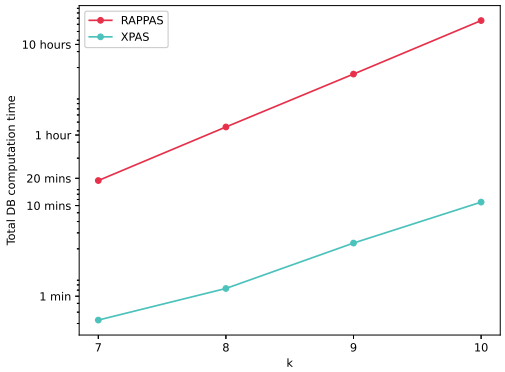


(a) Placement time for NEOTROP.

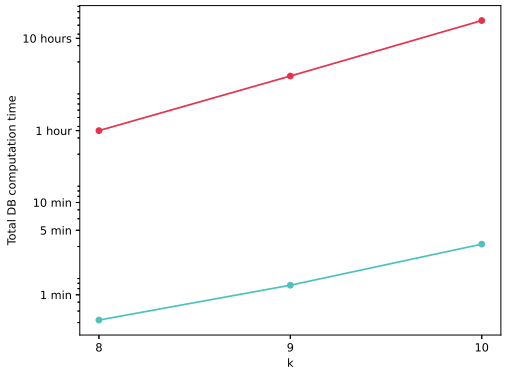


(b) Placement time for D155.

Figure 4-8 – Running time taken by RAPPAS, RAPPAS2, EPA-NG and APP-SPAM to place different number of query sequences for NEOTROP and D155. Preprocessing (i.e., query alignment for EPA-NG and phylo-k-mer database computation for RAPPAS and RAPPAS2) is not included. Solid lines correspond to default parameter values.

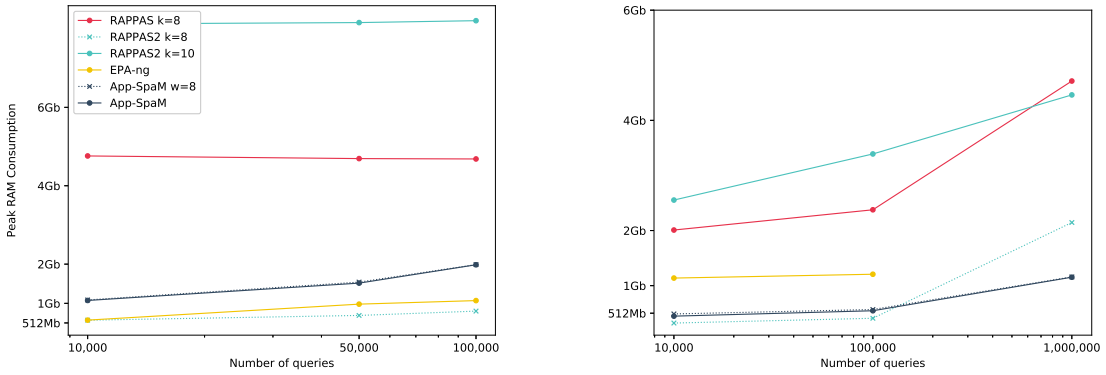


(a) Running time for preprocessing the NEOTROP dataset.



(b) Running time for preprocessing the D155 dataset.

Figure 4-9 – Running time of RAPPAS and XPAS needed to compute a database of phylo-k-mers for NEOTROP and D155 using the default threshold value of $\omega = 1.5$.



(a) Memory consumption of placement for NEOTROP.

(b) Memory consumption of placement D155.

Figure 4-10 – Peak memory consumption of phylogenetic placement tools for two datasets and different numbers of queries. Peak RAM during preprocessing is not included. Solid lines correspond to default parameter values.

values of k higher than 10 may produce databases that are too big in size, often without improving placement accuracy.

Memory consumption

Figure 4-10 gives a comparison of RAPPAS, RAPPAS2, EPA-NG and APP-SPAM in terms of peak RAM usage in the same experiments that measured the running time. The NEOTROP experiment showed that RAPPAS2 improved the memory consumption of RAPPAS from 4Gb to under 1Gb (if both tools are run with $k = 8$). RAM consumption of RAPPAS was the same while placing different number of queries, probably due to the Java machine reserving memory. In the D155 experiment, RAPPAS2 also improves the result of RAPPAS (reducing it from 4Gb to 2Gb for one million queries placed). APP-SPAM showed lower memory consumption than RAPPAS2 if the latter was run with $k = 10$. For $k = 8$, RAPPAS2 shows lower memory consumption than APP-SPAM for placing no more than one hundred thousand queries.

4.4.2 Placement accuracy

Placement accuracy experiments consisted of running the pruning-based accuracy pipeline of PEWO for several DNA datasets based on short alignments — D218 (bacterial 16S rRNA, 2.3kbp), D500 (chloroplast rbcL gene, 1.4kbp) from [20], D652 (bacterial 16S rRNA, 1.7kbp) from [130] — and two datasets based on longer alignments: the HIV-GENOME dataset (13kbp) containing whole-genome sequences of HIV from [189], and D155 (10kbp), a dataset of complete genomes of hepatitis C virus [130]. PAC experiments were run with different parameters for each software. RAPPAS was run as

```
$ java ... -jar RAPPAS.jar ... --gap-jump-thresh 1.0
```

to disable experimental gap jump strategies. XPAS was run as

```
$ python xpas.py build ... -u 1.0 --filter no-filter
```

to disable phylo-k-mer filtering. Under these conditions, RAPPAS and XPAS produce almost identical phylo-k-mers databases: differences are only in phylo-k-mers close to the threshold so that the floating-point operations in different implementations start to matter. Those differences are negligible. RAPPAS and XPAS+RAPPAS2 were run with several values of k. Other tools were run with their default parameters.

For every dataset, queries of different sizes were placed: 150, 1200 base pairs for datasets based on short alignments (D218, D500, D652), and 150, 10000 base pairs for viral datasets of longer alignments (D155, HIV-GENOME). For every software-parameter combination, the mean node distance among all prunings was calculated. The resulting values are shown in Figure 4-11. For RAPPAS and XPAS+RAPPAS2 results obtained for $k = 10$ are shown, since it is the default value of k for XPAS. For each experiment, mean node distance values, not averaged over different prunings, can be found in the appendix (Figures B-1, B-2, B-3, B-4, B-5). The results obtained for RAPPAS and XPAS+RAPPAS2 for other values of k can also be found there.

In most experiments, APPLES showed worse accuracy than other tools (Figure 4-11), which is consistent with the results of [22]. Since APPLES targets ultra-large trees, and these experiments were carried out for small trees (the largest was HIV-GENOME, with 881 leaves), I will not further discuss APPLES here. In all experiments, EPA-NG and PPLACER showed similar accuracy; because of this and because EPA-NG supports PPLACER's algorithm as an option, I will only discuss the results of EPA-NG. Both RAPPAS and XPAS+RAPPAS2 were run for three datasets: D218, D500, and D652 (Figures 4-11a, 4-11c). In these experiments, two versions of RAPPAS show identical accuracies, indicating that the RAPPAS results are reproducible with XPAS+RAPPAS2. Having this in mind, I will focus on comparing the accuracy of RAPPAS2 with that of EPA-NG and APP-SPAM.

Let us first consider short-alignment datasets (D218, D500, and D652, see Figures 4-11a and 4-11c). RAPPAS2 shows slight improvement over the competing methods when placing short sequences (Figure 4-11a). For dataset D218, this replicates and confirms the results of [22]. However, RAPPAS2 is slightly inferior to EPA-NG (but not APP-SPAM) when placing longer queries (1200 base pairs, Figure 4-11c), except for D652.

Now, let us consider long-alignment viral datasets (D155 and HIV-GENOME, see Figures 4-11b and 4-11d). RAPPAS was not run for these datasets because it takes too long to preprocess them using RAPPAS. Instead, only XPAS+RAPPAS2 were run to compare against other methods, and the results are inconsistent. APP-SPAM shows higher accuracy than EPA-NG and RAPPAS2 on the HIV dataset for short queries (Figure 4-11b), but not for long ones, for which RAPPAS2 shows the best result (Figure 4-11d). For D155, RAPPAS2 shows similar accuracy compared to EPA-NG in both short and long sequence placement (Figures 4-11b, 4-11d), while APP-SPAM shows worse accuracy in both cases.

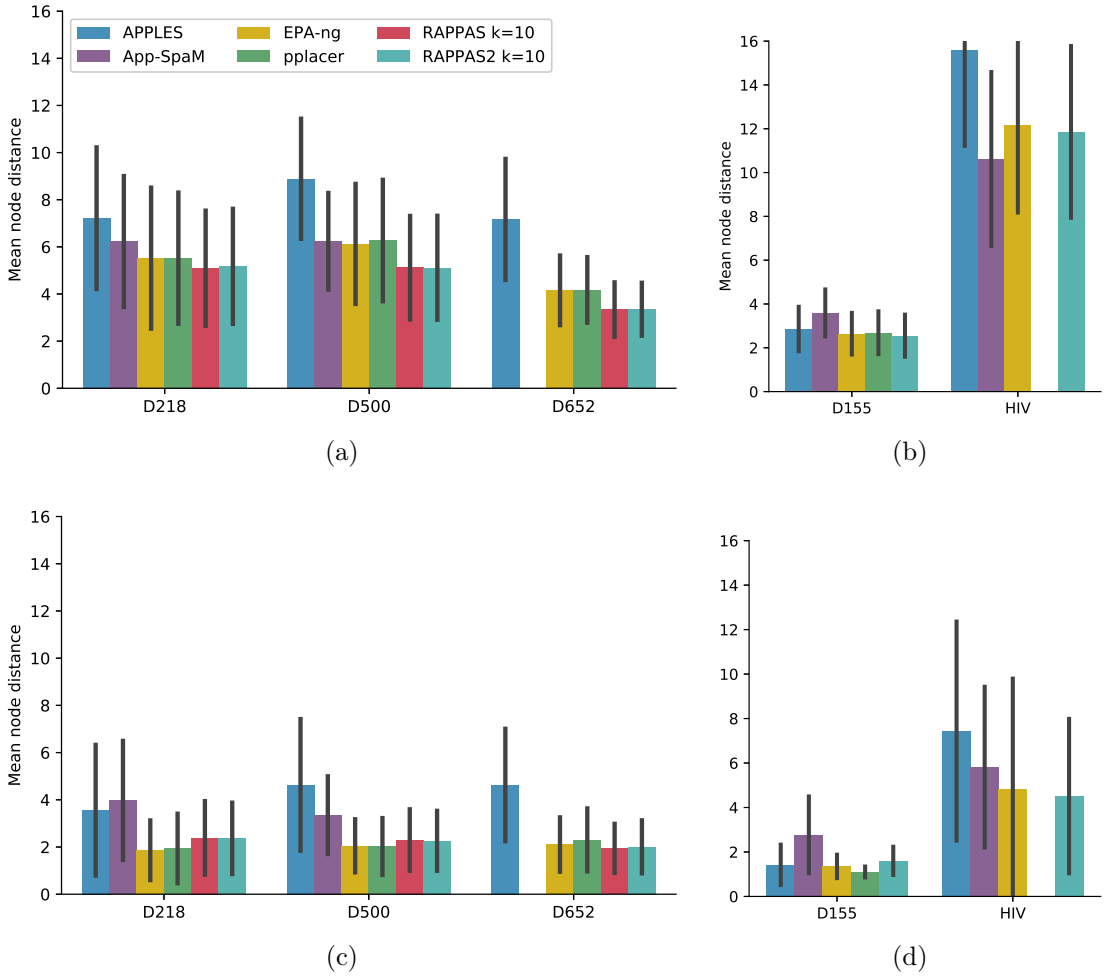


Figure 4-11 – Mean (colored bars) and standard deviation (black lines) of node distance obtained for PAC experiments on multiple datasets and different query sizes. Lower values are better. (a), (b): results for placing queries of 150 base pairs for short-alignment and long-alignment datasets, respectively. (c): results for placing queries of 1.2 Kbp in length for short-alignment datasets. (d): results for placing queries of 10 Kbp in length for long-alignment datasets.

4.5 Conclusion

I completely reimplemented RAPPAS, splitting it into two separate programs: XPAS and RAPPAS2. This is to make it more convenient to develop new software applying phylo-k-mers for other bioinformatics tasks. In addition to the new algorithm for calculating phylo-k-mers, XPAS also applies the filtering methods described in Chapter 3. To reduce memory consumption during filtering, I proposed and implemented the batch filtering approach. The XPAS Core Library, which underlies XPAS, can be used by other applications to load and use XPAS-built phylo-k-mer databases. It is currently used in three phylo-k-mers based software: RAPPAS2, SHERPAS, and CLAPPAS.

