Whole Genome Shotgun Sequencing Based Taxonomic Profiling Methods for Comparative Study of Microbial Communities

Dissertation zur Erlangung des Grades eines Doktors der Naturwissenschaften (Dr. rer. nat.) vorgelegt von

Temesgen Hailemariam Dadi

am Fachbereich Mathematik und Informatik
der Freien Universität Berlin

Berlin 2019

Gutachter:
Prof. Dr. Knut Reinert, Freie Universität Berlin, Deutschland (1. Gutachter)
Prof. Dr. Daniel Huson, Tübingen university, Deutschland

Datum der Disputation: 29.04.2019

In memory of my Father!

Contents

Preface

In this PhD thesis, I will describe two main computational methods relevant to *whole genome shotgun sequencing* (WGS) based taxonomic profiling tools. These methods were developed with the goal of advancing metagenomics as a means to study microbial communities. This thesis document is organized into four parts.

Part I, introduces the reader to different aspects of microbial communities (Chapter 1), the science of metagenomics (Chapter 2) and, read mapping in metagenomics (Chapter 3). The first chapter gives a broader overview of microbial communities and their importance with the intention of justifying the efforts made to study them. In the second chapter, I give an introduction to the science of metagenomics starting with the basics of genomics. In the third introduction chapter, I provide a review of read mapping with the context of metagenomics laying the ground work for the part III of this thesis.

In the second part, I focus on the taxonomic profile part of my work. I start by introducing the importance and challenges in taxonomic profiling. Then, I describe different approaches taken to tackle the challenges followed by a brief description of 13 different taxonomic profiling tools. In a following chapter, I give a detail description of SLIMM, a taxonomic profiling method I have developed. The last chapter of this part presents the evaluation of SLIMM compared to existing method. Most of the work described in taxonomic profiling part of this thesis is based on the following published work.

Temesgen Hailemariam Dadi, Bernhard Y. Renard, Lothar H. Wieler, Torsten Semmler, and Knut Reinert, SLIMM: species level identification of microorganisms from metagenomes *PeerJ*, 2017. doi:10.7717/peerj.3138

The third part of the thesis is about distributed read mapping for large databases which is composed of two chapters (Chapter 7 and Chapter 8). In Chapter 7, I introduce the *Dynamic seaRchablE pArallel coMpressed index* (DREAM index) framework which is designed to enable frequent updates of indices in read mapping. Then, I go on to describe DREAM-Yara, a distributed read mapper based on the *Dynamic seaRchablE pArallel coMpressed index* (DREAM index) framework. I will also describe the *interleaved*

Bloom filters (IBF), a novel data structure I developed in the process of materializing DREAM-Yara. Chapter 8 concludes this part of the thesis by providing an evaluation of DREAM-Yara against existing methods highlighting the significant improvements in index creation and update time while staying competitive in read mapping quality and speed. Results discussed in this chapters were presented at the 17th European Conference on Computational Biology (ECCB18) and subsequently published in:

Temesgen Hailemariam Dadi, Enrico Siragusa, Vitor C. Piro, Andreas Andrusch, Enrico Seiler, Bernhard Y. Renard, and Knut Reinert, DREAM-Yara: an exact read mapper for very large databases with short update time *Bioinformatics*, 2018.
doi:10.1093/bioinformatics/bty567

In the last part of the thesis, I conclude by giving a summarized discussion of what I did in my PhD work accompanied by general remarks and outlook related to the thesis.

In accordance with the standard scientific protocol, throughout this thesis I will use the personal pronoun *we* to indicate the reader and the writer, or my scientific collaborators and myself.

Additional Publications

During my PhD period, I have collaborated with others in publishing the following articles which are not covered in this thesis. My contributions involved data analysis work using own methods and maintaining the SeqAn software library.

Łukasz Grześkowiak, Beatriz Martínez-Vallespín, Femke-Anouska Heinsen, Temesgen Hailemariam Dadi, Judith Radloff, Salah Amasheh, Andre Franke, Knut Reinert, Wilfried Vahjen, Jürgen Zentek and Robert Pieper Formula Feeding Predisposes Neonatal Piglets to Clostridium difficile Gut Infection *The Journal of Infectious Diseases*, 2017.
doi:10.1093/infdis/jix567

Knut Reinert, Temesgen Hailemariam Dadi, Marcel Ehrhardt, Hannes Hauswedell, Svenja Mehringer, René Rahn, Jongkyu Kim, Christopher Pockrandt, Jörg Winkler, Enrico Siragusa, Gianvito Urgese and David Weese The SeqAn C++ template library for efficient sequence analysis: A resource for programmers.
Journal of Biotechnology, 2017.
doi:10.1016/j.jbiotec.2017.07.017

Vitor C. Piro, Temesgen Hailemariam Dadi, Enrico Seiler, Knut Reinert, and Bernhard Y. Renard, ganon: continuously up-to-date with database growth for precise short read classification in metagenomics
bioRxiv, 2018.
doi:10.1101/406017

Acknowledgments

Writing this thesis would not have been possible without the support of many people. First of all, I would like to thank my supervisor Knut Reinert for his continuous support, guidance and optimism throughout my PhD work. It has been an honor for me to work under his supervision. I also express my deepest gratitude to Bernhard Renard, Lothar H. Wieler, and Torsten Semmler for the valuable discussions and collaborations over the years. I also would like to thank my colleagues at the Freie Universität Berlin, for the amazing work environment maintained throughout my stay. Thank you Leon, not only for being the best office mate, but also for the tips and explanations which are too many to count. I highly appreciate your encouragement in the final thesis writing days.

I would like to thank the IMPRS organizers Kirsten and Fabian who supported me with administrative issues. I am specially indebted to Kirsten Kelleher, who sometimes went beyond her responsibilities to help me. I would also like to thank my fellow IMPRS students with whom I shared many good memories and after-work-beers.

My special thanks goes to my wife, Rahel Tesfaye, for proofreading the thesis and for being supportive, encouraging and understanding at times of frustration. I am grateful for my little daughter, Eden, for the refreshing joy she brought to my life. Thank you Zizu, Fiste, Konjit and Ruthi for all the good times we had together and for being there when needed.

Part I.

Introduction

1. Microbial Communities

Microbes, also known as microorganisms, are tiny organisms that can not be seen with a naked human eye. In other words, we need a microscope to see them. Both terms, i.e., microbe and microorganism, are interchangeably used to describe any living thing which is less than a tenth of a millimeter in diameter [1]. A microbe can be either a single cell organism or a cluster of cells as long as, this cluster is visible only with a microscope. The science that studies microbes is known as microbiology.

1.1. Types of Microbes

Microbes span all the three domains of life, namely Bacteria, Archaea and Eucarya [1, 2], as well as the philosophically debated "life forms," i.e., viruses. The issue of considering viruses living or non-living is beyond the scope of this thesis. However, since the focus of this study is (meta)genomics and viruses invariably contain some form of genetic information, we consider them as microbes throughout this document. It is a common observation that molecular studies of microbes often omit those under the domain Eucarya. However, this is usually due to the heterogeneity of organisms under this domain both in size and type. In other words, organisms in this domain are not solely microorganisms. In the following, we describe the different group of microbes.

Definition 1.1. Eukaryotes are organisms composed of cells with membrane-bound organelles most notably a membrane-bound nucleus and mitochondria. Eukaryotes make up one of the three domains of life.

Definition 1.2. Procaryotes opposed to eukaryotes, do not have membrane-bound organelles. Prokaryotes cover two domains of life namely bacteria and archaea. The majority of known microbes fall under this category.

Viruses are incredibly tiny particles composed of a protein coat surrounding other proteins and a genetic material in the form of either *ribonucleic acid* (RNA) or *deoxyribonucleic acid* (DNA). Viruses do not possess the necessary cell machinery neither to reproduce nor to grow. They can only reproduce by infecting a host cell.

Bacteria are prokaryotic unicellular microorganisms with a cell wall. Bacteria reproduce using a process called binary fission, a form of reproduction where the DNA material of a unicellular organism replicates and forms two virtually identical clusters. Then, the organism's cytoplasm splits into two around these clusters of DNA material which later become the nucleoid of each of the new progeny.

Archaea are also prokaryotic unicellular microorganisms. They differ from Bacteria, their prokaryote brothers, by their cell walls. Archaea do not have peptidoglycan cell walls [1]. These microorganisms constitute the domain Archaea also known as Archaebacteria. They are known to thrive in extreme environments. To mention few, some archaea species live in habitats with extremely high temperature and little or no oxygen [1, 3].

Fungi are eukaryotic organisms that exist in wide variety of shapes and sizes. Fungi such as yeast are microscopic and composed of a single cell. Mushrooms are also a type of fungi which are multicellular and are large enough to be visible. Fungi reproduction could be both asexual and sexual.

Algae are mostly aquatic eukaryotic organisms that are photosynthetic. Like fungi, algae can be unicellular and multicellular. They reproduce both sexually and asexually.

Protozoa are a diverse group of eukaryotic unicellular organisms with animal-like properties such as mobility and predation. They reproduce mostly through binary fission. Multiple fissions also are common reproduction mechanisms among protozoa.

Parasites are organisms that live inside or on another organism utilizing their host's resources. Parasites could be bacteria, viruses, protozoa and many small animals such as worms and insects. Unlike the above five groups, parasites by no means represent a taxonomic classification. Nevertheless, it is a common categorization of microbes worth mentioning.

Microbes exist as interconnected complex communities comprised of a wide variety of species. It is virtually impossible to find a single species (homogeneous) microbial community [4]. Members of communities exhibit different forms of interaction with each other. The interactions could be beneficial, indifferent, or detrimental to one or both of the engaged parties [5]. In other words, community members, whether they are clonal or not, exhibit all forms of symbiotic relations, including commensalism, mutualism, neutralism, amensalism, competition, parasitism, and predation. Some members of a community provide their neighbors with essential nutrients ranging from electron donors to amino acids and vitamins. Others help by detoxifying the habitat. The diversity in prevalence phenotype and genotype of community members makes it complicated to study and define and characterize their interaction. However, the same interactions are vital in characterizing microbial communities [6].

The complexity of microbial communities ranges from low complexity communities

composed of a handful of different species to high complexity communities containing thousands of different microbial species. Low complexity communities inhabit environments with extreme conditions such as extremely high temperature (up to 75 degree Celsius), high pressure and low pH. These findings are not surprising, since such habitats are deemed unlivable for most life forms [3]. On the other hand, profoundly complex microbial communities are common in environments such as soil and water.

1.2. Examples of Microbial Communities

Microbial communities of extreme environments are well studied for various scientific and economic interest. Among those are microbial communities found in *acid mine drainage* (AMD), acid runoff originating from the surface of mining sites [7]. These sites are known for their low pH and high concentrations of metals and sulfate [8, 7, 9, 10]. Due to their relatively low species diversity, Baker and Banfield suggest using AMD microbial communities as model systems for studying microbial community structures, functions, biogeochemical interactions, and feedback. Other known microbial communities of extreme environments include those living under high pressure in the deepest seabed (e.g.Mariana trench; 10,994 meters deep) and in hot vents of the deep ocean where the water temperature can reach well above 400 °C.

Microbial communities in soil habitats are astonishingly diverse [11], containing from thousands [11] to tens of thousands [12] different species. It should be clear that there is no representative soil microbial community across the globe. There is a considerable difference between compositions of two soil microbiomes even if they are 1cm apart [13, 6]. Bacteria and fungi are the most common types of microorganisms found in soil. Owing to their resistance to culturing, the majority of the microbial species living in the soil environments are either unknown or understudied. Soil microbial communities are an integral part of the soil ecosystem and play a unique role in making vegetation possible. For this reason, microbial species richness in soil is often used as an indicator of soil health.

Like most of the places on the planet, freshwaters such as lakes, springs, and rivers are not void of microorganisms. Microbes of freshwater lakes recently got the attention of the scientific community. Similar to soil habitats, lakes are sanctuaries for microbes of remarkable diversity. Analysis of community structure of Lake Lanier by Oh et al. [14] revealed that Bacteria and Archaea cover the majority of microbes (90%). The dominant bacterial groups found in this freshwater include phyla *Proteobacteria*, *Actinobacteria* and *Verrucomicrobia* in decreasing order of relative abundance [14, 15]. Bacteria play a significant role in freshwater environments and therefore are important to the living things living there. The quality of water in lakes and other freshwater sources is significantly tied to the composition of microbial communities in it [16, 17].

Other microbial communities that came to the spotlight are those living in the open waters around the globe. Researchers have studied microbial communities from different parts of the world's oceans for their composition and interaction with the ecosystem. There are a wealth of microorganisms living in ocean planktons, in different depths of the oceans as well as Sea sediments. The Tara ocean expedition, a three-year expedition around the world oceans, aimed at exploring the global ocean biodiversity and gave a particular focus for microbial communities among others. The follow-up studies based on the data collected from this expedition brought compelling insight about microbial communities of the open waters. A somewhat smaller scale studies are also done on microbial communities of many different regional seas such as the Sargasso Sea [18, 19, 20, 21].

Host-associated microbial communities take shelter inside or on their host organism occupying different body parts [22] of their host. Except for pathogens causing illness or even death, host-associated microbes usually exist in a positive or neutral symbiosis with their respective host. Much research has been carried out to define and characterize host-associated microbes, categorized by different body sites of the host. The majority of these studies showed that compositions of microbial communities differ clearly among the various body sites [23, 24].

Among the host-associated microbes, human-associated microbes are the earliest to be recognized. For obvious reasons, they are also the most studied. Until recent times, it was a common understanding that the microbial cells that live in and on a human outnumber human somatic cells ten to one [25, 22, 26, 27, 28]. However, latest studies show that the ten to one ratio is possibly an exaggeration [26, 29] and the actual rate is much closer to one to one, with microbes still outnumbering the human somatic cells. Human-associated microbial communities harbor different body sites such as skin, oral cavity, vagina, and gut. Interestingly, microbial communities of various body sites of the same host are distinct in their species composition. Another interesting observation was the fact that microbial communities always show some degree of difference from one person to the other.

1.3. Importance of Microbial Communities

Microorganisms and the communities they form are essential for all life forms. Every naturally occurring process in the biosphere is affected by microbes in one way or another. Microbes are the leading players in availing the main elements of life such as carbon and nitrogen into forms that are usable by living things. Nitrogen fixation is an excellent example of such activities performed by microbial communities. It is a process of converting atmospheric nitrogen (N_2) into ammonia or another nitrogen-based compound that can be directly consumed by plants and other life forms. Nitrogen-fixing bacteria like *Azotobacter* do most of the nitrogen fixation in the soil. Another

example is photosynthesis, where light energy is used to break down carbon dioxide into oxygen and sugar. Surprisingly, microorganisms of the globe have a higher capacity for photosynthesis compared to plants. Host-associated Microbial communities are integral parts of the hosts' life. They affect the health status as well as the day today biological functioning of their host in countless ways. In this section, we will describe some of the primary importance of microbial communities to us humans and other life forms. However, the description below is by no means a comprehensive list of the benefits of microbial communities.

1.3.1. Mining

Industrial scale mining with the help of microorganisms is a common practice, especially where the mining source involves low-grade metal ores. To make mining from such ores economically feasible, the end product metal has to be concentrated by a process called microbial leaching. In this process, acidophilic microbes such as *Acidithiobacillus ferrooxidans* facilitate acid production and dissolution of the low-grade ore so that it is washed to a pond where it gets further processed. Mining of copper from covellite (CuS) and gold from minerals containing arsenic (As) and pyrite (FeS_2) are good examples of mining that often involve microbial leaching.

1.3.2. Environmental Bioremediation

Environmental Bioremediation is the process of cleaning pollutants from the environment using microbes. Bacteria can help to deal with contaminants such as oil, toxic chemicals, uranium. Bioremediation involves introducing a new community of microorganisms or creating a favorable condition for the indigenous microbial communities in some fashion. The most common and successful usage of environmental remediation is crude oil spills and hydrocarbon leakages into both aquatic and land environments. Different bacterial, fungi and few algae species can break down petroleum products. To better facilitate the cleaning process inorganic nutrients such as nitrogen and phosphorus are added to encourage microbial growth. Some bacterial species such as *Alcanivorax borkumensis* are known to grow only in the presence of oil. Alcanivorax borkumensis secrets chemicals that make oil degradable and soluble. Microbes also have the potential to detoxify different chemicals such as pesticides introduced to the soil. Additionally, microbes play an important role in wastewater treatment.

1.3.3. Agriculture and Food

Microorganisms play a significant role in agriculture. Microbes living inside the roots of leguminous plants such as soybeans, beans, and peas are responsible for fixing atmospheric nitrogen into a form that these plants can uses. Due to this nitrogen fixation, leguminous plants do not require fertilizers to grow, saving the agricultural industry significant costs and minimizing the pollution impact of fertilizer runoff.

Microbes also have a great significance to the other side of agriculture, i.e., breading of animals. The majority of animals lack the enzymes necessary to digest cellulose and some other plant polysaccharides. It is only with the help of their gut microbial community that animals with plant-based diets can get the energy required to sustain life. Most of the domesticated animals including cows, sheep, and goats are ruminants. Ruminants are herbivorous animals with a dedicated organ called rumen. This organ is used to digest cellulose and other plant polysaccharides with the help of microbial communities living in it. For this reason, microorganisms are invaluable to the animal breeding industry.

The food industry is also in dept to microbes. Fermentation, anaerobic metabolism of sugar to produce alcohol or organic acids or gases, is the primary mechanism in which microorganisms are used to manufacture food products. The application of fermentation ranges from the leavening of bread, brewing, vinification, production of some liqueurs, vinegar yogurt and different cheese products.

1.3.4. Healthy Human Life

As it is pointed out in the previous subsection, animals need microbes to break down specific nutrients that they do not have the enzymes for and we humans are no exception. Many essential enzymes are known to be produced by different human gut microbial community members. Research also shows that microbes of the gut produce some of the essential amino acids, that our body can not synthesize adequately by itself. Gut microbes also play a role in the development of fully-fledged adult gastrointestinal tract that can perform efficient nutrients uptake. They also train our immune system enabling it to differentiate the commensal gut flora from intruding and potential pathogenic microorganism. They also serve as the first line of defense against colonization by a foreign entity.

1.3.5. Biotechnology and Medicine

Microbes can efficiently synthesize a variety of chemicals that have a broad impact on human life. An outstanding example in the context of this work is the production of *Taq*

polymerase, a product of the thermophilic hot spring bacterium *Thermus aquaticus*. This polymerase is popular in *polymerase chain reaction* (PCR) procedures for its stability in high temperatures up to 95°C. This property makes it immune to the denaturation step of the PCR. Microorganisms also play a significant role in genetic engineering which creates better breeds (high yield) of domestic animals, disease resistant and high yield plants..

Microbes are also used in the production of different medical products spanning antibiotics, insulin and different types of vaccines. Penicillin, the first ever antibiotics discovered by Alexander Fleming In 1929, was a product of a fungus called *Penicillium chrysogenum*. Today a wide variety of antibiotics are produced using different bacterial species such as multiple species under genus Streptomyces which are known to produce antibiotics like tetracyclines. Microbes also help manufacture various types of human hormones such as insulin and somatotropin (growth hormone).

2. Metagenomics

Metagenomics is the study of metagenomes, genetic material recovered directly from environmental samples. Metagenomics focuses on capturing the full DNA content of a given microbial community. By doing so, it facilitates understanding microbial community structures and functions much deeper than it is possible via traditional culture-based methods. Metagenomics is closely related to genomics. The term environmental genomics is used synonymously in place of metagenomics. That is why in this chapter we will explain what metagenomics as a field of science is and what it encompasses, beginning from basic genomics methods and technologies that are foundations to it.

2.1. Genomics Preliminaries

DNA, short for *deoxyribonucleic acid*, is a long macromolecule made from chained units called nucleotides. Nucleotides, in turn, are composed of three major subunits namely a nitrogenous base, a pentose sugar and a phosphate group. A nucleotide from a DNA can have one of the four different types of nitrogenous bases, i.e., adenine (A), guanine (G), cytosine (C) and thymine (T). The phosphate part of the nucleotide bonds with the sugar part of the next nucleotide to make an alternating sugar-phosphate backbone of the DNA chain. The sequence of nucleotides in such a chain provides the primary structure of a DNA molecule, and the corresponding sequence of nitrogenous bases accounts for the potential of DNA to hold genetic information. In cells, DNA exists as double-stranded molecule in which the two strands are glued to each other by a hydrogen bond formed between complementary nitrogen bases. Adenine makes two hydrogen bonds exclusively with thymine, and similarly, guanine makes three hydrogen bonds exclusively with cytosine.

Furthermore, the double-stranded DNA could exist in either linear or circular form. In both cases, DNA is packed together by a phenomenon called *supercoiling*. The number of nucleotide pairs (*base pairs*) in a double-stranded DNA can be used to describe its size. Due to a large number of base pairs found in a single DNA molecule, *kilobase pairs* (kbp) or even *megabase pairs* (mbp) are used as units to represent a thousand or a million base pairs respectively.

RNA, short for *ribonucleic acid*, similar to DNA is a long macromolecule made from chained units called nucleotides. There are two differences between DNA and RNA molecules. The first difference is in the sugar part of the nucleotides. DNA has a deoxyribose sugar whereas RNA has a ribose sugar which is reflected in the naming of the nucleic acids. The second difference is, in RNA, we have a uracil (U) nitrogen base instead of thymine(T).

A genome is the total complement of hereditary or genetic information that is carried by the organism as well as its cells. This genetic information is often stored in large DNA molecules except for RNA viruses. Such viruses store and pass their genetic information the form of RNA molecules. A genome is organized into physically standalone long DNA molecules called chromosomes. In addition to chromosomal DNA, a genome of an organism could include plasmid DNA and organellar DNA. Organellar DNA is limited to eukaryotes and includes DNA from mitochondria and chloroplasts. A genome of an organism provides the means to pass genetic information from an organism to its progeny and, the genetic information in a genome is stored as a sequence of nucleotides within one or more molecules of DNA or RNA (in case of some viruses).

Even though the genome includes other genetic elements in addition to chromosomal DNA, most of it is organized into one or more chromosomes. While Eukaryotes have multiple chromosomes, which are mostly linear, in their genomes, most of the prokaryotes have a single long circular chromosome with few exceptions bearing two or three chromosomes. Prokaryotic genomes pack millions of nucleotides within a chromosome. For example, the K-12 strain of E. coli, a model microorganism used for genetics has a 4.63 mbp long chromosome, and this is a typical genome size for other prokaryotes as well. Nevertheless, genomes of prokaryotes show a considerable variation in length ranging from 0.5 mbp to 13 mbp.

Genomics is a branch of molecular biology that studies the structure, function, evolution, and mapping or comparison of genomes. DNA sequencing, the process of calling the nitrogen bases of a DNA molecule in sequential order, is at the core of genomics. Genomics also involves bioinformatics algorithms and methods to make sense of the raw sequencing data which is usually fragmented and chaotic. The bioinformatics part of genomics includes assembling an entire genome of an organism, analyze its structure and function or comparing it to genomes of other organisms. The recent advances in genomics played a revolutionary role in understanding system biology and lead to many scientific discoveries. The rapid development of sequencing technologies, which are getting more accurate, faster and cheaper coupled with the development of bioinformatics methods enabled significant advances in genomics. In the next subsection, we mention some of the popular sequencing technologies and methods which are relevant to this work.

2.1.1. DNA Sequencing Technologies

Several DNA sequencing technologies have emerged during the last four decades, and they showed unprecedented advances leading to a sharp decline of cost and time needed to sequence DNA. In the following paragraphs, we describe different classes of sequencing technologies in chronological order. Each of these technologies either resulted in a significant decline of sequencing cost and time or introduced notable differences in the basic principles used for sequencing.

Sanger sequencing is the first widely used method for sequencing genomes which is why it has dubbed the term first-generation sequencing. The Sanger sequencing method is the first method to introduce sequencing by synthesis, i.e., using a single-stranded template DNA and determining the order or nucleotides while appending nucleotides to its complementary strand in a similar manner to DNA replication. The Sanger method utilizes a particular class of bases called dideoxyribonucleotides. These bases inhibit the elongation of the complementary strand at a random location. Four different solutions in separate tubes represent the four nucleotides. Each tube gets only one of the four corresponding dideoxyribonucleotides. The result is different size fragments of DNA ending in the specific base of the tube. The sizes of the fragments from each of the four tubes indicate the presence of an associated nucleotide base at a particular location. Although this sequencing method produces a relatively longer sequencing reads (up to 700 bp) with a little error rate, it is low throughput, and high cost coupled with the labor-intensive cloning involved could not make it the preferred method for metagenomics [30, 31].

Next-generation sequencing (NGS), also known as second-generation sequencing, is characterized by massive parallelization, i.e., sequencing multiple samples and a large number of fragments from each sample all at the same time. This is made possible due to miniaturization and fast computing capabilities. The Illumina/Solexia, and the now discontinued Roche/454 systems are very popular among other NGS sequencing platforms in metagenomic studies. Their popularity arises from their ability to produce from hundreds of millions to billions of short reads at a time with a reasonably lower cost compared to Sanger sequencing.

The massive parallelization of the sequencing process in NGS technologies is attributed mainly to the ability to immobilize a vast number of DNA fragments on a solid surface. The immobilization allows local sequencing reactions on individual fragments to happen simultaneously and thereby resulting millions of sequences at a time. Imaging systems require multiple fluorescent events to call a nucleotide base accurately. Amplification of DNA template fragments locally is responsible for multiple fluorescent events. For example, the Roche/454 system uses an amplification method called *emulsion PCR* (emPCR) whereas the Illumina/Solexia systems use another variant of clonal amplification called solid-phase amplification.

Most of the NGS platforms implement a variant of sequencing by synthesis developed based on the Sanger dideoxy approach. They use different mechanisms to sequentially incorporate nucleotides complementary to the template strand, one at a time. The Illumina/Solexia platforms use a method called *cyclic reversible termination* (CRT) and whereas the Roche/454 system uses a different method known as *single nucleotide addition* (SNA) or pyrosequencing.

In CRT florescent-marked nucleotides with removable terminator are added to the reaction chamber allowing DNA polymerase to incorporate just one nucleotide that is complementary to the template base. All the clonally amplified strands in a cluster get similar nucleotides, and individual clusters get their corresponding nucleotide bases. Next imaging system takes a snapshot of the entire surface capturing the nucleotides of all DNA clusters corresponding to the current cycle. After that, removing and washing away the terminators from the newly added nucleotides follows marking the completion of one cycle. The same process is repeated multiple times until a certain read length is reached. Having multiple fragments per cluster allows having consensus-based confidence in a base calling which is reported as per base quality stores.

In SNA or pyrosequencing, in every cycle, only a single type of nucleotide is added to millions of wells containing DNA-amplified beads. Hence the name *"single nucleotide."* If the added nucleotides are complementary to the current template DNA, their incorporation will release a pyrophosphate. In case of consecutive matches of similar and matching nucleotides, a comparable number of pyrophosphate will be released. The released pyrophosphates are then converted into visible light using a series of enzymatic reactions with an intensity proportional to the number of freed pyrophosphates. This process is repeated for all four nucleotides bases ensuring the elongation of the complementary strand at least by one nucleotide.

Third-generation sequencing technologies are the most recent sequencing technologies characterized by the ability to sequence a single DNA molecule effectively avoiding the need to do DNA amplification. These sequencing technologies produce long sequencing reads (up to tens of thousands base pairs long) in a short period. The nanopore sequencer by Oxford Nanopore Technologies and the Single Molecule Real-Time (SMRT) sequencer by Pacific Biosystems are leading examples of third-generation sequencing technologies. Even though the long reads produced by these sequencers are attractive, these reads have relatively higher per base error rates. Higher error rates, limited availability, and being still under constant development limit the application of third-generation sequencing technologies in metagenomics to being complementary to NGS methods.

2.1.2. Amplicon Sequencing

A specific part of a genome or even gene can be targeted, amplified and sequenced. Such type of sequencing is called amplicon sequencing, and the selectively amplified fragments are called amplicons. Using amplicon sequencing to target the 16S rRNA molecules of prokaryotes and 18S rRNA of eukaryotes, one can study the diversity of microbial communities. The conserved nature of 16S rRNA genes across prokaryotes makes them easy targets for amplicon sequencing, whereas the presence of enough differences among species enables the detection and quantification of member organisms. This method is cheaper compared to sequencing the entire environmental DNA because multiple samples can be sequenced at the same time using NGS sequencers such as the Illumina's MiSeq on desk sequencer. The main drawbacks of amplicon sequencing in a microbial community analysis are: 1) it is limited to identification of microorganisms and 2) it is subjected to PCR bias.

2.1.3. Shotgun Sequencing

Shotgun sequencing is a DNA library preparation method in which a long DNA molecule is randomly fragmented into smaller pieces suited for the sequencing technology. This method is used across generations of sequencing technologies. ***whole genome shotgun sequencing*** **(WGS)** is one form of shotgun sequencing in which the fragments span across the complete genome of an organism[1]. After sequencing, bioinformatics algorithms are used to get the original sequence by putting the fragments together. The main focus of this thesis will be the development of bioinformatics methods that facilitate the analysis of WGS data in particular which are our main contributions. The newly developed methods will be described in later chapters of this thesis particularly in Chapter 5 and 7).

2.2. Introduction to Metagenomics

2.2.1. Culture-based Genomics

In microbiology, the term culturing refers to the laboratory procedure of growing microorganisms in artificial medium containing nutrients. For microorganisms, growing means to increase the population in contrast to increase the size of individual organisms. Often, this process is manipulated by carefully choosing the nutrients or by setting

[1]The term ***whole metagenome shotgun sequencing*** **(WGMS)** is sometimes used to extend the same method to environmental sequencing. But due to lack of consensus in the terminology and the popularity of the term WGS, in this thesis we use just WGS in metagenomics context.

the environmental conditions such as pH and temperature towards favoring a single species so that only that particular species prevail. The end product of such process is called **pure culture** or **clonal culture**. The individual organisms in pure culture are clones of one species, and if some individuals belong to a different species in the culture, they are considered impurities. Culturing helps to zoom into a single type of microbe and consequently amplify it to understand the characteristics of that microbe better. Even though pure cultures are useful for different purposes in microbiology, they can also function as sources of DNA in studying the genome of a microorganism.

From a pure culture, DNA can be extracted, sequenced and then analyzed computationally with relative ease due to the prior information that the DNA stems from a single species. For example, starting from a pure culture of a microbe, one can extract and sequence their DNA, and assemble the resulting sequence fragments (reads) to get the complete genome of the microbe. Credit to this approach, about 1000 bacterial, 100 archaeal and 2000 viral genomes are available in public databases such as GenBank. Culturing also helps to get enough DNA material for sequencing resulting in better coverage of the genome, which is helpful for downstream computational algorithms.

2.2.2. Advantages of Metagenomics

Despite being a crucial way for studying individual microbes and their characteristics, culture-based studies are limited to a tiny fraction (less than 1%) of microbes that are cultivable to begin with [32, 33, 23]. The resistance to culturing comes from the complex nature of microbial communities and the network of interactions among them. Without this interaction, most of the microbes will not survive, which makes it impossible to cultivate them separately. It is also difficult to replicate the exact physical conditions for optimal microbial growth, especially in the case of extremophiles, i.e., organisms that thrive in an extreme environment. These reasons make it impossible to perform a genomic study on the majority of microorganisms using pure culture as well as to profile or characterize a given microbial community regarding diversity and individual member abundances.

Metagenomics, on the other hand, follows a culture free approach whereby all genetic materials come directly from the environment. This direct extraction will provide a close to complete snapshot of genetic materials of all types of organisms at the site including those that can not be cultured in a laboratory. In addition to capturing the genomes or unculturable microorganisms, metagenomics helps to understand microbial communities of great importance and the complex ecological interactions happening in them. Moreover, this helps to capture the ordinary and routine living condition of microbial communities in their natural environment as opposed to culture-based genomics.

Another advantage of metagenomics over culture-based studies is that it does not

involve the labor-intensive task of culturing itself. Even if the technology exists to grow each species from a microbial community, the process of culturing them all is a tedious manual, if not impossible, task primarily when the communities are composed of a large number of species.

Even though metagenomics is a crucial tool for studying microbial communities and has numerous advantages over culture-based methods, the former will not completely replace the later. Culture-based methods will remain essential and compliment metagenomic studies. Metagenomic studies will also help advance our capabilities of culturing to a broader spectrum of species through a better understanding of environmental conditions required to grow additional species in the laboratory.

2.3. Main Questions in Metagenomics

Metagenomics as a field of science attempts to answer different questions about a microbial community using genetic material extracted directly from their living environment. The following three fundamental questions about a microbial community community constitute most of the field. 1) *who is there?* 2) *What are they doing?* 3) *How are they doing it?* Answering these questions helps first to understand and then utilize microbial communities of a particular interest. Some of the application areas where microbial communities are used are described in section 1.3.

Among the questions mentioned above the first one, i.e., ***who is there?*** is within the scope of this thesis and will be discussed in depth in chapter 4. Nevertheless, we will give a rather short overview of all the three questions in the following subsections.

2.3.1. Who is there?

The types of member microorganisms living in it can be a good starting point to study a given microbial community. Types could be as specific as *species* or even *strains* or as generic as *superkingdom*. In between, there are different levels groupings called taxonomic ranks. In metagenomics studies, it is essential to identify which types of organisms are living in the sample under investigation and enumerate them on a given taxonomic level. It is also equally important to know the relative abundances of different taxa. This process of first enumerating the types of organisms and calculating the relative abundance of each type is called taxonomic profiling. We will provide a more detailed discussion on taxonomic profiling in chapter 4.

2.3.2. What are they doing?

Microbial communities affect and shape their surrounding ecosystem in many ways. It is strongly desirable to study the activities of microbial communities as a whole in their natural habitat alongside with identifying the types of member organisms. The activities include what kind of metabolites they are consuming from and releasing to their living medium. In metagenomics, it is possible to identify the proteins encoded by the collective DNA of the community without knowing from which organism that DNA originates.

In functional metagenomics, DNA is isolated from the environment, cloned and then expressed in the host organism to identify the enzymes encoded by the environmental DNA. The expression data helps to identify some of the proteins that can be produced by microbial communities. The results from functional metagenomics can also help to annotate sequences obtained using sequence-based metagenomics [34].

2.3.3. How are they doing it?

What are the mechanisms enabling microorganisms to affect their environment in specific ways? This question is complicated and involves understanding the network of interaction between different groups of member microorganisms intertwined in different forms of symbiosis. The more diverse a microbial community is, the more challenging it gets to enumerate and understand all the metabolic pathways in the community. Nevertheless, with the help of statistical models and bioinformatic tools, it is possible to shed some light on the microbial activities and metagenomics is an excellent addition to the methods required for this quest.

3. Read-Mapping in Metagenomics

Currently, NGS technologies are the most popular and economically feasible methods of choice to carry out both genomic and metagenomic studies involving DNA sequencing. One thing all NGS technologies have in common is that they involve fragmentation of DNA molecules into manageable small parts. The fragmentation process is entirely random, and there is no information about the relative origin of individual fragments within the DNA molecule. Sequencers turn these short fragments into DNA sequences of corresponding size. The resulting, often short, sequences are called sequencing reads or just reads.

Whether we use shotgun sequencing to target the entire DNA content of the sample or amplicon sequencing to sequence a small part of a genome, the output from sequencers is millions sequencing reads. The sequencers also accompany these short sequences with base quality scores measuring the probability of calling the individual pairs wrongly. These short reads by themselves are not informative unless they are rigorously processed using a serious of bioinformatic analysis. The analysis could be putting the pieces together to get a somewhat complete sequence of the original genome or targeted gene in case of amplicon sequencing (**assembly**), or it could be mapping reads to their original location in a reference genome (**read-mapping**). Assembly has to be done at least once per organism to get a reference genome first and thereby enable the later, i.e., read-mapping. In the following, we give a general overview of genome assembly and discuss read-mapping. Read-mapping is discussed in detail since it is within the scope of our contribution and therefore the scope of this thesis.

3.1. Assembly - Before Read-Mapping

The enormous amount of data produced by NGS technologies need a pipeline of cutting age bioinformatic solutions to achieve the desired research goals. Efficient algorithms are needed to process and make sense of millions of fragmented short reads. The first natural step after sequencing the DNA of an organism is to construct its entire genome by overlapping the resulting short reads. This process is called (*de novo*) genome assembly. The ultimate goal during assembly is to have one long sequence of nucleotides per each DNA molecule (e.g., chromosome) making up the organism's genome. It is often difficult to reach a final assembly of chromosome level due to the

computational challenges posed by repetitive regions inside the genome. This difficulty forces assemblers produce several contigs which are way shorter than chromosomes as a final result.