I conducted experiments to compare the accuracy of the phylogenetic placement of RAPPAS2 and the state-of-the-art placement tools. RAPPAS2 shows the same placement accuracy as RAPPAS, i.e., high accuracy, comparable to that of maximum-likelihood methods (EPA-ng and pplacer). In addition, I have conducted experiments measuring the speed of computing phylo-k-mer databases using XPAS and placement using RAPPAS2. XPAS is one to two orders of magnitude faster than RAPPAS in phylo-k-mer database computation; the improvement depends on the input parameters and the dataset. As for placement, RAPPAS2 is significantly faster than both RAPPAS (up to one order of magnitude) and EPA-ng (when run in a single thread).

The speed competitor for RAPPAS2 is App-SpaM, a recently released alignment-free phylogenetic placement software. App-SpaM, although it may outperform RAPPAS2 in placement speed (depending on the input parameters and the number of queries), is nevertheless inferior in placement accuracy to RAPPAS2 in most experiments. These results are consistent with those obtained by the authors of App-SpaM [22]. In addition, with the increasing number of query sequences, the running time of RAPPAS2 scales better than that of App-SpaM.

The faster speed of XPAS compared to RAPPAS allows for longer k-mer lengths, which may lead to higher placement accuracy. Consequently, the default value of k can be increased to 10. The lower memory consumption of XPAS, as well as the implemented phylo-k-mer filtering methods, increase the possibility of using XPAS and RAPPAS2 on machines with small amounts of RAM. Accurate alignment-free phylogenetic placement with phylo-k-mers for small and medium-size datasets can now be performed on any modern laptop.

Chapter 5

Conclusion and perspectives

In this final chapter, I summarize the results of my work and discuss some directions for future work on phylo-k-mers, and outline my thoughts on the future of the field.

The *raison d'être* of phylo-k-mers is to represent phylogenetic information in the form of k-mers, which are straightforward to use in designing new alignment-free methods. At the moment, there are three phylo-k-mer based methods for solving different bioinformatics problems: besides phylogenetic placement, phylo-k-mers are used by SHERPAS for recombination detection in viruses [189] and by CLAPPAS for protein family classification (unpublished results). Other methods may be developed in the future. They all need to have the phylogeny be represented with phylo-k-mers to work. RAPPAS presented the method and its implementation for this problem. However, it had limitations for peak memory usage and running time for large datasets. My work addressed those limitations, and the objective was to answer the question of *how to efficiently represent phylogenies with phylo-k-mers*.

I addressed this question from three different perspectives. First, I described the algorithmic side of the question: I considered the problem of computing phylo-k-mers that had earned little attention before. I provided a new algorithmic solution to this problem, described in Chapter 2. Under one reasonable condition, this problem is solved optimally. Nevertheless, algorithms showing faster performance in practice may exist; I will discuss this in Section 5.1.

Second, I addressed the question from a mathematical and information-theoretic point of view. Studying the problem of finding informative (for phylogenetic placement) phylo-k-mers led to a rethinking of *why and how exactly the method of* RAPPAS *works* from a machine learning and classification perspective. I connected RAPPAS to the Bernoulli Naive Bayes Classification used for classifying texts. Not only did it give insights on how to filter phylo-k-mers (which led to the results of Chapter 3), but it also provided food for thought on improving the RAPPAS method, which I will cover in Section 5.1. The phylo-k-mer filtering approach I suggested allows decreasing the size of phylo-k-mer databases dramatically. This in turn reduces RAM consumption during phylogenetic placement, which improves the scalability of RAPPAS for larger datasets.

Finally, I applied all my knowledge of software development to create more effective programs for phylo-k-mer computation (XPAS) and phylogenetic placement

(RAPPAS2, Chapter 4). Speed improvement of those programs is up to two orders of magnitude, which will allow applying phylo-k-mer based methods for larger phylogenies and larger amounts of sequencing data.

5.1 Future work

5.1.1 The optimal algorithm for phylo-k-mer computation

Section 2.3.3 described a new divide-and-conquer algorithm for computing phylo-k-mers in a k-sized window W of the alignment for a ghost node of a given branch. The algorithmic complexity of this algorithm is $\mathcal{O}(k \cdot \sigma^{k/2} + |\mathcal{Z}_\varepsilon^W|)$, where $\mathcal{Z}_\varepsilon^W$ is a set of k-mers obtaining scores higher than ε in this window. It is an improvement over the existed branch-and-bound algorithm. Moreover, it is easy to see that the new algorithm is optimal under the condition $|\mathcal{Z}_\varepsilon^W| = \Omega(k \cdot \sigma^{k/2})$.

Whether there is an algorithm computing phylo-k-mers in $\mathcal{O}(|\mathcal{Z}_\varepsilon^W|)$ time for *any* input remains unclear. Future work could further develop algorithms working in optimal time if $\mathcal{Z}^W$ contains only a few k-mers achieving scores higher than ε.

5.1.2 Alternative score model

A possible improvement to RAPPAS comes from a thorough reassessment of the meaning of phylo-k-mer scores, which has been the subject of an important reflection that we carried out as part of my work. I will try to summarize it here briefly.

RAPPAS views a query as a sequence of k-mers $W_1, W_2, \ldots, W_{m-k+1}$, where W_j is the k-mer starting at the jth site in the query. Each k-mer W_j can be seen as a random variable whose distribution depends on the phylogenetic origin of the query. This distribution is given by the $P_j^u(w)$ values defined in Section 1.5.2. In other words,

$$\mathbb{P}\left[W_j = w \mid \text{the query originates from node } u\right] = P_j^u(w).$$

We also assume, quite unrealistically, that the W_j for different values of j are statistically independent of each other. (In particular, this is unrealistic because consecutive k-mers such as W_j and W_{j+1} should in principle share a common $(k-1)$-mer, but the assumption of independence does not enforce this.)

Now let us denote "the query originates from node u" simply with u, and let us focus on the following probabilities:

$$\mathbb{P}\left[W_1, W_2, \ldots, W_{m-k+1} \text{ contains } w \mid u\right] = \mathbb{P}\left[\exists j : W_j = w \mid u\right]. \tag{5.1}$$

We can express them as follows:

$$\begin{aligned}
\mathbb{P}[\exists j : W_j = w \mid u] &= 1 - \mathbb{P}[\forall j : W_j \neq w \mid u] \\
&= 1 - \prod_{j=1}^{m-k+1} \mathbb{P}[W_j \neq w \mid u] \\
&= 1 - \prod_{j=1}^{m-k+1} \left(1 - \mathbb{P}\left[W_j = w \mid u\right]\right) \\
&= 1 - \prod_{j=1}^{m-k+1} \left(1 - P_j^u(w)\right)
\end{aligned} \tag{5.2}$$

We are now going to make a final assumption, and later we will examine how we should proceed if we do not wish to make this assumption. The assumption — which we call the *strong no co-occurrence hypothesis* — is that k is sufficiently large that the probabilities $P_1^u(w), P_2^u(w), \ldots, P_{m-k+1}^u(w)$ are all very close to 0 with the exception of at most one of them. This is similar, but not equivalent, to the assumption that no k-mer can occur more than once in a query of size m. The strong no co-occurrence hypothesis presumes further that there is at most one position in the query where the probability of observing w is non-negligible. Under this assumption, it is easy to see that

$$1 - \prod_{j=1}^{m-k+1} \left(1 - P_j^u(w)\right) \approx \max_{j=1}^{m-k+1} P_j^u(w). \tag{5.3}$$

Recall that $S^u(w) \approx \max_{j=1}^{m-k+1} P_j^u(w)$ (Equation 1.4), which combined with Equations 5.1, 5.2, 5.3, provides a straightforward interpretation for $S^u(w)$:

$$S^u(w) \approx \mathbb{P}\left[W_1, W_2, \ldots, W_{m-k+1} \text{ contains } w \mid u\right]. \tag{5.4}$$

Now recall that $S_y(w)$ is defined as the maximum of $S^u(w)$ across all ghost nodes $u \in G_y$ (Eqn. 1.5), which also gives us a relatively simple interpretation for the phylo-k-mer scores:

$$S_y(w) \approx \max_{u \in G_y} \mathbb{P}\left[W_1, W_2, \ldots, W_{m-k+1} \text{ contains } w \mid u\right]. \tag{5.5}$$

We now turn to the central question of this section: given that Equations 5.4 and 5.5 provide the intended meanings for $S^u(w)$ and $S_y(w)$, are there better ways to define $S^u(w)$ and $S_y(w)$, based on formulas whose validity does not depend on the strong no co-occurrence hypothesis?

The answer is trivially yes, and is obtained by simply using the left-hand side of Equation 5.3 instead of its right-hand side. This leads us to the following new definitions:

$$S_u^{\exists}(w) := \max\left\{1 - \prod_{j=1}^{m-k+1}\big(1 - P_j^u(w)\big), \varepsilon^{\exists}\right\}, \tag{5.6}$$

$$S_y^{\exists}(w) := \max_{u \in G_y} S_u^{\exists}(w) \tag{5.7}$$

Here we smooth out the very low probabilities by ensuring that no stored probability is smaller than $\varepsilon^{\exists}$, similarly to what is done for $S^u(w)$. In the original definition, ε could be interpreted as the minimum value allowed for $P_j^u(w)$. Then, the minimum value allowed for $S_u^{\exists}(w)$ is simply given by the following formula, which I take as definition of $\varepsilon^{\exists}$:

$$\varepsilon^{\exists} := 1 - (1 - \varepsilon)^{m-k+1}. \tag{5.8}$$

This completes the definition of the alternative model to score phylo-k-mers. One direction for future work may be to implement in XPAS the formulas 5.6, 5.7 and 5.8. I will then conduct experiments measuring phylogenetic placement accuracy on various datasets. It will be particularly interesting to test the new approach on long reference alignments, where m is large relative to k, and thus the strong no co-occurrence hypothesis is most likely to be violated.

5.1.3 Bernoulli-based phylogenetic placement

In Chapter 3, we discussed that the underlying model of RAPPAS is closely related to the Bernoulli model for text classification. I also described the method of phylogenetic placement based entirely on the Bernoulli model (see Section 3.4.1). Applying the same method of estimating scores of phylo-k-mers as in RAPPAS, or the alternative score model from Section 5.1.2, we can place the query sequences using Bernoulli's formula (Equation 3.26):

$$\hat{y}(q) = \operatorname*{arg\,max}_{y \in E(T_0)} \Big(\prod_{w:b_w=1} fS_y(w) \prod_{w:b_w=0} (1 - fS_y(w)) \Big) \tag{5.9}$$

with f being the parameter that depends on the query length:

$$f = \frac{|q| - k + 1}{m - k + 1} \tag{5.10}$$

Recall that RAPPAS uses this formula instead:

$$\hat{y}(q) = \operatorname*{arg\,max}_{y \in E(T_0)} \prod_{w:n_w>0} S_y(w)^{n_w} \tag{5.11}$$

An interesting difference between Formula 5.9 and 5.11 is that the first one takes the *absence of k-mers* into account. For any k-mers present in full-sized sequences originated from y with high probability, the Bernoulli placement formula penalizes y for those queries that do not contain those k-mers. In other words, if we expect w to be present with high probability in sequences from y, then the absence of w in the query is taken by Bernoulli placement (but not RAPPAS) as evidence that y *is not*

the correct placement for the query. This penalty is stronger the closer the value of f is to one, i.e., the closer the query size is to the reference alignment size.

How good such a classifier is in practice, it is possible to find out only experimentally. One direction of development of RAPPAS2 may be the implementation of Bernoulli placement and testing this model for different types of data. I suspect that this formula may help us achieve higher placement accuracy than the classic formula for trees inferred from conservative alignments, where the absence of important k-mers in the query may be informative. In addition, perhaps it may improve placement accuracy for those queries whose length approaches the length of the alignment.

5.1.4 Parallelism and distributed memory

From Figure 4-3, it is easy to see the parallel nature of computing and filtering phylo-k-mers. This is no coincidence: during the XPAS design process, I envisioned future parallelization of the phylo-k-mer database creation. This process can be seen as a two-stage parallel algorithm:

1. Compute phylo-k-mers for each ghost node of the tree in parallel. The results of these calculations are serialized on disk.

2. Merge the branch databases into batch databases.

3. Calculate filter values for each batch in parallel. After, serialize the results on disk.

4. Merge the previous step's results using the β-way merge algorithm and select filtered k-mers.

5. Form the final phylo-k-mer database from filtered k-mers.

Thus, XPAS can be implemented in distributed memory to increase the method's applicability and create phylo-k-mer databases for larger trees. Filtering the phylo-k-mers will allow controlling the memory consumption and the size of the resulting databases. For example, such an implementation would make it possible to prepare phylo-k-mer databases for large trees based on popular phylogenetic markers such as greengenes [52] (16S rRNA sequences of bacteria and archaea), SILVA [165] (small 16S/18S and large subunit 23S/28S rRNA of bacteria, archaea, and eukarya) or EzBioCloud 16S database [234] (a large collection of genome sequence-derived 16S rRNA sequences). Once computed, those databases can be published for accurate alignment-free placement on those phylogenies.

5.2 Final thoughts

I would like to wrap up my thesis with an historical example. In January 1954, a major event occurred in New York City for all machine translation researchers: IBM and George Washington University presented a new machine translation system [97]. This system could automatically translate short sentences from Russian into English, which was an incredible success at the time. Here are examples of translations made by this machine that day:

Original (romanized)	**Translation**
Mi pyeryedayem mislyi posryedstvom ryechi.	We transmit thoughts by means of speech.
Vyelyichyina ugla opryedyelyayatsya otnoshyenyiyem dlyini dugi k radyiusu.	Magnitude of angle is determined by the relation of length of arc to radius.

As a native Russian speaker, I can confirm that those translations are correct. This experiment brought much attention to the field of machine translation. The expectations of contemporaries were very high: one of researchers, Léon Dostert, stated that "five, perhaps three, years hence, interlingual meaning conversion by electronic process in important functional areas of several languages may well be an accomplished fact." [97]. Two years later, the famous Dartmouth Summer Research Project on Artificial Intelligence took place, which is considered to be the founding event of the field of artificial intelligence [157]. Nevertheless, a decade of research in this area led to a great disappointment. The problem of machine translation turned out to be much more challenging than it seemed. A critical report by the ALPAC (Automatic Language Processing Advisory Committee) reviewing the state of the field, published in 1966, dashed all hopes of a quick solution to the problem of machine translation [96, 95, 5]. This report led to a significant decrease in funding for the field of machine translation. Curiously, one of important applications of machine translation at the time was the automated translation of Russian-language scientific publications into English. However, the committee was so disappointed with the results of automatic translations that it admitted that it would be easier to learn Russian [5].

The field of machine translation has had a *very long* road to success. Rosenblatt introduced the perceptron model in 1958 [174]; backpropagation was proposed back in the 1970s and refined in 1986 ([176, 131]). The machine learning boom of the 1990s and early 2000s, and the development of deep learning in the 2010s finally led to the fact that anyone can benefit from quality machine translation services. It took not three, not five, but *more than fifty years* to bring the idea from proof-of-concept to ready-to-use solutions.

This is, of course, an extreme example. However, I think it can teach us a thing or two. First, the fact that our expectations sometimes may be unreasonably high. Projecting this onto bioinformatics, I am afraid that we can hardly expect to describe all the available diversity of life anytime soon, even with NGS technologies and new

methods in phylogenetics. On the other hand, this would mean that phylogenetic placement for sequence identification will remain relevant for a long time. The second lesson of this story is that it is not enough to develop new scientific methods. The essential condition for success is our ability to disseminate these methods and convince the public that they need them. Only by doing this can we engage the necessary resources and new people to move the field forward truly. The scientific community of the 1960s failed to convince ALPAC that the field of machine translation was worthy of funding, which slowed progress in the field for many years.

Returning to the thesis topic, I have to admit that I observe a poor awareness of the bioinformatics community about phylogenetic placement. It has begun to attract more and more interest with the emergence of new methods, but after ten years of research and development, it is rarely used in large pipelines for analyzing metabarcoding data. Despite the emerging evidence showing that for short 16S rRNA sequences, phylogenetic placement should be preferred to inferring phylogenies *de novo* [100], the latter is standard in the analysis of amplicon data. I cannot explain this by anything other than poor communication. The situation has changed somewhat with the integration of SEPP into the QIIME2 pipeline, but it has been almost ten years since SEPP was published.

Probably the main problem of communication in bioinformatics is its multidisciplinarity. Applied researchers have no time to learn new emerging methods, frameworks, and pipelines. Scientists inventing new methods do not always pay enough attention to bring their programs to a state where they are usable by someone far from programming. (I remember very well the day I first used a bioinformatics tool. The experience of using it was that the flag `--help` did not work.) A classic situation is when an applied researcher runs a complex program with default parameters, even if these parameters are not appropriate for the input data, simply because it is not easy to figure out what these parameters mean. The situation is similar with new scientific software: many applied researchers use the programs they are used to, even if these programs are obsolete.

I, of course, largely address this appeal to myself. I hope the reader will forgive me for being self-centered. Unfortunately, I have not yet done enough to disseminate the results of my work. Since my early days in bioinformatics, I have felt the lack of communication due to a strong division between *bio-bioinformatics* and *info-bioinformatics*. I hope that this division will be overcome in the future.

Appendix A

Tables and proofs

A.1 Proof of Lemma 3.6.2

Given the function

$$g(f) = -\sum_y \frac{1 - fS_y(w)}{N - fS_w} \log \frac{1 - fS_y(w)}{N - fS_w}$$

where log is the base a logarithm, its derivative is:

$$\frac{d}{df} g(f) = -\sum_y \frac{1}{\ln a} \frac{NS_y(w) - S_w}{(N - fS_w)^2} \Big(\ln \frac{1 - fS_y(w)}{N - fS_w} + 1 \Big)$$

Proof. Let us substitute $u = \frac{1 - fS_y(w)}{N - fS_w}$ and $v = \log u$.

Then,

$$\begin{aligned} \frac{d}{df} g(f) &= -\frac{d}{df} \sum_y u(f, y) \cdot v(f, y) \\ &= -\sum_y \frac{d}{df} (u(f, y) \cdot v(f, y)) \end{aligned}$$

Let us find u' and v':

$$\frac{d}{df} u = \frac{d}{df} \frac{1 - fS_y(w)}{N - fS_w} = \frac{-S_y(w)(N - fS_w) + (1 - fS_y(w))S_w}{(N - fS_w)^2} = \frac{S_w - NS_y(w)}{(N - fS_w)^2}$$

.

$$\frac{d}{df}v = \frac{d}{df}\log u = \frac{u'}{\ln a \cdot u} = \frac{N - fS_w}{\ln a \cdot (1 - fS_y(w))} \cdot \frac{S_w - NS_y(w)}{(N - fS_w)^2} =$$
$$= \frac{S_w - NS_y(w)}{\ln a \cdot (1 - fS_y(w))(N - fS_w)}$$

Finally, $g'(f) = -\sum_y (u'v + uv')$:

$$g'(f) = -\sum_y (u'v + uv') =$$
$$= -\sum_y \Big(\frac{S_w - NS_y(w)}{(N - fS_w)^2} \cdot \log \frac{1 - fS_y(w)}{N - fS_w} + \frac{1 - fS_y(w)}{N - fS_w} \cdot \frac{S_w - NS_y(w)}{\ln a \cdot (1 - fS_y(w))(N - fS_w)} \Big) =$$
$$= -\sum_y \frac{NS_y(w) - S_w}{(N - fS_w)^2} \Big(\log \frac{1 - fS_y(w)}{N - fS_w} + \frac{1}{\ln a} \Big) =$$
$$= -\sum_y \frac{1}{\ln a} \frac{NS_y(w) - S_w}{(N - fS_w)^2} \Big(\ln a \cdot \log \frac{1 - fS_y(w)}{N - fS_w} + 1 \Big) =$$
$$= -\sum_y \frac{1}{\ln a} \frac{NS_y(w) - S_w}{(N - fS_w)^2} \Big(\ln \frac{1 - fS_y(w)}{N - fS_w} + 1 \Big).$$

Q.E.D.

□

A.2 Tables

Software	Year	Ref	Language	Method	Conda	1st Author	PMID
PPlacer	2010	[146]	OCaml/C	Aln+ML	y	Matsen F.A.	21034504
EPA (RAxML)	2011	[20]	C	Aln+ML	n	Berger S.A.	21436105
LSHPLace	2013	[29]	N/A	Aln+AS	n	Brown D.G.	23424136
Phyclass	2015	[65]	N/A	Distances	n	Filipski A.	25923672
EPA-ng	2018	[15]	C++	Aln+ML	y	Barbera P.	30165689
RAPPAS	2019	[130]	Java	AS	y	Linard B.	30698645
Apples	2021	[11]	Python	Aln+ML	n	Balaban M.	31545363
AppSPAM	2021	[22]	C++	Distances	y	Blanke M.	N/A

Table A.1 – Auxiliary information about state-of-the-art phylogenetic placement tools. Abbreviations: not available (N/A), maximum-likelihod (ML), alignment (Aln.), ancestral sequences (AS).

Name	Year	Ref	Language	Type	Conda	1st Author	PMID
GUPPY	2011	NA	OCaml	multi-tool	y	Matsen F.A.	NA
Genesis	2019	[42]	C++	library	n	Czech L.	30169747
GAPPA	2019	[42]	C++	library	n	Czech L.	30169747
SEPP	2012	[155]	Python	pipeline	n	Mirarab S.	22174280
pplacerDC	2021	[112]	Python	pipeline	n	Koning E.	NA
pplacerXR	2021	[222]	Python	pipeline	n	Koning E.	NA
PhyloSift	2014	[44]	Perl	pipeline	n	Darling. A.	24482762
HmmUFOtu	2018	[237]	C++	pipeline	n	Zheng Q.	29950165
PhyloMagnet	2019	[191]	Nextflow/ Python	pipeline	n	Schön M.E.	31647547
PEWO	2020	[129]	Python	framework	y	Linard B.	32697844
SCRAPP	2020	[14]	Python	down	n	Barbera P.	32996237

Table A.2 – Phylogenetic placement-related libraries, pipelines, and dowstream analysis tools. Abbreviations: not available (N/A).

Appendix B

Figures

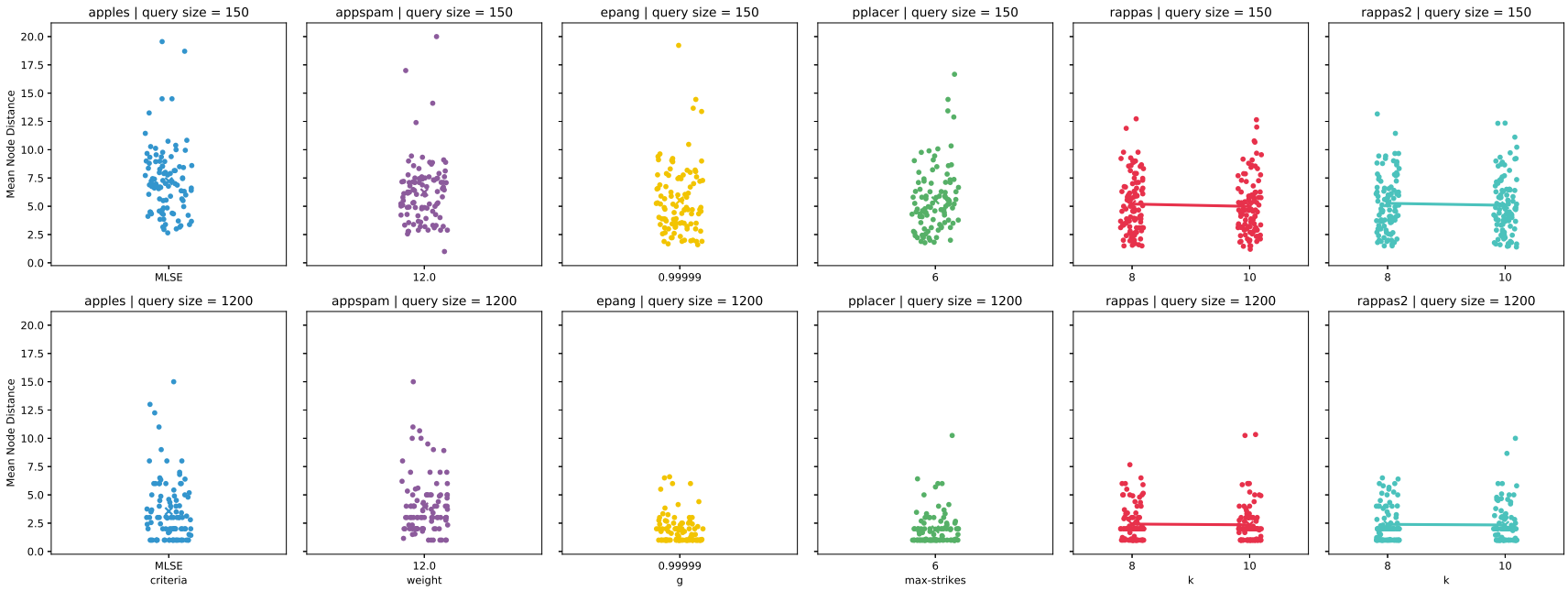


Figure B-1 – The results of PAC experiments on accuracy for D218.