In a single-genome assembly of NGS short reads, all sequence fragments originate from a single organism, and there is a relatively uniform depth of coverage spanning the genome. This type of information coupled with consensuses by high-quality base pairs helps to resolve the ambiguity of repetitive regions within the genome. There exist several de novo genome assemblers that utilize data structures such as de Bruijn graphs and implement different algorithms. SPAdes [35], Velvet [36], ABySS [37], and MaSuRCA [38] assemblers are few popular ones.

Using culturing methods to isolate, grow, and sequence the genome of a single microbial species many high-quality genomes assemblies are made available to the scientific community. The assemblers mentioned above are designed having single-genome-sequencing in mind. In contrast, metagenomics studies deal with multiple types of organisms, hence multiple genomes, and we can not assume uniform coverage due to unknown varying copy numbers from each type of organisms. These challenges make it difficult to get good quality assemblies resulting in somewhat incomplete and low-quality genome assemblies. The only exceptions are DNA samples taken from very low complexity microbial communities where the number of distinct microbial species are meager. From DNA sequences extracted from such samples, researchers successfully de novo assembled genomes of previously unknown species. However, there exist some genome assembly programs that designed explicitly for metagenomic shotgun sequences and built upon existing single genome assemblers. MetaVelvet [39] which is built upon the Velvet assembler is a good example.

Once a reference genome of an organism is constructed via de novo assemblers, it can be used for a multitude of other applications including reference assisted assembly (e.g., the accompanying assembler of SOAP2 read mapper [40]). Reference assisted assembly is computationally less expensive compared to de novo assembly. It is done by first mapping the short reads against an existing reference genome and finding consensus-based contigs from the overlapping reads. The resulting contigs can, in turn, be used to carry out variant calling between the new sample and the reference or perform genome-wide association studies. Another common usage of reference genomes is in read-mapping, which is discussed in detail in the following section.

3.2. Read-Mapping

Read-mapping is the process of locating one or more regions in a sequence that are similar to relatively much shorter sequences, dubbed reads. We call the larger sequence(s) that we search into *subject sequences* or simply *subjects*, whereas the term

query represents shorter fragments, that are being searched. The degree of similarity between queries and mapping regions of subject sequences varies from identical to having several mismatches, insertions, and deletions. Nevertheless, it is desirable to report regions with the highest similarity.

Given a set of subject sequences $\mathbb{G}$, and a set of short sequencing reads $\mathbb{R}$ and an error threshold or ϵ, for every read $r_i \in \mathbb{R}$ a read-mapping problem finds all the mapping locations $L_{jp} : 0 \leq l_{jp} < |g_j|$ within all of the subject sequences $g_j \in \mathbb{G}$, where r_i matches with ϵ or less number of differences with a segment of g_j that starts at location L_{jp}.

3.2.1. Indexing References Genomes

Classical approximate string matching algorithms have a complexity proportional to the length of the subject text [41, 42]. However, considering how big the subject text and the millions of queries in read-mapping, these algorithms are impractical to be adopted for read-mapping. Instead, a supporting index data structures which are created by preprocessing mostly the subject sequences are used to speedup the read-mapping process. Suffix trees [43] and suffix arrays [44] (space efficient alternatives) are among the first support data structures used to speedup read-mapping. These data structures provide a means to locate exact matches of seeds which are smaller stretches in a read. An exact match between a seed and a reference can be used as a starting point for approximate read-mapping. Suffix trees and suffix arrays have a time complexity of $O(m + occ)$ and $O(m + occ + \log n)$ to locate all occurrences of a pattern P respectively. Where m is the length of P and n is the length of the subject sequence, and occ is the number of occurrences of the P in the subject sequence [45]. Despite having an optimal runtime complexity, both data structures require an enormous amount of memory, i.e., $\Theta(n \log n)$ bits. This memory requirement is asymptotically larger than the size of the genome which takes $2.n$ bits assuming we can save a nucleotide using 2 bits. The memory requirement is prohibitively large even for the human genome [46].

The FM-index by Ferragina and Manzini [47] is the first index structure which is fast and memory efficient (proportional to the size of the subject sequence). It utilizes the Burrows-Wheeler transform of a text. The original text can be generated from an FM-index, making it also preferable for storage. Due to its appealing feature, i.e., fast search speed and low memory requirement, FM-Index become popular among modern day read mappers.

Another category of support data structures (indices) uses hash tables to store short sequences of length k known as k-mers and their corresponding locations in subject sequences. In this approach, the hash tables provide answers to the locations of k-mers within the subject sequence (s). The value of k determines the size of the hash table, and the heterogeneity of sequence(s) used. It is essential to consider the size and diversity

of subject sequences and the implementation of search strategies when choosing the length of k. The k-mer should be discriminative enough not to occur everywhere and small enough to be used as a seed for approximate string matching.

The data structures mentioned above provide a means of locating exact matches of fragments of a read with time complexity linear and sub-linear time to the length of the fragments. After locating the fragments, searches can be narrowed down to segments of the subject text using various filtering strategies. The cost of creating such index data structures is considerably high. However, due to the static nature of genomes, indexes can be built and stored on disk to be used repeatedly. The repeated usage of an index data structure once it is built guarantees the high cost of building the index amortizes over time.

In single organism genomics, read-mapping deals with a single reference genome containing a handful of chromosomes. The repeats in a genome are a result of gene recombination attributed to years of evolution. These repeats lead to multiple matching locations for a read which in turn causes ambiguity. The size of the reference genome ranges from a few thousand base pairs long in the case of viruses to billions of base pairs long. Without any preprocessing, it is prohibitively slow to locate all of the mapping locations of a read across the length of a genome(s). The standard solution to this problem is to construct different types of indices from a reference genome and use it for mapping sequencing reads. The indexing procedure is often time-consuming and also requires relatively higher computational resources. This cost gets even higher when the size of the database gets larger. However, once we have indexed a reference genome, and stored the resulting index on a disc, it can be used to map sequencing reads from multiple samples by merely loading it to memory without the need to rebuild it.

Contrary to single organism genomics in metagenomics, read-mapping has to deal with a set of references containing thousands of reference genomes. Although microbial genomes are small in size compared to mammalian genomes, their combined size could easily exceed the size of the biggest mammalian genome. As stated above, the larger the reference set is, the more difficult it gets to construct a corresponding index from it which is necessary to perform an approximate search. Another challenge unique to metagenomics is that the presence of similar genomes representing closely related species that present the same challenge as repeats within genomes. Due to this, a significant portion of metagenomic sequencing reads end up with a large number of mapping locations spanning a sheer number of genomes. From an indexing perspective, the ever-growing reference set of genomes due to new assemblies undermines the indexing process as one has to rebuild an index every time a new genome pops up. We address this problem in chapter 7 of this thesis.

3.2.2. Popular Read Mappers

In this subsection, we describe the notable features of popular read mappers and their unique features. We will also highlight the underlying methods used in the development of the tools as well as their strengths and shortcomings from mapping metagenomic reads to reference sequences of microbial genomes. We do not intend to make an exhaustive review of the individual read mappers, nor we make a comparison between them. However, we included four of the read mappers below in our exhaustive benchmarking process presented in chapter 8. We are aware that there are many more read mappers available. However, listing all of them is beyond the scope of this thesis.

Bowtie2 [48] is among the most popular read mapper, and it uses the FM-index to index reference genomes. Bowtie2 can handle alignments with large gaps. The latest version of Bowtie2 supports a parallel construction of indices which is an appealing feature in metagenomic read-mapping due to the size of reference databases. On the other hand, Bowtie2 does not guarantee to find all mapping locations for individual reads, which is a common phenomenon in metagenomic read-mapping due to having many similar genomes in the reference set.

BWA-SW [49] works by indexing both the reference and query sequences using FM-indices. It uses a heuristic algorithm to speedup alignment at the cost of probably missing some correct/better matches. BWA-MEM is a very popular read mapper available within the BWA software package which is more suited to longer and error-prone reads. BWA-MEM uses maximal exact matches for seeding to restrict the search space.

GEM [50]is a fast and sensitive read mapper that deploys a strata based mapping strategy. It uses pigeonhole-like filtering strategy using seeds which tolerate errors. The authors of GEM claim that it always perform complete searches, although our benchmarking (chapter 8) did not wholly corroborate with this claim. GEM missed a subtle amount of matches within its specified parameters. The exhaustive search performed by GEM for each read to report all mapping locations is desirable when mapping metagenomic reads against microbial reference genomes as multiplying reads are common occurrences.

Bitmapper [51] is an all-mapper which uses a hash table based index to locate small seeds (k-mers). It does filtering using the pigeonhole principle. It uses a modified Gene Myers' bit-vector algorithm to speedup the verification step where a read is checked whether it can align in a candidate location within the edit distance threshold.

mrsFAST [52] uses a collision-free hash table index to store k-mers of a genome and a list of their corresponding locations. mrsFast claims to be faster than existing methods while it accommodates only substitution errors. It uses the pigeonhole principle for

filtration. Within a given threshold, mrsFast reports all the mapping locations of a read accommodating only for substitution errors.

Hobbes 2 [53] is another hash table based fully sensitive read mapper. Hobbes 2 can handle indel and mismatch errors. Hobbes uses a filtering strategy based on the q-gram lemma with the help of an additional q-gram index to generate candidate locations and uses dynamic programming to extend them to a full alignment.

Yara Mapper [54] is an ultra-fast and fully sensitive read mapper of the SeqAn library [55]. Yara's filtering strategy uses spaced seeds in combination with the pigeonhole principle. The ultra-fast speed of Yara, its unmatched recall rate of alignments within a given error threshold, and its option to define strata based mapping results are among the reasons that makes it an attractive choice for metagenomic read-mapping.

3.3. Alternatives to Read-Mapping

In certain metagenomic pipelines such as taxonomic profiling or read assignment, the mapping location of a read within the stretch of a reference genome might be unnecessary depending on the required information for downstream analysis. In a taxonomic assignment of reads, mapping locations are ignored because it suffices to know whether or not a read has a mapping location in the genome. Such approaches save a significant amount of time and computing resources required by a precise read alignment to the exact matches and mismatches and all the mapping locations. Note that in procedures like this there is no need to list all the mapping locations exhaustively.

Another alternative to read-mapping is pseudo-alignment where the locality of a read is computed without the need to compute the complete base by base mapping of a read against a reference genome. Doing so effectively avoids the need for dynamic programming which is a time-consuming task. This concept was first introduced in Kalisto [56] for RNA-Seq and later adopted in meta-Kalisto [57] for taxonomic profiling.

Part II.

Taxonomic Profiling

4. Taxonomic Profiling

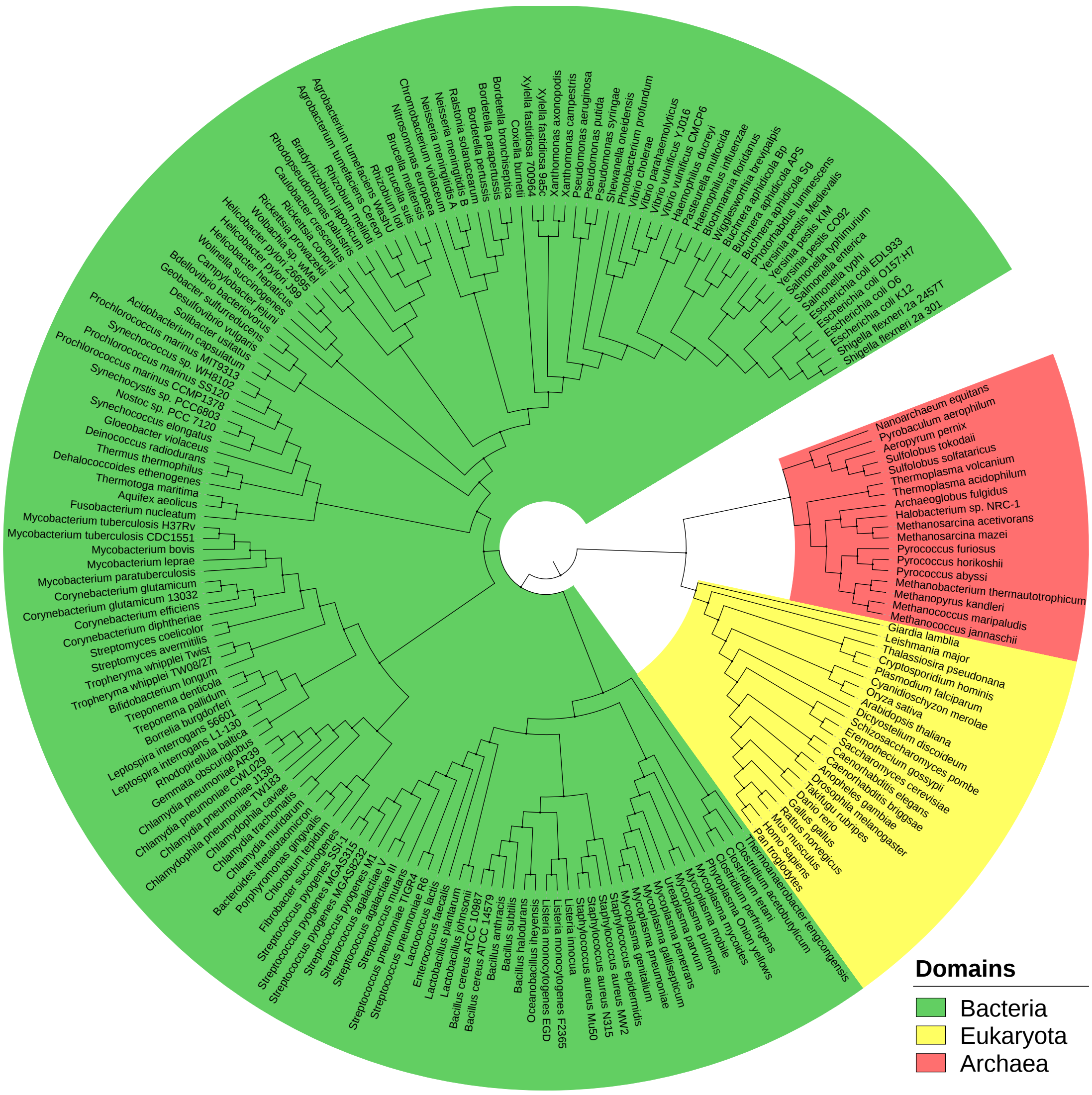


Figure 4.1.: Tree of life produced by iTOL [58]. Different colors indicate the three domains of life namely Archeae, Bacteria, and Eukaryota. The labels on leaf nodes represent different species.

4.1. Introduction

Biologists classify and organize living things into a hierarchical structure called taxonomic tree. Taxonomic trees reflect phylogenetic relationship. The taxonomic tree of life (figure 4.1) is organized using several hierarchical levels commonly known as taxonomic ranks. There are seven principal taxonomic ranks namely domain, phylum, class, order, family, genus, and species listed from generic to specific. Although not clearly defined there exist as well intermediate ranks such as super-order or sub-species which are more generic or more specific to a corresponding rank respectively. Each node in a taxonomic tree denotes a taxon (*plural taxa*) which represents a group of organisms exhibiting similar characteristics. All the subgroups under a node, i.e., the branch of a tree starting from that node, form a **clade**. In addition to exhibiting similar properties, organisms belonging to a clade are believed to be evolved from a common ancestor (see figure 4.2). Similarly, organisms belonging to the same clade have more similar genomes than those from a different clade. This genetic similarity is crucial in the identification of microorganisms using DNA sequences obtained through different NGS techniques. In metagenomics, the term taxonomic profiling refers to the

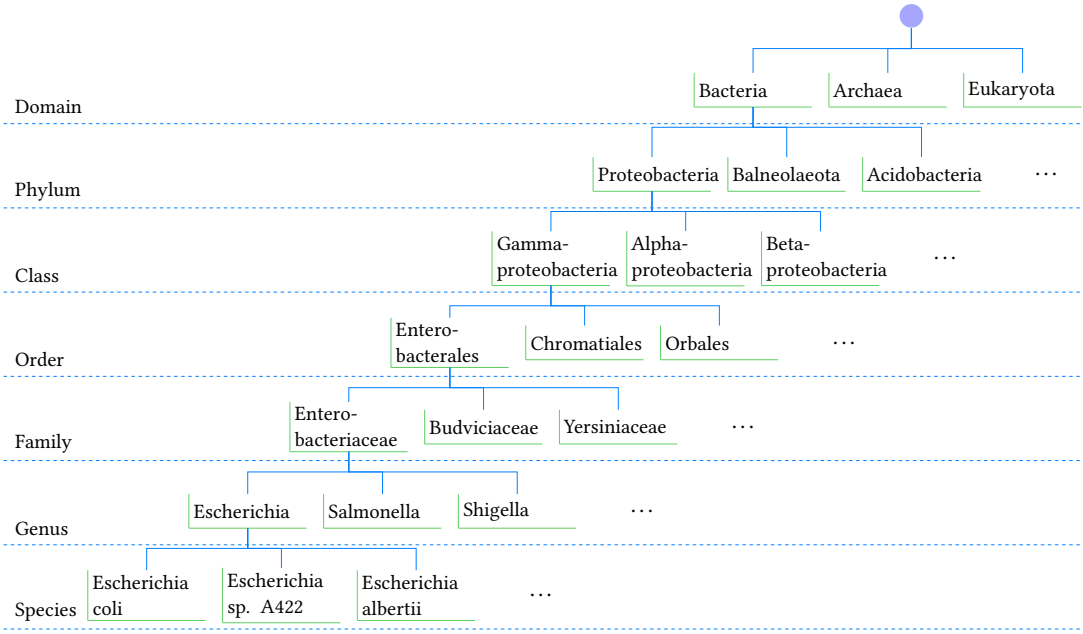


Figure 4.2.: A partial view of a taxonomic tree. All the seven priincipal taxonomic ranks are shown with few examples.

process of determining the composition of a given microbial community as a list of different biological taxa that make up the community and their relative abundance. The taxonomic profile of a microbial community is important due to the need to study species diversity of a single microbial community (i.e., alpha diversity) and the degree

to which a composition of microbial community changes [59]. The growing interest in studying microbial communities (see in section 1.3 for details) and the increasing availability of NGS data calls for efficient and robust taxonomic profiling methods.

There are two aspects of taxonomic profiling, qualitative and quantitative. An excellent taxonomic profiler has to deal with both adequately. The qualitative aspect deals with identifying the different group of microorganisms residing in the community. Whereas the quantitative aspect often deals with the prevalence or relative abundance of individual groups in the community. The resolution of a taxonomic profile depends on the choice of taxonomic rank/level.

4.2. Importance of Taxonomic Profiling

In scientific fields such as microbiology and ecology, there is a growing interest in studying the diversity of a microbial community residing in a specific habitat. Such diversity, which is known as alpha diversity [59], can be inferred using methods in metagenomics starting from sequencing the genetic material taken directly from the environment and analyzing it using suitable taxonomic profiling methods. Such tools enable the characterization of any given microbial community through their composition. Taxonomic profiling tools could also demonstrate the presence of a pathogen in the mix.

On the other hand, comparative metagenomics studies can be performed using taxonomic profiles of multiple microbial communities or multiple profiles of a single community at different time points. Such studies have a wide range of applications from probiotics to personalized medicine. The utilization of the gut microbial composition of farm animals such as pigs as an indicator of animal health in the animal industry is a more specific example.

For reasons mentioned above, quantification of microorganisms using DNA sequencing reads obtained by Next-Generation Sequencing (NGS) has become a subject of growing interest in the field of microbiology. The publication of numerous taxonomic profiling tools within the last decade only shows how appealing the subject indeed is. Lindgreen et al. [60] considered 14 different sequence classification tools based on various approaches in a recent review of such methods.

4.3. Challenges

There exist a multitude of different challenges making taxonomic profiling a complicated task. However, most of the challenges derive from either the absence of high-

quality reference genomes or the extreme similarity observed among the genomes of related species. In spite of the extensive efforts made to maintain public databases of reference genomes there remains a gap in providing a non-redundant but yet complete set of reference genomes spanning a wide range of microbial species.

When it comes to the similarity among reference genomes, it creates a challenge by making the assignment of sequencing reads to their origin ambiguous. The ambiguity gets worse when one wants to perform taxonomic profiling at higher resolution, i.e., at lower taxonomic ranks such as species as well as strains which is desirable. Figure 4.3 illustrates the extent of challenges posed by homologous genomic regions of closely related microorganisms. We merely took three sets of 20 different genomes and analyzed the number of unique 25-mers. Each set contains representative genomes from different genera species and strains within a single family, genus and species respectively. As it is shown in the plot, the average number of new 25-mers introduced along with an addition of a new genome is 16% for the strains set and 27% for species set. Only the set of genomes representing different genera exhibited a significant difference (84% per genome).

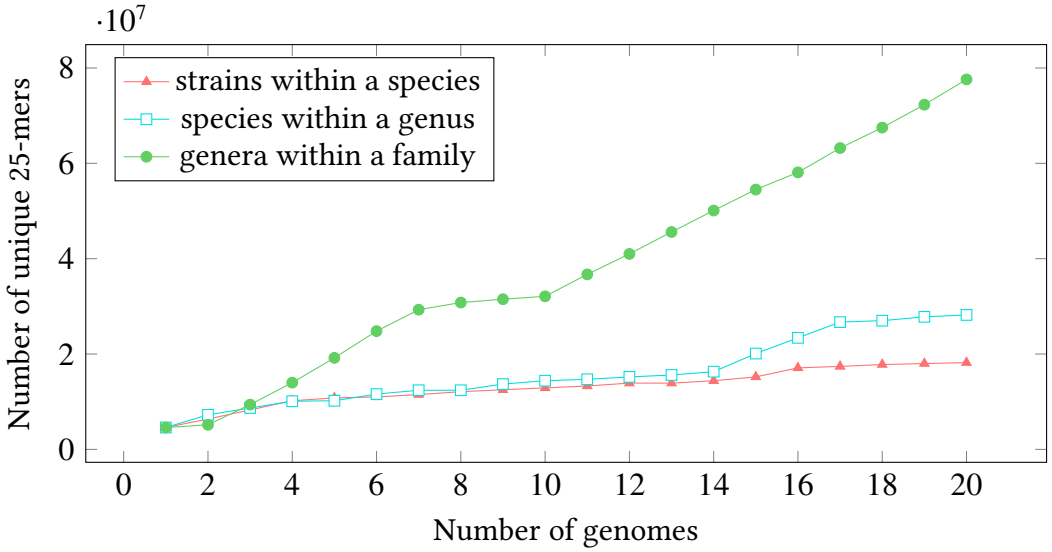


Figure 4.3.: The number of unique 25-mers from multiple genomes representing different strains, species and genera belonging to a single species, genus and family respectively. The amount of new 25-mers introduced along with an additional genome reflects how similar the genomes are.

Another challenge in taxonomic profiling comes from a pronounced range of variation in the relative abundance of individual groups in a microbial community. A high degree of variation in relative abundance creates difficulties to detect the least abundant microorganisms and to differentiate noise from the right signal. In other

words, it is difficult to detect under-represented members of a microbial community by metagenomic shotgun sequencing because the share of reads originating from these organisms is often lower than those assigned to a false positive genome due to sequence homology. This problem gets worse due to the high degree of variation in publicly available genome sequence quality and length of different microbes. The problems mentioned above make the identification and quantification of microbial communities a non-trivial task [61].

Due to the hurdles discussed above, especially the challenges posed by the homology of sequences among genomes of closely related microorganisms, several taxonomic profiling tools developed in earlier days produce taxonomic profiles at a higher taxonomic rank such as genus. Consequently, accuracy benchmarks were often performed at the genus or higher level of the taxonomic ranks. This choice is due to the shortcomings of many earlier tools to report species-level taxonomic profile with acceptable accuracy. However, a species-level resolution of microbial communities is desirable, and more modern taxonomic profiling tools offer species-level identification [62, 63, 60, 64].

4.4. Different Approaches to Taxonomic Profiling

Although there are other ways of performing taxonomic profiling on a given microbial community, in the scope of this text, taxonomic profiling is limited to DNA sequencing-based methods. There are two major approaches to obtain DNA sequences for taxonomic profiling, namely 16S-rDNA[1] sequencing, and WGS sequencing. The major difference lies in the DNA library preparation before sequencing. In WGS we blindly shred all the DNA extract sequences it directly whereas in 16S-rDNA we selectively amplify the 16S genes which are present in all prokaryotes by taking advantage of the conserved nature of this gene to design a universal primer across all prokaryotes. The follow-up analysis of a 16S-rDNA uses the variable regions within the gene to differentiate one organism from the other. Metagenomic sequences obtained in both approaches need a corresponding computational pipeline specific to the approach except for few taxonomic profiling methods that can handle sequences generated in both ways.

Both methods have their strengths and weaknesses. Since the 16S is very short (around 1500 bases long) compared to a typical genome of prokaryotes, one can sequence many samples in one run via multiplexing. This fact gives 16S-rDNA based studies a cost advantage over the WGS counterpart. Consequently, using 16S-rDNA methods, it is economically feasible to study hundreds and thousands of samples. Another

[1] 16S based methods are limited to prokaryotes. A similar method based on 18S rDNA is used to identify eukaryotes. If one wants to detect both prokaryotes and eukaryotes in a community via amplicon sequencing, the amplification process needs to be done separately.

advantage of 16S-rDNA based sequencing is the presence of well-maintained and matured reference database which helps the taxonomic profiling process [65, 66, 67, 68].

On the other hand, the presence of specific microorganisms interferes with the PCR process [69, 70] and certain primes favor 16S sequences of a particular group of organisms or disfavor others [71, 72, 73, 74], creating an artificial bias in both identification and quantification of a given microbial community. Moreover, the variations in the 16S genes are insufficient to discriminate microorganisms at species or strain level [75, 76]. Additionally, 16S-rDNA based methods are limited to prokaryotes as the gene is found only in prokaryotes which excludes viral and eukaryotic microorganisms such as fungi from the identification process. The limitation mentioned above of 16S-rDNA sequencing are naturally inexistent in WGS methods making WGS based taxonomic profiling a method of choice when the sequencing cost is bearable [77, 78, 19].

To tackle the challenges presented by ambiguous sequences (reads) that originate from genomic locations shared among multiple groups of organisms, Most taxonomic profiling tools implement two distinct approaches. The first approach is to prepare a signature-based database with sequences that are unique to a clade. In this approach, taxonomic clades are uniquely represented by sequences that do not share common regions with other clades of the same taxonomic rank. If we obtain the input sequence via 16S-rDNA sequencing, we have such a unique signature database for free. The 16S-rDNA of microorganisms contains alternating variable regions among organisms. However, in the case of WGS sequencing, one should curate, prepare and store this signature database comprised of marker genes in advance. Even if this approach uses the fraction of metagenomic data from the sequencer, it can guarantee to have only a unique assignment of sequencing reads to a clade. Tools like MetaPhlAn2 [79], GOTTCHA [80] and mOTUs [81] use this approach. The second approach works using the full set of reference sequences available as a database and assigning ambiguous reads to their *lowest common ancestor* (LCA) in a taxonomic tree. Kraken [82], a k-mer based read binning method, is an example of such an approach.

Both approaches mentioned above have certain advantages and disadvantages. The former has an advantage in speed and precision, but it is limited to utilizing the reads that can be mapped uniquely to the curated regions. The latter approach, on the other hand, suffers from the lack of uniquely-mapped reads at higher (more specific) taxonomic ranks since they are assigned to the LCA. Recent methods such as DUDes [62], SLIMM [83], and Bracken [84] follow a more elaborate routine to solve read ambiguity.

Depending on the method of choice for DNA sequencing, i.e., either 16S-rDNA or WGS, there are numerous existing taxonomic profiling tools available. Some tools take only one of 16S-rDNA or WGS data as their input. Few tools can handle both types of sequencing data. In the following section, we will enlist and provide a short description of 13 different tools. We are aware that, there are other taxonomic profiling tools out there and this is not a comprehensive list by no means. We included tools for this text

in a way that: a) we can cover different approaches and sequencing techniques; b) we included the most popular and better performing taxonomic profilers. Table 4.1 gives a quick overview of different taxonomic profilers along with the type of sequencing they can handle and what kind of databases they use.

Tool	Seq-Type		Database			Alphabet		Requires Alignment
	WGS	16S	Full Genomes	Marker Genes	k-mer based	DNA	Protein	
Bracken	✓		✓		✓	✓		
CLARK	✓				✓	✓		
DUDes	✓		✓			✓		✓
MGnify.	✓	✓	✓			✓		✓
GOTTCHA	✓			✓		✓		✓
Kraken	✓		✓		✓	✓		
LMAT	✓				✓	✓		
MEGAN	✓		✓			✓	✓	✓
MetaPhlAn	✓			✓		✓		✓
MetaPhlAn2	✓			✓		✓		✓
MG-RAST	✓	✓	✓			✓		✓
mOTUs	✓					✓		✓
SLIMM	✓		✓			✓		✓
QIIME		✓				✓		✓

Table 4.1.: A short summary of existing taxonomic profiling methods and their key features. The table shows the type of sequence the taxonomic profilers support (WGS/16S) and the database type they use. It also shows if a tool is alignment based or not

4.5. Existing Methods

Based on the final output of a method there are two categories of metagenomic classification tools, i.e., a read binning method and a taxonomic profiling method. A read binning method assigns every single read to a node in a taxonomic tree, whereas a taxonomic profiling method tries to report which organisms or clades are present in the sample with or without having to assign every read to a corresponding taxon. There exists an overlap between the two categories making it possible for some read binning methods to be used as a taxonomic profiling tool as well.

4.5.1. Bracken

Bracken [84] is a taxonomic profiler that focuses in better estimation of abundances at species level by using the read assignment output provided by Kraken (*described below in sub section* 4.5.5) as an input. It probabilistically redistributes reads originally assigned to a parent taxon back to the children and by that achieving a better read count and consequently reporting more close to real abundance values. Bracken uses a database of overlapping k-mers associated with a taxonomic node which is LCA to all the genomes containing that k-mer. See the subsection 4.5.5 for more detail.

4.5.2. CLARK

CLARK [85] focuses on classifying reads to target taxonomic groups with improved accuracy while staying fast. It uses a k-mer database extracted from reference genomes excluding k-mers belonging to multiple taxonomic groups (*targets*). The authors argue that the remaining unique to target k-mers can be used as signatures for assigning reads to targets. CLARK assigns reads to a group that has the maximum number of shared k-mers in the database. While building the database of CLARK users should decide at which taxonomic level they want to classify the sequencing reads. CLARK also offers a feature to exclude k-mers with a small number of occurrences in a group and confidence scores for read assignment. This approach makes CLARK fast and memory friendly.

4.5.3. DUDes

DUDes [62] uses a new top-down approach, i.e., utilizing the deepest uncommon descendent (DUD) in contrast to many methods that use the LCA method to resolve ambiguous reads mapping to multiple groups. Starting from a node at a higher level of a taxonomic tree, which has fewer ambiguities since groups are more distinct at such levels, DUDes tries to go for deeper taxonomic levels, even when ambiguities exist. DUDes uses alignment files (SAM format) obtained by mapping NGS reads against reference genomes using a read mapper of choice. The ability to perform well with a small fraction of NGS read-set is an appealing feature offered by DUDes.

4.5.4. GOTTCHA

GOTTCHA [80] is another taxonomic profiling tool that focuses on reducing the *false discovery rate* (FDR) of member microorganisms. To achieve its goal of low FDR, GOTTCHA uses databases containing unique segments of reference genomes dubbed as “unique genomes”. GOTTCHA’s databases are precomputed and made available for download. One should note that the selection of “unique genomes” is dependent on the target taxonomic rank which means, different databases for different taxonomic levels. After quality trimming NGS reads, GOTTCHA generates non-overlapping 30-mers from the reads and search for their exact matches using the BWA read mapper in its database. Using this matches, it creates coverage profile of the “unique genomes”. These coverage profiles are then used by GOTTCHA to create taxonomic profiles. We want to point out that taxonomic profiles generated by GOTTCHA are indeed highly specific, but they miss out many true-positives when the microbial communities are complex and diverse, i.e., contain a large number of species.

4.5.5. Kraken

Kraken [82] uses a pre-computed k-mer database that maps a k-mer and its corresponding node in a taxonomic tree. A node for a k-mer represents the LCA of all reference sequences that contain that k-mer. In order to make the k-mer lookup process faster, Kraken deploys a k-mer minimizer concept where k-mers sharing the same minimizer are stored close to each other effectively making the retrieval process cache in a friendly way. During read assignment, it looks up all the overlapping k-mers of a read in the hash table and classifies the read to a taxonomic node with the highest number k-mer hits. Whenever there is a tie between two nodes, Kraken resolves this by taking lowest common ancestor of the two. Kraken is a very fast read binning method, which is also often used to do taxonomic profiling. However, it is worth mentioning that it is not suited for abundance estimation.

4.5.6. LMAT

Similar to Kraken, LMAT [78] uses a database of k-mers extracted from a set of genomes. The database contains a mapping between each k-mer and the LCA of all the genomes containing it. A supplementary marker library called kML is generated by taking a subset of the k-mers and arranging them into disjoint sets. Any set that contains less than a thousand k-mer is excluded and k-mers whose LCA lies above the family rank are not considered. This library contains the most discriminative and informative k-mers reportedly. During a taxonomic assignment, a read goes to a node which is the LCA of most of the overlapping k-mers of that read. LMAT can also perform functional assignments to metagenomic reads.

4.5.7. MEGAN

MEGAN [86] takes the output of BLAST or similar programs including alignment output from short-read mappers, where metagenomic shotgun reads are mapped against protein reference databases (e.g., NCBI-nr) as an input. MEGAN then employs a simple LCA algorithm to assign reads to a taxonomic unit in the NCBI taxonomy using the alignment result obtained on the preprocessing step. The interactive graphical user interface of MEGAN enables users to explore a microbial community dipper. MEGAN also offers the possibility to compare multiple taxonomic profiles supported by graphical and statistical outputs. In addition to taxonomic-profiling, MEGAN also offers functional metagenome analysis on using different databases such as the SEED [87] hierarchy.

4.5.8. MetaPhlAn

MetaPhlAn [88] utilizes a large set of marker genes unique to a species or other higher level clade to map WGS metagenomic reads using the Bowtie [89] read mapper. MetaPhlAn then uses the mapping result and turn it to a taxonomic profile. This approach guarantees a unique match per read elevating the problem of ambiguity. The fraction of mapped reads is small compared to the total number of sequenced reads as the unique marker genes used for mapping are also a small fraction of the complete reference genomes. The fact that ambiguous reads are less critical for the process of taxonomic profiling and the computational performance gained justifies the low alignment rate.

4.5.9. MetaPhlAn2

MetaPhlAn2 is an improvement over MetaPhlAn by including more marker genes that span more species and markers that discriminate even between different strains of the same species. Additionally, MetaPhlAn2 supports eukaryotic and viral quantification and strain tracking. It also has improved speed as it uses Bowtie2 [48] instead of its predecessor.

4.5.10. MG-RAST

The MG-RAST webserver [90] offers a multitude of analysis for metagenomic sequencing data. The sequencing type can be either WGS or 16S-rDNA. After registering, users can upload their data sets for analysis and get a taxonomic profile as well as functional assignments. The taxonomic profiles provided by MG-RAST are generated by comparing sequences to both nucleotide and protein databases. For 16S-rDNA based sequences, MG-RAST offers three different alternative databases namely Greengenes [67], RDP II [91] and the European 16S RNA database [92].

4.5.11. MGnify

The European Molecular Biology Laboratory-European Bioinformatics Institute (EMBL-EBI) offers a free holistic web service to perform taxonomic profiling under its EBI Metagenomics web service [93] which now has a new name called MGnify. The user is required to upload his/her raw sequencing reads to the *European Nucleotide Archive* (ENA) together with standard metadata. The uploaded data then will go through a series of data analysis pipelines including quality control and adapter trimming steps using various open source tools. We recommend the reader to read the publication

by Mitchell et al. to get a closer look on which analysis steps are available and which tools are used by the web service. At the end of the analysis, taxonomic profiles and other statistics are displayed accompanied by different graphical presentations.

4.5.12. mOTUs

mOTUs [81] uses single copy universal marker genes to achieve a species-level abundance resolution of microbial communities. Like 16S-rRNA genes, these genes are universal across species but have only a single copy per genome alleviating the bias due to copy number variation of 16S-rDNA based methods. After scanning a large set of reference genomes for 10 of such universal marker genes, the hits were clustered to form “metagenomic operational taxonomic units (mOTUs). mOTUs computes taxonomic profiles based on the mapping result of metagenomic shotgun sequencing reads against this mOTUs database.