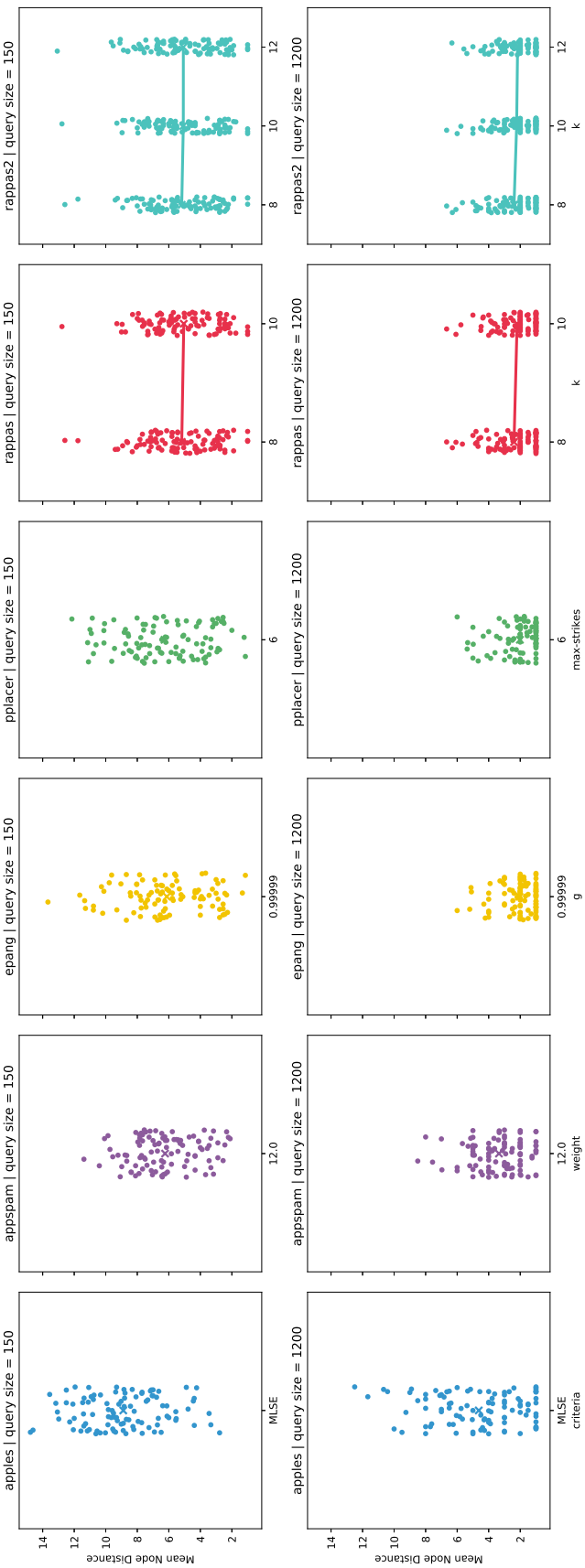


Figure B-2 – The results of PAC experiments on accuracy for D500.

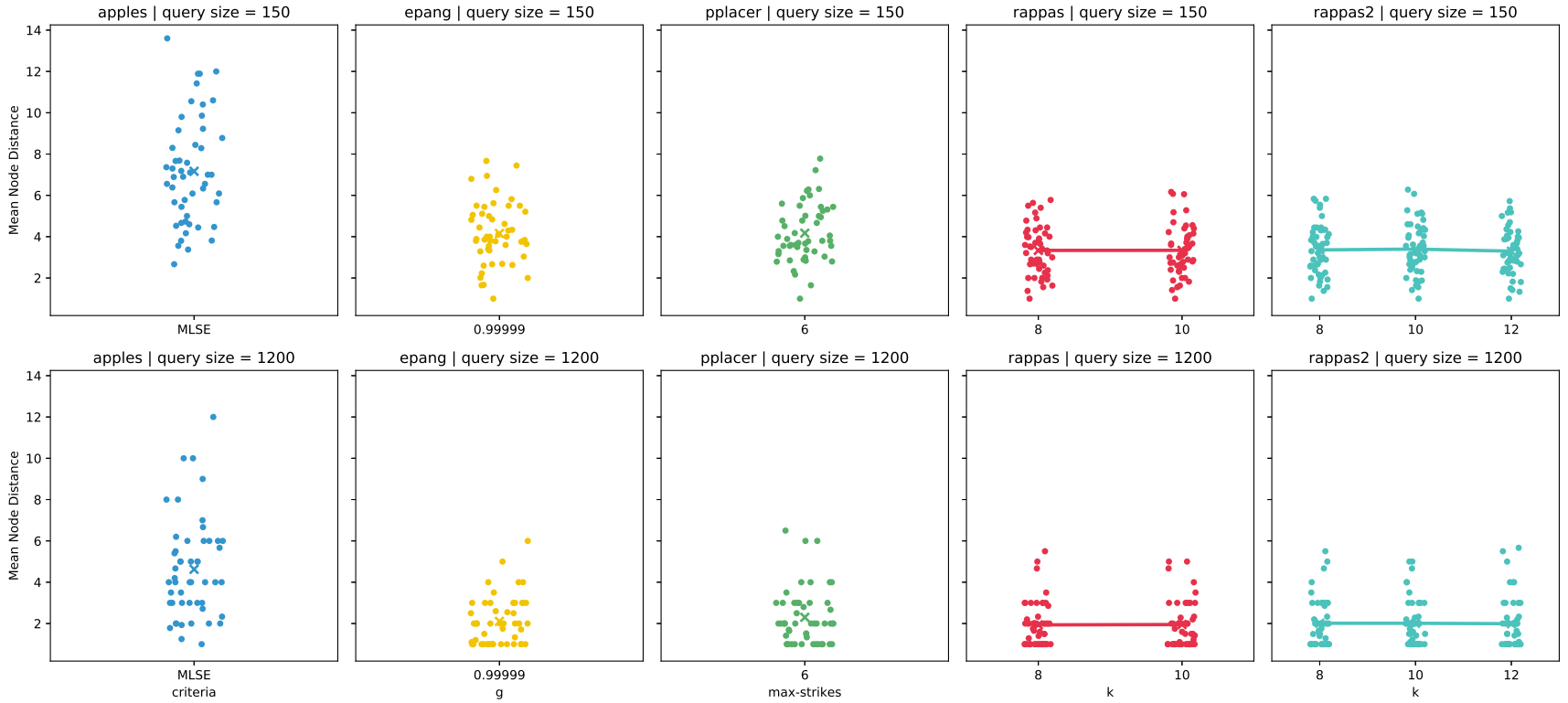


Figure B-3 – The results of PAC experiments on accuracy for D652.

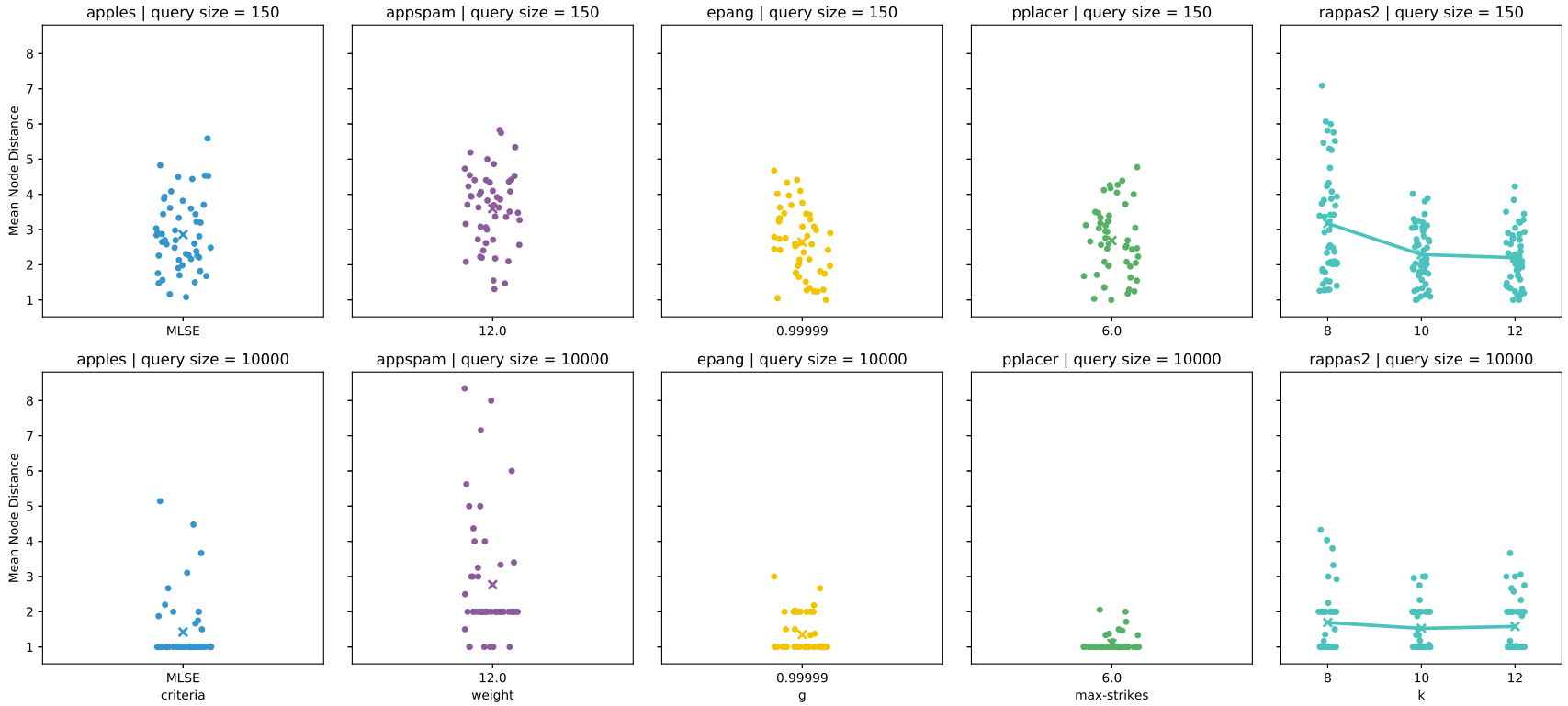


Figure B-4 – The results of PAC experiments on accuracy for D155.

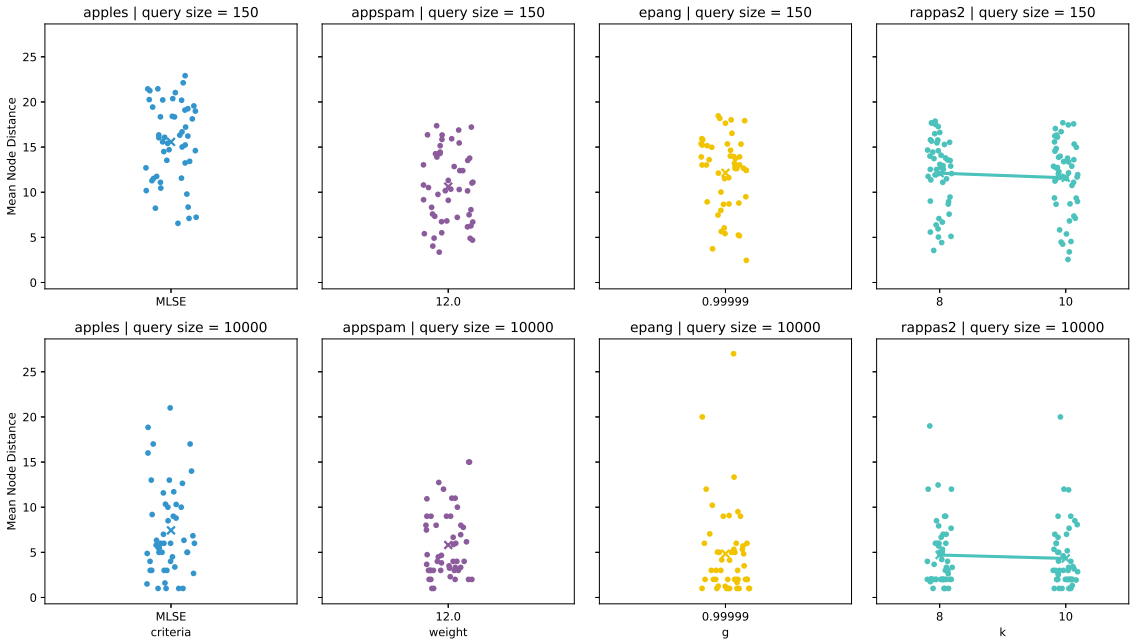


Figure B-5 – The results of PAC experiments on accuracy for HIV-genome.

Bibliography

[1] Mohamed Abdel-Basset, Weiping Ding, and Doaa El-Shahat. A hybrid Harris Hawks optimization algorithm with simulated annealing for feature selection. *Artificial Intelligence Review*, 54(1):593–637, 2021.

[2] Hervé Abdi and Lynne J. Williams. Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4):433–459, 2010.

[3] Mark D. Adams, Susan E. Celniker, Robert A. Holt, Cheryl A. Evans, Jeannine D. Gocayne, Peter G. Amanatides, Steven E. Scherer, Peter W. Li, Roger A. Hoskins, Richard F. Galle, et al. The genome sequence of *Drosophila melanogaster*. *Science*, 287(5461):2185–2195, 2000.

[4] Charu C. Aggarwal and ChengXiang Zhai. *A Survey of Text Classification Algorithms*, pages 163–222. Springer US, Boston, MA, 2012.

[5] ALPAC. *Language and Machines: Computers in Translation and Linguistics. A Report by the Automatic Language Processing Advisory Committee Division of Behavioral Sciences National Academy of Sciences National Research Council.* National Academies Press, Washington, D.C., January 1966.

[6] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, October 1990.

[7] Rudolf I. Amann, Wolfgang Ludwig, and Karl-Heinz Schleifer. Phylogenetic identification and *in situ* detection of individual microbial cells without cultivation. *Microbiological Reviews*, 59(1):143–169, 1995.

[8] Antonio Arauzo-Azofra, José Luis Aznarte, and José M Benítez. Empirical study of feature selection methods based on individual feature evaluation for classification problems. *Expert Systems with Applications*, 38(7):8170–8177, 2011.

[9] Oswald T. Avery, Colin M. MacLeod, and Maclyn McCarty. Studies on the chemical nature of the substance inducing transformation of pneumococcal types: Induction of transformation by a desoxyribonucleic acid fraction isolated from pneumococcus type III. *Journal of Experimental Medicine*, 79(2):137–158, 1944.

[10] Oluleye H. Babatunde, Leisa Armstrong, Jinsong Leng, and Dean Diepeveen. A genetic algorithm-based feature selection. *International Journal of Electronics Communication and Computer Engineering*, 5:889–905, 2014.

[11] Metin Balaban, Yueyu Jiang, Daniel Roush, Qiyun Zhu, and Siavash Mirarab. Fast and accurate distance-based phylogenetic placement using divide and conquer. *Molecular Ecology Resources*, 2021.

[12] Metin Balaban, Shahab Sarmashghi, and Siavash Mirarab. APPLES: scalable distance-based phylogenetic placement with or without alignments. *Systematic Biology*, 69(3):566–578, 2020.

[13] Madeleine Price Ball. Chemical structure of DNA, with colored label identifying the four bases as well as the phosphate and deoxyribose components of the backbone. Available at: `https://commons.wikimedia.org/wiki/File:DNA_chemical_structure.svg`, 2007. Accessed: 2021-10-08. The original image was published under multiple licenses. It is used here under the Creative Commons CC0 1.0 Universal license. See `https://creativecommons.org/publicdomain/zero/1.0/deed.en`.

[14] Pierre Barbera, Lucas Czech, Sarah Lutteropp, and Alexandros Stamatakis. SCRAPP: A tool to assess the diversity of microbial samples from phylogenetic placements. *Molecular Ecology Resources*, 21(1):340–349, 2021.

[15] Pierre Barbera, Alexey M. Kozlov, Lucas Czech, Benoit Morel, Diego Darriba, Tomás Flouri, and Alexandros Stamatakis. EPA-ng: Massively parallel evolutionary placement of genetic sequences. *Systematic Biology*, 68(2):365–369, Sep 2018.

[16] Paul H. Barrett. A transcription of Darwin's first notebook On Transmutation of Species. *Bulletin of the Museum of Comparative Zoology*, 122(6):246, 1960.

[17] Sam Behjati and Patrick S. Tarpey. What is next generation sequencing? *Archives of Disease in Childhood-Education and Practice*, 98(6):236–238, 2013.

[18] Simon Bennett. Solexa Ltd. *Pharmacogenomics*, 5(4):433–438, June 2004.

[19] David R. Bentley, Shankar Balasubramanian, Harold P. Swerdlow, Geoffrey P. Smith, John Milton, et al. Accurate whole human genome sequencing using reversible terminator chemistry. *Nature*, 456(7218):53–59, November 2008.

[20] Simon A. Berger and Alexandros Stamatakis. Aligning short reads to reference alignments and trees. *Bioinformatics*, 27(15):2068–2075, Jun 2011.

[21] Simon A. Berger and Alexandros Stamatakis. PaPaRa 2.0: a vectorized algorithm for probabilistic phylogeny-aware alignment extension. *Heidelberg Institute for Theoretical Studies*, 2012.

[22] Matthias Blanke and Burkhard Morgenstern. App-SpaM: phylogenetic placement of short reads without sequence alignment. *Bioinformatics Advances*, 1(1):vbab027, 2021.

[23] Avrim L. Blum and Pat Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97(1-2):245–271, 1997.

[24] Kristine Bohmann, Alice Evans, M. Thomas P. Gilbert, Gary R. Carvalho, Simon Creer, Michael Knapp, Douglas W. Yu, and Mark de Bruyn. Environmental DNA for wildlife biology and biodiversity monitoring. *Trends in Ecology & Evolution*, 29(6):358–367, 2014.

[25] Evan Bolyen, Jai Ram Rideout, Matthew R. Dillon, Nicholas A. Bokulich, Christian C. Abnet, et al. Reproducible, interactive, scalable and extensible microbiome data science using QIIME 2. *Nature Biotechnology*, 37(8):852–857, August 2019.

[26] Ingwer Borg and Patrick J.F. Groenen. *Modern Multidimensional Scaling: Theory and Applications*. Springer Science & Business Media, 2005.

[27] Annina Breen. This graph shows the main taxonomic ranks: domain, kingdom, phylum, class, order, family, genus, and species. this graph demonstrates how taxonomic ranking is used to designate related animals, the example used here is the red fox (*Vulpes vulpes*). Available at: `https://commons.wikimedia.org/wiki/File:Taxonomic_Rank_Graph.svg`, 2015. Accessed: 2021-10-09. This image is licensed under the Creative Commons Attribution-Share Alike 4.0 International. See `https://creativecommons.org/licenses/by-sa/4.0/deed.en`.

[28] Thomas D. Brock. The study of microorganisms *in situ*: progress and problems. In *Symp. Soc. Gen. Microbiol.*, number 41, pages 1–17, 1987.

[29] Daniel G. Brown and Jakub Truszkowski. LSHPlace: Fast phylogenetic placement using locality-sensitive hashing. In *Biocomputing 2013*, pages 310–319, Kohala Coast, Hawaii, USA, November 2012. World Scientific.

[30] Gavin Brown, Adam Pocock, Ming-Jie Zhao, and Mikel Luján. Conditional likelihood maximisation: a unifying framework for information theoretic feature selection. *The Journal of Machine Learning Research*, 13:27–66, 2012.

[31] Arthur J. Cain. The principal taxonomic ranks used in modern taxonomy. Available at: `https://www.britannica.com/science/taxonomy`, 2021. Encyclopedia Britannica. Accessed: 2021-10-09.

[32] J. Gregory Caporaso, Justin Kuczynski, Jesse Stombaugh, Kyle Bittinger, Frederic D. Bushman, Elizabeth K. Costello, et al. QIIME allows analysis of high-throughput community sequencing data. *Nature Methods*, 7(5):335–336, May 2010.

[33] Ellen C. Carbo, Igor A. Sidorov, Jessika C. Zevenhoven-Dobbe, Eric J. Snijder, Eric C. Claas, Jeroen F.J. Laros, Aloys C.M. Kroes, and Jutte J.C. de Vries. Coronavirus discovery by metagenomic sequencing: a tool for pandemic preparedness. *Journal of Clinical Virology*, 131:104594, October 2020.

[34] Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

[35] Chen-Shan Chin, David H. Alexander, Patrick Marks, Aaron A. Klammer, James Drake, Cheryl Heiner, Alicia Clum, Alex Copeland, John Huddleston, Evan E. Eichler, Stephen W. Turner, and Jonas Korlach. Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data. *Nature Methods*, 10(6):563–569, June 2013.

[36] Charles Y. Chiu and Steven A. Miller. Clinical metagenomics. *Nature Reviews Genetics*, 20(6):341–355, June 2019.

[37] Carlton Chu, Ai-Ling Hsu, Kun-Hsien Chou, Peter Bandettini, ChingPo Lin, Alzheimer's Disease Neuroimaging Initiative, et al. Does feature selection improve classification accuracy? Impact of sample size and feature selection on classification using anatomical magnetic resonance images. *Neuroimage*, 60(1):59–70, 2012.

[38] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[39] Thomas M. Cover. *Elements of Information Theory*. John Wiley & Sons, 1999.

[40] Michael A.A. Cox and Trevor F. Cox. Multidimensional scaling. In *Handbook of Data Visualization*, pages 315–347. Springer, 2008.

[41] Mark Craven, Andrew McCallum, Dan PiPasquo, Tom Mitchell, and Dayne Freitag. Learning to extract symbolic knowledge from the World Wide Web. Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh, 1998.

[42] Lucas Czech, Pierre Barbera, and Alexandros Stamatakis. Genesis and Gappa: processing, analyzing and visualizing phylogenetic (placement) data. *Bioinformatics*, 36(10):3263–3265, May 2020.

[43] Lucas Czech and Alexandros Stamatakis. Scalable methods for analyzing and visualizing phylogenetic placement of metagenomic samples. *PLoS One*, 14(5):e0217050, 2019.

[44] Aaron E. Darling, Guillaume Jospin, Eric Lowe, Frederick A. Matsen, Holly M. Bik, and Jonathan A. Eisen. PhyloSift: phylogenetic analysis of genomes and metagenomes. *PeerJ*, 2:e243, January 2014.

[45] Charles Darwin. Charles Darwin's 1837 sketch, his first diagram of an evolutionary tree from his first notebook on transmutation of species (1837). Available at: `https://en.wikipedia.org/wiki/File:Darwin_tree.png`, 1837. Accessed: 2021-10-09. This image was released into the public domain.

[46] Charles Darwin. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. John Murray, 1859.

[47] Sanmay Das. Filters, wrappers and a boosting-based hybrid for feature selection. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, page 74–81, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[48] Manoranjan Dash and Huan Liu. Feature selection for classification. *Intelligent Data Analysis*, 1(1-4):131–156, 1997.

[49] Richard Dawkins and Nicola Davis. *The Selfish Gene*. Macat Library, 2017.

[50] David Deamer, Mark Akeson, and Daniel Branton. Three decades of nanopore sequencing. *Nature Biotechnology*, 34(5):518–524, May 2016.

[51] Todd Z. DeSantis, Philip Hugenholtz, Keith Keller, Eoin L. Brodie, Neils Larsen, Yvette M. Piceno, R. Phan, and Gary L. Andersen. NAST: a multiple sequence alignment server for comparative analysis of 16S rRNA genes. *Nucleic Acids Research*, 34(Web Server):W394–W399, July 2006.

[52] Todd Z. DeSantis, Philip Hugenholtz, Neils Larsen, Mark Rojas, Eoin L. Brodie, Keith Keller, Thomas Huber, Daniel Dalevi, Ping Hu, and Gary L. Andersen. Greengenes, a chimera-checked 16S rRNA gene database and workbench compatible with ARB. *Applied and Environmental Microbiology*, 72(7):5069–5072, 2006.

[53] Mohamed S. Donia, Peter Cimermancic, Christopher J. Schulze, Laura C. Wieland Brown, John Martin, Makedonka Mitreva, Jon Clardy, Roger G. Linington, and Michael A. Fischbach. A systematic analysis of biosynthetic gene clusters in the human microbiome reveals a common family of antibiotics. *Cell*, 158(6):1402–1414, September 2014.