4.5.13. QIIME

QIIME [94] is an integrated software suite that contains a collection of scripts and other third-party applications to perform a range of analysis on amplicon data. QIIME have several scripts that can perform various quality control and data analysis at various stages of the taxonomic profiling process. In QIIME 16S-rDNA or other amplicon sequences are clustered into OTUs in a process called OTU picking which serves as a basis for taxonomic profiling. QIIME features different OTU picking strategies namely de novo, closed-reference, and open-reference OTU picking strategies. De novo OTU picking works by clustering reads in the absence of references whereas in closed-reference OTU picking clusters are formed on the bases of existing OTUs with references and unclustered reads are left out from further analysis. In open-reference OTU picking, on the other hand, unclustered reads will be subjected to de novo OTU picking. Then, taxonomic profiles are inferred based on the number of reads in the individual clusters taking the copy number of 16S genes into account. QIIME also does functional assignments based on the KEGG [95] functional database.

5. SLIMM

In this chapter, we will present *species level identification of microorganisms from metagenomes* (SLIMM), a novel and easy to use taxonomic profiling method, that better addresses the problem of ambiguous reads. As we are going to demonstrate in the coming sections of this chapter our novel approach enables SLIMM to outperform its competitors in both specificity and sensitivity as well as in calling the correct relative abundance of discovered species. As the name implies, SLIMM is well suited to perform taxonomic profiles at species level resolution. However, SLIMM can also produce taxonomic profiles at higher levels of taxonomy such as genus upon the users' request. SLIMM is open source and freely available for download from https://github.com/seqan/slimm.

5.1. SLIMM Motivation

As discussed in the previous chapter, one of the main challenges in taxonomic profiling comes from shared regions of genomes among reference sequences. Existing tools deal with this either by reducing the reference genomes to a much smaller representative marker genes that are unique per clade or by simply assigning sequencing reads mapping to multiple references to the LCA. Both approaches have their shortcomings. The former has to rely on the presence of reads overlapping with the selected unique marker genes which are only a tiny fraction of the complete genome. This causes significantly less abundant organism to be easily overlooked. The latter approach, on the other hand, assigns a majority of the reads to a higher rank, leaving the lower level taxonomic assignment sparse. Consequently, taxonomic profiles at lower level taxa (higher resolution) are biased and less accurate. Besides, there is a high chance of missing a species just because there exists another highly similar species is in the database, and all of the reads, mapping to it are pushed up to a higher level taxa.

Another issue we wanted to address is the accuracy of abundances reported for individual species. In addition to reporting all the member organisms of a microbial community, an excellent taxonomic profiler should assign accurate relative abundances to the detected organisms. This property is crucial when one does comparative metagenomics across multiple communities or across different time point of the same community. Computing correct abundance profile of a microbial community using

next-generation sequencing (NGS) necessitates considering the number of genomes representing a member organism in a database and their sizes and the possibility of reads shared by an organism that is not present but have a close relative in the community.

With these ideas in mind, we designed SLIMM to be an accurate taxonomic profiler in the context of both listing the correct member organisms and reporting their corresponding abundances. To achieve such accuracy we utilize the read coverage landscape of genomes and exclude reference genomes with poor coverage from consideration. By doing so, SLIMM effectively increases the number of quasi-unique reads available at lower taxonomic ranks. We also engineered SLIMM to be computationally efficient by reducing its runtime and memory footprint.

5.2. SLIMM Pipeline

The core algorithm of SLIMM requires two inputs. The primary input is an alignment/mapping file in SAM or BAM format, obtained by aligning short metagenomic shotgun sequencing reads against a library of reference genomes of interest. A secondary input that we refer to as the SLIMM database is also instrumental for the taxonomic analysis. This database holds the taxonomic information related to all the reference sequences being considered. The information stored inside SLIMM database includes the names and hierarchical relations of all taxonomic clades covered by the set of references used to perform read-mapping. The database is built by extracting a subset of NCBI's taxonomic tree specific to the genomes under consideration and a mapping from accession numbers to a taxonomic identifier (tax_id). The SLIMM pre-processor facilitates all these inputs required to run the core algorithm of SLIMM. Figure 5.1 illustrates an overview of the SLIMM pipeline, and we will provide a detailed description of the individual steps in the following subsections.

5.2.1. Preprocessing Module

SLIMM preprocessing module is designed to make the process of assembling a set of reference genomes easy. It includes three python scripts that enable the user to select download and prepare reference genomes from the NCBI's *RefSeq* (ftp://ftp.ncbi.nlm.nih.gov/genomes/refseq) or *GenBank* (ftp://ftp.ncbi.nlm.nih.gov/genomes/genbank/) databases in highly configurable manner. These three scripts, namely *select_refs.py*, *collect_refs.py*, and *merge_refs.py* can be executed sequentially to obtain a single multi FASTA file that contains the desired set of references ready to be used by a read mapper of choice either directly or after creating a suitable index. In the following, we will describe all of the three python scripts and the features they offer.

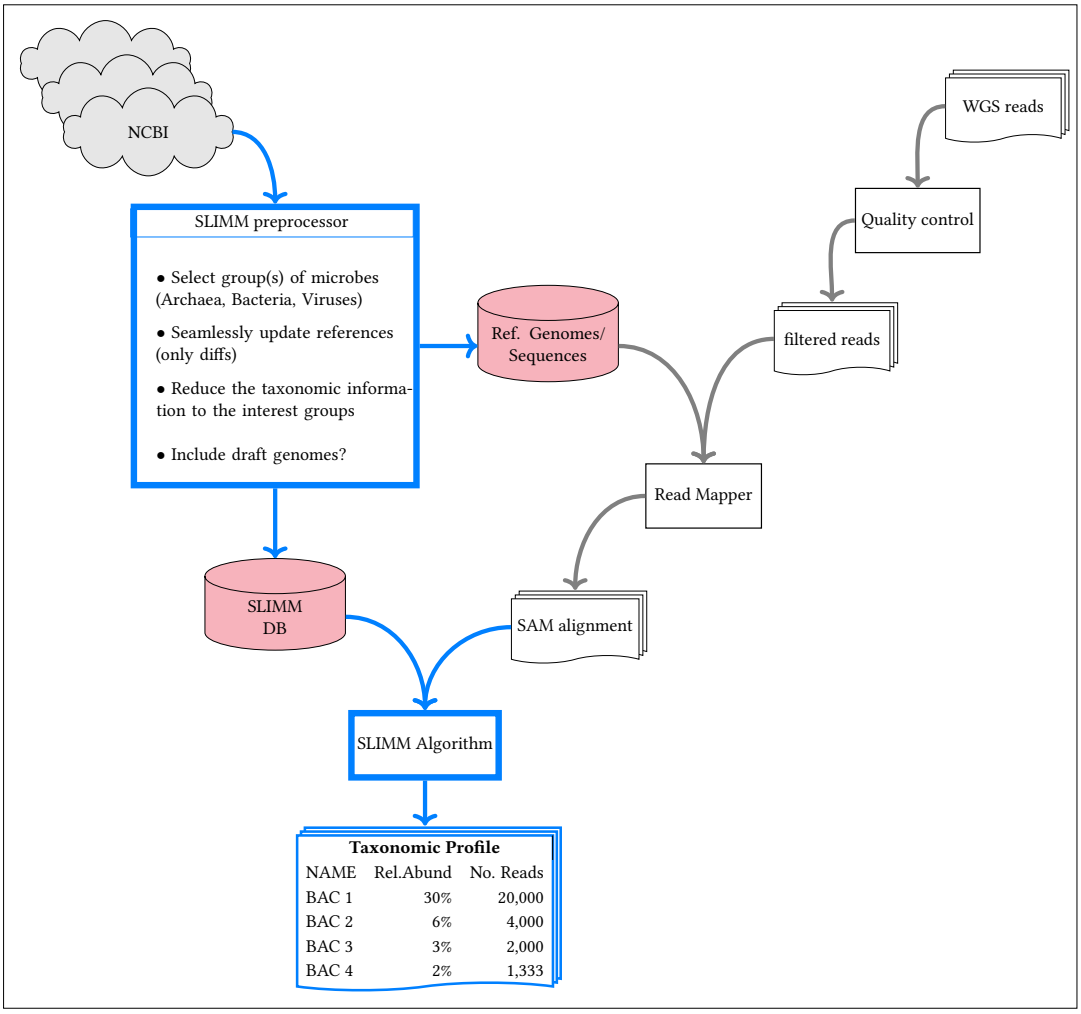


Figure 5.1.: SLIMM Pipeline: The preprocessing module of SLIMM downloads/updates all available genomes of a particular interest group and creates a corresponding SLIMM database containing names and taxonomic assignments of the selected genomes. A read mapper is then used to map reads to these reference sequences. Then SLIMM algorithm uses the mapping results to produces taxonomic profile reports.

select_refs.py allows users to configure their reference set based on their interest. Users have the option to select which group(s) of organisms (i.e., Bacteria, Archaea, Viroids, and fungi) to include in their reference set. They have additional option to include individual species by providing a list of taxonomic ids making it convenient to include host genomes or possible contaminants in the reference set. Users can also choose between RefSeq and GenBank databases. They can decide to use only complete genomes or consider genomes with any level of assembly. There is also an option to consider only a single genome per species in case one is interested in doing species level assembly. In the end, the script produces a list of genomes and their corresponding download URL accompanied by different metadata.

select_refs.py enables users to compile a non-redundant reference database through multiple configurable options. By adjusting the parameters to this scripts, it is possible to cover a large number of species without making the cumulative size of reference genomes too high. One can also make a compromise between covering a wide range of species and the quality (level of assembly) of genomes to consider. For more information see the command line options of this script listed in appendix A.1.

collect_refs.py takes the list produced by *select_refs.py* and download the genomes into local directory. We implemented a mechanism to avoid downloading the same file multiple times by providing an option to provide a local buffer directory that contains already download files. For each entry in the list, *collect_refs.py* checks this buffer directory for a presence of the corresponding file before attempting to download it. If it exists, it simply creates a symbolic link in the output directory effectively optimizing disc and network usage. This feature is attractive when one needs to create multiple sets with overlapping reference content. For more information see appendix A.2.

merge_refs.py takes the individual FASTA files produced by *collect_refs.py* and merge them in a specific way. The individual files are representative genomes of a single taxonomic unit, but they contain multiple FASTA entries due to incomplete assembly or plasmids. *Merge_refs.py* excludes all plasmids and merge the remaining FASTA entries coming from one file by adding a sequence of N's between them. This procedure avoids meaningless read mappings at the merging point of two contigs. The final result is a single multi FASTA file containing multiple entries equal to the number of files as well as the number of genomes in the set. The exhaustive lilts of options for this script are shown in appendix A.3.

SLIMM database builder (slimm_build) is an auxiliary program that takes a FASTA file and the NCBI taxonomic mapping files, namely *names.dmp* and *nodes.dmp* available at ftp://ftp.ncbi.nih.gov/pub/taxonomy, to create a SLIMM database specific to the list of organisms represented by genomes inside the FASTA file. Since we only generate taxonomic reports for a one of the seven major taxonomic ranks namely superkingdom (domain), phylum, class, order, family, genus, and species, we purged the taxonomic tree from intermediate ranks and kept only these taxonomic ranks. The SLIMM database contains 1. the mapping from reference genomes accessions and the associated

taxonomic identifier 2. the taxonomic lineage from a taxonomic identifier all the way to the rank of superkingdom (domain). The length of the linage is always seven. This reduction saves a significant amount of computational time as assigning a read to its LCA is computationally expensive. The resulting SLIMM database serves as a second important input for the main SLIMM program.

5.2.2. Read-Mapping

As it is discussed before, SLIMM's primary input is an alignment/mapping file in SAM or BAM format containing a list of sequencing reads and their mapping location within one or multiple genomes. In order to obtain such a SAM or BAM input file, one has to perform read-mapping using a read mapper of choice. A read mapper takes a set of short sequencing reads (queries), does an approximate search in one or more reference sequences (subject) and report the mapping locations for each read. In our case, query sequences are short metagenome shotgun sequencing reads, and the subject is a library of reference genomes representing different microorganisms. Such a library can be assembled using SLIMMs' preprocessing scripts. Once the reference library is ready, it has to be indexed by the indexing module of the read mapper of choice. There exist read mappers that can use the FASTA file directly. However, such read mappers are not suited for large databases as they will be extremely slow. After performing the standard quality control on WGS metagenomic reads, the resulting filtered reads will be mapped against the indexed reference genomes using the chosen read-mapping program.

The Read-mapping can be done using a read mapper of choice. Nevertheless, SLIMMs' overall pipeline could benefit from a faster but yet accurate read mapper as this pre-processing step is relatively time-consuming. It is crucial to allow the read-mapping program output secondary alignments because 1) it is very likely to have a sequencing read mapped to multiple targets, 2) a read might have multiple best hits and 3) the best hit of a read might not be its true origin. It is up to SLIMM to resolve this ambiguity using coverage landscape information as shown in Figure 5.3.

In our preliminary experiments, we tried Bowtie 2 [48] and Yara [54] because they are fast read mappers with multi-threading options. Since the Yara read mapper is several times faster, does not employ heuristics and its resulting alignments produced better profiles in some of the cases, we used it as the default mapper for this study.

5.3. SLIMM Algorithm

As introduced in the previous section, the SLIMM algorithm utilizes a read alignment file in SAM/BAM format. Then it parses and processes the alignment records in the

input file to produce a taxonomic profile after performing major operations described in figure 5.2 sequentially. The following subsection describes each of the individual steps in detail.

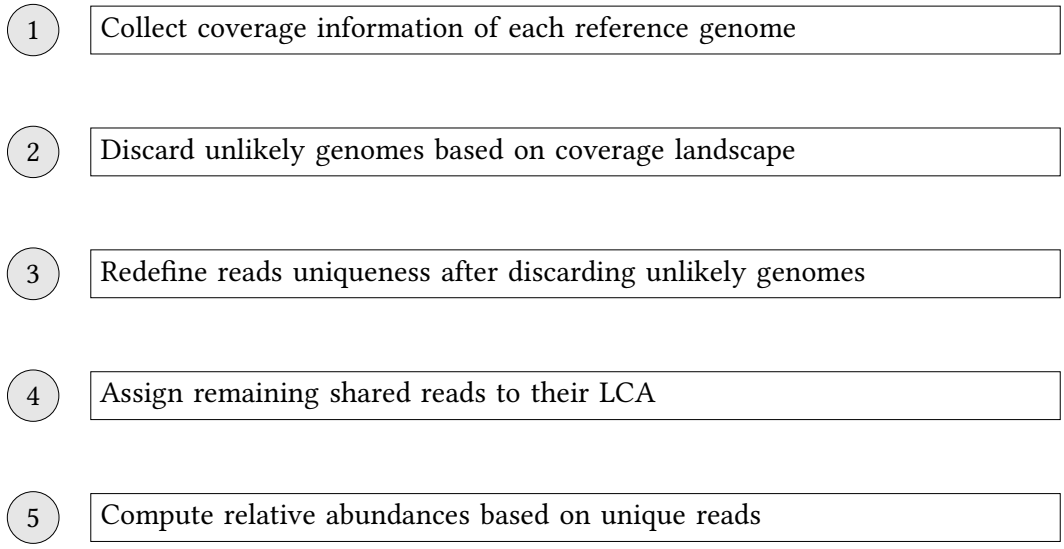


Figure 5.2.: The main algorithmic steps of SLIMM. SLIMM discards spurious genomes based on coverage landscape information collected in the form of read coverage depth. Then read uniqueness is recalculated considering freed reads.

5.3.1. Collect coverage information of genomes

After parsing the SAM/BAM input files, we identify which reads are mapped to which reference genomes. Then we separate the reads uniquely assigned to a single reference sequence from those assigned to multiple reference sequences. We consider reads that are mapped to multiple locations within a reference as uniquely mapped. During this stage, SLIMM collects information like the number of reference genomes with mapping reads, the total number of reads and the average read length, which It latter uses to discard reference genomes. We then map reads into bins of specific width across each reference genome based on the location of their mapping. The binning is done twice, once for mapped reads in general and once only for uniquely mapped reads. The user has the option to set the width of a bin. However, SLIMM uses the average length of sequencing reads as a default value for bin width. Higher bin width implies fewer bins across genomes and faster runtime, but it could lead to underrepresentation of coverage information which in turn is based on whether a bin is empty or not. The bin number corresponding to a read mapped to a reference given by the central position of its mapping location divided by the width of the bins (integral part only).

Formally, The bin number i of a read mapped to a reference genome starting from loci L_s all the way to loci L_e is given by:

$$i = \left\lfloor \frac{L_s + L_e}{2 \times w} \right\rfloor \tag{5.1}$$

Where w is the bin width used to partition references.

All mapped reads will be binned to a single bin using equation 5.1 including those reads that lie on bin borders. This process is done twice, first using all sequencing reads and then using only uniquely mapping sequencing reads. After binning is completed the next step is to calculate genome coverage C as the percentage of the genome length covered by non zero bins according to equation 5.2. We also compute the average coverage depth D as a function of the sum of read length within a bin divided by the width of a bin as shown in equation 5.3.

$$C = \frac{\left|B^1\right|}{|B|} \times 100 \tag{5.2}$$

$$D = \frac{\sum_{i=1}^{|B|} \sum_{j=1}^{n_i} \left|R_{ij}\right|}{|B|} \tag{5.3}$$

Where $\left|B^1\right|$ is the number of bins with 1 or more reads, $|B|$ is the total number of bins in the reference, n_i is the number of reads in bin i and $\left|R_{ij}\right|$ is the length of read in nucleotides bases.

5.3.2. Discard unlikely genomes based on coverage landscape

The main difference between SLIMM and other taxonomic profilers is that, SLIMM utilizes the coverage information of reference genomes to exclude false positive hits mainly due to homologous regions among reference genomes. Figure 5.3 depicts how SLIMM decides to exclude a reference genome with mapping reads from consideration. For instance, it is clear that reference genomes $G3$ and $G4$ are closely related and share homologous regions. They are identical except for bins 3 and 5. When we bin-map sequencing reads against genomes, we see that both genomes have similar coverage by shared reads except in bin 3 and bin 5 exactly where the two genomes differ. Moreover, $G4$ is covered by unique reads around bins where $G3$ is missing any mapped reads. This scenario indicates that $G3$ is indeed a spurious hit and got all of those mapped reads due to its homology with $G4$. When it comes to $G2$, it has poor coverage both by shared and unique reads making it another false hit.

Practically we use a percentile based threshold to discard reference sequences with low coverage percentages. The threshold is calculated based on a user-defined percentile (default 0.001) of all coverage percentages of the genomes. In other words, after sorting the reference sequences based on their coverage percentages in descending order we take the top N sequences that cover 99.999% of the sum of all coverage percentages. This step is done for both coverage percentage by reads that mapped on multiple references and uniquely mapped reads. This process eliminates many genomes even if they have a lot of reads mapping to them as long as they do not have a good enough coverage. Furthermore, this method was also proven to eliminate reference sequences that acquire a stack of reads only in one or two bins across their genomes which could be a result of either a sequencing artifact or homologous region in the genome among distant relatives (*G*3 in figure 5.3).

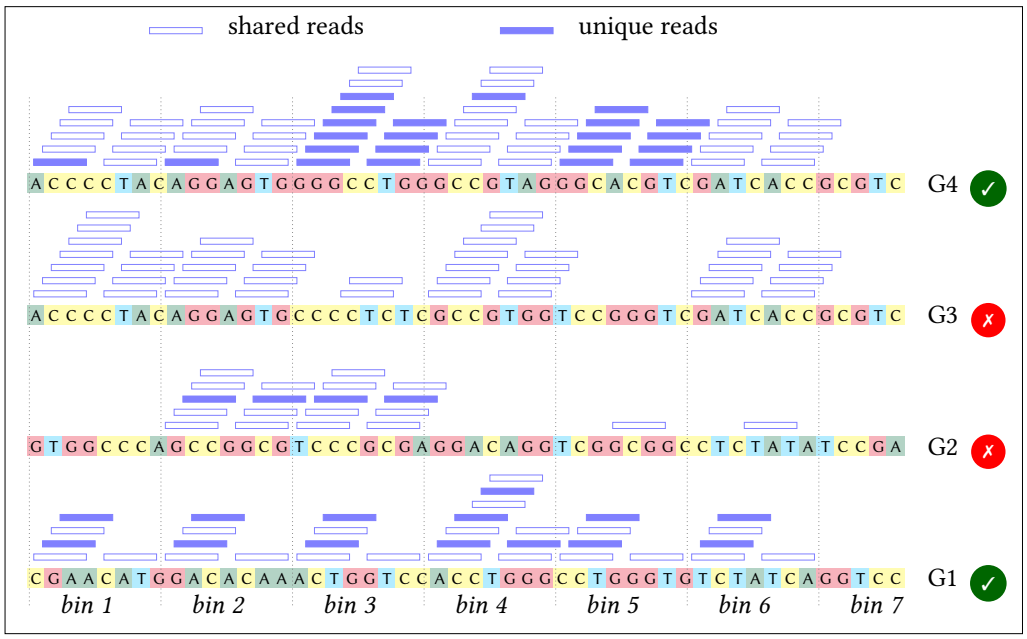


Figure 5.3.: A simplified illustration of how SLIMM uses reference filtering based on coverage information: G2 and G3 could not pass the filtering steps because they did not contain enough coverage by uniquely mapped reads and all reads respectively.

5.3.3. Redefine reads uniqueness

We have described how we excluded genomes that have sequencing reads mapped to them but are unlikely to be the source of those mapped reads. Once we have done that we can redefine the uniqueness of each reads within the scope of the remaining genomes. For example, all the shared reads mapping to *G*3 in figure 5.3 will be free once we exclude *G*3 and will probably be uniquely mapping to *G*4. Depending on

the number of genomes we manage to exclude, our total number of unique reads will increase significantly. This reevaluation of reads uniqueness increases the number of uniquely mapped reads assigned to lower-level clades in a taxonomic tree and is shown to enhance the abundance estimation of reported clades.

5.3.4. Assign remaining shared reads to their LCA

After discarding unlikely reference genomes and redefining read uniqueness, there will be sequencing reads that are still shared among the remaining reference genomes. SLIMM assigns these shared reads to the LCA of the reference genomes that are sharing them. LCAs are computed based on the NCBI taxonomic tree contained in the SLIMM database generated at the pre-processing stage. Note that this process could result in a higher number of reads assigned to clades of a rank compared to clades of a lower rank.

5.3.5. Compute relative abundances based on unique reads

The last step of the SLIMM algorithm is computing the relative abundance of each clade (taxonomic unit) at the desired rank. The relative abundance of a clade at a rank $A(c, r)$ is defined as the number of reads that are assigned to that clade N_c divided by the total number of reads assigned to any clade at the same rank N_r, i.e.,

$$A(c,\ r) = \frac{N_c}{N_r} \tag{5.4}$$

Additionally, we report extra information alongside with the relative abundance such as an aggregated coverage depth of each clade defined as in equation 5.5, the original number of reads assigned to a reference (shared and unique) and the revised number of unique reads after the exclusion of unlikely genomes.

$$D(c) = \frac{\sum_{i=1}^{N_c} |R_i|}{\sum_{j=1}^{|M_c|} \left|G_j\right|} \tag{5.5}$$

Where $D(c)$ is coverage depth of a clade, N_c is the number of reads assigned to a clade, $|R_i|$ is the length of the i^{th} read in nucleotide bases, $|M_c|$ is the number of children genomes under clade c that contribute at least one read and $\left|G_j\right|$ is the length of the j^{th} genome in nucleotide bases.

5.4. SLIMM Application

In a study investigating the sudden outbreaks of Clostridium difficile infection (CDI) in newly born piglets, we applied SLIMM to investigate the microbial composition of different groups of piglets. Fecal samples were isolated and used for metagenome analysis. The fecal samples were from suckling piglets, formula-fed piglets, and formula-fed and C. difficile–infected piglets. After generating WGS libraries and sequencing them using the Illumina NextSeq500 system. Flexbar [96] was used to remove adapter sequences, trim low-quality bases and subsequently remove reads shorter than 100 bases long. Reads that passed the quality filter were mapped against a set of reference genomes using the Yara read mapper. We used SLIMM to analyze the resulting mapping files and produce taxonomic profiles. The taxonomic profiles revealed differences in diversity, evenness and number of organisms among different groups of piglets. The results were included in [97].

6. SLIMM Evaluation

Evaluating a computational method involves both the assessment of computational performance and accuracy of produced results. Computational performance covers the amount of time and resources (RAM and disk space) needed to finish a particular task. Using accuracy as an evaluation criterion is applicable where the methods in question fail to produce complete or exact results in a deterministic way. The reason could be the complexity of the problem, the absence of sufficient data, or lack of resources (computational). Taxonomic profiling methods are among such methods. In the previous chapter, we gave a detail description of SLIMM, a novel taxonomic profiling method we developed. In this chapter, we will assess SLIMM's performance by comparing it with existing methods.

6.1. Benchmarked Methods

We picked Kraken, GOTTCHA, and mOTUs for benchmarking. The reason for choosing these existing methods are 1) their popularity among the bioinformatics community, 2) the novelty of their approach and 3) their claimed performance compared to their predecessors at the time of development. We refer the reader to section 4.5 of chapter 4 for a description of the considered methods.

6.2. Datasets

6.2.1. Reference Set

Two different sets of reference genomes were downloaded from NCBI GenBank (ftp://ftp.ncbi.nlm.nih.gov/genomes/genbank/) using the SLIMM preprocessing module on the date 21.05.2016. The first set contains complete reference genomes available at the download time under the domains archaea and bacteria. We called this reference set *small_DB*. It contains 4915 genomes covering 2163 different species of bacteria and archaea. The second set of references (*large_DB*) aims at covering a large number of different species. We achieved this by configuring the SLIMM preprocessing module to

compile a single reference per species while considering draft (not complete) genomes as well. *Large_DB* contains 13192 reference genomes and covers the same number of different species.

As it is described in chapter 5, we select one reference genome per species in the following order. 1) reference genome, 2) representative genome, 3) complete genome, 4) chromosome level assembly, 5) scaffold level assembly and at last 6) contig level assembly. Whenever more than one assembles are present in a selected category, we consider merely the latest one. After experimenting on multiple different read-sets, the *large_DB* was proven to produce better quality taxonomic profiles at the species level. Therefore we used the *large_DB* as reference database in all of our benchmarks.

6.2.2. Metagenomic Reads

In order to evaluate the accuracy of SLIMM and compare it to similar methods, we assembled 18 different metagenomic read-sets from a wide variety of sources and simulation strategies. We firmly believe that evaluating methods with a heterogeneous mix of multiple read-sets is necessary to prove their robustness. Moreover, by including read-sets which are not produced by us in the evaluation process, we assure the fairness of the evaluation process.

The first type of read-sets were mock community metagenomes synthetically produced in laboratories and later sequenced using the Illumina Genome Analyzer II. We considered three metagenomic read-set of this type; the first two are obtained from the Human Microbiome Project (HMP) [25] containing genomes of 22 microorganisms. The two read-sets from HMP are similar in the species they contain. They only differ in the abundance distribution. One contains an even abundance distribution of the microorganisms whereas the other contains a differing abundance distribution of the 22 microorganisms. The third and last read-sets in this category came from the study [98] where they produced a synthetic community of bacteria and archaea with 64 member species.

The second type of read-sets considered are simulated metagenomes that resemble community profile of a real metagenome as identified by MetaPhlAn2 [79] a popular metagenomic profiling tool based on clade-specific marker genes. We took the reported taxonomic profile and used it as a basis for the simulation. This way we can ensure that our simulated metagenomic read-set is a partial reflection of the natural world. With this approach we simulated two metagenomes using a human gut and a freshwater metagenome obtained from the human microbiome project [79] and the Lake Lanier study [14] respectively.

	Read-sets	Number of Reads	Read Length	Paired	Species Count	Abundance	Source
Mock	MG01	109.63 M	75	Yes	64	EV	[98] - SRR606249
	MG02	6.56 M	75	No	21	EV	HMP [25] - SRR172903
	MG03	7.93 M	75	No	21	ST	HMP [25] - SRR172902
Mimic.Sim	MG04	18.38 M	100	Yes	40	ST	HMP ([79])
	MG05	18.38 M	100	Yes	134	ST	Lake Lanier study ([14])
Random.Sim	MG06	18.38 M	100	Yes	50	EV	Randomly simulated reads with 3 different abundance distributions and 3 different number of member organisms.
	MG07	18.38 M	100	Yes	50	ST	
	MG08	18.38 M	100	Yes	50	ST	
	MG09	18.38 M	100	Yes	200	EV	
	MG10	18.38 M	100	Yes	200	ST	
	MG11	18.38 M	100	Yes	200	ST	
	MG12	18.38 M	100	Yes	500	EV	
	MG13	18.38 M	100	Yes	500	ST	
	MG14	18.38 M	100	Yes	500	ST	
CAMI	MG15	7.44 M	100	Yes	199	ST	Medium complexity CAMI challenge toy read-sets which are publicly available at https://data.cami-challenge.org/participate.
	MG16	7.43 M	100	Yes	199	ST	
	MG17	149.14 M	100	Yes	199	ST	
	MG18	149.03 M	100	Yes	199	ST	

Table 6.1.: List of read-sets and their primary properties used in the evaluation process of SLIMM against other existing methods

In the third category, we simulated randomly created metagenomic communities of a diverse number of member organisms and abundance compositions using the NeSSM [99] simulation program. We considered three communities with randomly selected member organisms. The number of organisms in these communities is 50, 200 and 500. We then chose three different ranges of relative abundances, i.e., even, [1-100] and [1-1000]. Doing so provided us with a total of 9 randomly created metagenomes with varying complexity both regarding diversity and in abundance differences. The different settings of metagenomic read-sets are essential to make sure that the tested methods work with a broad range of inputs. To resemble an actual metagenome and to make the taxonomic profiling more difficult, we contaminated all the simulated read-sets with real-world metagenomic reads sequenced by Illumina MiSeq. We first removed the reads that could be mapped to any of the prokaryotic genomes in our database.

The last category of read-sets contains the Medium complexity CAMI (The Critical Assessment of Metagenome Interpretation) challenge toy read-sets that are publicly available at https://data.cami-challenge.org/participate.

Details of all the read-sets used for evaluation can be found in Table 6.1. The table summarizes the metagenomic read-sets and their key properties used for the evaluation process. We believe that, this collection is representative enough for most of the metagenomic communities that a taxonomic identifier will have to handle.

	Alignment + SLIMM	Kraken	GOTTCHA	mOTUs
Avg. Runtime (Seconds)	422.1 + 61.0	157.4	1727.1	1526.6
Peak Memory (GB)	33.67 + 5.2	102	4	1.6

Table 6.2.: Average Runtime and Memory Comparison of SLIMM against existing methods

6.3. Computational Performance

6.3.1. Infrastructure and Parameters

We used a computer equipped with an Intel(R) Xeon(R) CPU 3.30GHz processor (32 cores) and 378GB of RAM to evaluate the computational performance of all the tools. Run time and peak memory usage were averaged over all the read-sets, excluding the CAMI read-sets. The CAMI read-sets are not included in the runtime and memory comparison. That is because we could not run Kraken with large_DB on the same machine since it required 500GB of memory. Instead, we run Kraken on a cluster for these particular read-sets. The command line scripts used to run the individual methods considered for benchmarking are listed in appendix A.4.

6.3.2. Runtime and Memory Footprint

Table 6.2 shows the average runtime and the average peak memory usage of the tools across runs on 14 out of the 18 different read-sets (excluding the CAMI read-set for the reason mentioned in the previous paragraph). Without the time needed for generating the required SAM files SLIMM is proven to be faster than any of the other tools considered while using a fair amount of memory footprint. With the preprocessing, Kraken is faster than SLIMM but with a much more memory footprint. SLIMM is faster than GOTTCHA and mOTUs. Appendix A.5 shows the individual runtime of each tool across 14 different read-sets.

6.4. Accuracy Evaluation

6.4.1. List of Reported Organisms

After obtaining the taxonomic profiles reported by each tool for each read-sets, we computed three different accuracy measures namely precision (specificity), recall (sensitivity) and F1-Score as an assessment of quality on the profile produced. The comparison between generated profiles by individual tools against the known composition of either a mock community or a simulated read-set is the basis for computing accuracy measures. We defined each of the measurements as defined in equations 6.1 - 6.3. The values of the confusion matrix used in the equations are interpreted as it is elaborated in Figure 6.1.

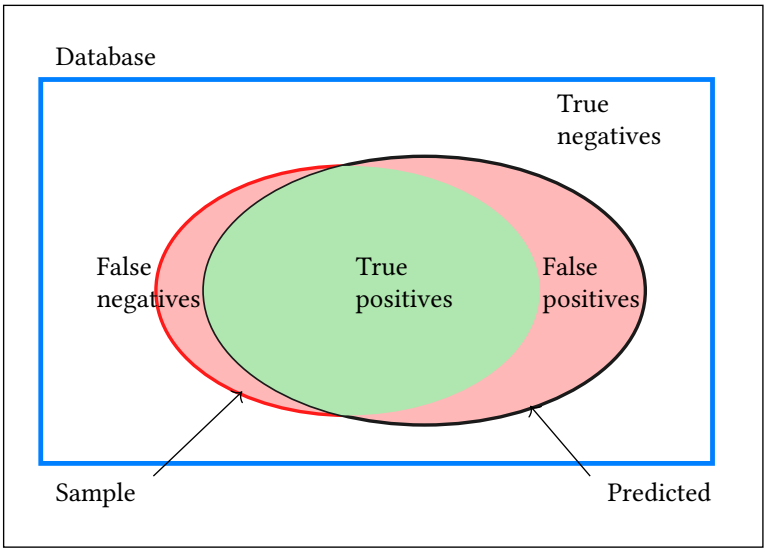


Figure 6.1.: Elaboration of the confusion matrix values (i.e., TP, FP, TN FN) used for evaluating the accuracy of taxonomic profile methods.

$$precision = \frac{TP}{TP + FP} \tag{6.1}$$

$$recall = \frac{TP}{TP + FN} \tag{6.2}$$

$$F1\ Score = \frac{2 \cdot (precision \times recall)}{precision + recall} \tag{6.3}$$

		Precision				Recall				F1			
	Read-set	G OTTCHA	m OTUs	K raken	S LIMM	G OTTCHA	m OTUs	K raken	S LIMM	G OTTCHA	m OTUs	K raken	S LIMM
Mock	MG01	0.9808	1.0000	0.6264	0.8923	0.8226	0.8065	0.9194	0.9355	0.8947	0.8929	0.7451	0.9134
	MG02	1.0000	1.0000	0.8400	0.9545	0.9524	0.8571	1.0000	1.0000	0.9756	0.9231	0.9130	0.9767
	MG03	1.0000	1.0000	0.6897	0.9524	0.8571	0.4286	0.9524	0.9524	0.9231	0.6000	0.8000	0.9524
Mimic.Sim	MG04	0.6000	0.9474	0.4250	1.0000	0.6176	0.5294	1.0000	1.0000	0.6087	0.6792	0.5965	1.0000
	MG05	0.8714	0.9630	0.6650	1.0000	0.4656	0.1985	1.0000	1.0000	0.6070	0.3291	0.7988	1.0000
Random.Sim	MG06	0.6897	0.8718	0.4352	0.9783	0.8333	0.7083	0.9792	0.9375	0.7547	0.7816	0.6026	0.9574
	MG07	0.6964	0.9091	0.4352	0.9783	0.8125	0.6250	0.9792	0.9375	0.7500	0.7407	0.6026	0.9574
	MG08	0.7143	0.8824	0.4299	0.9783	0.8333	0.6250	0.9583	0.9375	0.7692	0.7317	0.5935	0.9574
	MG09	0.8396	0.9286	0.7220	0.9929	0.5855	0.3421	0.9737	0.9211	0.6899	0.5000	0.8291	0.9556
	MG10	0.7949	0.9574	0.7178	0.9930	0.4079	0.2961	0.9539	0.9276	0.5391	0.4523	0.8192	0.9592
	MG11	0.8058	0.9464	0.7164	0.9928	0.5461	0.3487	0.9474	0.9079	0.6510	0.5096	0.8159	0.9485
	MG12	0.7333	0.9773	0.8284	0.9855	0.0377	0.1473	0.9589	0.9315	0.0717	0.2560	0.8889	0.9577
	MG13	0.8095	0.9811	0.8237	0.9855	0.0582	0.1781	0.9281	0.9315	0.1086	0.3014	0.8728	0.9577
	MG14	0.8000	0.9811	0.9857	0.9851	0.0548	0.1781	0.9452	0.9041	0.1026	0.3014	0.9650	0.9429
CAMI	MG15	0.7397	0.8000	0.7644	0.9261	0.2714*	0.1206*	0.7990	0.8191	0.3971*	0.2096	0.7813	0.8693
	MG16	0.6883	0.8462	0.7027	0.8377	0.2663*	0.1106*	0.7839	0.8040	0.3841*	0.1956	0.7411	0.8205
	MG17	0.4531	0.7368	0.7608	0.9302	0.1457*	0.1407*	0.7990	0.8040	0.2205*	0.2363	0.7794	0.8625
	MG18	0.4839	0.7778	0.6996	0.8223	0.1508*	0.1407*	0.7839	0.8141	0.2299*	0.2383	0.7393	0.8182

Table 6.3.: Comparison of SLIMM against different tools regarding precision and recall on species-level: The highest values in each row are highlighted in strong green for both precision and recall. *GOTTCHA and mOTUs have unfairly lower recall and F1 values due to their database which does not contain the complete set of references for the corresponding read-sets

Where, TP is the number of species which are originally in the sample and called by the tools, whereas FP is the number of species which are not initially in the samples but are yet reported by the tools. Similarly, FN are the number of species which are initially in the sample but got missed by the tools.