[54] Robert P. W. Duin and Marco Loog. Linear dimensionality reduction via a heteroscedastic extension of LDA: the Chernoff criterion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(6):732–739, 2004.

[55] Sean R. Eddy. Profile hidden Markov models. *Bioinformatics*, 14(9):755–763, October 1998.

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

[57] John Eid, Adrian Fehr, Jeremy Gray, Khai Luong, John Lyle, Geoff Otto, Paul Peluso, David Rank, Primo Baybayan, Brad Bettman, et al. Real-time DNA sequencing from single polymerase molecules. *Science*, 323(5910):133–138, January 2009.

[58] C. elegans Sequencing Consortium. Genome sequence of the nematode *C. elegans*: a platform for investigating biology. *Science*, 282(5396):2012–2018, 1998.

[59] Isaac Elias. Settling the intractability of multiple alignment. *Journal of Computational Biology*, 13(7):1323–1339, September 2006.

[60] Steven N. Evans and Frederick A. Matsen. The phylogenetic Kantorovich-Rubinstein metric for environmental sequence samples. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 74(3):569–592, June 2012.

[61] James S. Farris. Methods for computing Wagner trees. *Systematic Zoology*, 19(1):83, March 1970.

[62] Joseph Felsenstein. The number of evolutionary trees. *Systematic Zoology*, 27(1):27, March 1978.

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

[64] Joseph Felsenstein. *Inferring Phylogenies*. Sinauer Associates, Sunderland, Mass, 2004.

[65] Alan Filipski, Koichiro Tamura, Paul Billing-Ross, Oscar Murillo, and Sudhir Kumar. Phylogenetic placement of metagenomic reads using the minimum evolution principle. *BMC genomics*, 16(1):1–9, 2015.

[66] Robert D. Finn, Jody Clements, and Sean R. Eddy. HMMER web server: interactive sequence similarity searching. *Nucleic Acids Research*, 39(suppl):W29–W37, July 2011.

[67] Ronald A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.

[68] Walter M. Fitch. The molecular evolution of cytochrome c in eukaryotes. *Journal of Molecular Evolution*, 8(1):13–40, March 1976.

[69] Walter M. Fitch and Emanuel Margoliash. Construction of phylogenetic trees. *Science*, 155(3760):279–284, January 1967.

[70] Robert D. Fleischmann, Mark D. Adams, Owen White, Rebecca A. Clayton, Ewen F. Kirkness, Anthony R. Kerlavage, Carol J. Bult, Jean-Francois Tomb,

Brian A. Dougherty, Joseph M. Merrick, et al. Whole-genome random sequencing and assembly of *Haemophilus influenzae* Rd. *Science*, 269(5223):496–512, 1995.

[71] Forluvoft. Simple diagram of double-stranded DNA. Available at: `https://commons.wikimedia.org/wiki/File:DNA_simple2.svg`, 2008. Accessed: 2021-10-08. The image was released into the public domain.

[72] Anthony Foucher, Olivier Evrard, G. Francesco Ficetola, Ludovic Gielly, Julie Poulain, Charline Giguet-Covex, J. Patrick Laceby, Sébastien Salvador-Blanes, Olivier Cerdan, and Jérôme Poulenard. Persistence of environmental DNA in cultivated soils: implication of this memory effect for reconstructing the dynamics of land use and cover changes. *Scientific Reports*, 10(1):10502, June 2020.

[73] Limin Fu, Beifang Niu, Zhengwei Zhu, Sitao Wu, and Weizhong Li. CD-HIT: accelerated for clustering the next-generation sequencing data. *Bioinformatics*, 28(23):3150–3152, December 2012.

[74] Jean-loup Gailly and Mark Adler. Zlib compression library. Available at: `http://www.dspace.cam.ac.uk/handle/1810/3486`, 2004. Accessed: 2021-11-03.

[75] Mohammadreza Ghodsi, Bo Liu, and Mihai Pop. DNACLUST: accurate and efficient clustering of phylogenetic marker genes. *BMC Bioinformatics*, 12(1):271, December 2011.

[76] Thibaut Goetghebuer-Planchon. A C++ implementation of a fast hash map and hash set using robin hood hashing. Available at: `https://github.com/Tessil/robin-map`, 2017. Accessed: 2021-10-30. MIT License. See `https://github.com/Tessil/robin-map/blob/master/LICENSE`.

[77] André Goffeau, Bart G. Barrell, Howard Bussey, Ronald W. Davis, Bernard Dujon, Heinz Feldmann, Francis Galibert, Jörg D. Hoheisel, Claude Jacq, Michael Johnston, et al. Life with 6000 genes. *Science*, 274(5287):546–567, 1996.

[78] Julia K. Goodrich, Sara C. Di Rienzi, Angela C. Poole, Omry Koren, William A. Walters, J. Gregory Caporaso, Rob Knight, and Ruth E. Ley. Conducting a microbiome study. *Cell*, 158(2):250–262, July 2014.

[79] Fred Griffith. The significance of pneumococcal types. *Epidemiology & Infection*, 27(2):113–159, 1928.

[80] Stéphane Guindon, Jean-François Dufayard, Vincent Lefort, Maria Anisimova, Wim Hordijk, and Olivier Gascuel. New algorithms and methods to estimate maximum-likelihood phylogenies: Assessing the performance of PhyML 3.0. *Systematic Biology*, 59(3):307–321, 05 2010.

[81] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3(Mar):1157–1182, 2003.

[82] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1):389–422, 2002.

[83] Ernst Haeckel. *Generelle Morphologie der Organismen. Allgemeine Grundzüge der organischen Formen-Wissenschaft, mechanisch begründet durch die von C. Darwin reformirte Descendenz-Theorie*, volume 1. Berlin, G. Reimer, 1866.

[84] Ernst Haeckel. Generelle morphologie der organismen: allgemeine grundzüge der organischen formen-wissenschaft, mechanisch begründet durch die von c. darwin reformirte decendenz-theorie. Available at: `https://commons.wikimedia.org/wiki/File:Haeckel_arbol_bn.png`, 1866. Accessed: 2021-10-09. This image was released into the public domain.

[85] Torben Hagerup. Sorting and searching on the word RAM. In Michel Morvan, Christoph Meinel, and Daniel Krob, editors, *STACS 98*, pages 366–398, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[86] Mark A. Hall. *Correlation-based Feature Selection for Machine Learning*. PhD thesis, University of Waikato Hamilton, 1999.

[87] Alfred D. Hershey and Martha Chase. Independent functions of viral protein and nucleic acid in growth of bacteriophage. *Journal of General Physiology*, 36(1):39–56, 1952.

[88] John H. Holland et al. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial intelligence*. MIT Press, April 1992.

[89] Ying Huang, Beifang Niu, Ying Gao, Limin Fu, and Weizhong Li. CD-HIT Suite: a web server for clustering and comparing biological sequences. *Bioinformatics*, 26(5):680–682, March 2010.

[90] John P. Huelsenbeck. Bayesian inference of phylogeny and its impact on evolutionary biology. *Science*, 294(5550):2310–2314, December 2001.

[91] Laura A. Hug, Brett J. Baker, Karthik Anantharaman, Christopher T. Brown, Alexander J. Probst, et al. A new view of the tree of life. *Nature Microbiology*, 1(5), Apr 2016.

[92] Laura A. Hug, Brett J. Baker, Karthik Anantharaman, Christopher T. Brown, Alexander J. Probst, et al. A novel representation of the tree of life. 92 bacterial phyla, 26 archaael phyla, and all eukaryotic phyla are given in this tree of life built from 16 ribosomal proteins. Available at: `https://commons.wikimedia.org/wiki/File:A_Novel_Representation_Of_The_Tree_Of_Life.png`, 2016. Accessed: 2021-10-09. This image is derived from [91] and is licensed under the Creative Commons Attribution-Share Alike 4.0 International. See `https://creativecommons.org/licenses/by-sa/4.0/deed.en`.

[93] Philip Hugenholtz. Exploring prokaryotic diversity in the genomic era. *Genome Biology*, 3(2):1–8, 2002.

[94] Philip Hugenholtz, Brett M. Goebel, and Norman R. Pace. Impact of culture-independent studies on the emerging phylogenetic view of bacterial diversity. *Journal of Bacteriology*, 180(18):4765–4774, 1998.

[95] W. John Hutchins. The history of machine translation in a nutshell. Available at: `https://aclanthology.org/www.mt-archive.info/10/Hutchins-2014.pdf`. Accessed: 2022-14-03.

[96] W. John Hutchins. ALPAC: the (in) famous report. *Readings in Machine Translation*, 14:131–135, 2003.

[97] W. John Hutchins. The Georgetown-IBM experiment demonstrated in January 1954. In *Conference of the Association for Machine Translation in the Americas*, pages 102–114. Springer, 2004.

[98] Aapo Hyvärinen and Erkki Oja. Independent component analysis: algorithms and applications. *Neural Networks*, 13(4-5):411–430, 2000.

[99] Ekaterina A. Ivanova, Ilia O. Korvigo, Boris F. Aparin, Evgenii L. Chirak, Elizaveta V. Pershina, Nikolay S. Romaschenko, Nikolai A. Provorov, and Evgeny E. Andronov. The preservation of microbial DNA in archived soils of various genetic types. *PLOS One*, 12(3):e0173901, March 2017.

[100] Stefan Janssen, Daniel McDonald, Antonio Gonzalez, Jose A. Navas-Molina, Lingjing Jiang, Zhenjiang Zech Xu, Kevin Winker, Deborah M. Kado, Eric Orwoll, Mark Manary, Siavash Mirarab, and Rob Knight. Phylogenetic placement of exact amplicon sequences improves associations with clinical information. *mSystems*, 3(3), June 2018.

[101] Richard Jensen and Qiang Shen. Semantics-preserving dimensionality reduction: rough and fuzzy-rough-based approaches. *IEEE Transactions on Knowledge and Data Engineering*, 16(12):1457–1471, 2004.

[102] Thorsten Joachims. A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh, 1996.

[103] Mark Johnson, Irena Zaretskaya, Yan Raytselis, Yuri Merezhuk, Scott McGinnis, and Thomas L. Madden. NCBI BLAST: a better web interface. *Nucleic Acids Research*, 36(Web Server):W5–W9, May 2008.

[104] Alexandre Jousset, Christina Bienhold, Antonis Chatzinotas, Laure Gallien, Angélique Gobet, Viola Kurm, et al. Where less may be more: how the rare biosphere pulls ecosystems strings. *The ISME Journal*, 11(4):853–862, April 2017.

[105] Paschalia Kapli, Sarah Lutteropp, Jiajie Zhang, Kassian Kobert, Pavlos Pavlidis, Alexandros Stamatakis, and Tomas Flouri. Multi-rate Poisson tree processes for single-locus species delimitation under maximum likelihood and Markov chain Monte Carlo. *Bioinformatics*, 33(11):1630–1638, 01 2017.

[106] Kazutaka Katoh, Kei-ichi Kuma, Hiroyuki Toh, and Takashi Miyata. MAFFT version 5: improvement in accuracy of multiple sequence alignment. *Nucleic Acids Research*, 33(2):511–518, January 2005.

[107] Samina Khalid, Tehmina Khalil, and Shamila Nasreen. A survey of feature selection and feature extraction techniques in machine learning. In *2014 Science and Information Conference*, pages 372–378. IEEE, 2014.

[108] Kenneth K. Kidd and Laura A. Sgaramella-Zonta. Phylogenetic analysis: concepts and methods. *American Journal of Human Genetics*, 23(3):235, 1971.

[109] Scott Kirkpatrick, C. Daniel Gelatt, and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[110] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial intelligence*, 97(1-2):273–324, 1997.

[111] Daphne Koller and Mehran Sahami. Toward optimal feature selection. Technical report, Stanford InfoLab, 1996.

[112] Elizabeth Koning, Malachi Phillips, and Tandy Warnow. pplacerDC: a new scalable phylogenetic placement method. In *Proceedings of the 12th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*, pages 1–9, 2021.

[113] Sotiris B Kotsiantis, Ioannis Zaharakis, P Pintelas, et al. Supervised machine learning: A review of classification techniques. *Emerging Artificial Intelligence Applications in Computer Engineering*, 160(1):3–24, 2007.

[114] Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura Barnes, and Donald Brown. Text classification algorithms: A survey. *Information*, 10(4), 2019.