Table 6.3 shows the results on accuracy of taxonomic profiles reported by different metagenomic classifiers including SLIMM. The table lists the precision, recall, and F1-Score of the method on 18 different read-sets. SLIMM showed a superior performance regarding precision over the other tools (13 of the 18 cases). SLIMM and Kraken showed comparable and good results in the recall category. SLIMM is second place exceeding Kraken occasionally. However, Kraken produced a higher number of false positives to attain this recall, hence the lower numbers in precision. GOTTCHA performed well with the HMP read-sets while it underperformed in the rest of the read-sets in general. mOTUs does not perform well in all of the read-sets. We provided F1-Score in the table as a measure of the right balance between precision and recall. SLIMM outperforms all the other tools both in precision and F1-Score in 17 of the 18 cases while Kraken is slightly better in recall for the majority of the cases.

We performed a PR curve analysis to assess the robustness of methods to cutoff values. Two read-sets were considered for this purpose, the staggered HMP mock community read-set and one of the CAMI challenge read-sets. We first sorted the list of predicted species by reported abundances in decreasing order. Then, we considered more species from the sorted list and followed the progression of precision and recall to obtain PR

curves in iterative way. The resulting PR curves in Figure 6.2 indicate that SLIMM has a better recall rate than the other tools while staying precise. This is an indication that, SLIMM is more resilient to cutoff values than the competing methods. Interestingly GOTTCHA has not detected any false positive hits for the HMP mock community read-set. Due to that, it stayed precise the whole time but could not achieve the recall level of SLIMM and Kraken as it misses 2 organisms in its report. mOTUs exhibited similar behavior, but it misses more than half of the 22 organisms in the sample.

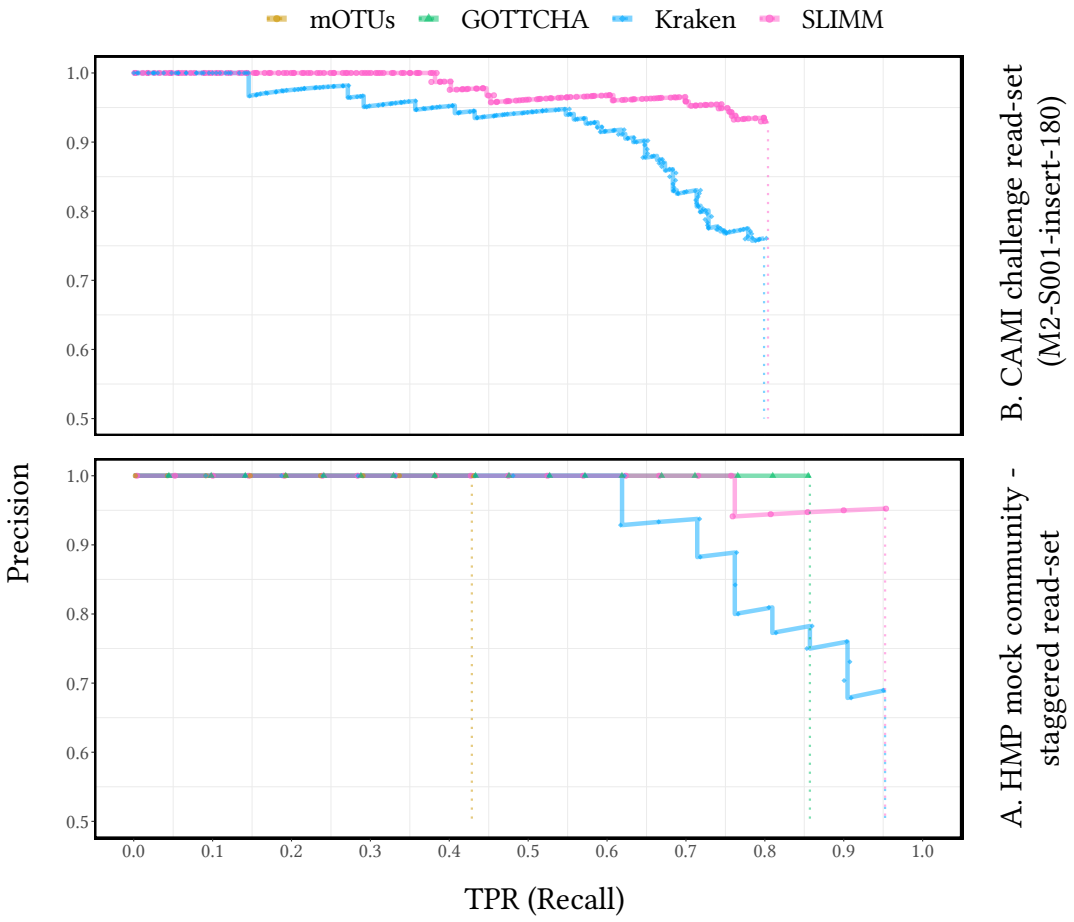


Figure 6.2.: PR curves used to compare SLIMM against existing methods. True positive rate (TPR)/recall is drawn against precision. SLIMM showed the highest performance. GOTTCHA did not discover any false positives but is low in recall.

In a similar experiment, we investigated the effect of our coverage landscape based filtering procedure as well as the influence of read mappers used in the alignment step. We have also performed a digital normalization on the raw input reads before the alignment step following the recommendation by Piro et al.. Figure 6.3, clearly shows that turning off the filtering procedure caused a significant loss in performance. The choice of read mapper also affects the accuracy of taxonomic profiles produced by SLIMM significantly. Mapping results obtained using the Yara read mapper have better precision and recall compared to those produced using Bowtie2. We believe the

sensitivity of the Yara read mapper in reporting all the alignments locations of a read within a user-defined mapping quality is the main reason for this. The effect of digital normalization in improving the quality of taxonomic profiles generated by SLIMM was negligible contrary to the findings by Piro et al..

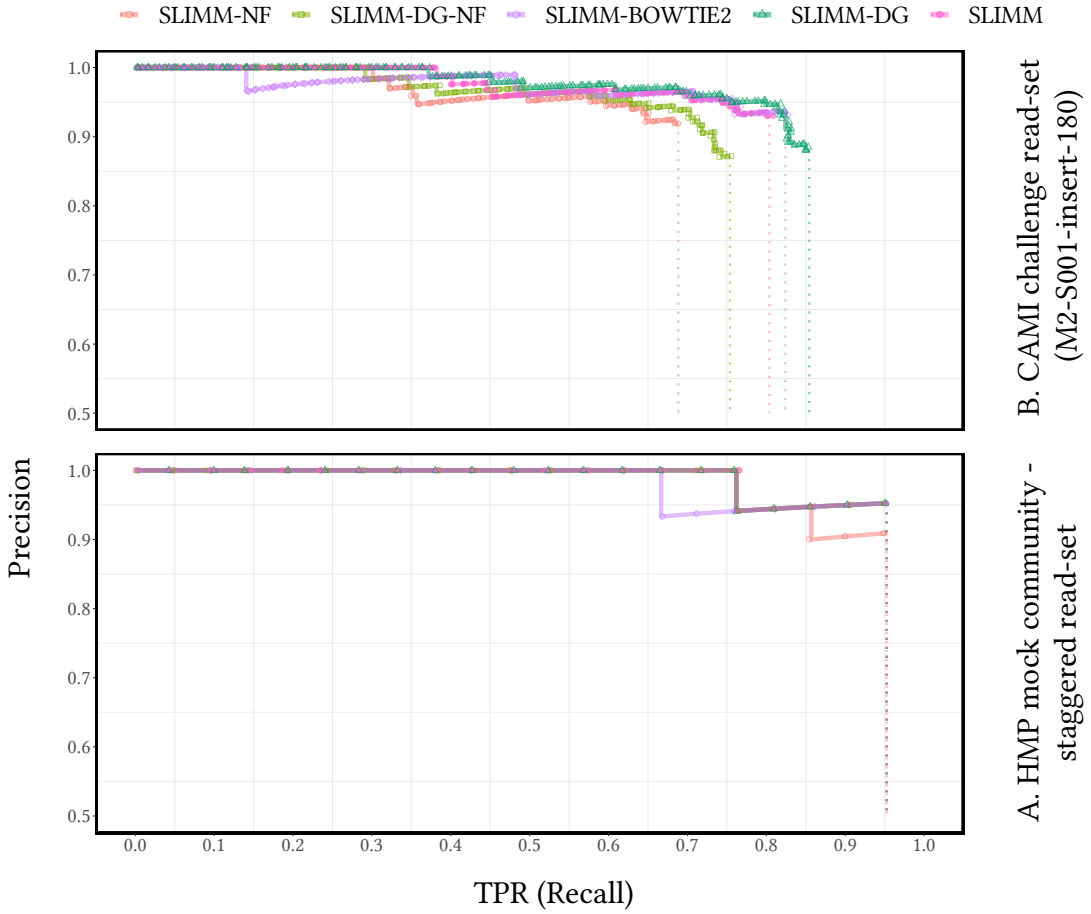


Figure 6.3.: PR curves used to compare different variants of SLIMM. SLIMM-DG (with digital normalization), SLIMM-NF (without filtration), SLIMM-NF-DG (without filtration but with digital normalization) and SLIMM-BOWTIE2 using the Bowtie2 read mapper are included.

6.4.2. Correctness of Abundance

Another crucial aspect of taxonomic profiling is how accurate the abundances assigned to detected organisms. SLIMM's performed better than existing methods by reporting abundances closer to the actual abundances which are the basis of simulation. The scatter plots in Figure 6.4 A) and B), where the true abundance of organisms used for simulation are drawn against their predicted abundances, show that SLIMM's abundances are the closest to the real abundances. The two scatter plots are based

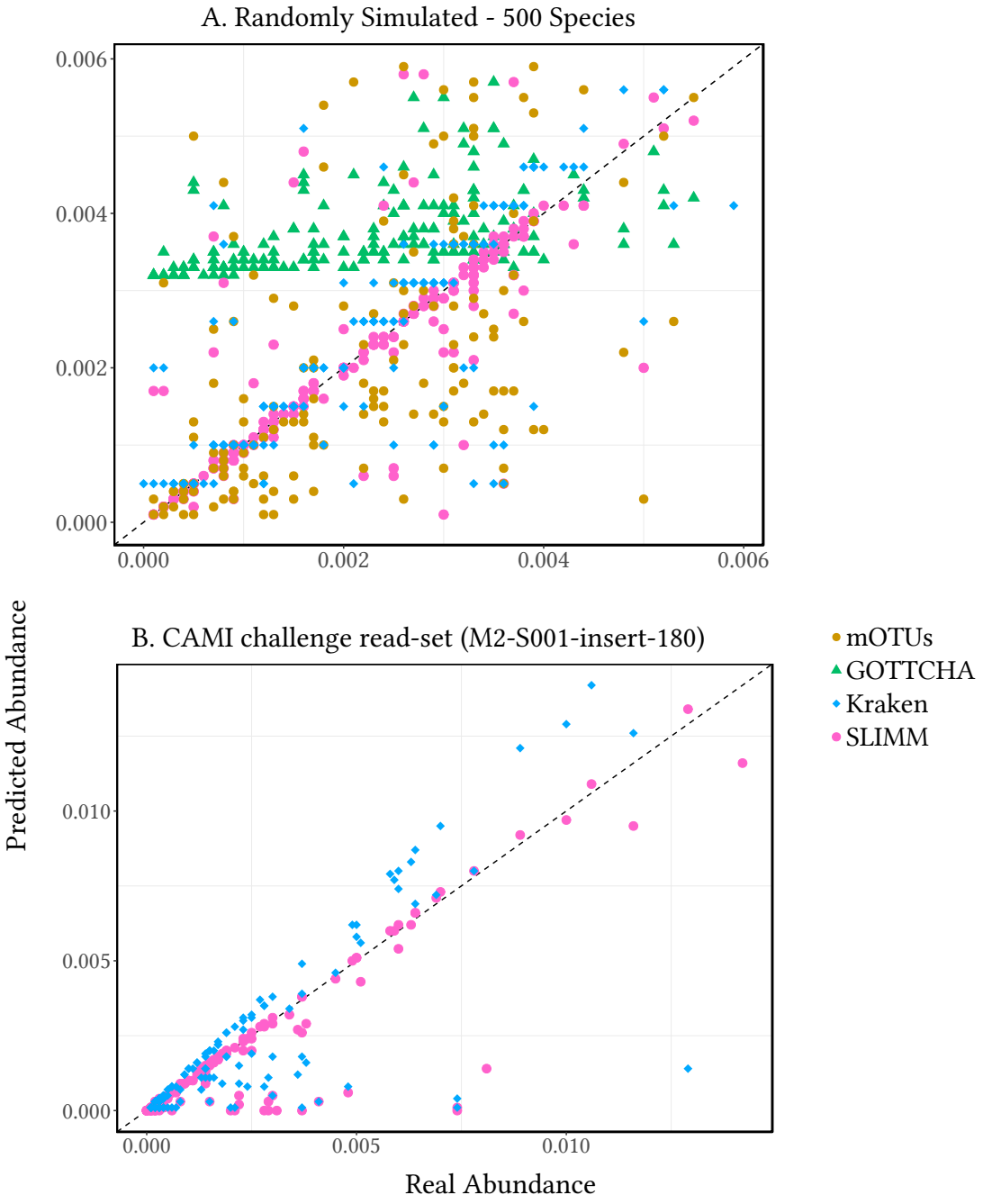


Figure 6.4.: Scatter plots showing the divergence between abundances predicted by different tools and the actual abundances used in simulation. SLIMM predicted the abundances more accurately than the other tools. Kraken overestimates the abundance values. GOTTCHA and mOTUs did not perform well in predicting the correct abundances.

on taxonomic profiles generated by different methods using one of the randomly simulated read-set and one of the CAMI challenge read-set. From these plots, it can be seen that SLIMM predicts the abundance more accurately. Even though it was not originally developed for abundance estimation, the next best tool is Kraken which slightly overestimates the true abundance. mOTUs and GOTTCHA do not perform well at predicting the abundances. Similar plots on more read-sets are provided at A.7.

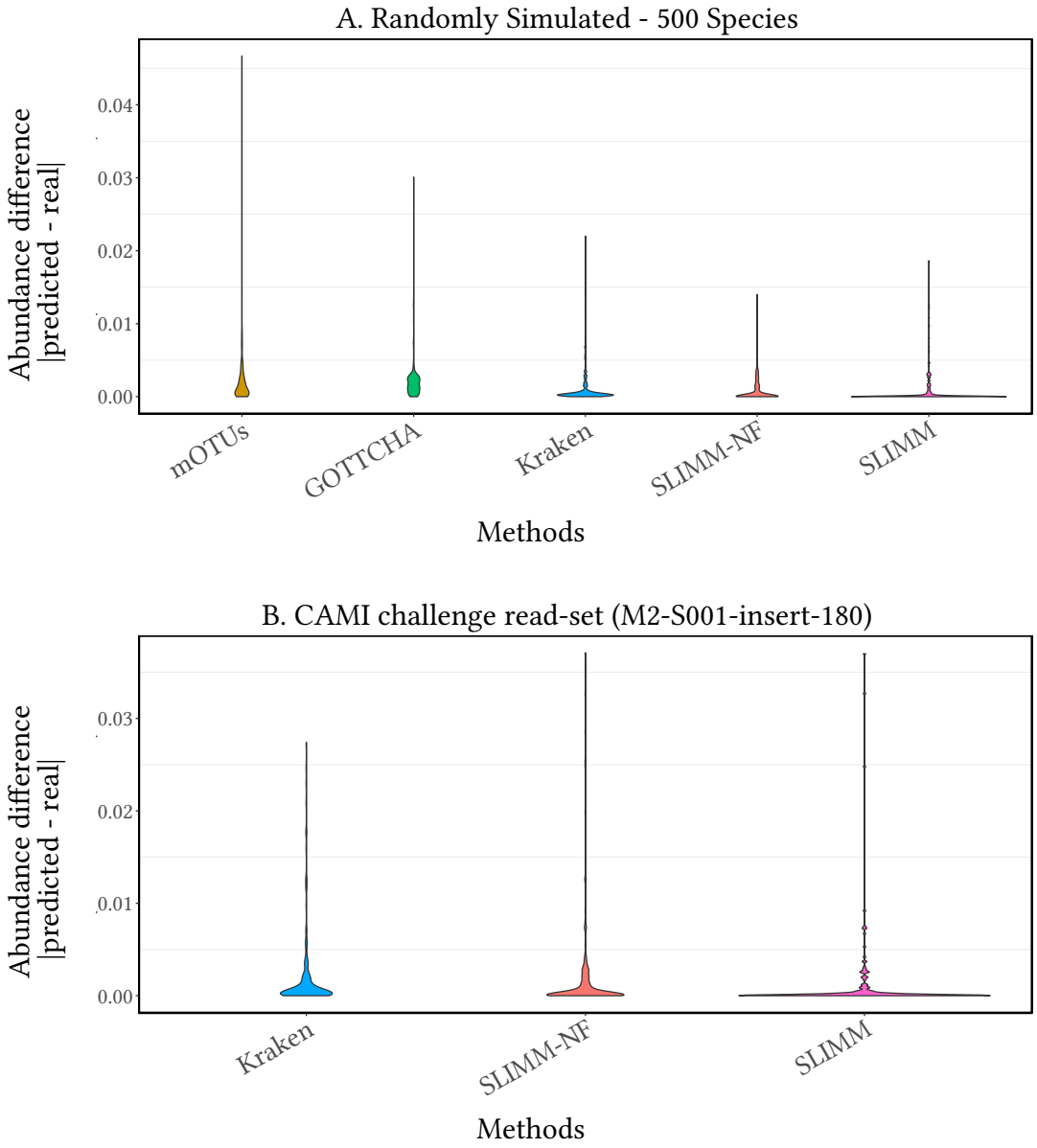


Figure 6.5.: Violin plots showing the divergence of predicted abundances from the actual abundances. SLIMM has the lowest divergence from true abundances. SLIMM's abundances are better with the coverage landscape based filter than without.

Violin plots are similar to box plots, but additionally, they visualize the density distribution of different data points. The violin plots in Figure 6.5 show how good the different

tools predicted the abundances compared to the actual abundances. In these plots, we can see that SLIMM has the lowest divergence from true abundance values. For the randomly simulated read-set, SLIMM has an average absolute difference of 0.00073, and Kraken has an average absolute difference of 0.00116 which is 159% higher compared to SLIMM. For the same read-set, GOTTCHA and mOTUs have an average absolute difference of 0.00206 and 0.00273 respectively. SLIMM also received the most correct (closer) abundances with absolute differences of first quartile (Q1)=0.00002 and third quartile (Q3)=0.00016. Kraken is the second-best tool in this regard with values Q1=0.00018, Q3=0.00065.

Similar to the PR curve analysis, we did a comparison among different configurations of SLIMM by switching off our novel coverage landscape filtering (*SLIMM-NF*), by applying digital normalization prior to read-mapping (*SLIMM-DG*), the combination of the two (*SLIMM-DG-NF*) and using the Bowtie2 read mapper for alignment step (*SLIMM-BOWTIE2*). The filtering step in SLIMM is a crucial part of the method as it is shown in the violin plots of figure 6.5. The scatter plots of predicted vs. real abundances in figure 6.6 show similar evidence about the importance of the filtering module. Other factors such as the read mapper used and the application of digital normalization have a small effect on the correctness of reported abundances. In Appendices (A.6, A.7, and A.8), we provide plots on more read-sets.

6.5. Summary

In this chapter, we presented a detailed evaluation of SLIMM as a taxonomic profiling tool. Using 18 different read-set from different sources and compositions, we evaluated the accuracy of SLIMM and compared it to three other existing methods. The heterogeneity of read-sets and their sources, including third-party sources, is a testimony to how rigorous our benchmark is. The results show SLIMM is a computationally efficient, and accurate taxonomic profiler which produces a precise and yet sensitive list of member organisms of a microbial community. We have also shown the individual relative abundances produced by SLIMM are the closest to the real abundances used for simulation among all the methods considered in this evaluation.

Our detailed investigations on the effect of the new coverage based filter reveal that, the filter is at the core of SLIMM's performance. The results show this filtration module is crucial to create an accurate list of organisms and predict their abundances correctly. We have also noted that using a fully sensitive read mapper such as Yara has a positive effect on the taxonomic profiling process.

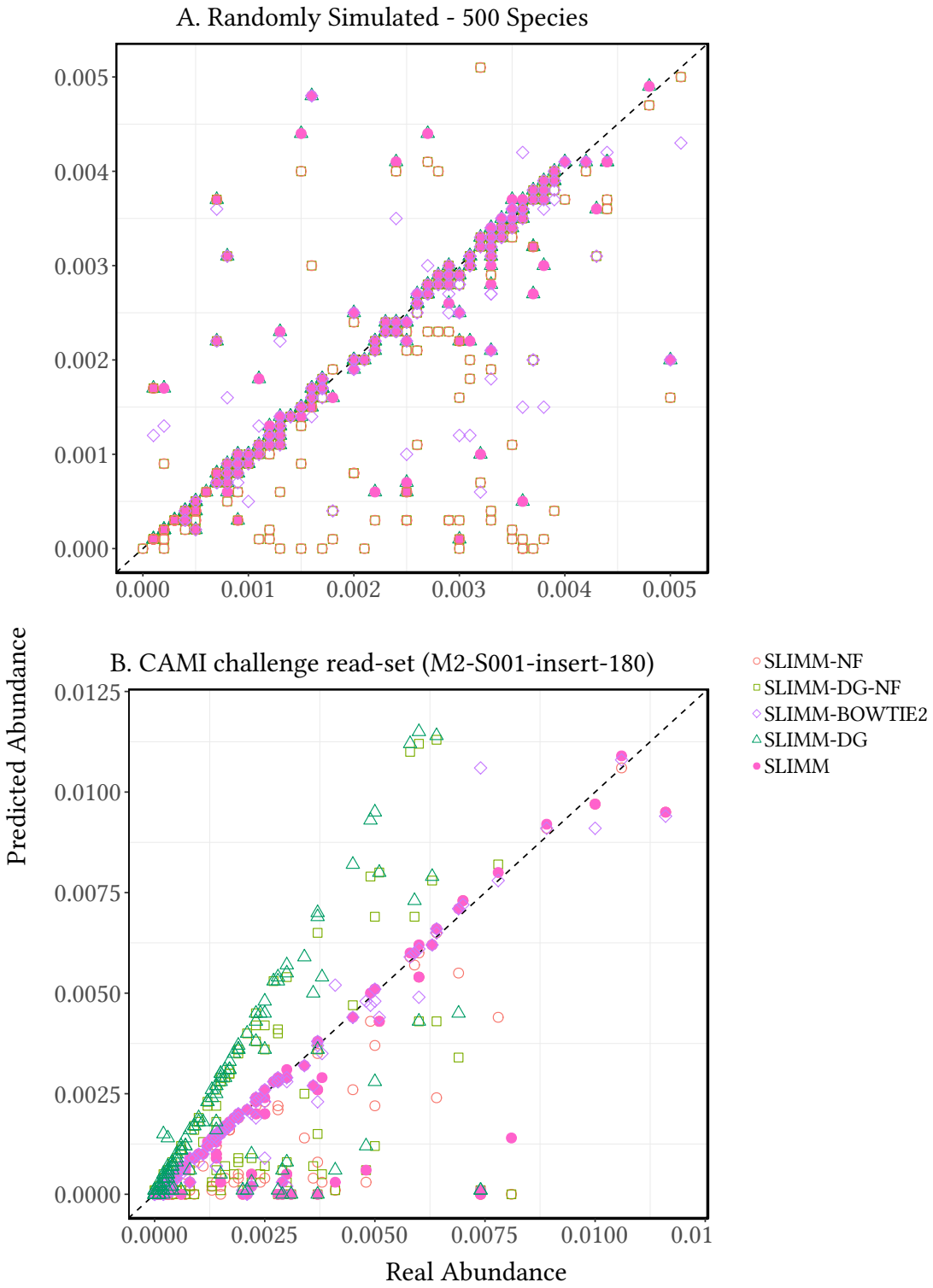


Figure 6.6.: Scatter plots showing the divergence between abundances predicted by by different variants of SLIMM and the actual abundances used in simulation.

Part III.

Read Mapper for Large Databases

7. DREAM-Yara

In this chapter, we propose and explain the DREAM index framework to address the problem posed by relying on a single large index where databases are growing and changing frequently. Then we describe a working implementation of the framework which consists of three significant contributions. The first contribution is a taxonomy based clustering/binning method for a collection of database sequences (e.g. bacterial genomes). Our second contribution is a novel data structure for quickly distributing reads across bins that relies on a combination of Bloom filters [100] and k-mer counting. The last contribution is a distributed, parallel version of the Yara read mapper [101]. Implementation of the IBF is under review for merging in the SeqAn library. DREAM-Yara code and software is available for download at https://github.com/temehi/dream_yara

7.1. Introduction to DREAM Index Framework

Modern sequencing technologies brought a super-exponential growth of sequencing capacities that led to a progressive generation of raw sequencing reads. These reads fuel the continued growth of reference databases via either a de novo or guided assembly of genomes. The raw sequencing reads are also often stored in public databases where they can be re-examined for a hypothesis outside their original experimental scope. However, our focus is on reference genome databases and how to cope with their growing size, mainly regarding making search operations viable.

Public genomic reference databases are expanding in daily bases both in diversity (genomes of new species) and redundancy (re-sequencing species which already have a genome). For example, GenBank's WGS section reference database has become more than 25x bigger in nucleotide bases and 15x bigger in the number of sequences in the past ten years. Even though the number of species with reference genomes has increased, a significant portion of the growth can be attributed to re-sequencing species with an already existing genome. In case of prokaryotes, sequencing different strains of the same species is a common practice. Figure 7.1 shows the annual increase of genome assemblies stored in GenBank. The figure also highlights the number of species which did not have a reference genome before. The plot clearly shows the majority of the new sequences added to the GenBank database are not coming from

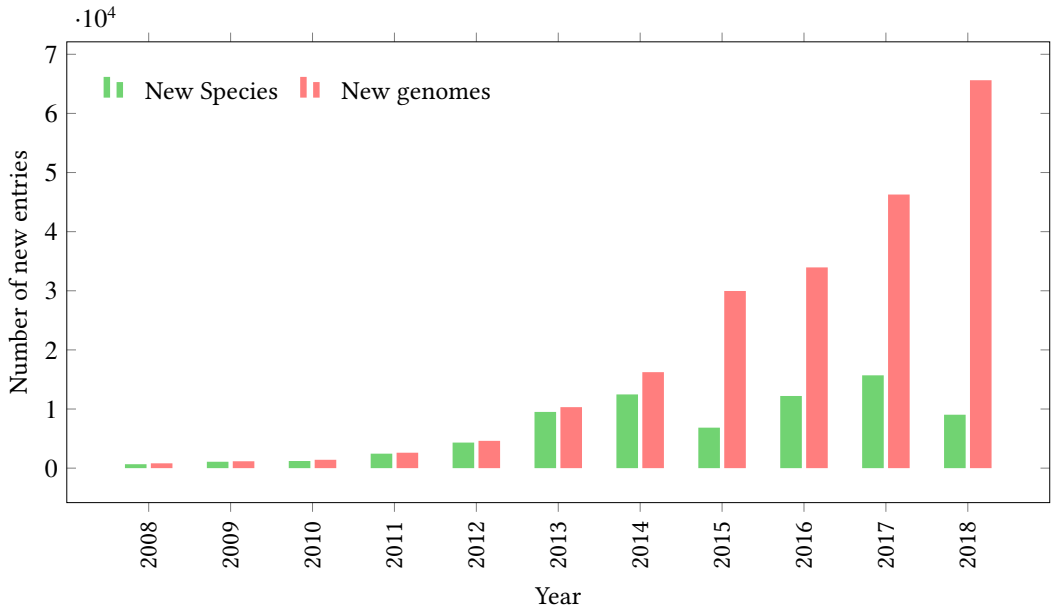


Figure 7.1.: The number of new assemblies (genomes) added to the GenBank database per year in the last decade. The number of new species introduced is getting relatively smaller compared to the total number of genomes added to the database.

species new to the database. This suggests high level of sequence similarity between multiple entries. Note that two genomes of different species can also have a significant similarity depending on their taxonomic relationship.

Searching millions of sequencing reads against reference databases, considering *approximate matches* is ubiquitous in computational biology. The search process is computationally expensive due to the overwhelming number of queries and the size of the database. Traditionally this is undertaken by indexing the reference database using either Burrows-Wheeler-Transform (BWT) based FM-indices (also often referred to as compressed suffix arrays (CSA) [47] or hash-based indexing. Even though it is computationally expensive to build it, such an index reduces the search time dramatically. The cost for indexing amortizes over time due to the static nature of the reference. This assumption does not hold when databases are changing faster than the index building process, because frequent updating of indices is required.

Most researches on indexing large genomic databases depend on a strong assumption that once an index is built it can be used repeatedly, for a long period, without the need for an update. A good example is the human genome (hg38) which is five years old. Popular read mappers can provide an already built index of hg38 to avoid the lengthy process of building an own index by the user. The repeated usage assumption does not hold when we are dealing with a set of multiple reference genomes that are changing daily (e.g., reference genomes of all prokaryotes). In general, the current

state of the art indexing data structures are not designed considering frequent *changes* of the underlying databases.

The main bottleneck comes from the static nature of the data structures used for indexing. The resulting index is often big, non-modular and monolithic data structure, which makes it costly (a) to change small parts of a sequence, or (b) to add or delete complete sequences while maintaining the ability to support fast approximate string searches. For example, in metagenomics, this problem becomes more and more recurrent. Many metagenomics search tools (e.g. [102, 62]) and read mappers [49, 101] use FM-indices [47]) and have to index about 50 to 200 gigabases. Due to constant uploads from the community, the database changes on a daily or weekly basis and thus require a newly constructed index. Recomputing a single index of this size is quite costly regarding space and time, even if approaches of merging BWTs are used [103, 104]. It takes about one day to compute the index for the databases used in our experiments. On the other hand, the ability for fast approximate searches in such an index is crucial. It is used either directly to find all approximate occurrences of a (short) string or parts of it in seed-and-extend approaches.

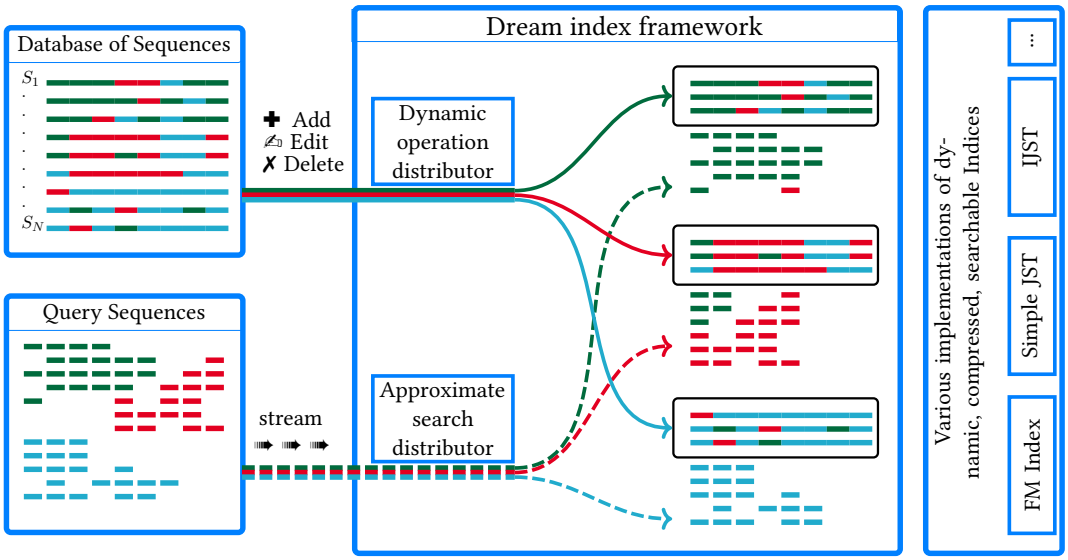


Figure 7.2.: Sketch of the DREAM index framework. The red sequence piece among the green ones symbolizes that we do not require a perfect partitioning allowing us to use fast methods. The boxes on the right symbolize the potential use of different index implementations. Note that we use solely FM-indices in the context of this thesis.

The DREAM index framework offers various solutions for the problems discussed above by introducing modularity through clustering of sequences in a reference database and creating multiple indices on the clusters opposed to one big index. As depicted in 7.2 the DREAM index has two major components, i.e., the *dynamic operations distributor*, which handles the creation of clusters and corresponding indices, and the *approximate*

search distributor, which distributes search queries to a subset of indices with a potential mapping location. The implementation of these two components of the framework can vary depending on some key parameters of the input set (size of the input, amount of redundancy, the importance of rebuilding time vs. search time). In the next sections, we will present a detailed and working implementation of the DREAM index framework with the goal of creating a scalable read mapper that can handle frequently changing reference databases.

In a similar work, Mohamadi et al. presented a method where references are partitioned and indexed separately based on size. Then they used on-the-fly constructed Bloom filters to dispatch, and map reads against the individual partitions. This approach comes short when reference databases are rather large, and there is a need to create many partitions. While investigating this method, we noticed two crucial bottlenecks. First, read dispatching was inefficient, meaning the method mapped almost all reads against all partitions resulting in slower overall mapping time. Besides, the dispatched/-partitioned reads are written to disc creating an additional IO overhead. Second, the merging step to create one alignment result from individual alignment files takes too long.

7.2. Dynamic Operations Distributor

The *dynamic operations distributor* is one of the two main components in the DREAM index framework. It is responsible for creating many small indices given a set of database sequences (figure 7.2). The *dynamic operations distributor* achieves this by creating non-overlapping partitions of the input database. We call these partitions **bins**. The efficiency of the framework is highly dependent on the ability to create "good quality" clusters. A high degree of similarity and redundancy inside each cluster and a clear difference between different clusters is considered as "good quality". Various clustering approaches such k-mer based sequence clustering can be employed to realize the partitioning of large reference databases into smaller parts. The choice of clustering strategy is highly dependent on the nature of the reference databases.

The high level of redundancy, which is common among large reference databases is desirable for clustering. A typical example is the set of prokaryotic references genomes which contains many strains of bacteria belonging to the same species. Reference genomes from the 100,000 genome project [106] are also great examples of large reference sets with a high degree of similarity. For prokaryotic genomes one can use the existing taxonomic classification to make clusters as in [107] or opt for a more complex and taxonomy agnostic k-mer based sequence clustering. However, reference genomes from the 100,000 genome project need a different clustering/partitioning strategy than prokaryotic genomes. For example, if we create bins based on chromosome numbers, that would effectively create 23 bins. This approach can be further extended by cutting

the chromosomes into chunks and put chunks from the same locus into the same bin. Since sequences belonging to the same bin are highly similar, they are naturally amenable to compression techniques (e.g. [108, 109]). On the other hand, compression usually makes it costly to implement the main operations on the data, namely *finding approximate matches* of (many) queries (approximate in the sense of edit distance).

After partitioning the original set of reference sequences into many smaller bins, corresponding indices are created for each bin. The indexing method could vary as long as it supports the desired approximate search that will follow. We address the resulting individual indices as sub-indices, since they cover only a fraction of our original search space. Sub-indices are computationally cheap to build due to the small size of the bins. The *dynamic operations distributor* should also be able to add/remove sequences to/from the collection of sub-indices quickly in a two-step process. The first step is to decide into which bin the sequence will be added or from which it will be removed. We decide by cross-checking the clustering strategy used to create the bins. Then the sub-indices of an affected bin can be rebuilt to reflect the changes. Updating a sequence is equivalent to deleting the original sequence and adding the updated version.

7.3. Dynamic Search Distributor

The second major component of the DREAM index framework is *approximate search distributor*. This layer of the framework deals with the distribution of sequencing reads among sub-indices during an approximate search. By applying a fast filtering strategy, it should avoid searching all sequencing reads against all sub-indices. This kind of distribution significantly reduces the number of search operations and facilitates data parallelization. By doing so, we can compensate for the overhead created by searching multiple small sub-indices compared to searching against a single big index. Another purpose of the *approximate search distributor* is to collect and consolidate the search results from each sub-indices. The consolidation process deals with identifying best matches and valid matches for a read in a global context.

7.4. DREAM Index Framework - Implementation

In this section, we discuss our implementation of DREAM index framework with the goal of creating a full-fledged read mapper with an indexing module that can handle frequent changes in the reference database. We built the distributed read mapper on top of the Yara read mapper which uses a standard FM-index. Many but small FM-indices are used as sub-indices since they support fast approximate queries and give a coarse-grained dynamization by simply rebuilding a sub-index if needed. We used a

k	k-mer space	unique k-mers	unique k-mers / k-mer space
12	$1.68E+07$	$8.39E+06$	0.5001
13	$6.71E+07$	$3.36E+07$	0.5000
14	$2.68E+08$	$1.34E+08$	0.5000
15	$1.07E+09$	$5.32E+08$	0.4951
16	$4.29E+09$	$1.85E+09$	0.4303
17	$1.72E+10$	$4.43E+09$	0.2576
18	$6.87E+10$	$7.18E+09$	0.1046
19	$2.75E+11$	$9.16E+09$	0.0333
20	$1.10E+12$	$1.03E+10$	0.0094

Table 7.1.: The k-mer content of the RefSeq prokaryotic database downloaded on the date 2017-09-26.

novel data structure based on bloom filters in conjunction with the q-gram lemma as an approximate search distributor. In the following subsections, we will discuss the implementation details of different components that make up the *dynamic operations distributor* and the *approximate search distributor* of the DREAM index framework.

7.4.1. Mapping Metagenomic Reads

The need to create a read mapper for metagenomic reads, which can accommodate NCBI's RefSeq database [27] of complete prokaryotic genomes, heavily influenced our implementation of the DREAM index framework. As a result, we started by analyzing the DNA content of RefSeq complete bacterial and archaeal genomes downloaded on the date 2017-09-26. The dataset comprises 15,250 sequences representing 2,991 species, summing up to a total of 31.34 billion nucleotide bases. Since we use a k-mer based pre-filter before read-mapping, we have also analyzed the k-mer content of the database. Table 7.1 shows the number of unique k-mers in the database in comparison with the total number of possible k-mers (k-mer space) for the selected values of k.

We have partitioned the reference database into 64, 256 and 1024 bins using two different clustering strategies: 1) Based on the NCBI taxonomy and 2) Using a k-mer based sequence clustering algorithm. We will discuss both approaches and present the result of our investigation in the following section. We devised a simple metric called *effective binning ratio* (EBR) to measure clustering efficiency as follows. First, we define an effective text size $ETS(T, k)$ of a given sequence database T in the context of its k-mer content as the number of unique k-mers it contains. The EBR of a clustering that creates b bins is given by

$$EBR = \frac{\sum_{i=1}^{b} ETS(T_i, k)}{ETS(T, k)} \tag{7.1}$$

Where T_i represents the sequence in bin number i. Naturally the value of EBR is between 1 (where all the bins contain disjoint unique k-mers) and b (when all bins have identical unique k-mers). For a fixed value of k, a value of EBR closer to 1 indicates a better clustering.

7.4.2. Clustering Sequences Using NCBI's Taxonomy

The NCBI taxonomy is a form of clustering by itself where similar organisms are grouped hierarchically. TaxSBP[1] (https://github.com/pirovc/taxsbp), is an implementation of the approximation algorithm for the hierarchically structured bin packing problem [107] based on the NCBI Taxonomy database [68]. We create bins in such a way that they span a small number of branches of the taxonomic tree. In other words, sequences belonging to the same species are more likely to be in the same bin. One can easily control the number of bins as desired. This clustering method is very efficient, given that it uses the "pre-clustered" taxonomic tree information to generate similarly sized groups of closely related sequences. In this work, we will consider contiguous sequences in the given reference genomes as the smallest unit of sequences that to be clustered into bins. That means we will not split those sequences into smaller parts. Adding and removing sequences is also straightforward. When a new sequence arrives, we simply add it into a bin that contains the sequence of a close relative according to the taxonomy.

Using TaxSBP, we created three different clustering of the RefSeq databases by varying the number of bins (64, 256 and 1024) and analyzed the effectiveness of the clustering using the metrics defined above. The results shown in table 7.2 indicate that shorter k-mers (k<15) are not discriminating enough among clusters. The EBR corresponding to k=19 and k=20 are less than 1.5 which indicates good clustering performed by TaxSBP.

There are two drawbacks of taxonomic based clustering, i.e., 1) Its dependency on the availability of a taxonomic tree. In our case-study (mapping metagenomics reads to the RefSeq prokaryotic database) there exists an already defined taxonomic tree. 2) It does not consider the real sequences similarity. Although rare, if two sequences are very similar, but their taxonomic assignment is far apart, they will end up in different bins. In the next subsection, we discuss a k-mer based clustering which can be used in the absence of taxonomic information and considers the actual content of the sequences to perform clustering.

[1] TaxSBP is implemented and maintained by V. Piro, one of the co-authors of the DREAM-Yara paper. The author of this thesis only used TaxSBP.

Method	# bins	k								
		12	13	14	15	16	17	18	19	20
TaxSBP	64	56.58	44.90	27.15	11.95	4.65	2.27	1.52	1.24	1.13
	256	180.13	109.84	49.06	17.11	5.81	2.64	1.70	1.37	1.23
	1024	510.95	234.66	82.23	24.68	7.72	3.36	2.12	1.68	1.50
k-mer based	64	41.16	32.94	21.09	10.12	4.22	2.14	1.46	1.21	1.11
	256	92.72	62.97	32.84	13.14	4.85	2.31	1.52	1.24	1.12
	1024	180.04	103.49	45.07	15.64	5.31	2.42	1.56	1.26	1.13

Table 7.2.: The *effective binning ratio* (EBR) of clustering the RefSeq prokaryotic database into 64, 256 and 1024 bins using TaxSBP and k-mer based clustering.

7.4.3. k-mer Based Clustering of Sequences

Using frequencies of canonical k-mers from each sequence as an input vector, we applied a k-means clustering. The EBR of the clusters was better than that of taxonomic based clustering (Table 7.2). However, k-means clustering produced uneven clusters. Some of the bins were too large, which undermines the DREAM index framework. We tried to refine the clustering to achieve even distribution across the bins. After we have done the initial clustering, we further refined the bins that are too big and merging the small bins. However, this approach was not efficient, and the overall process was computationally expensive. Besides, updating the bins were difficult as we can not easily decide in which bin to add a new sequence. Due to this we abandoned the small improvement in EBR by this clustering method and used the more straightforward and faster taxonomic based clustering (TaxSBP) for our implementation of the DREAM index framework.

7.4.4. Binning Dictionary and q-gram Lemma

Using TaxSBP, we can partition the database text T into b bins in such a way, that a bin B_i contains similar parts of T denoted by T_i. Sub-indices can be created fast for each bin using T_i as an input. Updates can be performed quasi-dynamically by rebuilding only the sub-indices affected by the change. This process covers the *dynamic operations distributor* layer of the DREAM index framework. For the *approximate search distributor* we use what we call *binning dictionary* together with the well-known q-gram lemma.

A *binning dictionary* D contains the membership information of all k-mers in the original database across the bins for a given value of k. A binning directory returns a *binning* bit vector in which the i-th bit is set if the k-mer is present in bin B_i for a given a k-mer, a. There are three main requirements for a binning directory. 1) As it

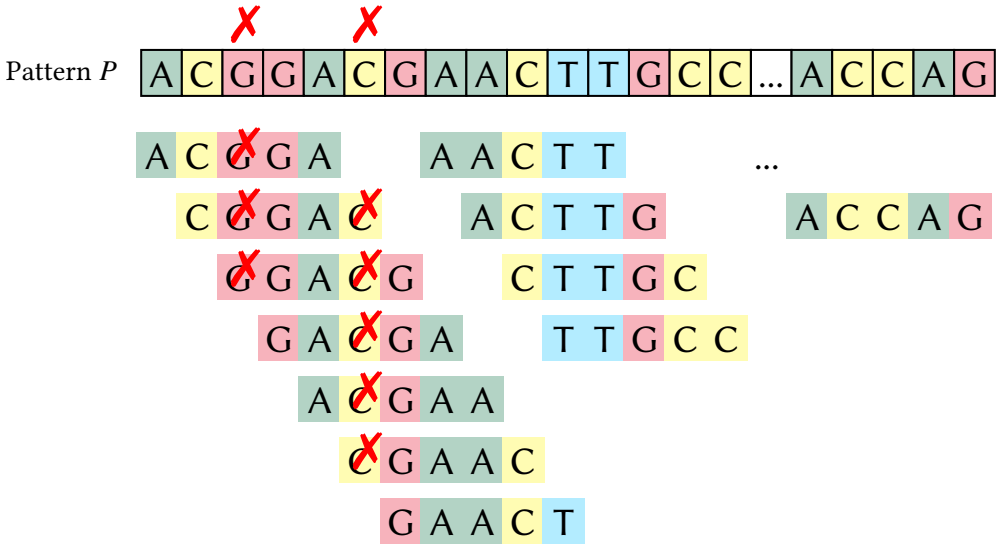


Figure 7.3.: Illustration of the q-gram Lemma: One mismatch in a pattern can destroy q q-grams (5-mers). Here, the first error at (G) will destroy the first three 5-mers. The second error at (C), however destroys 5 5-mers

will be used for filtration before an approximate search, a binning directory should be sufficiently sensitive. 2) It has to be fast so that it can offset the overhead of dealing with b-many sub-indices. 3) It should support fast (preferably partial) updates when there is a change in some of the bins.

Lemma 7.1. For a given k and number of errors e, there are $k_p =\mid p \mid -k+1$ many k-mers in p and an approximate occurrence of p in T has to share at least $t = (k_p - k \cdot e)$ k-mers.

We combine our *binning dictionary* D with the q-gram lemma to distribute sequencing reads across the bins for approximate search. The lemma is based on the observation that one error can destroy at most q many q-mers. Figure 7.3 illustrates the reason behind the q-gram lemma. Whereas figure 7.4 shows how we can decide which bins have a potential for an approximate match for a read in a single batch operation. First, we zero initialize a count vector of size b. For every k-mer in a sequencing read the *binning dictionary* returns a bitvector of size b. We increment the count vector at positions where the bitvector is set. After repeating the process for all the k-mers of the read, we check the count vector for values exceeding the threshold given by the q-gram lemma and search the corresponding bins for the read. This process provides a means to have an approximate search distributor within the DREAM index framework.

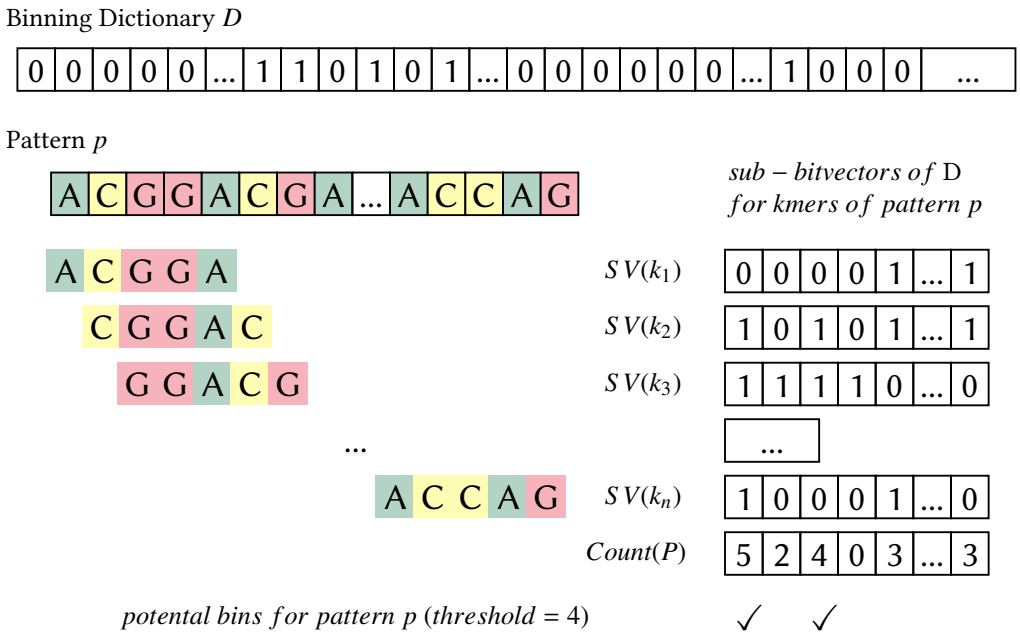


Figure 7.4.: The q-gram lemma using *binning dictionary* (D). For each k-mer k_i generated from a pattern p we extract binning sub-bitvectors $SV(k_i)$ representing the bins containing k-mer k_i. For all set bits in $SV(k_i)$ we increment the counter of corresponding bin. Bins whose counter is greater than or equals to the threshold (in this case 4) will be searched for an approximate match for p.

7.5. Interleaved Bloom Filters (IBF)

We now discuss how we implemented an efficient, easy to build and update *binning dictionary* using a novel data structure that we named IBF. Even though we are using b-many bloom filters as underlying data structures, our interleaved arrangement of the bloom filters is different than that of Mohamadi et al..

7.5.1. Bloom Filters

Bloom filters [100] are data structures that are used to perform set membership queries with a flexible option to trade-off false positive rate with the size of the data structure. It is defined by a bitvector of adjustable size n and a set of h independent hash functions that map a key value, in our case a k-mer, to one of the bit positions. To insert a k-mer to a Bloom filter we simply set h bit positions defined by h hash functions to 1 (see figure 7.5 for illustration). Bloom filters have zero false negative rate, i.e., they guarantee to report the presence of a member object if it is in the set. Depending on the size of the bitvector (n), collisions could happen within a single hash function (a hash function returning identical hash values for different k-mers) or among different hash functions (two hash functions returning the same hash value for different k-mers). These collisions are the source of false positive answers by bloom filters. However, one can tune-down the probability of having a false positive answer to an acceptable level by increasing the bitvector size.

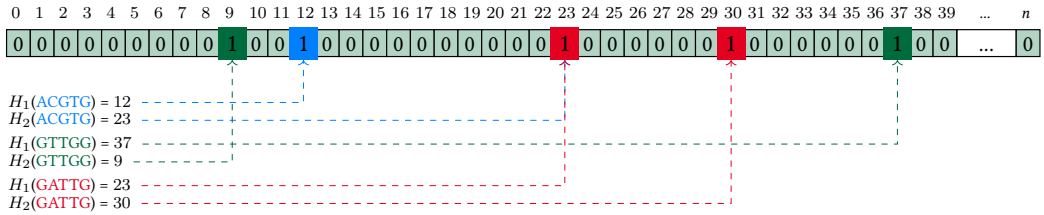


Figure 7.5.: Adding k-mers to a Bloom filter using two hash functions. k-mers are added by setting bits at position returned by the hash functions. $H_2(ACGTG)$ and $H_1(GATTG)$ both yielded 23 creating a collision.

During lookup, a k-mer is considered present in the Bloom filter, if all h positions are set to one. Note that a Bloom filter can give a false positive answer if other k-mers set the h bits. However, if the Bloom filter size is large enough, the probability of a false positive answer is low. A Bloom filter of size n bits with h different hash functions and m elements inserted has the following probability of giving a false positive answer.

$$p_{fp} = \left(1 - \left(1 - \frac{1}{n}\right)^{h \cdot m}\right)^{h}. \tag{7.2}$$

It is recommended to allocate sufficient space (n), such that p_{fp} does not become too large and hurt the precision of the answer. A good rule of thumb is to make n greater than $h \times m$.

In our scenario, we have b-many different sets where b is the number of bins we partitioned our reference database into, and the objects are k-mers generated from each bin. If we use conventional Bloom filters to answer k-mer membership queries, we need b-many Bloom filters and assemble the results into a binning bitvector. This approached was used in [105] where they used a stand-alone Bloom filters for each bin. The problem is executing k-mer membership quires millions of times against b-many Bloom filters creates a computational overhead as it involves too many cash misses.

Our solution to the problem is combining b Bloom filters (one for each bin) with *identical* hash functions into a *single* composite Bloom filter by interleaving them. To put it differently, we replace each bit in the single Bloom filter by a (sub)-bitvector of size b, where the i-th bit "belongs" to the Bloom filter for bin B_i. We call the resulting Bloom filter an *Interleaved* Bloom Filter (IBF). The IBF has a size of $b \times n$. When inserting a k-mer from bin B_i into the IBF, we compute all h hash functions which point us to the position of the block where the sub-bitvectors are and then simply set the i-th bit from the respective beginnings. Hence, we effectively interleave b Bloom filters in a way that allows us to retrieve the binning bitvectors for the h hash functions easily. When querying in which bins a k-mer is, we would retrieve the h sub-bitvectors and apply a logical AND to them which results in the required binning bitvector indicating the membership of the k-mer in the bins. This approach has a significant advantage in query time as retrieving a (sub)-bitvector is extremely cache-friendly. The procedure is depicted in Figure 7.6.

As it is explained in the binning dictionary, we decide which bins are a potential target for a read by applying the q-gramm lemma (lemma 7.1). The IBF tells in which bins a given k-mer occurs by returning a binning bitvector. Hence, we can look up each k-mer from a pattern in the IBF, retrieve the binning bitvector that marks the k-mer's bin-membership, and update an array of counters for each bin. If the counter exceeds the threshold for the bin, the pattern will be searched approximately in the bin, otherwise not. Figure 7.4 depicts this approach is where a generalized binning dictionary is used instead of an IBF.

The IBF fulfills all the three requirements of a binning dictionary, which is instrumental to implement an approximate search distributor. 1) It is fully sensitive as Bloom filters are. 2) It is fast thanks to the cash-friendly retrieval of k-mer membership across the

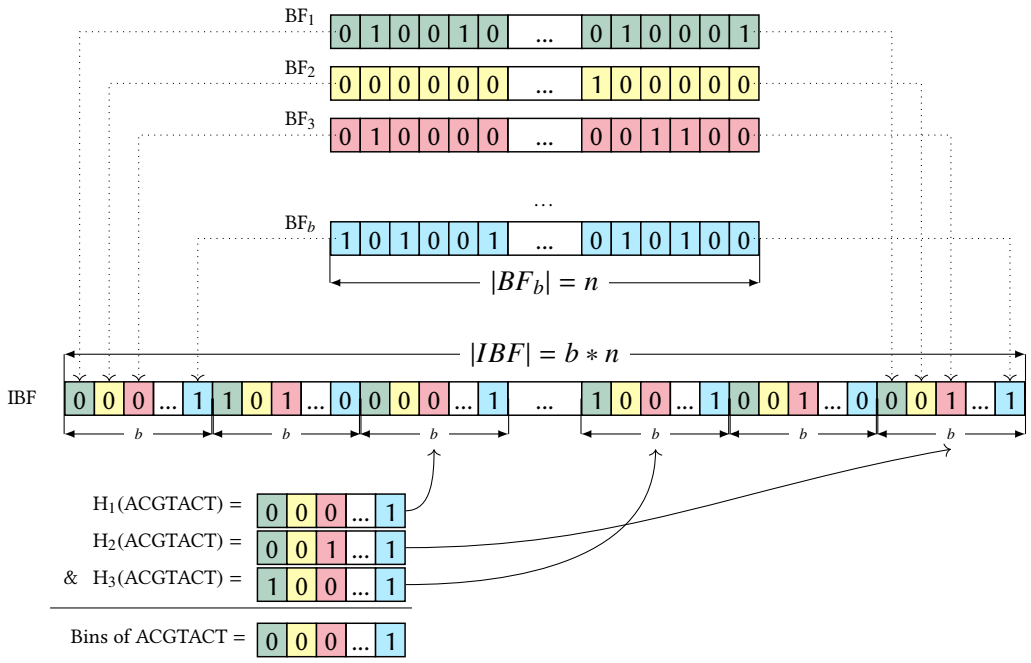


Figure 7.6.: A schematic of the IBF. Differently colored Bloom filters of length n for the b bins are shown in the top. The individual Bloom filters are interleaved to make an IBF of size $b \times n$. In the example, we retrieve 3 positions for a k-mer (ACGTACT) using 3 different hash functions. The corresponding sub bitvectors are combined with a bitwise & giving us the needed binning bitvector.

	32 GB IBF				16 GB IBF			
k	17	18	19	20	17	18	19	20
64	1.00E-08	2.00E-08	2.00E-08	2.00E-08	9.00E-08	1.20E-07	1.40E-07	1.50E-07
256	2.00E-08	2.00E-08	3.00E-08	3.00E-08	1.60E-07	1.90E-07	2.00E-07	2.10E-07
1024	1.70E-07	1.70E-07	1.70E-07	1.80E-07	1.32E-06	1.34E-06	1.37E-06	1.40E-06

Table 7.3.: Average false positive rate of bloom filters which are interleaved within the IBF on the bins created by TaxSBP. Three different binning (64, 256 and 1024), a range of k-mer sizes (17-20), and two bitvector sizes (16 GB and 32 GB) are shown.

bins. 3) It can be partly updated in a straightforward way to reflect changes in bins. Consider the contents of the i^{th} bin has changed. First, we reset the corresponding i^{th} bits from every sub-bitvector of the IBF. Then we add the k-mers from the same updated bin to the IBF. We can simultaneously update multiple affected bins in parallel.

7.5.2. Performance of the IBF on Metagenomic Data

Dataset: In order to check the performance of the IBF we used the NCBI's RefSeq database of complete prokaryotic genomes described at the beginning of this section. The database contains 31.34 GB of DNA sequences. We build IBFs for three binning scenarios of this database (64 bins, 256 bins, and 1024 bins) created using the TaxSBP program. We tried both k-mers of length 19 and 20 to create the IBF. Even though the effective binning ratio is better for 20-mers than it is for 19-mers (table 7.2), the final filtration efficiency based on the q-gram lemma is comparable between the two k-mer sizes when checked with real data. We believe, this is because the threshold from q-gram lemma, given the number of errors and the length of reads, is higher for 19-mers, which boosted the filtering efficiency.

Update set: To demonstrate the partial updating of the IBF, we consider the same database after approximately three months (downloaded on the date 2017-12-19). We specifically consider the updated genomes of *Escherichia Coli* which sums up to 0.23 Gbp. 155 new sequences and one removed sequence were part of the update. The update affected 5 out of 64 bins, 15 out of 256 bins, and 42 out of 1024 bins.

Metagenomic reads: We have downloaded a publicly available sequencing run with SRA id *SRR6504858*. The source of the DNA was a human gut and the sequencing was done by Nanfang Hospital of Southern Medical University. This read-set contains paired sequences, 150 bases long each, produced by Illumina HiSeq X Ten instrument. For practical reasons we considered only the forward pair of reads to evaluate the IBF. The number of reads is 42,692,634.

	IBF Size (GB)	k	Max Resident Mem (GB)	Time (hh:mm:ss) IBF build	IBF update	IBF load	Filter reads	Avg. Reads per bin	Filter Factor
64 bins	16	19	16.21	1:12:45	0:06:15	0:00:33	0:41:03	313416	136
	16	20	16.21	1:12:54	0:06:17	0:00:33	0:40:44	317363	135
	32	19	32.21	1:19:40	0:08:21	0:01:04	0:40:43	305419	140
	32	20	32.21	1:18:27	0:08:21	0:01:04	0:40:08	308439	138
256 bins	16	19	16.20	1:12:56	0:05:51	0:00:33	0:52:10	105752	404
	16	20	16.20	1:14:03	0:06:01	0:00:32	0:51:45	108344	394
	32	19	32.20	1:19:54	0:07:56	0:01:04	0:48:05	102794	415
	32	20	32.20	1:18:12	0:07:55	0:01:04	0:47:37	104764	408
1024 bins	16	19	16.17	1:12:40	0:04:23	0:00:33	1:34:21	51051	836
	16	20	16.17	1:13:05	0:04:21	0:00:33	1:33:38	53252	802
	32	19	32.17	1:18:04	0:06:14	0:01:04	1:15:54	48584	879
	32	20	32.17	1:18:07	0:06:15	0:01:04	1:15:18	50058	853

Table 7.4.: Evaluation of the *interleaved Bloom filters* (IBF) on multiple setup of parameters. The evaluation is performed on the clustering the RefSeq prokaryotic database into 64, 256 and 1024 bins using TaxSBP and k-mer values 19 and 20. The IBF performed well for both bitvector sizes of 16 GB and 32 GB.

Infrastructure We used a workstation computer equipped with Intel(R) Core(TM) i5-8500 CPU and 64GB of memory. Even though our implementation of the IBF supports parallelization that scales linearly, we used a single thread to run the commands.

The results of performance evaluation on the IBF are shown in table 7.4. These Results show that the IBF is robust to parameterization both concerning computational performance and filtering efficiency. It took an average of approximately 1 hour and 16 minutes to build the IBF for 31GB of genomic reference database on a single thread. We want to mention that our implementation of the IBF has a multi-threading support and the observed speedup is close to the number of threads used. The 32GB version of the IBF took slightly (5-7 minutes) longer to build than 16 GB version. There is virtually no difference in build time associated neither with the number of bins nor with the length of k-mers used, which indicates that the IBF is a scalable data-structure that can easily support even larger databases.

Updating the IBF for the update set mentioned above took less than 7 minutes. This is again on a single thread and multi-threading support is available in our implementation. The time required for updating the IBF is independent of its size and the number of bins. On top of a fixed amount of time required to load the data structure from disk and save it back, the update time is only sensitive to the size of the update. As shown in table 7.4, the amount of RAM required both to build and update the data structure is the same as the size of the IBF bit vector with a small addition for buffering sequences

and “bookkeeping”.

In order to investigate the filtration and distribution efficiency, we applied different configurations of the IBF on metagenomic reads (table 7.4). If we have done approximate searching, without a filter (in this case the IBF) all 42 M reads would have been distributed to every bin. The average number of reads in the table shows that a significantly less number of reads are kept after the filtration. In this experiment we are looking for approximate matches up to 5 (3% of errors) mismatches. The “*filter factor*” is the ratio between number of filtered reads per bin and the number of reads before filtering. In our setup the “*filter factor*” is higher for the smaller value of k=19, compared to k=20. This observation is counter-intuitive and related the dynamics of q-gram lemma threshold in relation to the number of errors and read length. In our example the filter threshold is 18 ($(150-19+1)-19\times5$) and 11 ($(150-20+1)-20\times5$) for for k=19 and k=20 respectively, which creates a more tight filter for k=19. More detail, on how the q-gram lemma behaves under different read length, error rates and k-mer sizes, is given at appendix A.9.

On the other hand the “*filter factor*” gets better with a larger number of bins as the individual bins are getting smaller and more specialized. A small increase in “*filter factor*” is observed in the 32 GB version of the IBF compared to 16 GB version. This is due to the difference in the false positive rate of bloom filter in relation to the bit vector size(table 7.3). However the difference is negligible and can be ignored if one needs to optimize for memory usage or to speedup index loading time. More performance evaluations on the IBF, including the effect of parallelization, will be presented in chapter 8 in combination with the evaluation of the DREAM-Yara read mapper.

We strongly anticipate, that the IBF will be a useful data structure for a wide range of applications. Applications that involve set membership queries across multiple bins/groups/sets, especially for assessing k-mer content can benefit from the IBF. When writing this manuscript for DREAM-Yara ([110]), it was brought to our attention that Bradley et al. independently thought of a similar data structure in [111], although they do not use them in conjunction with the q-gram lemma and do not employ the strategy of interleaving them.

7.6. FM-indices per Bin

After partitioning the databases smaller partitions (bins), standard FM-indices are built on individual bins. We call the small indices sub-indices. The sub-indices can be built in parallel as they are independent. The main advantage of having such sub-indices is the modularity they provide during an update operation. One needs to rebuild only the sub-indices of affected bins. In our experiments, we modified the indexing module of the Yara read mapper [54, 110] to create FM-indices in parallel. In addition to the

speedup gained by multi-threading, building all sub-indices on smaller partition took three times less time than building a single index. Multi-threaded construction of FM-indices is not currently available in standard yara_indexer implementation. Another advantage of building small sub-indices is the low memory footprint which makes index construction possible on a standard laptop. Nevertheless, the short update time remains the main advantage of creating small sub-indices.

7.7. Distributed Yara - A Trivial Distribution

Here we describe a straightforward approach of distributed read-mapping as it was done in [105] using the sub-indices created. In this naive approach, we take our set of reads and iteratively map them to the corresponding sub-indices of each bin as opposed to using an approximate search distributor of the DREAM index framework. This approach has all the advantages related to index construction and update time. However, the overhead created by the sub-indices and the reduction mapping speed outweighs the benefits (see table 8.1 in chapter 8 for more details). Another issue is the validity of mapping results across the bins. Mapping results obtained from each sub-indices have to be consolidated, which involves identifying the final best/primary match among others. Nevertheless, we did benchmark the trivial distributed read-mapping in combination with the Yara read mapper and called it "Distributed Yara". Our benchmark results indicate that the distributed indices are too slow unless they are complemented with a smart filter (*approximate search distributor*) and in memory consolidation of mapping results. In the next section, we present our solution to the problem as the DREAM-Yara software.

7.8. DREAM-Yara

Now, we describe the fully featured read mapper we implemented based on Yara read mapper to support the DREAM index framework. DREAM-Yara was engineered to counterbalance the effect of having multiple sub-indices 1) by using a lightweight filter to distribute reads into bins with a mapping potential. 2) Consolidating mapping results before writing them to disk. In order to explain its integration with the DREAM index framework as DREAM-Yara, in the following we provide a rough description of the Yara read mapper itself. We limit the explanation to aspects of the read mapper relevant to the integration process. For more detailed read, we recommend reading publications [54, 110].

7.9. The Yara Read Mapper

The Yara read mapper [54, 110] is currently state of the art read mapper of the SeqAn library [55, 112]. It is an exact read mapper that reports all mapping locations of a read within a user-defined error threshold in contrast to popular read mappers that deploy a heuristic such as Bowtie 2 [48] and BWA [49]. The reason for using heuristics is mostly related to reducing the computational time. Even if Yara mapper does not use any heuristic, it is faster than the methods which do so. The efficiency of Yara stems from a well-engineered set of known computational algorithms implemented in the *SeqAn* library. Yara combines known algorithmic concepts in a novel way to achieve superior performance, which we leverage in our DREAM-Yara adoption.

Yara uses the concept of strata based mapping where stratum is defined by the distance between a read and the genome at a mapping location. Matches with similar distance fall under the same stratum. Given a maximum distance k allowed for a match, an all-mapper should report all the matches for a stratum of distance [0, k]. For practical purposes best+x mapping is offered by Yara. In best+x mapping, matches at a stratum corresponding to the best match and x extra strata of matches will be reported. The number of extra strata is lower than the maximum distance allowed (k). Note that a stratum is always reported exhaustively. Yara always reports exhaustively all equally good matches of a read. In other words, no read match with the same quality/error will be left out, which is not the case for heuristic-based read mappers.

In the beginning, read pairs are treated independently by Yara. After reporting all the matches, the library information (insert size of library length) is used to choose the primary mapping location. That is if two equally viable mapping locations exist for each pair, the one that respects the insert size is regarded as a primary match.

7.10. DREAM-Yara Adoption

We design DREAM-Yara on top of the original Yara mapper to support sub-indices. DREAM-Yara takes bin-many FM-indices and an IBF as an additional input compared to Yara. All the other parameters and Input settings are identical to Yara. Alignment files produced by DREAM-Yara are identical to those produced by the standard Yara mapper using one BIG index which reassures users that there is no compromise concerning mapping result quality. DREAM-Yara includes all the parameters of Yara, hence can be configured to report all-mapping, best+x mapping, and best mapping similar to Yara. DREAM-Yara inherits all the good qualities of Yara by using it as a core mapper.

The DREAM-Yara software collection is a first complete implementation of the DREAM index framework. DREAM-Yara's indexer has modules to create many sub-indices (FM-indices) on smaller bins, and update bins that are affected. These modules make

a *dynamic operation distributor* under the DREAM index framework. DREAM-Yara uses TaxSBP to cluster sequences into a desired number of bins before indexing. The indexer also has a sub-module that can create and update IBFs. Which for later usage to distribute reads in a filtered way using the q-gram lemma. In the end, it consolidates all the mapping results obtained from individual bins and cross-validates them. The validation process involves 1) checking the best stratum, 2) identifying the best matches of a read across the bins, and selecting a primary match among the best matches. The IBF based filtration to distribute reads and the match consolidation process of DREAM-Yara represent the *approximate search distributor* layer of the DREAM index framework.

By adding a filtration layer between loading reads from a disc and performing the actual mapping using Yara's algorithm, we were able to counter the overhead created by using many small indices. The significant decrease in the number of approximate searches (table 7.4) is essential for making distributed indices a viable alternative compared to using one big index. We show this by comparing the trivial distribution of reads without filtering against DREAM-Yara. We present detailed results in the next chapter.

In DREAM-Yara, sequencing reads are loaded to memory in batches. Next, each read goes through an IBF based filter to decide which bins have a potential for a mapping location for that read. This is done by generating all k-mers from a read and looking them up in the bins using an IBF. For each k-mer, the IBF returns a bitvector of length b, where b is the number of bins. The bitvector indicates in which bins the k-mer exists. We use this bitvector to increment a count vector of length b and repeat the process for each k-mer of the read. Then we look at the count vector to decide which of the bins shared more k-mers than required by the q-gram lemma threshold depending on read length and the maximum allowed mismatch/error. This process creates b different subsets out of the loaded reads representing the different bins. We handle reverse complements and paired reads in a generous fashion. That is, a read belongs to a subset R_{bi} if it, or its reverse complement, or its mate pair shares enough number of k-mers with bin B_i.

After filtration, we use Yara's core mapping algorithm to find all mapping locations for each of the corresponding reads assigned to that bin. Then we collect mapping results from all subsets and merge them. While merging, DREAM-Yara creates a global ranking of all mapping locations based on mapping qualities. This ranking is the basis to decide which matches are best matches and primary matches. Some matches which are within parameters set by a user in a context of a bin might get discarded in the merging process. For example, if the user wants only the best matches ($strata - 0$), what was reported as the best match for a bin might not be so in a global context. In the end, the mapping result is written to a single SAM/BAM file. The merged mapping file DREAM-Yara produces is identical to that of standard Yara, provided that the same set of parameters, reads and references are used.

7.11. Summary

In this chapter, we presented the DREAM index framework which addresses the static nature of indices in an approximate search. By partitioning a large set of references into smaller clusters, we introduced modularity to the indices, enabling quick partial updates. Besides, the total time and peak memory required to build sub-indices are significantly lower than building an equivalent big index. We mentioned that having many-indices instead of one is against the core principle of indexing itself and will slow down the search process. The DREAM index framework addresses this issue using the concept of an *approximate search distributor* which significantly reduces the total job done by filtering and then distributing reads.

In our implementation of a read mapper within the DREAM index framework, we used a novel data structure called IBF as a filter. We showed that the IBF is a scalable, fast and robust data structure to simultaneously lookup k-mers in many bins. With the help of the IBF, we designed and implemented a read mapper that works with distributed indices. The next chapter focuses on the evaluation of DREAM-Yara as a read mapper that works with distributed indices.