[115] Alexey M. Kozlov, Diego Darriba, Tomáš Flouri, Benoit Morel, and Alexandros Stamatakis. RAxML-NG: a fast, scalable and user-friendly tool for maximum likelihood phylogenetic inference. *Bioinformatics*, 35(21):4453–4455, 05 2019.

[116] Joseph B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

[117] Joseph B. Kruskal. Nonmetric multidimensional scaling: a numerical method. *Psychometrika*, 29(2):115–129, 1964.

[118] Joseph B. Kruskal and Myron Wish. *Multidimensional scaling*. Number 11 in SAGE's Quantitative Applications in the Social Sciences. Sage, 2009.

[119] Laura S. Kubatko. Inference of phylogenetic trees. In *Tutorials in Mathematical Biosciences IV*, pages 1–38. Springer, 2008.

[120] Thomas Navin Lal, Olivier Chapelle, Jason Weston, and André Elisseeff. Embedded methods. In *Feature Extraction*, pages 137–165. Springer, 2006.

[121] Eric S. Lander, Lauren M. Linton, Bruce Birren, Chad Nusbaum, Michael C. Zody, Jennifer Baldwin, Keri Devon, Ken Dewar, Michael Doyle, William FitzHugh, et al. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, 2001.

[122] Pat Langley. *Elements of Machine Learning*. Morgan Kaufmann, 1996.

[123] Pat Langley and Stephanie Sage. Induction of selective Bayesian classifiers. In *Uncertainty Proceedings 1994*, pages 399–406. Elsevier, 1994.

[124] Chris-André Leimeister, Salma Sohrabi-Jahromi, and Burkhard Morgenstern. Fast and accurate phylogeny reconstruction using filtered spaced-word matches. *Bioinformatics*, 33(7):971–979, January 2017.

[125] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P Trevino, Jiliang Tang, and Huan Liu. Feature selection: A data perspective. *ACM Computing Surveys (CSUR)*, 50(6):1–45, 2017.

[126] Weizhong Li and Adam Godzik. Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*, 22(13):1658–1659, July 2006.

[127] Weizhong Li, Lukasz Jaroszewski, and Adam Godzik. Clustering of highly homologous sequences to reduce the size of large protein databases. *Bioinformatics*, 17(3):282–283, March 2001.

[128] Shih-Wei Lin, Zne-Jung Lee, Shih-Chieh Chen, and Tsung-Yuan Tseng. Parameter determination of support vector machine and feature selection using simulated annealing approach. *Applied Soft Computing*, 8(4):1505–1512, 2008.

[129] Benjamin Linard, Nikolai Romashchenko, Fabio Pardi, and Eric Rivals. PEWO: a collection of workflows to benchmark phylogenetic placement. *Bioinformatics*, 36(21):5264–5266, 2021.

[130] Benjamin Linard, Krister Swenson, and Fabio Pardi. Rapid alignment-free phylogenetic identification of metagenomic sequences. *Bioinformatics*, 35(18):3303–3312, Jan 2019.

[131] Seppo Linnainmaa. Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, 16(2):146–160, 1976.

[132] Haoyue Liu, MengChu Zhou, and Qing Liu. An embedded feature selection method for imbalanced data classification. *IEEE/CAA Journal of Automatica Sinica*, 6(3):703–715, 2019.

[133] Huan Liu and Hiroshi Motoda. *Computational Methods of Feature Selection*. CRC Press, 2007.

[134] Mingxin Liu, Laurence J. Clarke, Susan C. Baker, Gregory J. Jordan, and Christopher P. Burridge. A practical guide to DNA metabarcoding for entomological ecologists. *Ecological Entomology*, 45(3):373–385, June 2020.

[135] Karen G. Lloyd. Great plate count translated. Available at: `https://github.com/klloydbeaufort/great-plate-count-translated`, 2016. Accessed: 2021-10-10. The repository does not indicate the license used. All rights are reserved.

[136] Harvey Lodish, Arnold Berk, Chris A. Kaiser, Chris Kaiser, Monty Krieger, Matthew P. Scott, Anthony Bretscher, Hidde Ploegh, Paul Matsudaira, et al. *Molecular Cell Biology*. Macmillan, 2008.

[137] Nicholas J. Loman, Chrystala Constantinidou, Martin Christner, Holger Rohde, Jacqueline Z.-M. Chan, Joshua Quick, Jacqueline C. Weir, Christopher Quince, Geoffrey P. Smith, Jason R. Betley, Martin Aepfelbacher, and Mark J. Pallen. A culture-independent sequence-based metagenomics approach to the investigation of an outbreak of Shiga-toxigenic *Escherichia coli* O104:H4. *JAMA*, 309(14):1502, April 2013.

[138] Shih Keng Loong, Chee Sieng Khor, Faizatul Lela Jafar, and Sazaly AbuBakar. Utility of 16S rDNA sequencing for identification of rare pathogenic bacteria. *Journal of Clinical Laboratory Analysis*, 30(6):1056–1060, November 2016.

[139] Catherine A. Lozupone, Micah Hamady, Scott T. Kelley, and Rob Knight. Quantitative and qualitative β diversity measures lead to different insights into factors that structure microbial communities. *Applied and Environmental Microbiology*, 73(5):1576–1585, March 2007.

[140] Frédéric Mahé, Colomban de Vargas, David Bass, Lucas Czech, Alexandros Stamatakis, Enrique Lara, David Singer, Jordan Mayor, John Bunge, Sarah Sernaker, et al. Parasites dominate hyperdiverse soil protist communities in neotropical rainforests. *Nature Ecology & Evolution*, 1(4):1–8, 2017.

[141] Frédéric Mahé, Torbjørn Rognes, Christopher Quince, Colomban de Vargas, and Micah Dunthorn. Swarm: robust and fast clustering method for amplicon-based studies. *PeerJ*, 2:e593, September 2014.

[142] Frédéric Mahé, Torbjørn Rognes, Christopher Quince, Colomban de Vargas, and Micah Dunthorn. Swarm v2: highly-scalable and high-resolution amplicon clustering. *PeerJ*, 3:e1420, December 2015.

[143] Marcel Margulies, Michael Egholm, William E. Altman, Said Attiya, Joel S. Bader, et al. Genome sequencing in microfabricated high-density picolitre reactors. *Nature*, 437(7057):376–380, September 2005.

[144] Thomas Marill and D. Green. On the effectiveness of receptors in recognition systems. *IEEE transactions on Information Theory*, 9(1):11–17, 1963.

[145] Frederick A. Matsen, Noah G. Hoffman, Aaron Gallagher, and Alexandros Stamatakis. A format for phylogenetic placements. *PLOS One*, 7(2):e31009, February 2012.

[146] Frederick A. Matsen, Robin B. Kodner, and E. Virginia Armbrust. pplacer: linear time maximum-likelihood and bayesian phylogenetic placement of sequences onto a fixed reference tree. *BMC bioinformatics*, 11(1):1–16, 2010.

[147] Allan M. Maxam and Walter Gilbert. A new method for sequencing DNA. *Proceedings of the National Academy of Sciences*, 74(2):560–564, 1977.

[148] Allan M. Maxam and Walter Gilbert. Sequencing end-labeled DNA with base-specific chemical cleavages. In *Methods in Enzymology*, volume 65, pages 499–560. Elsevier, 1980.

[149] Andrew McCallum, Kamal Nigam, et al. A comparison of event models for Naive Bayes text classification. In *AAAI-98 Workshop on Learning for Text Categorization*, volume 752, pages 41–48, 1998.

[150] W. Richard McCombie, John D. McPherson, and Elaine R. Mardis. Next-generation sequencing technologies. *Cold Spring Harbor Perspectives in Medicine*, 9(11):a036798, November 2019.

[151] Ronen Meiri and Jacob Zahavi. Using simulated annealing to optimize the feature selection problem in marketing applications. *European Journal of Operational Research*, 171(3):842–858, 2006.

[152] Joachim Messing, Roberto Crea, and Peter H. Seeburg. A system for shotgun DNA sequencing. *Nucleic Acids Research*, 9(2):309–321, 1981.

[153] Donald Michie, David J. Spiegelhalter, and Charles C. Taylor. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood series in artificial intelligence. Ellis Horwood, New York, 1994.

[154] Sebastian Mika, Gunnar Ratsch, Jason Weston, Bernhard Scholkopf, and Klaus-Robert Mullers. Fisher discriminant analysis with kernels. In *Neural Networks for Signal Processing IX: Proceedings of the 1999 IEEE Signal Processing Society Workshop*, pages 41–48. IEEE, 1999.

[155] Siavash Mirarab, Nam Nguyen, and Tandy Warnow. SEPP: Saté-enabled phylogenetic placement. In *Biocomputing 2012*, pages 247–258. World Scientific, 2012.

[156] Luis Carlos Molina, Lluís Belanche, and Àngela Nebot. Feature selection algorithms: A survey and experimental evaluation. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 306–313. IEEE, 2002.

[157] James Moor. The Dartmouth College artificial intelligence conference: The next fifty years. *AI Magazine*, 27(4):87–87, 2006.

[158] Benoit Morel, Alexey M. Kozlov, and Alexandros Stamatakis. ParGenes: a tool for massively parallel model selection and phylogenetic tree inference on thousands of genes. *Bioinformatics*, 35(10):1771–1773, October 2018.

[159] Bernard M.E. Moret, Usman Roshan, and Tandy Warnow. Sequence-length requirements for phylogenetic methods. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Roderic Guigó, and Dan Gusfield, editors, *Algorithms in Bioinformatics*, volume 2452, pages 343–356. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[160] National Human Genome Research Institute. Human Genome Project FAQ. Available at: `https://www.genome.gov/human-genome-project/Completion-FAQ`. Accessed: 2021-08-10.

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

[162] Stjepan Oreski and Goran Oreski. Genetic algorithm-based heuristic for feature selection in credit risk assessment. *Expert Systems with Applications*, 41(4):2052–2064, 2014.

[163] Karl Pearson. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11):559–572, 1901.

[164] Vikas Peddu, Ryan C. Shean, Hong Xie, Lasata Shrestha, Garrett A. Perchetti, Samuel S. Minot, Pavitra Roychoudhury, Meei-Li Huang, Arun Nalla, Shriya B. Reddy, Quynh Phung, Adam Reinhardt, Keith R. Jerome, and Alexander L. Greninger. Metagenomic analysis reveals clinical SARS-CoV-2 infection and bacterial or viral superinfection and colonization. *Clinical Chemistry*, 66(7):966–972, July 2020.

[165] Christian Quast, Elmar Pruesse, Pelin Yilmaz, Jan Gerken, Timmy Schweer, Pablo Yarza, Jörg Peplies, and Frank Oliver Glöckner. The SILVA ribosomal RNA gene database project: improved data processing and web-based tools. *Nucleic Acids Research*, 41(D1):D590–D596, 2012.

[166] Baranidharan Raman and Thomas R. Ioerger. Enhancing learning using feature and example selection. *Journal of Machine Learning Research (submitted for publication)*, 2003.

[167] А. С. Разумов. Прямой метод учета бактерий в воде. Сравнение его с методом Коха. *Микробиология*, 1(2):131–146, 1932. In Russian: Direct method of counting bacteria in water. Comparing it with Koch's method. The English translation can be found in [135].

[168] Irina Rish et al. An empirical study of the naive Bayes classifier. In *IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence*, volume 3, pages 41–46, 2001.

[169] Jarmo Ritari, Jarkko Salojärvi, Leo Lahti, and Willem M. de Vos. Improved taxonomic assignment of human intestinal 16S rRNA sequences by a dedicated reference database. *BMC Genomics*, 16(1):1056, December 2015.

[170] Torbjørn Rognes, Tomáš Flouri, Ben Nichols, Christopher Quince, and Frédéric Mahé. VSEARCH: a versatile open source tool for metagenomics. *PeerJ*, 4:e2584, October 2016.

[171] Nikolai Romashchenko. The principal taxonomic ranks used in modern taxonomy, 2021. This image is based on [27]. I simplified it by removing the illustration and taxonomic ranking of *Vulpes vulpes*. The image is licensed under the Creative Commons Attribution-Share Alike 4.0 International. See `https://creativecommons.org/licenses/by-sa/4.0/deed.en`.

[172] Nikolai Romashchenko. Sequencing cost per raw megabase of DNA, 2021. Licensed under the Creative Commons Attribution-Share Alike 4.0 International. See `https://creativecommons.org/licenses/by-sa/4.0/deed.en`.

[173] Fredrik Ronquist and John P. Huelsenbeck. MrBayes 3: Bayesian phylogenetic inference under mixed models. *Bioinformatics*, 19(12):1572–1574, August 2003.

[174] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386, 1958.