8. DREAM-Yara Evaluation

In this chapter we will present a thorough evaluation of the DREAM-Yara read mapper we presented in the previous chapter. We evaluate DREAM-Yara by benchmarking it against three existing read mappers, the standard Yara mapper and two trivially distributed read mappers. The three popular read mappers considered in our evaluation are Bowtie 2 [48], BWA-MEM [49] and, GEM [50]. The brief description of the read mappers was provided in 3.2.2. We have included Yara read mapper in the evaluation to help the reader understand the impact of using distributed indices within a similar set of algorithms. The evaluation focuses on three aspects of the read mappers, i.e., the time required to create and update a required index, mapping speed, and mapping sensitivity.

8.1. Evaluation Setup

8.1.1. Dataset

As metagenomic read-mapping is the main target for DREAM-Yara, we used a reference set of archaeal and bacterial complete genome sequences downloaded from NCBI's RefSeq database [27]. The dataset was downloaded on 2017-09-26 and amounts to 15,250 sequences, summing up to a total of 31.34 Gbp. The database represents 2,991 species. This reference set was partitioned into 64, 256 and 1024 non-overlapping bins using TaxSBP program for the mappers that required distributed indices. This included DREAM-Yara, distributed Yara and DIDA-BWA. Clustering times with TaxSBP are negligible and were not included in the evaluation process

The changes to the same database were downloaded after approximately three months (on the date 2017-12-19). For practicality we considered sequence additions and deletions specific to *Escherichia Coli* species. This 0.23 Gbp update includes one removed sequence and 155 new sequences which we believe to be typical set of sequences for which we want to update our index. When TaxSBP redistributes this update to the original bins, 5 out of 64 bins, 15 out of 256 bins, and 42 out of 1024 bins were affected by the change. Both the original set and the update set are similar to the ones used for the evaluation of the IBF in the previous chapter. Information on how to download the reference sets in different binning configuration is available at appendix A.10.

To evaluate the computational performance and sensitivity of the read mappers, we use a publicly available sequencing run (SRA/ENA id: SRR6504858) submitted by Nanfang Hospital of Southern Medical University. DNA material was extracted from fecal material representing a gut metagenome and sequenced using an Illumina HiSeq X Ten instrument. The resulting paired reads sequences were of length 150. Each pair contains 42 M reads. We used only the first pair for practical reasons. In the case of sensitivity benchmarks by Rabema, we used only the first 1 M reads as computing full list of mapping locations using Razars3 [113] is prohibitively slow to do it for the whole set of reads.

8.1.2. Infrastructure

All the experiments were carried out on a compute server equipped with 32 (Intel(R) Xeon(R) CPU E5-2650 v3 2.30GHz) processors and 130GB of memory. All of the tools were run using 8 threads with the exception of indexing modules of the BWA-MEM and standard Yara mapper. The index building process in the two read mappers does not offer multi-threading support. Bowtie 2 and GEM are the only exceptions of a non-distributed mapper with multi-threading index building. The IBF used in DREAM-Yara is built with 18-mers and has a bit vector size of 16GB (137,438,953,472 bits). The exact and complete commands used in this evaluation are available in appendix A.11.

8.2. Results

In this section, we present the benchmark results of DREAM-Yara, trivial Distribution of Yara, the DIDA distributed indices framework using BWA-MEM as a mapper, and other three state of the art read mappers. We organized the results in two parts. 1) Runtime and memory consumption required to build and update indices, 2) Read mapper performance including throughput, memory consumption during read-mapping, and sensitivity analysis using a well defined benchmarking method, Rabema [114].

8.2.1. Build and Update Indices

The time and memory required to build and update indices for the metagenomic reference database described in section 8.1.1 is listed in table 8.1. DIDA framework using BWA, Distributed-Yara (*a trivial distribution similar to the DIDA framework*), DREAM-Yara, standard Yara, Bowtie2, GEM, and BWA are included in the comparison. The table clearly shows that the advantage of having small sub-indices in improving the time required to build and update indices. In a standard single index scenario, the fastest indexer took close to 10 hours, whereas the distributed indices with 1024

	bins	Time (hr:min:ss)		Peak Memory
		Build	*Update	(GB)
DIDA (BWA)	1024	56:07	2:51	0.06
Distributed-Yara	1024	49:10	3:15	8.32
DREAM-Yara	1024	1:07:03	6:43	16.15
	256	1:44:02	9:42	16.16
	64	1:57:01	19:04	16.32
Yara		27:17:54	*27:17:54	85.07
Bowtie 2		9:51:42	*9:51:42	89.80
BWA-MEM		19:33:24	*19:33:24	43.83
GEM		20:41:01	*20:41:01	104.00

Table 8.1.: Wall clock time and peak memory required for building and updating indices. Peak Memory refers to the maximum resident memory occupied by a program (*all threads in case of multi-threading*) during execution. * Since it is not possible to partially update indices for standard mappers, similar values as build time are reported

bins took about 1 hour. That is a ten fold speedup. As some of the read mappers used multi-threading for indexing, we do not advise a direct comparison of reported times. Nevertheless, we believe that parallel building of a single big index doesn't scale well with the number of threads. This is further supported by the results from Bowtie2 and GEM which support multi-threading for indexing and were run with 8 threads. But the speedup was just 1.5× for GEM and 3× for Bowtie 2.

We have evaluated DREAM-Yara in three different bin setups to study the effect of clustering into different number of bins. As it is shown in the table more number of bins means less index building and updating time. The values reported for the three different bin-configuration of DREAM-Yara are aggregations of two values, one from small FM-indices and another one from an IBF filter. We took the sum in the case of runtime, and the maximum of the two in the case of peak memory consumption. That is why the build time of DREAM-YARA in 1024 bins configuration, is slightly higher than that of the trivial distribution (Distributed Yara). Distributed-Yara uses the same FM-indices as DREAM-Yara, but there is no need for an IBF as no filtration is performed. So in the case of Distributed-Yara, we omit the time and memory related to IBF construction. A detailed breakdown of these values is presented in table 8.2.

It took the 1024 bins version of DREAM-Yara 1 hour and 7 minutes to index 31.34 Gb of reference sequences. That is approximately 9 times faster than the next fastest indexer (Bowtie2) and 26 times faster than the slowest indexer (standard Yara). Distributed-Yara

bins	Build time (hr:min:ss)			Update time (hr:min:ss)			Peak Memory (GB)		
	IBF	FM	TOTAL	IBF	FM	TOTAL	IBF	FM	MAX
1024	17:52	49:10	1:07:03	3:28	3:15	6:43	16.15	8.32	16.15
256	21:28	1:22:33	1:44:02	3:40	6:02	9:42	16.16	12.38	16.16
64	20:04	1:36:56	1:57:01	9:23	9:41	19:04	16.32	16.32	16.18

Table 8.2.: A break down of runtime and memory for index building in DREAM-Yara between FM-indices and an IBF.

and the DIDA framework using BWA exhibited the best time in creating and updating indices. Despite of having the least computational requirements during indexing step, we do not find them to be practical considering how slow they make the mapping process as it is shown in the next section. DREAM-Yara is only 18 minutes slower in creating and 3 minutes slower in updating indices than Distributed- Yara. These differences are due to the time needed for building and updating the complementing IBF which are shown in table 8.2.

Even more appealing result is the time required to update indices. As we have explained before, in the case of standard one big index, there is no possibility to partially update the indices. One has to build the index again from scratch irrespective of how small change is. That is why we reported the same values under the update column of table 8.1 for the standard read mapper. This are not actual values recorded, hence the * mark on the values. We safely assumed it would take at least the same amount of time as the updated set is larger. On the other hand it took only few minutes to update the affected sub-indices. The more bins we have, the faster incorporating the same amount of update gets. In the 1024 bins setup updating the affected (42) sub-indices took less than 7 minutes. In the 256 and 64 bins setup updating the corresponding affected sub-indices took 10 and 19 minutes respectively. This is a significant improvement compared to several hours required to rebuild a single index.

Another interesting result is the extremely lower memory required to build and update distributed indices. All the standard read mappers required an enormous amount of memory to index the 31.34 Gbp set of reference sequences, while DREAM-Yara indexer required 62% less memory compared to BWA which is the next best method concerning memory. The efficiency in memory requirement of BWA is reflected in DIDA-BWA which is requiring the least amount of memory off all cases. Distributed-Yara and the DIDA framework using BWA also have the smallest amount memory footprint. As it is shown in table 8.2, the IBF is responsible for the peak memory consumption of DREAM-Yara indexer. Memory consumtion reported in the table are coming from the index construction in particular. The memory requirement of building distributed indices is low enough to allow building it on average current day laptop, even for huge databases.

8.2.2. Read-Mapping

We compared the mapping speed of the read mappers using read-mapping throughput in *giga base pairs per hour* (Gbp/h) and memory consumption. That is the amount of reads mapped against a fixed set of reference sequences in one hour. Yara [s=0] is the fastest read mapper with the highest throughput (16.59 Gbp/h) followed by GEM (12.73 Gbp/h) and Bowtie 2 (7.71 Gbp/h). The results are shown in figure 8.1.

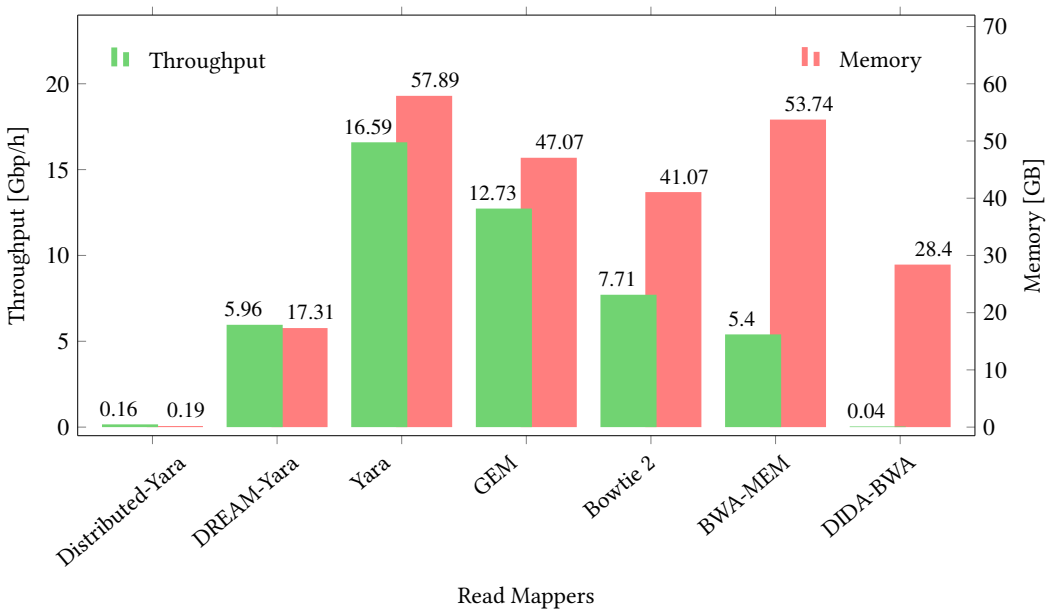


Figure 8.1.: Read mappers performance - throughput and peak memory evaluated using 42 M metagenomic reads (SRA/ENAid: SRR6504858)

The trivial distribution of Yara using 1024 small sub-indices is 103.69× slower than standard Yara. This does not include merging results from each bins. DIDA-BWA is the slowest of all in mapping speed at 0.04 Gbp/h (135× slower than the standard BWA-MEM). These numbers showcase the overhead created by having smaller sub-indices. DREAM-Yara is only 2.78 times slower than standard Yara and competitive with the other read mappers. DREAM-Yara showed 37× better throughput compared to the trivial distribution of Yara. DREAM-Yara requires significantly less memory during read-mapping compared to Bowtie2, BWA and standard Yara. Its memory requirements are second only to the trivial distribution of Yara. The results are confirmations that DREAM-Yara removes the bottleneck of large index reconstruction successfully while remaining in speed and memory consumption competitive to the standard read mappers.

The *Rabema benchmark* [114] can be used to measure the sensitivity of read mappers in finding "relevant" mapping locations of genomic reads within a reference(set). In

our context we define all mapping location which are as good as the best mapping location to be relevant locations. In other words, mapping locations in the best strata are considered relevant. We refer to mapping locations with equal number of edit distance co-optimal mapping locations. Rabema computes the *sensitivity* of each tool as the fraction of relevant mapping locations found by the tool. Rabema further organizes mapping locations into groups by their *error rate* then computes sensitivity within each error rate group allowing an in-depth evaluation.

The Rabema benchmark produces two kinds of sensitivity reports, which differ by the weight given to matches of a read depending on the number of mapping locations that a read has. In Absolute kind of sensitivity we simply divided the number of co-optimal matches reported by the number of ground truth co-optimal locations. Every match weighs equally. In a normalized sensitivity report, on the other hand, matches from a read that maps everywhere are weighted less and a match unique to a read weighs the best. The final sensitivity is given by the average sensitivity calculated for each read. For instance, if two reads r_1 and r_2 map to the database with e.g. 0 errors and assume r_1 maps uniquely and r_2 maps at 100 locations. Assume a read mapper finds the unique location of r_1 and only 50 locations for r_2. In the normalized sensitivity case, Rabema will report a sensitivity of $\frac{(1/1+50/100)}{2}$ =0.75 and in the absolute sensitivity case, it will report a sensitivity of $\frac{1+50}{1+100}$ =0.5049.

We used RazerS 3 [113], another fully sensitive read mapper from the SeqAn library, to build Rabema's gold standard. RazerS 3 was run in full-sensitive mode within 5 % error rate. Subsequently, we provided the reads as unpaired to each tool, as the Rabema benchmark is not suited to work with paired-end reads.

Table 8.3 shows both normalized and absolute Rabema results. The left panel shows percent scores normalized by the number of valid mapping locations. Hence, as pointed out above, repetitive reads have less weight. Distributed-Yara, DREAM-Yara and Yara [s=0] are 100% sensitive both in normalized and absolute sensitivity measures. In the brake down of the sensitivity in error groups, indicated by the small numbers in table 8.3, the full sensitivity of Yara and derivatives stays 100% for all error categories. There is no loss in mapping sensitivity due to the distribution of mapping. To our surprise GEM is not full-sensitive even though it claims to be so; it loses small fraction of normalized locations starting from 2 % error rate up. Bowtie 2 and BWA are neither fully sensitive nor designed to be so. They loose a number of co-optimal mapping locations especially the ones with higher error rate.

In metagenomics "repetitive" reads are often a result of multiple genomes of similar organisms as opposed to repetitive regions in a genome. Until decisions are made which genomes are the actual source of a read, it is desirable to gather all mapping locations for non-trivial downstream analysis. In other words, all mapping locations are significant for downstream analysis. The right panel of table 8.3 shows Rabema results where we considered the absolute number of co-optimal locations without any normalization. Here, DREAM-Yara, Distributed-Yara and Standard Yara [s=0] are the

	Co-opt. locations								
	[% Normalized]				[% Absolute]				
Distributed-Yara [s=0]	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	
		100.0	100.0	100.0		100.0	100.0	100.0	
DREAM-Yara [s=0]	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	
		100.0	100.0	100.0		100.0	100.0	100.0	
Yara [s=0]	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	
		100.0	100.0	100.0		100.0	100.0	100.0	
GEM	99.8	100.0	100.0	99.9	96.4	100.0	99.8	95.4	
		99.5	99.7	94.8		74.2	67.6	79.3	
Bowtie 2	99.8	99.9	100.0	99.9	81.6	80.8	90.4	89.9	
		99.7	99.5	97.3		67.6	63.4	81.3	
BWA-MEM	99.3	100.0	100.0	99.8	85.2	83.6	92.1	93.6	
		98.2	93.8	90.1		81.5	72.7	84.5	
DIDA-BWA	99.5	100.0	100.0	99.8	85.2	83.6	92.1	93.6	
		98.2	93.8	–		81.5	72.7	84.5	

Table 8.3.: Rabema benchmark results on 1 M metagenomic reads (SRA/ENA id: SRR6504858) mapped against 31.34 GB archaeal and bacterial references from NCBI's RefSeq database. The colored panels show the results of finding all co-optimal mapping locations of the reads; Big numbers show total Rabema scores, while small numbers show marginal scores for the mapping locations at $\binom{0\ 1\ 2}{3\ 4\ 5}$ % error rate. The left panel shows the sensitivity of mappers normalized by the number of locations reported per read, while the right panel shows absolute sensitivity.

clear winners in sensitivity. They found all co-optimal locations; GEM loses 5.6 % of locations at 2 % error rate.

8.3. Results Summary

Our benchmark results in this chapter show that DREAM-Yara removes the bottleneck of large index reconstruction successfully while remaining in speed and memory consumption competitive to state-of-the-art read mappers. This is supported by the 37.25× speedup on a trivial distribution without a filter and in memory consolidation of results. Looking at time and space requirements for mapping, DREAM-Yara needed 56% less memory when compared against Bowtie2, BWA and standard Yara. DREAM-Yara allows to both create index and do read-mapping of metagenomic reads against the complete prokaryotic reference genomes of RefSeq (31.4 Gbp) on a personal computer as it requires only a maximum of 17 GB primary memory.

Part IV.

Conclusion and Miscellaneous

9. Conclusion

9.1. Discussion

In this thesis, we presented two computational methods that utilize WGS data for studying microbial communities. We started by motivating the need to study microbial communities by elaborating their importance in different sectors. Then we came to the topic of taxonomic profiling, which is instrumental in studying microbial communities comparably. The ambiguity resulted from the similarity between the genomes of different microbial organisms is a challenge posed to the taxonomic profiling process. SLIMM, our contribution to improving taxonomic profiling regarding accuracy, uses the coverage landscape of genomes to resolve such ambiguities and achieves a high precision in reporting member organisms at a species level. It also looks deeper into the aspect of reporting correct abundances for detected groups and offers more close to real abundances than computing methods.

Taxonomic profiling methods relay on reference databases which are often substantial. Large databases might not be directly the case with marker-based methods such as MetaPhlAn2 [79]. However, the markers themselves need to be generated from big databases. These databases are getting larger due to cheaper and more advanced sequencing technologies. On one hand, the rate of change made to these databases is getting faster. On the other hand, existing methods rely on the static natures of the indexing data structure. Researchers often justify, the high cost of creating indices and the lack of means to incorporate changes into indices with the argument "*How often one builds an index?*". However, in metagenomics, reference databases are changing in daily basis.

It is widely accepted that, in the context of taxonomic profiling, read mapping is too slow for metagenomic reads [115]. The growing number of taxonomic classifiers which solely use k-mer information and avoid read alignments is a testimony for that. However, in critical situations like diagnosis, alignment-based methods offer a better level of confidence. Hence, it is essential to develop them further so that they could catch up with the growth of databases. The growth of databases also challenges k-mer based methods. In a recent paper, Nasko et al. investigate how the growth of databases influence k-mer based methods. They argue that the growing species-to-genus ratio will challenge read assignment at a species level. Another astonishing fact they pointed

out is the doubling of the number of unique 31-mers every 1.5 years. That is an alarming rate of an increase considering the Terabyte memory requirement of k-mer indices these tools need at the moment. There are ongoing efforts in creating a reduced version of metagenomic reference databases [117].

Naturally, the results of taxonomic profiling can only be as good as the database used. Any taxonomic profiler can identify a species, using environmental sequences, if that species has a representation in the database used. We believe, it is imperative to consider the quadrupled number of species represented in the NCBI's RefSeq in the past five years. It is questionable to use taxonomic profiles generated with a database from five years ago. The prohibitively slow task of building indices for databases and the tedious job of cleaning up and customization required before indexing is tempting to use a relatively old database.

In a systematic approach to the problem, we proposed the DREAM index framework and discussed it in this thesis. We introduced sub-indices instead of one big index in the interest of modularity. We build sub-indices from bins formed by clustering similar sequences together. Having multiple sub-indices enables partial updates on affected part of the database. If widely applied, the DREAM index framework could help ease the decision of using an up-to-date database. It is evident that using multiple small indices creates a prohibitive overhead in the read mapping process. We demonstrated this overhead using experiments involving a trivial distributed read mapping. The proposed DREAM index framework aims to counter that using its *approximate search distributor* layer, where reads are distributed for approximate search after quick filtering.

Coming back to the problem of read-mapping in metagenomics, we presented DREAM-Yara, a distributed index read mapper developed under the DREAM index framework. To realize that, we needed an implementation of a lightweight filter that distributes reads before mapping. In the process, we came up with a novel way of using many bloom filters in a cache-friendly way by interleaving them and treating them as a single bloom filter. We named the resulting data structure a IBF. The IBF can answer k-mer membership queries for a large number of bins simultaneously and quickly. It also scales well to the number of bins and size of databases. Combining the IBF with the famous q-gram lemma, we created a working implementation of an *approximate search distributor*. In the end, we combined these concepts to create an in-memory distributed version of the Yara read mapper and called it DREAM-Yara.

Sub-indices for DREAM-Yara can be created order-of-magnitude faster than the standard read mappers, and It takes only a few minutes to update them. DREAM-Yara is competitive in speed with state-of-the-art read mappers. DREAM-Yara's memory requirement both to create indices and perform read mapping is low enough to allow indexing and searching the 31GB RefSeq database of genomes in a typical desktop computer. More importantly, updates on references can be easily incorporated to metagenomic pipelines without the nightmare of a day-long indexing process.

9.2. Future Work

We firmly believe that DREAM-Yara will be a very practical, exact read mapper for metagenomic short reads. Full integration of SLIMM and DREAM-Yara would result in an accurate taxonomic profiler that can handle frequent changes in databases. If DREAM-Yara is used to obtain the alignment files required as an input in SLIMM's pipeline, the overall memory requirement of the pipeline will drop significantly. This will broaden the accessibility towards groups and individuals without a big computing server. More importantly the availability of complete and up-to-date database coupled with SLIMM's high precision has a potential application for reliable microbiome (as well as pathogen) surveillance.

To use DREAM-Yara in a broader context other than metagenomic read mapping, we need a clustering method that does not require a taxonomy as an input. This is instrumental in order to generalize the read mapper to any reference sets. A considerable amount of investigation of clustering methods suitable for this task is required. Methods like Cd-hit [118] and a more recent method MeShClust [119] work well in clustering a decent set of sequences. Whether or not they scale to more extended set of genomes needs to be checked. In addition to scaling, to fully utilize the IBF for filtering, it is essential to have bin-sizes evenly distributed. The individual filters, corresponding to bins, in the IBF, has to be equally sized. If there are bins with significantly higher sizes than the rest, it would mean, either making all bloom filters large enough for these outliers or accepting the high false positive rate for the outliers. It is also essential that, the clustering method is transparent enough to add a new sequence into existing cluster.

The work on IBF can be further extended to different applications other than read mapping. There are ongoing efforts to apply it in searching RNA-Seq experiments from the SRA. Piro et al. have developed a metagenomic read classifier using the IBF which can easily handle the WGS (including draft genomes) version of RefSeq. If engineered carefully and a proper clustering method is devised, it could also speed up blast like searches.

Appendices

A.1. Command Line Options of select_refs.py

Command line options of select_refs.py under SLIMM pre-processing module

```
$ python preprocessing/select_refs.py --help
usage: select_refs.py [-h] [-g GROUPS] [-s] [-c] [-t TAXA_IDS]
                      [-d {refseq,GenBank}] -o OUTPUT_DIR

Selects microbial reference genomes (assemblies) to download based on various
criteria. (NB. This script does not download the actual FASTA files. Instead
it simply identifies a set of genomes/assemblies and their corresponding
download locations.)

optional arguments:
  -h, --help            show this help message and exit
  -g GROUPS, --groups GROUPS
                        Which group of microbes to consider any combination of
                        the letters [A], [B], [V] and [F] where B = Bacteria,
                        A = Archaea and V = Viruses and Viroids (default: AB)
  -s, --species_only    download one reference per species.
  -c, --complete        download only complete genomes (includes chromosome
                        level assembly)
  -t TAXA_IDS, --taxa_ids TAXA_IDS
                        comma separated list of taxonomic ids to be included
                        (in addition to --groups) into the reference database.
                        This way you might even add the genome of Eukaryotes.
                        e.g. the host genome
  -d {refseq,GenBank}, --database {refseq,GenBank}
                        From which database references should be downloaded
                        (default: refseq)
  -o OUTPUT_DIR, --output_dir OUTPUT_DIR
                        Path of a directory where (intermediate) results will
                        be saved
```

A.2. Command Line Options of collect_refs.py

Command line options of collect_refs under SLIMM pre-processing module

```
$ python preprocessing/collect_refs.py --help
usage: collect_refs.py [-h] -t TSV_FILE [-tr {1,2,3,4,5,6,7,8,9,10}]
                       [-bd BUFFER_DIR] -o OUTPUT_DIR

Download reference genomes using a list compiled by select_refs.py.

optional arguments:
  -h, --help            show this help message and exit
  -t TSV_FILE, --tsv_file TSV_FILE
                        Path to a TSV file containing genomes to download and
                        their corresponding FTP path. (a result from running
                        select_refs.py)
  -tr {1,2,3,4,5,6,7,8,9,10}, --threads {1,2,3,4,5,6,7,8,9,10}
                        number of threads for downloading in parallel in the
                        range 1..10 (default: 1)
  -bd BUFFER_DIR, --buffer_dir BUFFER_DIR
                        directory containing already downloaded files to avoid
                        repetitive downloads. Symbolic link will be created to
                        already existing files in this directory
  -o OUTPUT_DIR, --output_dir OUTPUT_DIR
                        Path of a directory where (intermediate) results will
                        be saved
```

A.3. Command Line Options of merge_refs.py

Command line options of merge_refs under SLIMM pre-processing module

```
$ python preprocessing/merge_refs.py --help
usage: merge_refs.py [-h] -t TSV_FILE -i INPUT_DIR -o OUTPUT_FILE

Merge downloaded reference genomes into one single FASTA file. Contigs will
be delimited by a sequence of NNNs

optional arguments:
  -h, --help            show this help message and exit
  -t TSV_FILE, --tsv_file TSV_FILE
                        Path to a TSV file containing genomes to download and
                        their corresponding FTP path. (a result from running
                        select_refs.py)
  -i INPUT_DIR, --input_dir INPUT_DIR
                        Path to directory containing downloaded FASTA files.
                        (a result from running collect_refs.py)
  -o OUTPUT_FILE, --output_file OUTPUT_FILE
```

A.4. Command Lines Used in SLIMM Evaluation

The following command lines were used to run different taxonomic profiling tools, in order to produces the results described in chapter 6.

```
/usr/bin/time -v \
  slimm --rank species --output-prefix ${SAMPLE_NAME} ${SLIMM_DB} ${SAMPLE_NAME}.sam

kraken-build --download-taxonomy --db ${K_DB};
kraken-build --db ${K_DB} --clean;
/usr/bin/time -v \
  kraken --fastq-input --output ${SAMPLE_NAME}.kr --db ${K_DB} ${SAMPLE_NAME}.fastq
/usr/bin/time -v \
  kraken-report --db ${K_DB} ${SAMPLE_NAME}.kr > ${SAMPLE_NAME}.kr.tsv

/usr/bin/time -v \
  gottcha.pl --prefix ${SAMPLE_NAME} --outdir ${OUT} --database ${GOTCHA_DB} \
  --input  ${SAMPLE_NAME}.fastq;

/usr/bin/time -v \
  mOTUs.pl --prefix ${SAMPLE_NAME} --output-directory ${OUT} ${SAMPLE_NAME}.fastq;
```

A.5. Detailed Runtime Across Multiple Read-Sets

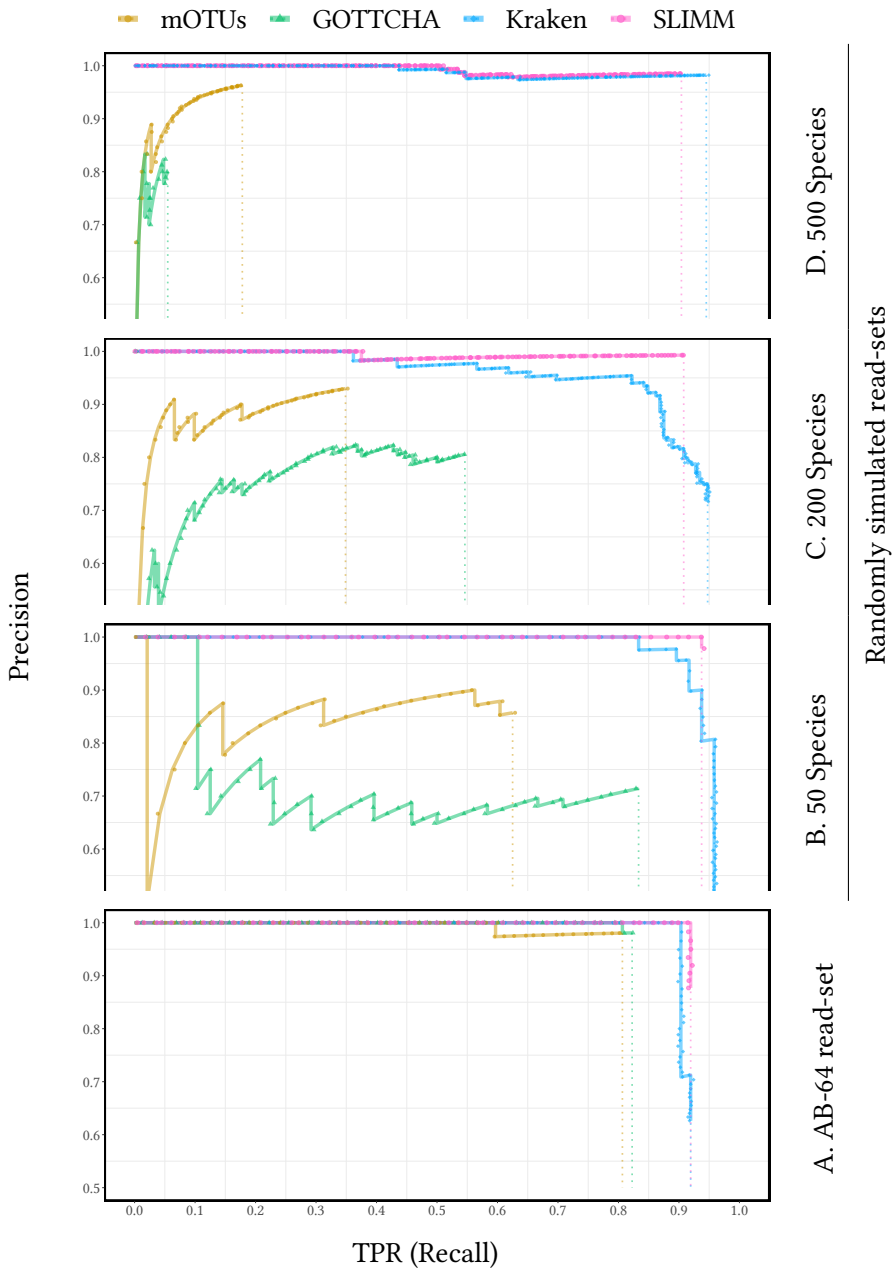
The following table shows the runtime required to run SLIMM and other methods on different read-sets.

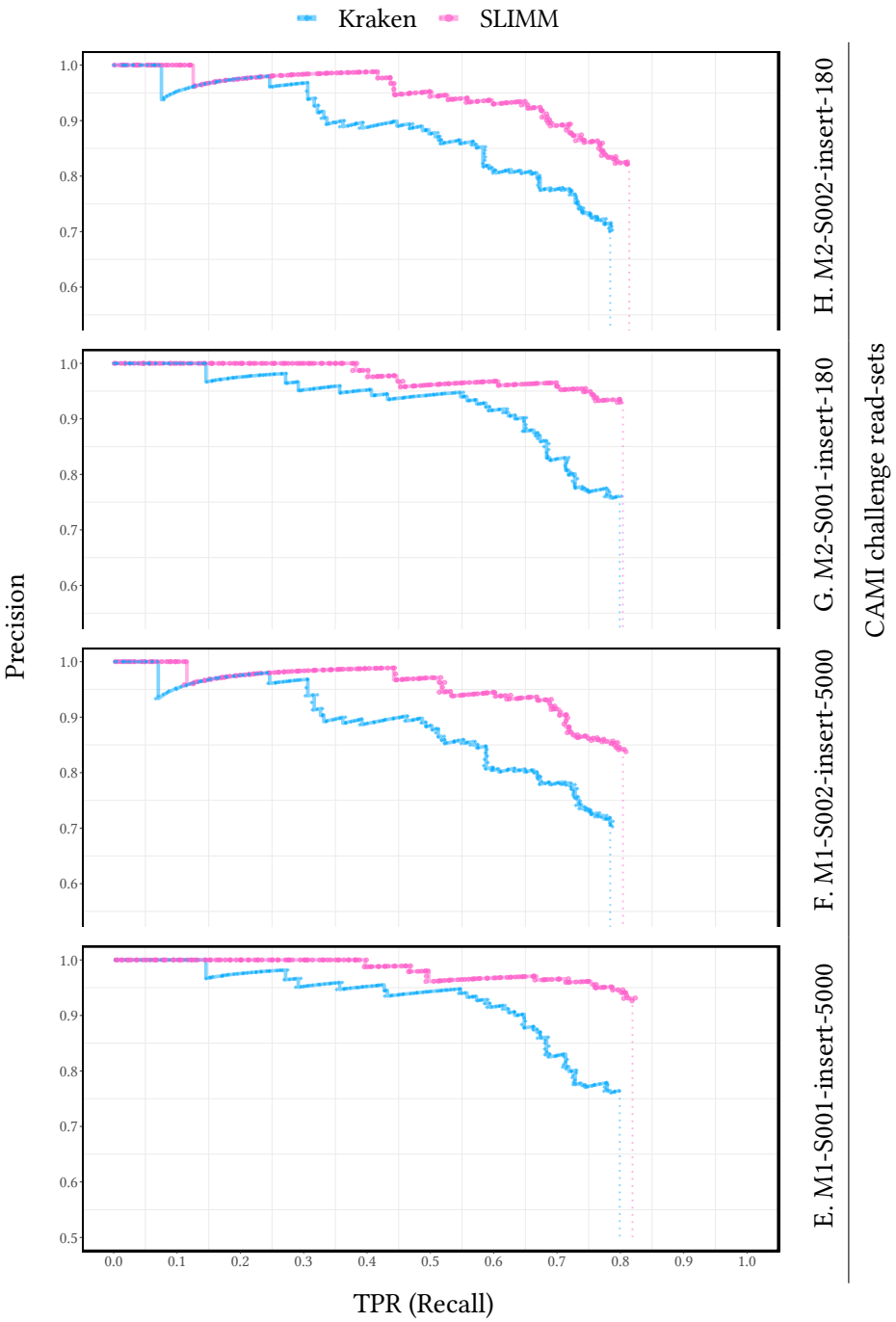
		Read Mapper		Taxonomic Profiler			
	Read-set	Yara	Bowtie2	SLIMM	Kraken	GOTTCHA	mOTUs
Mock	MG01	1403	1869	4	359	8428	6551
	MG02	93	474	100	68	334	211
	MG03	160	893	249	74	334	288
Mimic.Sim	MG04	306	344	14	314	1813	1289
	MG05	308	406	19	198	1979	1261
Random.Sim	MG06	373	710	50	200	1971	1286
	MG07	360	674	45	201	1889	1319
	MG08	368	737	53	200	1925	1285
	MG09	394	723	53	193	2052	1291
	MG10	424	720	54	200	2035	1336
	MG11	450	726	54	196	1937	1308
	MG12	414	746	61	198	2052	1300
	MG13	438	792	63	204	1917	1332
	MG14	418	736	61	197	2007	1315
Sum		5909	10550	880	2802	30673	21372
Average		422.1	753.6	62.9	200.1	2190.9	1526.6

A.6. Extra Precision - Recall Curves

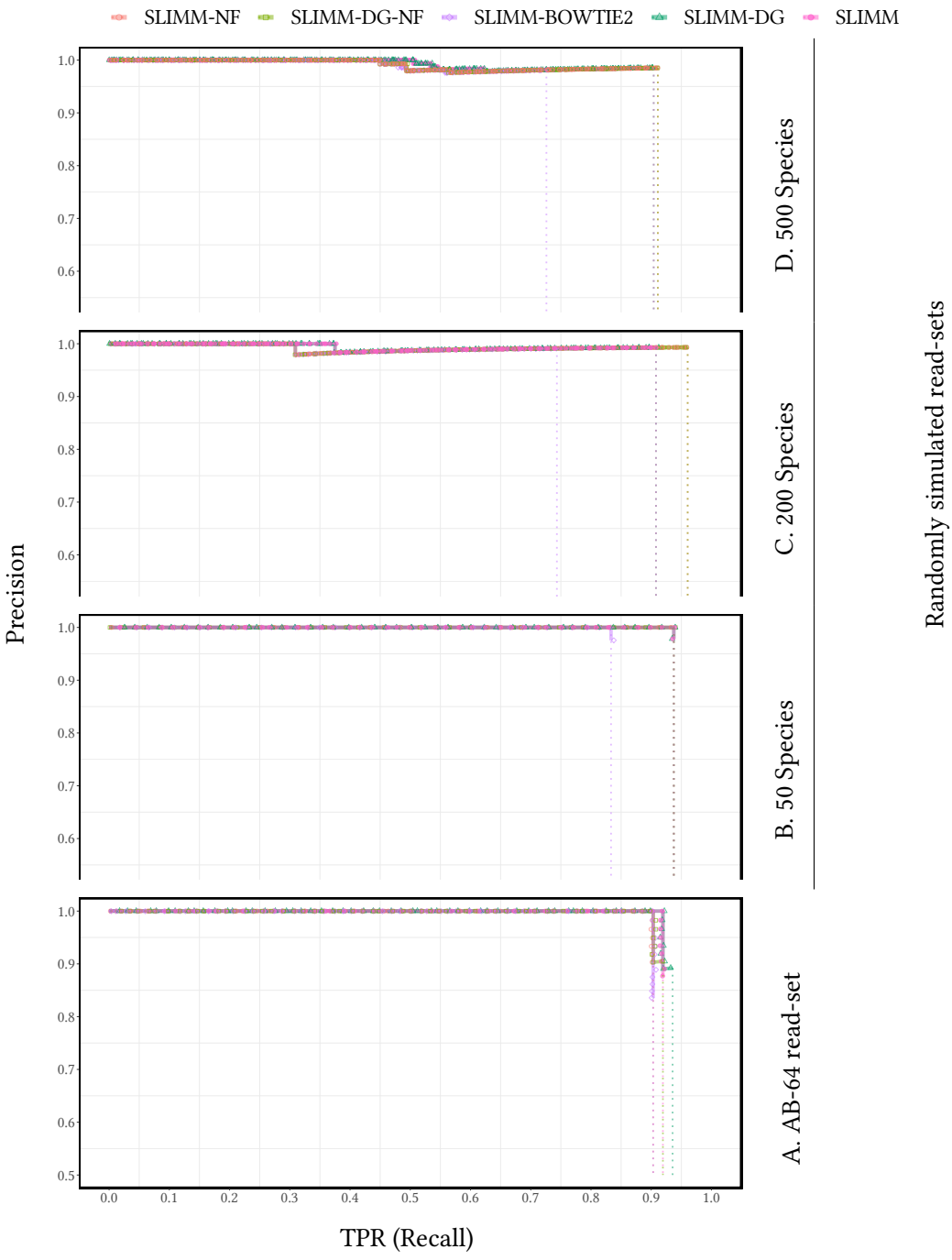
Here we show additional precision - recall curves (True Positive Rate (TPR)/recall drawn against precision.) for 8 different read-sets. SLIMM received the highest performance for all of the read-sets shown here by detecting most of the microorganisms in each sample while staying precise.

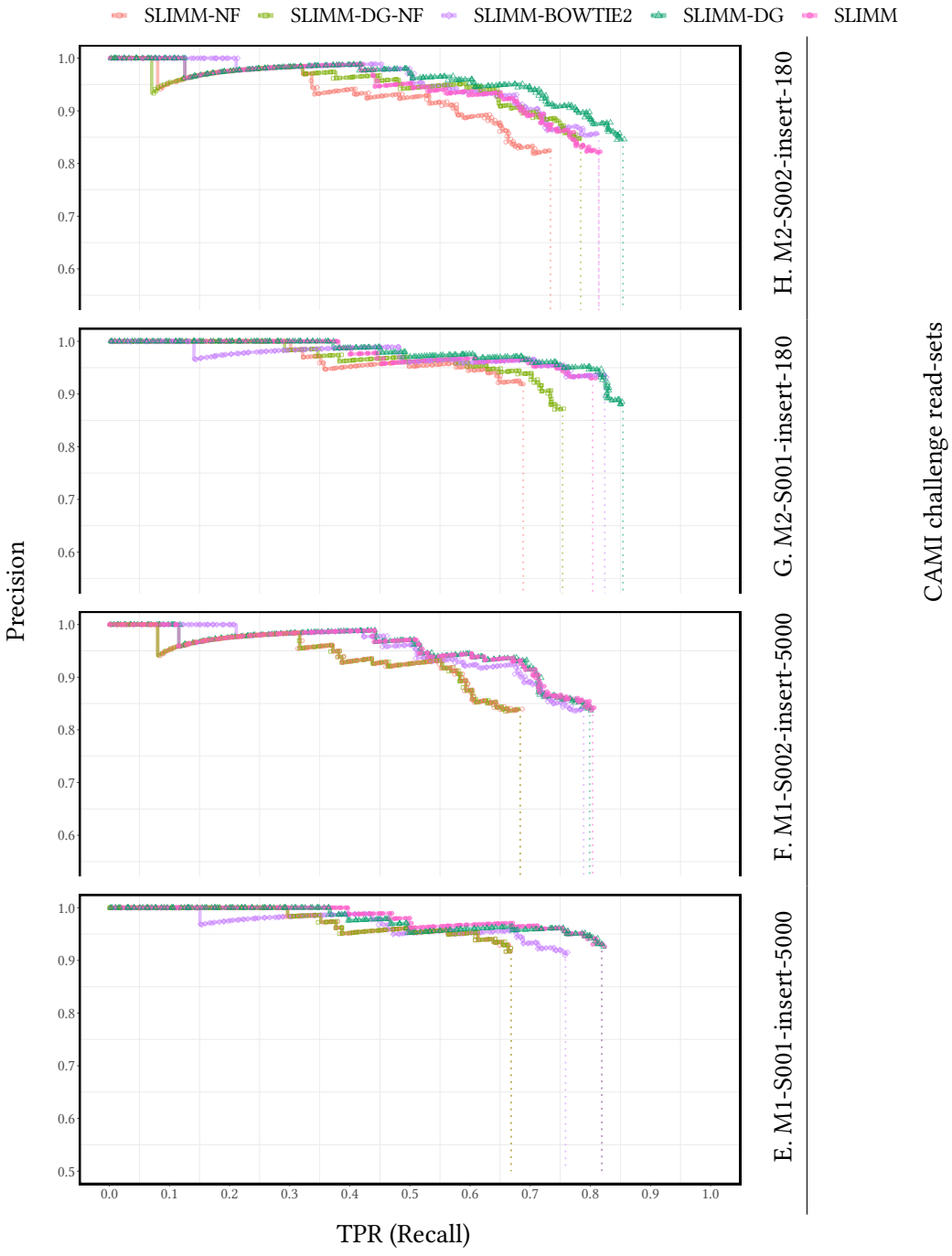
SLIMM vs Existing Methods




Kraken
SLIMM
Precision
TPR (Recall)
CAMI challenge read-sets
H. M2-S002-insert-180
G. M2-S001-insert-180
F. M1-S002-insert-5000
E. M1-S001-insert-5000

Different Versions of SLIMM

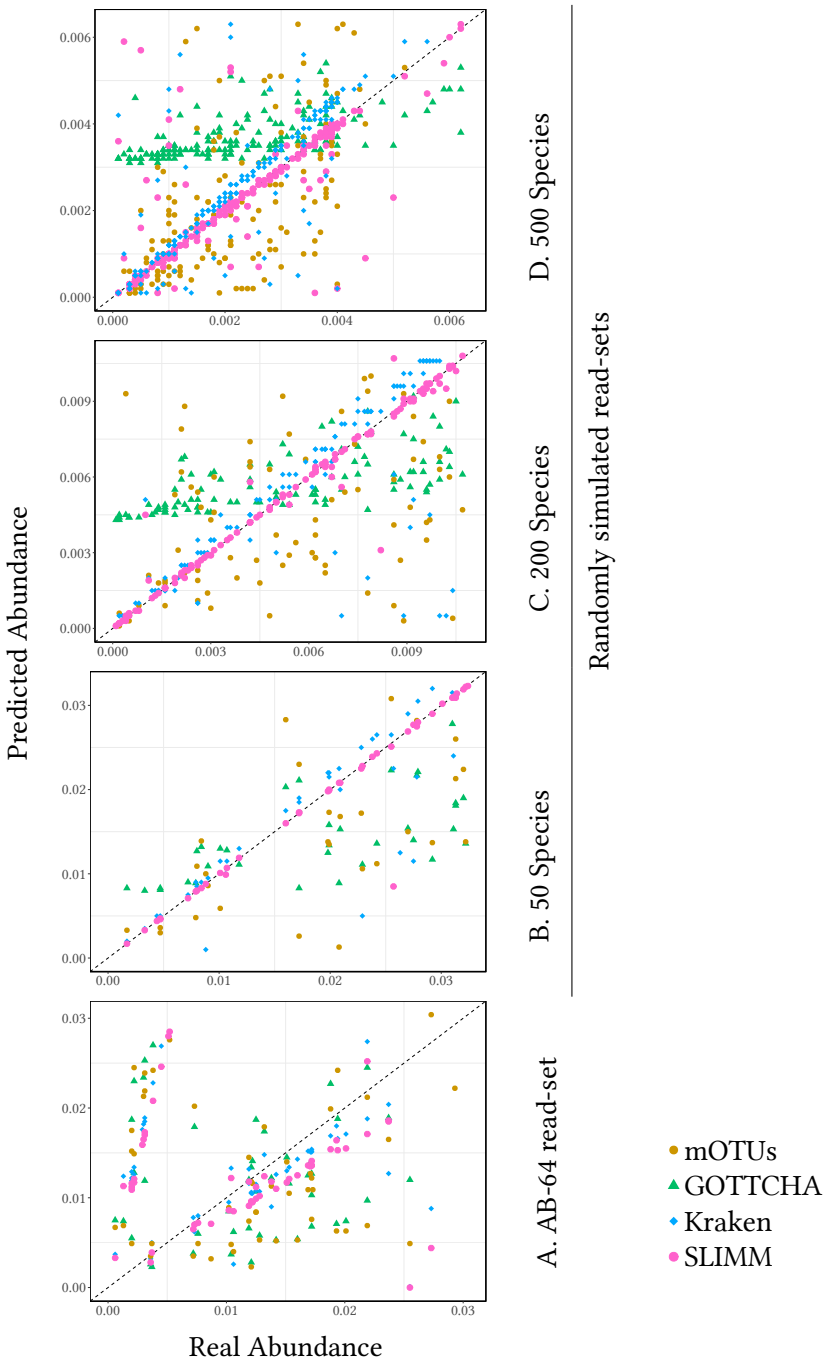


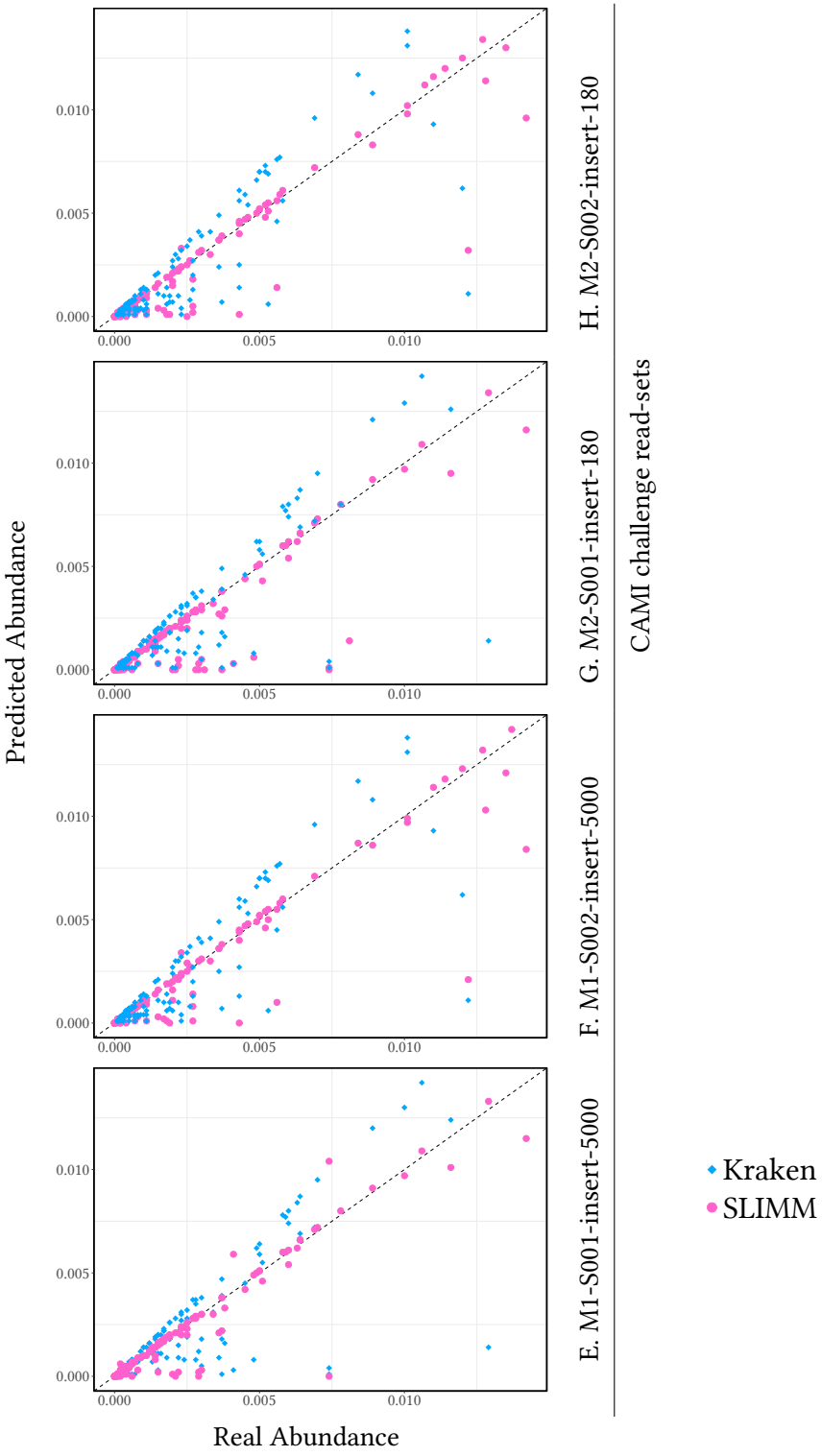

SLIMM-NF
SLIMM-DG-NF
SLIMM-BOWTIE2
SLIMM-DG
SLIMM
Precision
TPR (Recall)
CAMI challenge read-sets
H. M2-S002-insert-180
G. M2-S001-insert-180
F. M1-S002-insert-5000
E. M1-S001-insert-5000

A.7. Extra Scatter Plots

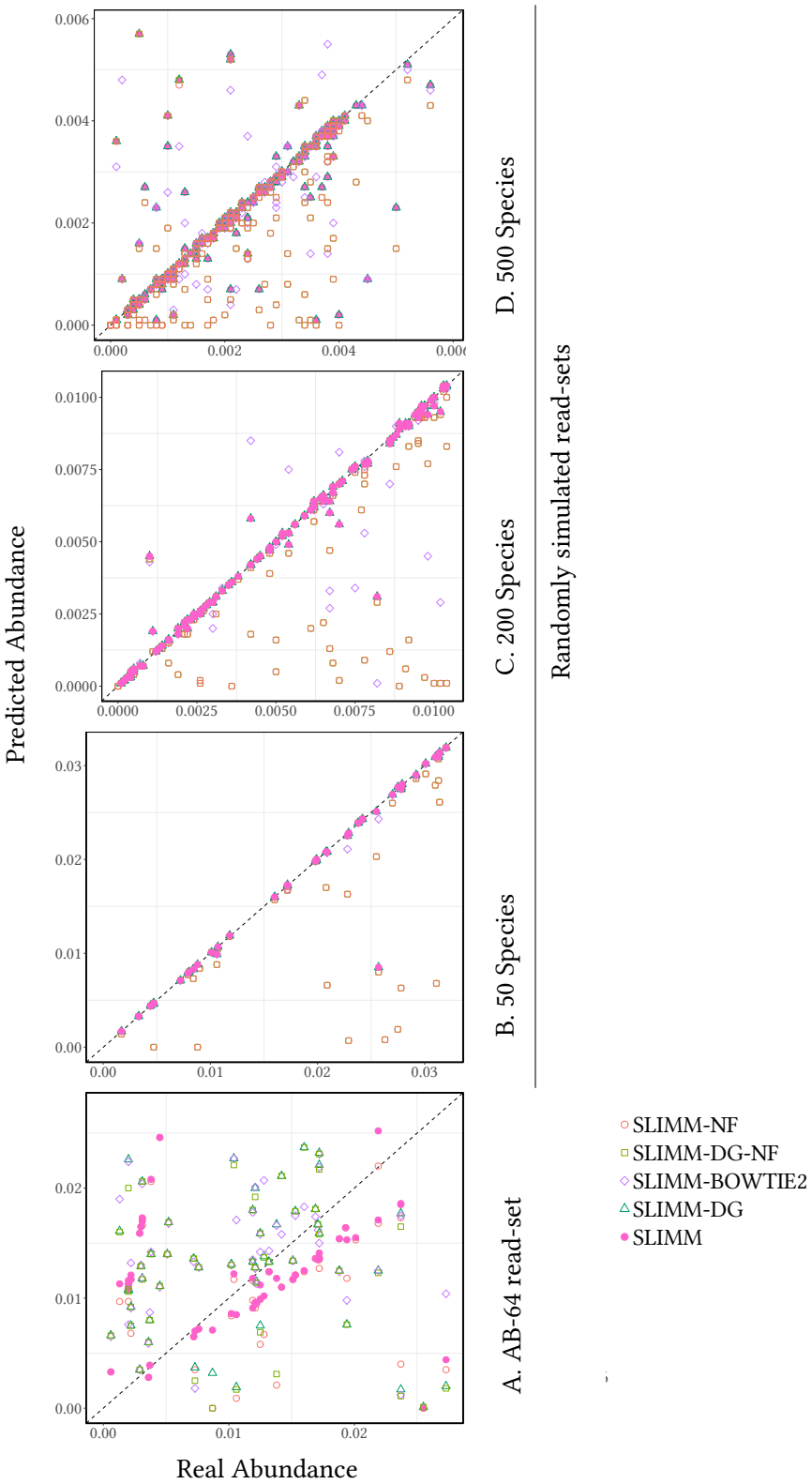
Here we provide additional scatter plots (true abundance in the simulation plotted against predicted abundance) for eight different read-sets.

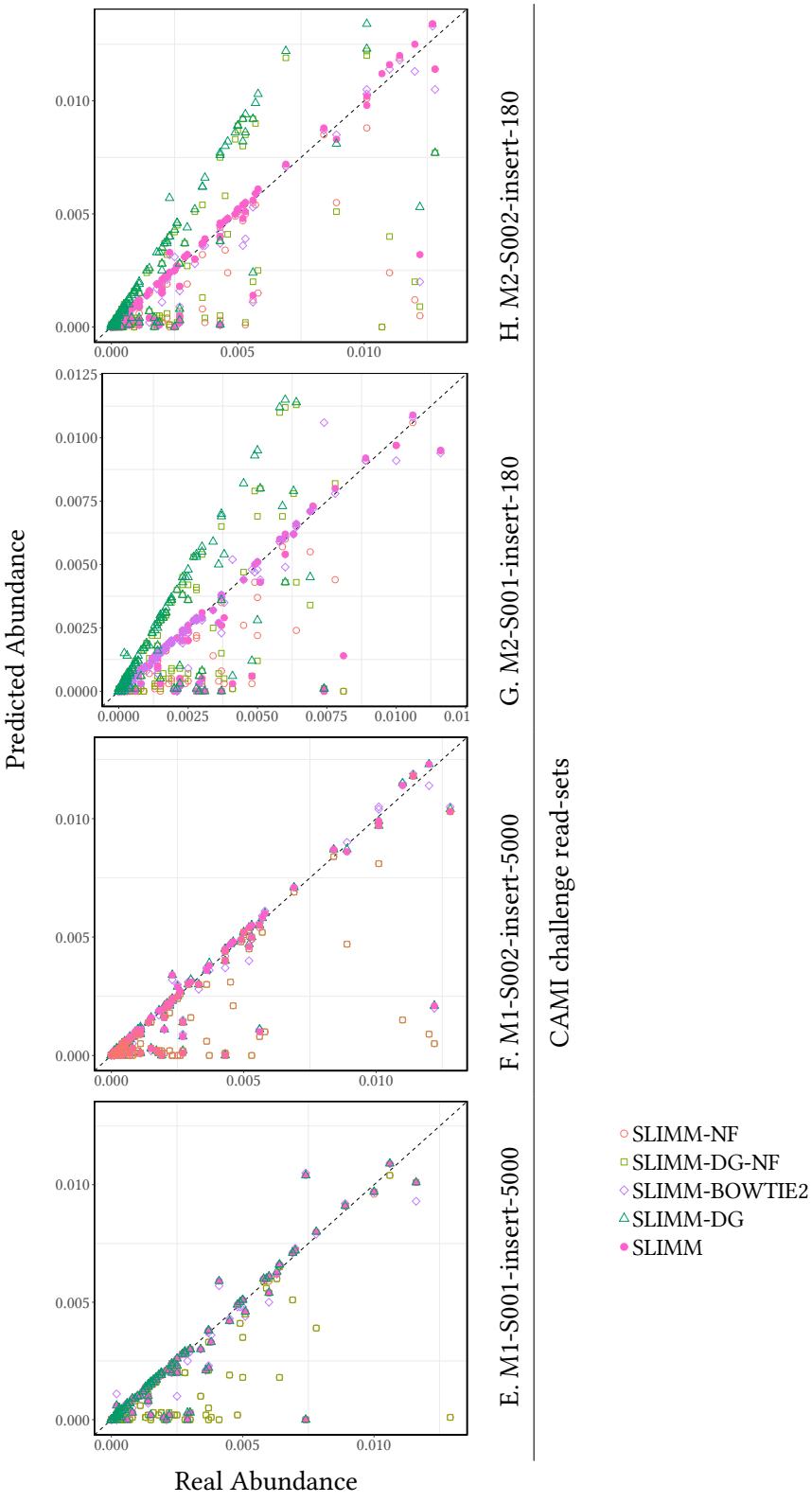
SLIMM vs Existing Methods




Predicted Abundance
Real Abundance
0.000
0.005
0.010
H. M2-S002-insert-180
G. M2-S001-insert-180
F. M1-S002-insert-5000
E. M1-S001-insert-5000
CAMI challenge read-sets
Kraken
SLIMM

Different Versions of SLIMM

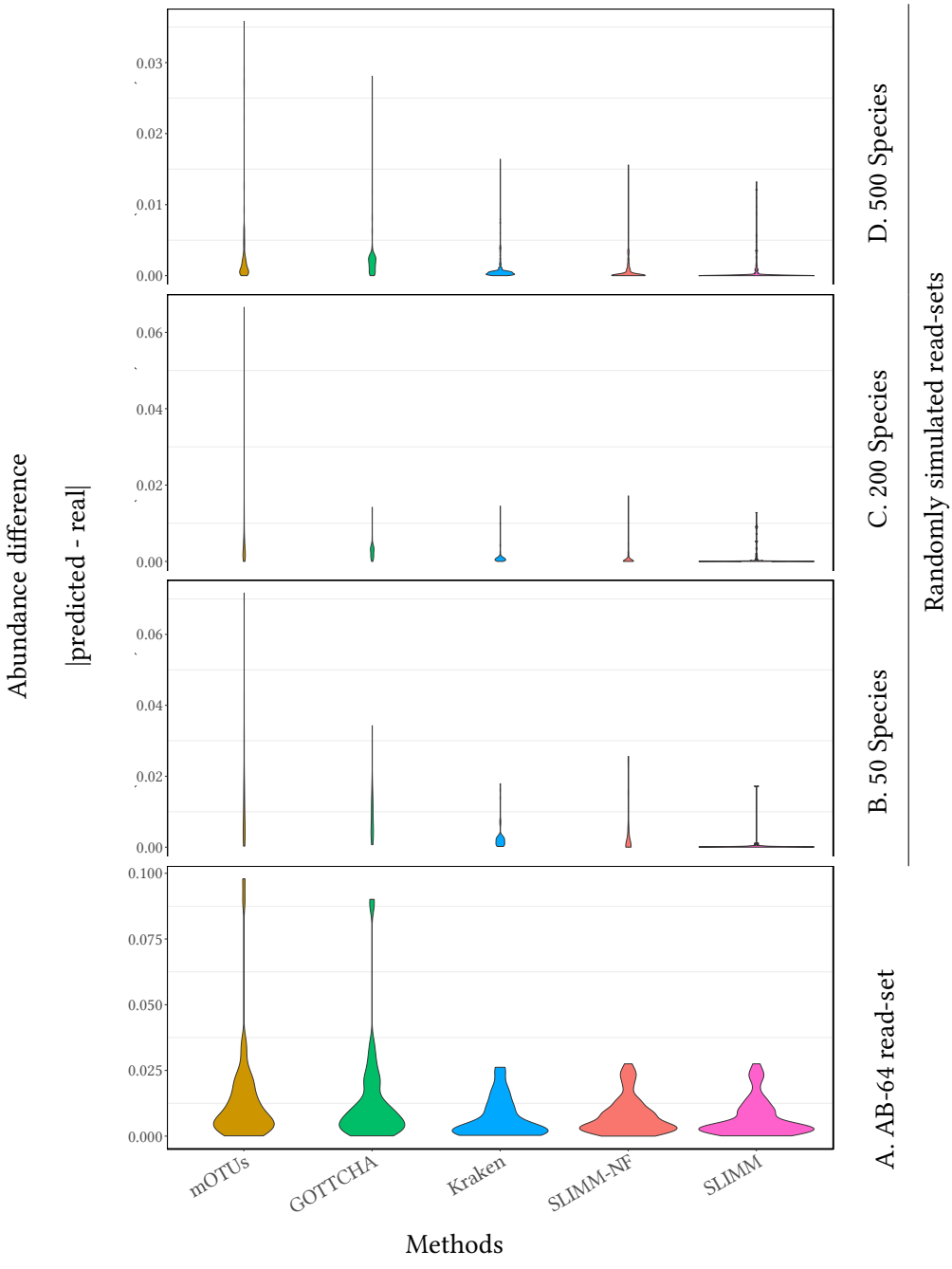


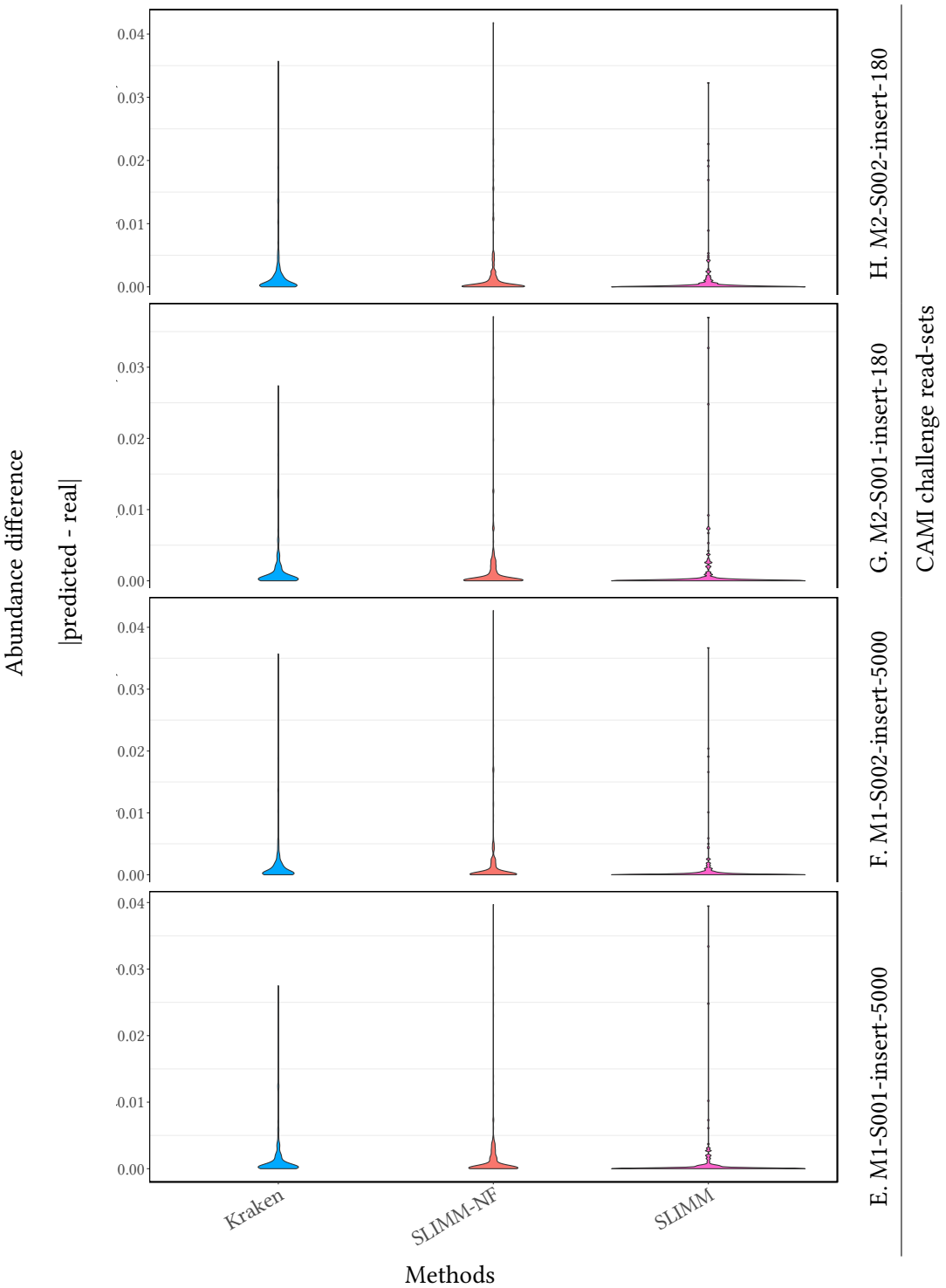

Predicted Abundance
Real Abundance
CAMI challenge read-sets
H. M2-S002-insert-180
G. M2-S001-insert-180
F. M1-S002-insert-5000
E. M1-S001-insert-5000
SLIMM-NF
SLIMM-DG-NF
SLIMM-BOWTIE2
SLIMM-DG
SLIMM

A.8. Extra Violin Plots

Here we provide more violin plots additional to the ones shown in 6.4.2. The violin plots show the distribution of divergence between predicted abundances and actual abundances across eight different read-sets.

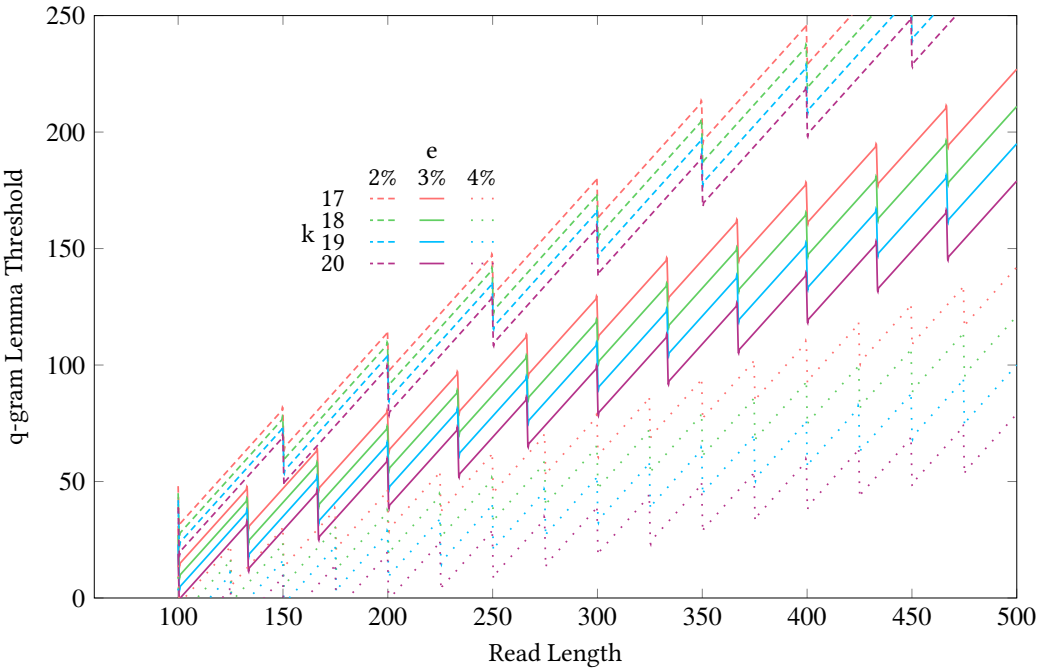
SLIMM vs Existing Methods




Abundance difference
|predicted - real|
0.04
0.03
0.02
0.01
0.00
Kraken
SLIMM-NF
SLIMM
Methods
E. M1-S001-insert-5000
F. M1-S002-insert-5000
G. M2-S001-insert-180
H. M2-S002-insert-180
CAMI challenge read-sets

A.9. q-gram Lemma Threshold

A plot showing how the threshold of the q-gram lemma react on changing values of read length, k-mer sizes, and percent error rates. The absolute errors has to be rounded up in order to preserve the guarantee provided by the Lemma. While using DREAM-Yara, one has to choose the maximum error rate carefully by considering the read length the k-mer used to build the filter and the maximum error rate desired. DREAM-Yara prompts the user when the chosen parameters result in a threshold equal to zero. This makes the mapper extremely slow as it turns off the filter.



A.10. Reference Datasets

Bellow are the clustered reference genomes used in benchmarking DREAM-Yara in chapter 8. The table lists zipped filenames corresponding to different binning sizes. Both original clustering and update sets are listed. The files contain the RefSeq database of prokaryotic genomes after clustering them into the specified bins. To build the full index it is enough to concatenate the individual files from one of the binning schemes and use them as single multi-FASTA file. The database files can be downloaded from the FTP site ftp://ftp.mi.fu-berlin.de/pub/dadi/dream_yara_data/.

	Number of bins	File name
Full set from Sept. 26, 2017	64 bins	A_B_refseq_20170926_taxo_64.tar.gz
	256 bins	A_B_refseq_20170926_taxo_256.tar.gz
	1024 bins	A_B_refseq_20170926_taxo_1024.tar.gz
Update set from: Sept. 26, 2017	64 bins	A_B_refseq_20170926_taxo_64_up.tar.gz
	256 bins	A_B_refseq_20170926_taxo_256_up.tar.gz
	1024 bins	A_B_refseq_20170926_taxo_1024_up.tar.gz

A.11. Command Lines Used in DREAM-Yara Evaluation

The following command lines were used in the benchmarking process of DREAM-Yara, in order to produces the results described in chapter 8.2. We grouped the set of command lines into indexing, read mapping, and Rabema benchmark.