[175] D. B. Roszak and Rita R. Colwell. Survival strategies of bacteria in the natural environment. *Microbiological Reviews*, 51(3):365–379, September 1987.

[176] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[177] Krista M. Ruppert, Richard J. Kline, and Md Saydur Rahman. Past, present, and future perspectives of environmental DNA (eDNA) metabarcoding: A systematic review in methods, monitoring, and applications of global eDNA. *Global Ecology and Conservation*, 17:e00547, January 2019.

[178] Nicole Rusk. Torrents of sequence. *Nature Methods*, 8(1):44–44, January 2011.

[179] Yvan Saeys, Inaki Inza, and Pedro Larranaga. A review of feature selection techniques in bioinformatics. *Bioinformatics*, 23(19):2507–2517, 2007.

[180] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A bayesian approach to filtering junk e-mail. In *Learning for Text Categorization: Papers from the 1998 workshop*, volume 62, pages 98–105. Citeseer, 1998.

[181] Noelia Sánchez-Maroño, Amparo Alonso-Betanzos, and María Tombilla-Sanromán. Filter methods for feature selection — a comparative study. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 178–187. Springer, 2007.

[182] Frederick Sanger, Gilian M. Air, Bart G. Barrell, Nigel L. Brown, Alan R. Coulson, John C. Fiddes, C.A. Hutchison, Patrick M. Slocombe, and Mo Smith. Nucleotide sequence of bacteriophage φX174 DNA. *Nature*, 265(5596):687–695, 1977.

[183] Frederick Sanger, Steven Nicklen, and Alan R. Coulson. DNA sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences*, 74(12):5463–5467, 1977.

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

[185] Patrick D. Schloss. Reintroducing mothur: 10 years later. *Applied and Environmental Microbiology*, 86(2), January 2020.

[186] Patrick D. Schloss, Sarah L. Westcott, Thomas Ryabin, Justine R. Hall, Martin Hartmann, et al. Introducing mothur: Open-source, platform-independent, community-supported software for describing and comparing microbial communities. *Applied and Environmental Microbiology*, 75(23):7537–7541, December 2009.

[187] Jeremy Schmutz, Jeremy Wheeler, Jane Grimwood, Mark Dickson, Joan Yang, Chenier Caoile, Eva Bajorek, Stacey Black, Yee Man Chan, Mirian Denys, et al. Quality assessment of the human genome sequence. *Nature*, 429(6990):365–368, 2004.

[188] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998.

[189] Guillaume E. Scholz, Benjamin Linard, Nikolai Romashchenko, Eric Rivals, and Fabio Pardi. Rapid screening and detection of inter-type viral recombinants using phylo-k-mers. *Bioinformatics*, 36(22–23):3303–3312, Dec 2020. btaa1020.

[190] Stephan C. Schuster. Next-generation sequencing transforms today's biology. *Nature Methods*, 5(1):16–18, January 2008.

[191] Max E. Schön, Laura Eme, and Thijs J.G. Ettema. PhyloMagnet: fast and accurate screening of short-read meta-omics data using gene-centric phylogenetics. *Bioinformatics*, 36(6):1718–1724, 2020.

[192] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys (CSUR)*, 34(1):1–47, 2002.

[193] Bo Segerman. The most frequently used sequencing technologies and assembly methods in different time segments of the bacterial surveillance and RefSeq genome databases. *Frontiers in Cellular and Infection Microbiology*, 10:527102, October 2020.

[194] Rudy Setiono and Huan Liu. Neural-network feature selector. *IEEE Transactions on Neural Networks*, 8(3):654–662, 1997.

[195] Neethu Shah, Haixu Tang, Thomas G. Doak, and Yuzhen Ye. Comparing bacterial communities inferred from 16S rRNA gene sequencing and shotgun metagenomics. In *Biocomputing 2011*, pages 165–176. World Scientific, 2011.

[196] Jay Shendure, Gregory J. Porreca, Nikos B. Reppas, Xiaoxia Lin, John P McCutcheon, et al. Accurate multiplex polony sequencing of an evolved bacterial genome. *Science*, 309(5741):1728–1732, September 2005.

[197] Lloyd M. Smith, Jane Z. Sanders, Robert J. Kaiser, Peter Hughes, Chris Dodd, Charles R. Connell, Cheryl Heiner, Stephen B.H. Kent, and Leroy E. Hood. Fluorescence detection in automated DNA sequence analysis. *Nature*, 321(6071):674–679, 1986.

[198] Temple F. Smith and Michael S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, March 1981.

[199] Robert Reuven Sokal and Charles Duncan Michener. *A Statistical Method for Evaluating Systematic Relationships*. University of Kansas, 1958.

[200] Sujatha Srinivasan, Noah G. Hoffman, Martin T. Morgan, Frederick A. Matsen, Tina L. Fiedler, et al. Bacterial communities in women with bacterial vaginosis: high resolution phylogenetic analyses reveal relationships of microbiota to clinical criteria. *PLOS One*, 7(6):e37818, 2012.

[201] James T. Staley and Allan Konopka. Measurement of *in situ* activities of nonphotosynthetic microorganisms in aquatic and terrestrial habitats. *Annual Review of Microbiology*, 39(1):321–346, October 1985.

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

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

[204] Kathryn A. Stewart. Understanding the effects of biotic and abiotic factors on sources of aquatic environmental DNA. *Biodiversity and Conservation*, 28(5):983–1001, April 2019.

[205] Pierre Taberlet. *Environmental DNA: for Biodiversity Research and Monitoring*. Oxford University Press, Oxford, United Kingdom, first edition, 2018.

[206] Luis Talavera. An evaluation of filter and wrapper methods for feature selection in categorical clustering. In *International Symposium on Intelligent Data Analysis*, pages 440–451. Springer, 2005.

[207] Feng Tan, Xuezheng Fu, Yanqing Zhang, and Anu G. Bourgeois. A genetic algorithm-based method for feature subset selection. *Soft Computing*, 12(2):111–120, 2008.

[208] Philip Francis Thomsen and Eske Willerslev. Environmental DNA – An emerging tool in conservation for monitoring past and present biodiversity. *Biological Conservation*, 183:4–18, March 2015.

[209] S.L. Ting, W.H. Ip, Albert H.C. Tsang, et al. Is Naive Bayes a good classifier for document classification. *International Journal of Software Engineering and Its Applications*, 5(3):37–46, 2011.

[210] Ioannis Tsamardinos and Constantin F. Aliferis. Towards principled feature selection: Relevancy, filters and wrappers. In *International Workshop on Artificial Intelligence and Statistics*, pages 300–307. PMLR, 2003.

[211] Nathaniel Valiere and Pierre Taberlet. Urine collected in the field as a source of DNA for species and individual identification. *Molecular Ecology*, 9(12):2150–2152, December 2000.

[212] A. Valouev, J. Ichikawa, T. Tonthat, J. Stuart, S. Ranade, H. Peckham, K. Zeng, J. A. Malek, G. Costa, K. McKernan, A. Sidow, A. Fire, and S. M. Johnson. A high-resolution, nucleosome position map of *C. elegans* reveals a lack of universal sequence-dictated positioning. *Genome Research*, 18(7):1051–1063, July 2008.

[213] J. Craig Venter, Mark D. Adams, Eugene W. Myers, Peter W. Li, Richard J. Mural, Granger G. Sutton, Hamilton O. Smith, Mark Yandell, Cheryl A. Evans, Robert A. Holt, et al. The sequence of the human genome. *Science*, 291(5507):1304–1351, 2001.

[214] Jorge R. Vergara and Pablo A. Estévez. A review of feature selection methods based on mutual information. *Neural Computing and Applications*, 24(1):175–186, 2014.

[215] Carl von Linné. *Systema naturae; sive, Regna tria naturae: systematice proposita per classes, ordines, genera & species*. Haak, 1735. In Latin: The system of nature or the three kingdoms of nature, according to classes, orders, genera and species.

[216] Carl von Linné. Caroli linnaei systema naturae: a photographic facsimile of the first volume of the tenth edition (1758): Regnum animale. *British Museum (Natural History), London*, 823, 1758.

[217] Carl von Linné et al. Systema naturae per regna tria naturae, secundum classes, ordines, genera, species, cum characteribus, differentiis, synonymis, locis. *Laurentius Salvius, Stockholm*, 1758. 10th Edition. In Latin: System of nature through the three kingdoms of nature, according to classes, orders, genera and species, with characters, differences, synonyms, places.

[218] Lipo Wang, Yaoli Wang, and Qing Chang. Feature selection methods for big data bioinformatics: A survey from the search perspective. *Methods*, 111:21–31, 2016.

[219] Lusheng Wang and Tao Jiang. On the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1(4):337–348, January 1994.

[220] James D. Watson and Francis H.C. Crick. Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid. *Nature*, 171(4356):737–738, 1953.

[221] James D. Watson and Francis H.C. Crick. The structure of DNA. In *Cold Spring Harbor symposia on quantitative biology*, volume 18, pages 123–131. Cold Spring Harbor Laboratory Press, 1953.

[222] Eleanor Wedell, Yirong Cai, and Tandy Warnow. Scalable and accurate phylogenetic placement using pplacer-XR. In *International Conference on Algorithms for Computational Biology*, pages 94–105. Springer, 2021.

[223] Sarah L. Westcott and Patrick D. Schloss. De novo clustering methods outperform reference-based methods for assigning 16S rRNA gene sequences to operational taxonomic units. *PeerJ*, 3:e1487, December 2015.

[224] K.A. Wetterstrand. DNA sequencing costs: Data from the NHGRI Genome Sequencing Program (GSP). Available at: `https://www.genome.gov/sequencingcostsdata`, 2020. Accessed: 2022-14-03.

[225] A. Wayne Whitney. A direct method of nonparametric measurement selection. *IEEE Transactions on Computers*, 100(9):1100–1103, 1971.

[226] Carl R. Woese and George E. Fox. Phylogenetic structure of the prokaryotic domain: The primary kingdoms. *Proceedings of the National Academy of Sciences*, 74(11):5088–5090, November 1977.

[227] Carl R. Woese, Otto Kandler, and Mark L. Wheelis. Towards a natural system of organisms: proposal for the domains Archaea, Bacteria, and Eucarya. *Proceedings of the National Academy of Sciences*, 87(12):4576–4579, 1990.

[228] Carl R. Woese, Linda J. Magrum, and George E. Fox. Archaebacteria. *Journal of Molecular Evolution*, 11(3):245–252, 1978.

[229] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, 2(1-3):37–52, 1987.

[230] Patrick C.Y. Woo, Susanna K.P. Lau, Jade L.L. Teng, Herman Tse, and Kwok Y. Yuen. Then and now: use of 16S rDNA gene sequencing for bacterial identification and discovery of novel bacteria in clinical microbiology laboratories. *Clinical Microbiology and Infection*, 14(10):908–934, October 2008.

[231] Wenzhu Yang, Daoliang Li, and Liang Zhu. An improved genetic algorithm for optimal feature subset selection from multi-character feature set. *Expert Systems with Applications*, 38(3):2733–2740, 2011.

[232] Z. Yang. PAML 4: Phylogenetic analysis by maximum likelihood. *Molecular Biology and Evolution*, 24(8):1586–1591, April 2007.

[233] Jieping Ye, Ravi Janardan, and Qi Li. Two-dimensional linear discriminant analysis. *Advances in Neural Information Processing Systems*, 17:1569–1576, 2004.

[234] Seok-Hwan Yoon, Sung-Min Ha, Soonjae Kwon, Jeongmin Lim, Yeseul Kim, Hyungseok Seo, and Jongsik Chun. Introducing EzBioCloud: a taxonomically united database of 16S rRNA gene sequences and whole-genome assemblies. *International Journal of Systematic and Evolutionary Microbiology*, 67(5):1613, 2017.

[235] Harry Zhang. The optimality of naive Bayes. In *Proceedings of the 17th International Florida Artificial Intelligence Research Society Conference*, pages 562–567. AAAI Press, 2004.

[236] Alice Zheng and Amanda Casari. *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*. O'Reilly Media, Inc., 2018.

[237] Qi Zheng, Casey Bartow-McKenney, Jacquelyn S. Meisel, and Elizabeth A. Grice. HmmUFOtu: An HMM and phylogenetic placement based ultra-fast taxonomic assignment and OTU picking tool for microbiome amplicon sequencing studies. *Genome Biology*, 19(1):82, December 2018.

[238] Emile Zuckerkandl and Linus Pauling. Molecules as documents of evolutionary history. *Journal of Theoretical Biology*, 8(2):357–366, March 1965.