Indexing

```
# Yara indexer
/usr/bin/time -v \
    yara_indexer \
        --output-prefix ${reference}.yara.index \
        ${reference}.fasta;

# DREAM-Yara indexer - IBF
/usr/bin/time -v \
    yara_build_filter \
        --number-of-bins ${nb} \
        --threads 8 --kmer-size 18 --filter-type bloom \
        --bloom-size 16 --num-hash 3 \
        --output-file ${reference}.dyara_taxo_${nb}.filter;

# DREAM-Yara indexer - IBF - update
/usr/bin/time -v \
    yara_update_filter \
        --threads 8  \
        ${reference}.dyara_taxo_${nb}_up.filter \
        ${reference}_taxo_${nb}_up/*.fasta;

# DREAM-Yara indexer - FM Index
/usr/bin/time -v \
    yara_indexer_dis \
        --threads 8 \
         --output-prefix ${bin_indices_dir} <bin_ref_dir>/*.fasta;

# DREAM-Yara indexer - FM Index - update
/usr/bin/time -v \
     yara_indexer_dis \
         --threads 8 \
         --output-prefix ${bin_indices_dir}
 <bin_ref_update_dir>/*.fasta;

# Distributed yara indexer - FM Index
# Same as DREAM-Yara indexer - FM Index

# Distributed yara indexer - FM Index - update
# Same as DREAM-Yara indexer - FM Index - update


# Bowtie2 indexer
/usr/bin/time -v \
    bowtie2-build \
        --threads 8  \
        ${reference}.fasta \
        ${reference}.bowtie2.index;


. . .
```

```
# BWA indexer
/usr/bin/time -v \
     bwa index  -a bwtsw  \
     -p ${reference}.bwa.index  \
     ${reference}.fasta;

# GEM indexer
/usr/bin/time -v \
     gem-indexer \
         --threads 8  \
         -i ${reference}.fasta  \
         -o ${reference}.gem.index;

# DIDA(-BWA) indexer
/usr/bin/time -v \
    sh -c 'ulimit -n $((nb * 2));
    mkdir didabwa.dir; cd didabwa.dir;
    prt \
        --partition ${nb} ${reference}.fasta && \
    seq ${nb} | xargs -P 8 -I {}  \
        bwa index mref-{}.fa;'
```

Read Mapping

```
# Yara mapper
/usr/bin/time -v \
    yara_mapper \
        --threads 8 \
        --error-rate 4 \
        --secondary-alignments record \
        --rabema-alignments \
        --sensitivity full \
        --version-check OFF -v \
        --output-file ${algn}.yara.bam \
        ${reference}.yara.index \
        ${reads}.fastq 2> ${algn}.yara.bam.log;

# Distributed Yara mapper
/usr/bin/time -v \
    sh -c 'seq 0 $((${nb}-1)) | \
        xargs -I {} sh -c " yara_mapper  \
            --threads 8 \
            --error-rate 4 \
            --secondary-alignments record \
            --rabema-alignments \
            --sensitivity full \
            --version-check OFF -v \
            --output-file ${algn}.distyara_taxo_${nb}/{}.bam \
            ${bin_indices_dir}{} ${reads}.fastq " ';

# Distributed Yara mapper - Merge results
/usr/bin/time -v \
     seq 0 $((${nb}-1)) | xargs -I {} \
        samtools view \
            -H ${algn}.distyara_taxo_${nb}/{}.bam | \
            grep @SQ | \
            awk -F":|\t" '{print $3"\t"$5}' > header.h;

    seq 0 $((${nb}-1)) | xargs -I {} \
        samtools view -\@ 8 \
            -F 4 ${algn}.distyara_taxo_${nb}/{}.bam | \
            samtools view \
            -b -t header.h - > ${algn}.distyara_taxo_${nb}.bam;

# DREAM-Yara mapper
/usr/bin/time -v \
    yara_mapper_dis \
         --threads 8 \
         --filter-type bloom \
         --reads-batch 1000000 \
         --error-rate 4 \
         --secondary-alignments record \
         --rabema-alignments \
         --sensitivity full \
         --version-check OFF -v \
         --output-file ${algn}.dyara_taxo_${nb}.bam \
         --bloom-filter ${reference}.dyara_taxo_${nb}.filter;
        ${bin_indices_dir} ${reads}.fastq;




. . .
```

```
# Bowtie2 mapper
/usr/bin/time -v \
    bowtie2 \
        -x ${reference}.bowtie2.index \
        -U ${reads}.fastq  \
        --threads 8  -k 100  \
        --end-to-end  \
        --rg-id none \
        --rg SM:none | \
        samtools view -@ 8 -bS - > ${algn}.bowtie2.bam;

# BWA mapper
/usr/bin/time -v \
    /bin/bwa mem \
    -t 8 \
    -a ${reference}.bwa.index ${reads}.fastq \
    -R '@RG\tID:none\tSM:none'| \
    samtools view -bS - > ${algn}.bwa.bam;

# GEM mapper
/usr/bin/time -v \
    sh -c ' gem-mapper \
        --threads 8 \
        --quality-format ignore \
        -m 0.04 -e 0.04 \
        -I ${reference}.gem.index.gem \
        -i ${reads}.fastq && \
    gem-2-sam \
        --quality-format offset-33 \
        --sequence-lengths \
        --index ${reference}.gem.index.gem \
        --expect-single-end-reads | \
        samtools view -bS - > ${algn}.gem.bam';

# DIDA(-BWA) mapper
/usr/bin/time -v \
    sh -c 'ulimit -n $((nb * 2));
        mkdir didabwa.dir;
        cd didabwa.dir; dsp \
        --partition ${nb}  \
        --threads 8 \
        --alen 19 \
        --bmer 19 \
        --step 1 \
        --fq \
        --se  ${reads}.fastq && \
        seq ${nb} | xargs -P 8 -I {} \
        sh -c "bwa mem \
            -o aln-{}.sam mref-{}.fa \
            mreads-{}.fastq"; \
        mrg \
            --partition ${nb} \
            --aligner bwa \
            --mode fast;'
```

Rabema Benchmark

```
# Rabema gold standard preparation
/usr/bin/time -v \
    ./bin/razers3 \
    --thread-count 0 \
    --percent-identity 96 \
    --recognition-rate 100 \
    --distance-range 0 \
    --dont-shrink-alignments \
    --max-hits 10000000 \
    --version-check OFF -v \
    --output ${algn}.razers3_gold_unprep.bam \
    ${reference}.fasta ${reads}.fastq;

/usr/bin/time -v \
    ./bin/samtools sort -@ 8 \
    -n ${algn}.razers3_gold_unprep.bam \
    ${algn}.razers3_gold_unprep.qname;

/usr/bin/time -v \
    ./bin/rabema_prepare_sam \
        -i ${algn}.razers3_gold_unprep.qname.bam \
        -o ${algn}.razers3_gold_prep.bam;

/usr/bin/time -v \
    ./bin/samtools sort -@ 8 \
    ${algn}.razers3_gold_prep.bam \
    ${algn}.razers3_gold_prep.coord;

/usr/bin/time -v \
    ./bin/rabema_build_gold_standard \
    --max-error 4 \
    --distance-metric edit \
    --reference ${reference}.fasta \
    --in-bam ${algn}.razers3_gold_prep.coord.bam \
    --out-gsi razers3_gold.e4.gsi.gz

# Process results of mappers
./bin/samtools sort -@ 8 \
    -n ${algn}.distyara_taxo_${nb}.bam ${algn}.distyara_taxo_${nb}.qname;
./bin/samtools sort -@ 8 \
    -n ${algn}.dyara_taxo_${nb}.bam ${algn}.dyara_taxo_${nb}.qname;
./bin/samtools sort -@ 8 \
    -n ${algn}.yara.bam ${algn}.yara.qname;
./bin/samtools sort -@ 8 \
    -n ${algn}.bowtie2.bam ${algn}.bowtie2.qname;
./bin/samtools sort -@ 8 \
    -n ${algn}.bwa.bam ${algn}.bwa.qname;
./bin/samtools sort -@ 8 \
    -n ${algn}.gem.bam ${algn}.gem.qname;

# Run Rabema evaluation
/usr/bin/time -v \
    ./bin/rabema_evaluate \
        --DONT-PANIC \
        --max-error 4 --trust-NM \
        --distance-metric edit \
        --benchmark-category all-best \
        --extra-pos-tag XP \
        --reference ${reference}.fasta \
        --in-bam ${algn}.${mapper}.qname.bam \
        --in-gsi razers3_gold.e4.gsi.gz \
        --out-tsv ${algn}.${mapper}.e4.all-best.rabema_report_tsv
.
```

Abstract

Microorganisms, typically occurring as large, species diverse communities, are a ubiquitous part of nature. These communities are a vital part of their environment, influencing it through various layers of interaction. Host-associated microbial communities are particularly scrutinized for their influence on the host's health. Additionally, there is a growing interest in microbial communities due to their role in livestock, agriculture, waste treatment, mining, and biotechnology. Metagenomics is a relatively young scientific field that aims to study such microbial communities based on genetic material recovered directly from an environment. Advances in DNA sequencing have enabled us to perform taxonomic profiling, i.e. to identify microbial species quantitatively and qualitatively at increasing depth.

In whole genome shotgun sequencing (WGS), environmental DNA is taken directly from an environment and sequenced after being fragmented, without PCR amplification. Taxonomic profiling methods based on such sequencing data introduce less PCR bias compared to their amplicon based counterparts such as 16S-rDNA based profiling methods. However, the challenges posed by the enormous and redundancy of databases and the high degree homology among reference genomes of microorganisms put WGS methods at a disadvantage. In this thesis, we will present and discuss two separate computational methods that address both challenges.

The first method is a taxonomic profiler that leverages coverage landscapes created by mapping sequencing reads across reference genomes to address the challenge posed by homologous regions of genomes. By carefully evaluating the coverage profile of reference genomes we drop spurious references from consideration. This filtration strategy results in more uniquely mapping reads to the remaining reference genomes improving both the resolution and accuracy of the taxonomic profiling process. We have also shown that this method improves the quality of relative abundances assigned to each detected member organism.

The second method is a distributed read mapper which addresses the issue of large and frequently changing databases by systematically partitioning it into smaller bins. It significantly reduces the time, and computational resources required to build indices from such large databases by orders of magnitudes and updates can be performed very quickly in a few minutes compared to days in earlier methods. To achieve a competitive mapping speed while maintaining many small indices, we implemented a novel, fast and lightweight filtering data structure called interleaved bloom filter. With that, we are able to achieve the described improvements in the index building and updating time without compromising the read-mapping speed.

Zusammenfassung in deutscher Sprache

Mikroorganismen, typischerweise in Form von großen Gemeinschaften aus einer Vielzahl von Spezies, sind ein allgegenwärtiger Bestandteil unserer Umwelt. Solche Gemeinschaften sind ein wesentlicher Bestandteil ihrer Umgebung und beeinflussen diese auf verschiedenen Ebenen. Besonders Wirt-assoziierte Mikroben werden wegen ihres Einflusses auf die menschliche Gesundheit intensiv untersucht. Darüber hinaus entwickelt sich ein wachsendes Interesse an mikrobiellen Gemeinschaften wegen ihrer Rolle in der Landwirtschaft, Abfalltechnik, im Bergbau und in der Biotechnologie. Metagenomik ist ein vergleichsweise neues wissenschaftliches Feld, welches mikrobielle Gemeinschaften auf der Basis von genetischem Material aus einer definierten Umgebung untersucht. Technische Fortschritte bei der DNA Sequenzierung haben es möglich gemacht, auf diese Weise taxonomisches Profiling durchzuführen, d.h. die mikrobiellen Spezies qualitativ und quantitativ zu erfassen.

Bei der *whole genome shotgun sequencing (WGS)* Methode wird die DNA aus der Probe direkt fragmentiert und sequenziert. Taxonomische Profiling-Methoden, welche auf diesem Verfahren beruhen, sind weniger anfällig für PCR Biase im Vergleich zu anderen Methoden, wie z.B. 16S-rDNA basierten Verfahren. Allerdings stellt hierbei die enorme Größe und Redundanz der Datenbanken sowie der hohe Grad an Homologie unter den in den Datenbanken erfassten Organismen einen Nachteil dar. In dieser Arbeit stellen wir zwei rechnergestützte Verfahren vor, die beide Probleme adressieren.

Die erste Methode ist ein taxonomischer Profiler, mit dem Ziel, die Mehrfachzuweisungen von Reads zu Referenzsequenzen homologer Spezies auf der Basis der unterschiedlichen Abdeckungsprofile zu korrigieren. Durch die sorgfältige Auswertung der Read-Abdeckungen werden hierbei falsch positive Referenzgenome von der Auswahl entfernt. Durch diese Filterstrategie erhöht sich die Genauigkeit und Auflösung des Verfahrens, da ein größerer Teil der Reads eindeutig einem Genom zugeordnet werden kann. Wir zeigen darüberhinaus, dass durch die Methode auch die Häufigkeiten der Organismen präziser bestimmt werden können.

Die zweite Methode ist ein verteilter Read-Mapper, welcher das Problem der großen und sich häufig ändernden Referenzdatenbanken in der Metagenomik dadurch adressiert, dass die Referenzdatenbanken systematisch in Partitionen unterteilt werden. Hierdurch kann der Bedarf an Rechenzeit und Speicher für die Berechnung von Indizes um Größenordnungen verringert und Index Aktualisierungen in wenigen Minuten anstelle von Tagen durchgeführt werden. Um trotz der hohen Zahl von kleinen Indizes eine hohe Performanz beim alignieren der Reads zu erreichen, haben wir eine neue, schnelle und kompakte Filter-Datenstruktur entwickelt, den interleaved bloom filter. Dadurch sind wir in der Lage, die beschriebenen Verbesserungen beim Erzeugen und Aktualisieren der Indizes ohne Einbußen bei der Mapping-Geschwindigkeit zu erreichen.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel und Quellen verwendet habe. Ich erkläre weiterhin, dass ich die vorliegende Arbeit oder deren Inhalt nicht in einem früheren Promotionsverfahren eingereicht habe.

Berlin, den 28. Januar 2019
Temesgen Hailemariam Dadi

Bibliography

[1] T. Betsy and J. Keogh. *Microbiology DeMYSTiFieD*, volume 53. 2012. ISBN 0071471340. doi:10.1036/0071446508.

[2] C. R. Woese, O. Kandler, and M. L. Wheelis. Towards a natural system of organisms: proposal for the domains Archaea, Bacteria, and Eucarya. *Proceedings of the National Academy of Sciences*, 87(12):4576–4579, 1990. ISSN 0027-8424. doi:10.1073/pnas.87.12.4576.

[3] R. Amils, C. Ellis-Evans, and H. Hinghofer-Szalkay. Life in extreme environments. *Life in Extreme Environments*, 409(September 2000):1–450, 2007. ISSN 00280836. doi:10.1007/978-1-4020-6285-8.

[4] E. E. Allen and J. F. Banfield. Community genomics in microbial ecology and evolution. *Nature Reviews Microbiology*, 3(6):489–498, 2005. ISSN 17401526. doi:10.1038/nrmicro1157.

[5] K. Zengler and L. S. Zaramela. The social network of microorganisms - How auxotrophies shape complex communities. *Nature Reviews Microbiology*, 16(6): 383–390, 2018. ISSN 17401534. doi:10.1038/s41579-018-0004-5.

[6] S. Mitri and K. Richard Foster. The Genotypic View of Social Interactions in Microbial Communities. *Annual Review of Genetics*, 47(1):247–273, 2013. ISSN 0066-4197. doi:10.1146/annurev-genet-111212-133307.

[7] C. Méndez-García, A. I. Peláez, V. Mesa, J. Sánchez, O. V. Golyshina, and M. Ferrer. Microbial diversity and metabolic networks in acid mine drainage habitats. *Frontiers in Microbiology*, 6:475, 2015. ISSN 1664-302X. doi:10.3389/fmicb.2015.00475.

[8] B. J. Baker and J. F. Banfield. Microbial communities in acid mine drainage. *FEMS Microbiology Ecology*, 44(2):139–152, 2003. ISSN 01686496. doi:10.1016/S0168-6496(03)00028-X.

[9] L. X. Chen, M. Hu, L. N. Huang, Z. S. Hua, J. L. Kuang, S. J. Li, and W. S. Shu. Comparative metagenomic and metatranscriptomic analyses of microbial communities in acid mine drainage. *ISME Journal*, 9(7):1579–1592, 2015. ISSN 17517370. doi:10.1038/ismej.2014.245.

[10] L. xing Chen, L. nan Huang, C. Méndez-García, J. liang Kuang, Z. shuang Hua, J. Liu, and W. sheng Shu. Microbial communities, processes and functions in acid mine drainage ecosystems. *Current Opinion in Biotechnology*, 38:150–158, 2016. ISSN 18790429. doi:10.1016/j.copbio.2016.01.013.

[11] W. H. King. Isotope shift and configuration interaction in U i. *Journal of Physics B: Atomic and Molecular Physics*, 12(3):383–386, 1979. ISSN 00223700. doi:10.1088/0022-3700/12/3/015.

[12] T. P. Curtis, W. T. Sloan, and J. W. Scannell. Estimating prokaryotic diversity and its limits. *Proceedings of the National Academy of Sciences*, 99(16):10494–10499, 2002. ISSN 0027-8424. doi:10.1073/pnas.142680199.

[13] N. Fierer. Embracing the unknown: Disentangling the complexities of the soil microbiome. *Nature Reviews Microbiology*, 15(10):579–590, 2017. ISSN 17401534. doi:10.1038/nrmicro.2017.87.

[14] S. Oh, A. Caro-Quintero, D. Tsementzi, N. DeLeon-Rodriguez, C. Luo, R. Poretsky, and K. T. Konstantinidis. Metagenomic insights into the evolution, function, and complexity of the planktonic microbial community of Lake Lanier, a temperate freshwater ecosystem. *Applied and Environmental Microbiology*, 77(17):6000–6011, 2011. ISSN 00992240. doi:10.1128/AEM.00107-11.

[15] R. J. Newton, S. E. Jones, A. Eiler, K. D. McMahon, and S. Bertilsson. A Guide to the Natural History of Freshwater Lake Bacteria. *Microbiology and Molecular Biology Reviews*, 75(1):14–49, 2011. ISSN 1092-2172. doi:10.1128/MMBR.00028-10.

[16] K. Deiner, J. C. Walser, E. Mächler, and F. Altermatt. Choice of capture and extraction methods affect detection of freshwater biodiversity from environmental DNA. *Biological Conservation*, 183:53–63, 2015. ISSN 00063207. doi:10.1016/j.biocon.2014.11.018.

[17] A. W. Reid. Digital subtraction angiography. *Scottish Medical Journal*, 32(6): 172–177, 1987. ISSN 1365294X. doi:10.1111/mec.12985.

[18] Llobet-Brossa, Rosselló-Mora, and Amann. Microbial Community Composition of Wadden Sea Sediments as Revealed by Fluorescence In Situ Hybridization. *Applied and environmental microbiology*, 64(7):2691–2696, 1998.

[19] J. C. Venter, K. Remington, J. F. Heidelberg, A. L. Halpern, D. Rusch, J. A. Eisen, D. Wu, I. Paulsen, K. E. Nelson, W. Nelson, D. E. Fouts, S. Levy, A. H. Knap, M. W. Lomas, K. Nealson, O. White, J. Peterson, J. Hoffman, R. Parsons, H. Baden-Tillson, C. Pfannkoch, Y. H. Rogers, and H. O. Smith. Environmental Genome Shotgun Sequencing of the Sargasso Sea. *Science*, 304(5667):66–74, 2004. ISSN 00368075. doi:10.1126/science.1093857.

[20] M. Bahram, F. Hildebrand, S. K. Forslund, J. L. Anderson, N. A. Soudzilovskaia, P. M. Bodegom, J. Bengtsson-Palme, S. Anslan, L. P. Coelho, H. Harend, J. Huerta-Cepas, M. H. Medema, M. R. Maltz, S. Mundra, P. A. Olsson, M. Pent, S. Põlme, S. Sunagawa, M. Ryberg, L. Tedersoo, and P. Bork. Structure and function of the global topsoil microbiome. *Nature*, 560(7717):233–237, 2018. ISSN 14764687. doi:10.1038/s41586-018-0386-6.

[21] E. Karsenti, S. G. Acinas, P. Bork, C. Bowler, C. de Vargas, J. Raes, M. Sullivan, D. Arendt, F. Benzoni, J. M. Claverie, M. Follows, G. Gorsky, P. Hingamp, D. Iudicone, O. Jaillon, S. Kandels-Lewis, U. Krzic, F. Not, H. Ogata, S. Pesant, E. G. Reynaud, C. Sardet, M. E. Sieracki, S. Speich, D. Velayoudon, J. Weissenbach, P. Wincker, C. Abergel, D. Arslan, S. Audic, J. M. Aury, N. Babic, L. Beaufort, L. Bittner, E. Boss, C. Boutte, J. Brum, M. Carmichael, R. Casotti, A. Chambouvet, P. Chang, C. Chica, C. Clerissi, S. Colin, F. M. Cornejo-Castillo, C. Da Silva, S. De Monte, J. Decelle, Y. Desdevises, C. Dimier, J. Dolan, M. Duhaime, X. Durrieu

de Madron, F. D'Ortenzio, F. D'Ovidio, I. Ferrera, L. Garczarek, M. J. Garet-Delmas, S. Gasmi, J. M. Gasol, N. Grimsley, R. Heilig, J. Ignacio-Espinoza, J. L. Jamet, L. Karp-Boss, M. Katinka, H. Khalili, Z. Kolber, N. Le Bescot, H. Le Gofff, G. Lima-Mendez, F. Mahe, M. G. Mazzocchi, M. Montresor, P. Morin, B. Noel, C. Pedros-Alio, E. Pelletier, Y. Perez, M. Picheral, G. Piganeau, O. Poirot, J. Poulain, N. Poulton, F. Prejger, J. Prihoda, I. Probert, J. Rampal, G. Reverdin, S. Romac, J. B. Romagnan, F. Roullier, C. Rouviere, G. Samson, S. Santini, H. Sarmento, A. Sciandra, S. Solonenko, L. Stemmann, L. Subirana, S. Sunagawa, A. Tanaka, P. Testor, A. Thompson, V. Tichanne-Seltzer, L. Tirichine, E. Toulza, S. Tozzi, A. Veluchamy, and A. Zingone. A holistic approach to marine Eco-systems biology. *PLoS Biology*, 9(10):7–11, 2011. ISSN 15457885. doi:10.1371/journal.pbio.1001177.

[22] C. Bombardelli, J. H. Ayuso, and R. G. Pelayo. Collision avoidance maneuver optimization. *Advances in the Astronautical Sciences*, 152(7402):1857–1870, 2014. ISSN 00653438. doi:10.1038/nature11209.

[23] C. J. Robinson, B. J. M. Bohannan, and V. B. Young. From Structure to Function: the Ecology of Host-Associated Microbial Communities. *Microbiology and Molecular Biology Reviews*, 74(3):453–476, 2010. ISSN 1092-2172. doi:10.1128/MMBR.00014-10.

[24] R. E. Ley, M. Hamady, C. Lozupone, P. J. Turnbaugh, R. R. Ramey, J. S. Bircher, M. L. Schlegel, T. A. Tucker, M. D. Schrenzel, R. Knight, and J. I. Gordon. Evolution of mammals and their gut microbes. *Science*, 320(5883):1647–1651, 2008. ISSN 00368075. doi:10.1126/science.1155725.

[25] J. Peterson, S. Garges, M. Giovanni, P. McInnes, L. Wang, J. A. Schloss, V. Bonazzi, J. E. McEwen, K. A. Wetterstrand, C. Deal, C. C. Baker, V. Di Francesco, T. K. Howcroft, R. W. Karp, R. D. Lunsford, C. R. Wellington, T. Belachew, M. Wright, C. Giblin, H. David, M. Mills, R. Salomon, C. Mullins, B. Akolkar, L. Begg, C. Davis, L. Grandison, M. Humble, J. Khalsa, A. Roger Little, H. Peavy, C. Pontzer, M. Portnoy, M. H. Sayre, P. Starke-Reed, S. Zakhari, J. Read, B. Watson, and M. Guyer. The NIH Human Microbiome Project. *Genome Research*, 19(12): 2317–2323, 2009. ISSN 10889051. doi:10.1101/gr.096651.109.

[26] J. A. Gilbert, M. J. Blaser, J. G. Caporaso, J. K. Jansson, S. V. Lynch, and R. Knight. Current understanding of the human microbiome. *Nature Medicine*, 24(4):392–400, 2018. ISSN 1546170X. doi:10.1038/nm.4517.

[27] D. H. Haft, M. DiCuccio, A. Badretdin, V. Brover, V. Chetvernin, K. O'Neill, W. Li, F. Chitsaz, M. K. Derbyshire, N. R. Gonzales, M. Gwadz, F. Lu, G. H. Marchler, J. S. Song, N. Thanki, R. A. Yamashita, C. Zheng, F. Thibaud-Nissen, L. Y. Geer, A. Marchler-Bauer, and K. D. Pruitt. RefSeq: An update on prokaryotic genome annotation and curation. *Nucleic Acids Research*, 46(D1):D851–D860, 2018. ISSN 13624962. doi:10.1093/nar/gkx1068.

[28] D. C. Savage. Microbial Ecology of the Gastrointestinal Tract. *Annual Review of Microbiology*, 31(1):107–133, 1977. doi:10.1146/annurev.mi.31.100177.000543.

[29] R. Sender, S. Fuchs, and R. Milo. Are We Really Vastly Outnumbered? Revisiting the Ratio of Bacterial to Host Cells in Humans. *Cell*, 164(3):337–340, 2016. ISSN 10974172. doi:10.1016/j.cell.2016.01.013.

[30] X. Yin, J. Zhang, and X. Wang. *Sequential injection analysis system for the determination of arsenic by hydride generation atomic absorption spectrometry*, volume 32. Pearson Education, Inc, 14 edition, 2004. ISBN 9788578110796. doi:10.1017/CBO9781107415324.004.

[31] T. Thomas, J. Gilbert, and F. Meyer. Metagenomics - a guide from sampling to data analysis. *Microbial Informatics and Experimentation*, 2(1):3, 2012. ISSN 2042-5783. doi:10.1186/2042-5783-2-3.

[32] W. R. Streit and R. A. Schmitz. Metagenomics - The key to the uncultured microbes. *Current Opinion in Microbiology*, 7(5):492–498, 2004. ISSN 13695274. doi:10.1016/j.mib.2004.08.002.

[33] N. R. C. Committee on Metagenomics: Challenges and Functional Applications. *The New Science of Metagenomics*. National Academies Press, 2007. ISBN 978-0-309-10676-4. doi:10.17226/11902.

[34] K. N. Lam, J. Cheng, K. Engel, J. D. Neufeld, and T. C. Charles. Current and future resources for functional metagenomics. *Frontiers in Microbiology*, 6(OCT): 1–8, 2015. ISSN 1664302X. doi:10.3389/fmicb.2015.01196.

[35] A. Bankevich, S. Nurk, D. Antipov, A. A. Gurevich, M. Dvorkin, A. S. Kulikov, V. M. Lesin, S. I. Nikolenko, S. Pham, A. D. Prjibelski, A. V. Pyshkin, A. V. Sirotkin, N. Vyahhi, G. Tesler, M. A. Alekseyev, and P. A. Pevzner. SPAdes: A New Genome Assembly Algorithm and Its Applications to Single-Cell Sequencing. *Journal of Computational Biology*, 19(5):455–477, 2012. ISSN 1066-5277. doi:10.1089/cmb.2012.0021.

[36] D. R. Zerbino and E. Birney. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Research*, 18(5):821–829, 2008. ISSN 10889051. doi:10.1101/gr.074492.107.

[37] J. T. Simpson, K. Wong, S. D. Jackman, J. E. Schein, S. J. Jones, and I. Birol. ABySS: A parallel assembler for short read sequence data. *Genome Research*, 19 (6):1117–1123, 2009. ISSN 10889051. doi:10.1101/gr.089532.108.

[38] A. V. Zimin, G. Marçais, D. Puiu, M. Roberts, S. L. Salzberg, and J. A. Yorke. The MaSuRCA genome assembler. *Bioinformatics*, 29(21):2669–2677, 2013. ISSN 13674803. doi:10.1093/bioinformatics/btt476.

[39] T. Namiki, T. Hachiya, H. Tanaka, and Y. Sakakibara. MetaVelvet: An extension of Velvet assembler to de novo metagenome assembly from short sequence reads. *Nucleic Acids Research*, 40(20):e155, 2012. ISSN 03051048. doi:10.1093/nar/gks678.

[40] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, and R. Durbin. The Sequence Alignment/Map format and SAMtools. *Bioinformatics (Oxford, England)*, 25(16):2078–2079, 2009. ISSN 13674803. doi:10.1093/bioinformatics/btp352.

[41] R. S. Boyer and J. S. Moore. A Fast String Searching Algorithm. *Communications of the ACM*, 20(10):762–772, 1977.

[42] D. E. Knuth, J. H. Morris, Jr., and V. R. Pratt. Fast Pattern Matching in Strings. *SIAM Journal on Computing*, 6(2):323–350, 1977. ISSN 15221946. doi:10.1002/ccd.22046.

[43] A. Apostolico. The myriad virtues of suffix trees. In A. Apostolico and Z. Galil, editors, *Combinatorial Algorithms on Words*, volume 12, pages 85–96, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg. ISBN 978-3-642-82456-2.

[44] U. Manber and G. Myers. Suffix Arrays: A New Method for On-Line String Searches. *SIAM Journal on Computing*, 22(5):935–948, 1993. doi:10.1137/0222058.

[45] D. Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology*, volume 28. Cambridge University Press, 1997. ISBN 0521585198. doi:10.1145/270563.571472.

[46] S. Canzar and S. L. Salzberg. Short Read Mapping: An Algorithmic Tour. *Proceedings of the IEEE*, 105(3):436–458, 2017. doi:10.1109/JPROC.2015.2455551.

[47] P. Ferragina and G. Manzini. Opportunistic data structures with applications. *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 390–398, 2000. ISSN 0272-5428. doi:10.1109/SFCS.2000.892127.

[48] B. Langmead and S. L. Salzberg. Fast gapped-read alignment with Bowtie 2. *Nature Methods*, 9(4):357–359, 2012. ISSN 15487091. doi:10.1038/nmeth.1923.

[49] H. Li and R. Durbin. Fast and accurate long-read alignment with Burrows-Wheeler transform. *Bioinformatics*, 26(5):589–595, 2010. ISSN 13674803. doi:10.1093/bioinformatics/btp698.

[50] S. Marco-Sola, M. Sammeth, R. Guigó, and P. Ribeca. The GEM mapper: Fast, accurate and versatile alignment by filtration. *Nature Methods*, 9(12):1185–1188, 2012. ISSN 15487091. doi:10.1038/nmeth.2221.

[51] H. Cheng, H. Jiang, J. Yang, Y. Xu, and Y. Shang. BitMapper: An efficient all-mapper based on bit-vector computing. *BMC Bioinformatics*, 16(1):192, 2015. ISSN 14712105. doi:10.1186/s12859-015-0626-9.

[52] F. Hach, F. Hormozdiari, C. Alkan, F. Hormozdiari, I. Birol, E. E. Eichler, and S. C. Sahinalp. MrsFAST: A cache-oblivious algorithm for short-read mapping. *Nature Methods*, 7(8):576–577, 2010. ISSN 15487091. doi:10.1038/nmeth0810-576.

[53] J. Kim, C. Li, and X. Xie. Improving read mapping using additional prefix grams. *BMC Bioinformatics*, 15(1):42, 2014. ISSN 14712105. doi:10.1186/1471-2105-15-42.

[54] E. Siragusa. *Approximate string matching for high-throughput sequencing*. Thesis, PhD Thesis, FREIEN UNIVERSITÄT BERLIN, 2015.

[55] A. Döring, D. Weese, T. Rausch, and K. Reinert. SeqAn an efficient, generic C++ library for sequence analysis. *BMC Bioinformatics*, 9:11, 2008. ISSN 14712105. doi:10.1186/1471-2105-9-11.

[56] N. L. Bray, H. Pimentel, P. Melsted, and L. Pachter. Near-optimal probabilistic RNA-seq quantification. *Nature Biotechnology*, 34(5):525–527, 2016. ISSN 15461696. doi:10.1038/nbt.3519.

[57] L. Schaeffer, H. Pimentel, N. Bray, P. Melsted, and L. Pachter. Pseudoalignment for metagenomic read assignment. *Bioinformatics*, 33(14):2082–2088, 2017. ISSN 14602059. doi:10.1093/bioinformatics/btx106.

[58] I. Letunic and P. Bork. Interactive tree of life (iTOL) v3: an online tool for the display and annotation of phylogenetic and other trees. *Nucleic acids research*, 44(W1):W242–W245, 2016. ISSN 13624962. doi:10.1093/nar/gkw290.

[59] N. Worathumrong and A. J. Grimes. The Effect of o-Salicylate upon Pentose Phosphate Pathway Activity in Normal and G6PD-Deficient Red Cells. *British Journal of Haematology*, 30(2):225–231, 1975. ISSN 13652141. doi:10.1111/j.1365-2141.1975.tb00536.x.

[60] S. Lindgreen, K. L. Adair, and P. P. Gardner. An evaluation of the accuracy and speed of metagenome analysis tools. *Scientific Reports*, 6:19233, 2016. ISSN 20452322. doi:10.1038/srep19233.

[61] A. Brady and S. L. Salzberg. Phymm and PhymmBL: Metagenomic phylogenetic classification with interpolated Markov models. *Nature Methods*, 6(9):673–676, 2009. ISSN 15487091. doi:10.1038/nmeth.1358.

[62] V. C. Piro, M. S. Lindner, and B. Y. Renard. DUDes: A top-down taxonomic profiler for metagenomics. *Bioinformatics*, 32(15):2272–2280, 2016. ISSN 14602059. doi:10.1093/bioinformatics/btw150.

[63] M. S. Lindner and B. Y. Renard. Metagenomic profiling of known unknown microbes with MicrobeGPS. *PLoS ONE*, 10(2):e0117711, 2015. ISSN 19326203. doi:10.1371/journal.pone.0117711.

[64] O. E. Francis, M. Bendall, S. Manimaran, C. Hong, N. L. Clement, E. Castro-Nallar, Q. Snell, G. B. Schaalje, M. J. Clement, K. A. Crandall, and W. E. Johnson. Pathoscope: Species identification and strain attribution with unassembled sequencing data. *Genome Research*, 23(10):1721–1729, 2013. ISSN 10889051. doi:10.1101/gr.150151.112.

[65] M. Balvočiute and D. H. Huson. SILVA, RDP, Greengenes, NCBI and OTT - how do these taxonomies compare? *BMC Genomics*, 18(Suppl 2):1–8, 2017. ISSN 14712164. doi:10.1186/s12864-017-3501-4.

[66] J. R. Cole, Q. Wang, J. A. Fish, B. Chai, D. M. McGarrell, Y. Sun, C. T. Brown, A. Porras-Alfaro, C. R. Kuske, and J. M. Tiedje. Ribosomal Database Project: Data and tools for high throughput rRNA analysis. *Nucleic Acids Research*, 42 (D1):D633–D642, 2014. ISSN 03051048. doi:10.1093/nar/gkt1244.

[67] T. Z. DeSantis, P. Hugenholtz, N. Larsen, M. Rojas, E. L. Brodie, K. Keller, T. Huber, D. Dalevi, P. Hu, and G. L. Andersen. Greengenes, a Chimera-Checked 16S rRNA Gene Database and Workbench Compatible with ARB. *Applied and Environmental Microbiology*, 72(7):5069–5072, 2006. ISSN 0099-2240. doi:10.1128/AEM.03006-05.

[68] S. Federhen. The NCBI Taxonomy database. *Nucleic Acids Research*, 40(D1): D136–D143, 2012. ISSN 03051048. doi:10.1093/nar/gkr1178.

[69] Z. J. Jay and W. P. Inskeep. The distribution, diversity, and importance of 16S rRNA gene introns in the order Thermoproteales. *Biology Direct*, 10(1), 2015. ISSN 17456150. doi:10.1186/s13062-015-0065-6.

[70] K. Raymann, A. H. Moeller, A. L. Goodman, and H. Ochman. Unexplored Archaeal Diversity in the Great Ape Gut Microbiome. *mSphere*, 2(1):e00026–17, 2017. ISSN 2379-5042. doi:10.1128/mSphere.00026-17.

[71] I. C. Starke, W. Vahjen, R. Pieper, and J. Zentek. The Influence of DNA Extraction Procedure and Primer Set on the Bacterial Community Analysis by Pyrosequencing of Barcoded 16S rRNA Gene Amplicons. *Molecular Biology International*, 2014:1–10, 2014. ISSN 2090-2182. doi:10.1155/2014/548683.

[72] F. Fouhy, A. G. Clooney, C. Stanton, M. J. Claesson, and P. D. Cotter. 16S rRNA gene sequencing of mock microbial populations-impact of DNA extraction method, primer choice and sequencing platform. *BMC Microbiology*, 16(1):123, 2016. ISSN 14712180. doi:10.1186/s12866-016-0738-z.

[73] S. Chakravorty, D. Helb, M. Burday, N. Connell, and D. Alland. A detailed analysis of 16S ribosomal RNA gene segments for the diagnosis of pathogenic bacteria. *Journal of Microbiological Methods*, 69(2):330–339, 2007. ISSN 01677012. doi:10.1016/j.mimet.2007.02.005.

[74] J. Pollock, L. Glendinning, T. Wisedchanwet, and M. Watson. The madness of microbiome: Attempting to find consensus "best practice" for 16S microbiome studies. *Applied and Environmental Microbiology*, 84(7):521, 2018. ISSN 10985336. doi:10.1128/AEM.02627-17.

[75] T. Větrovský and P. Baldrian. The Variability of the 16S rRNA Gene in Bacterial Genomes and Its Consequences for Bacterial Community Analyses. *PLoS ONE*, 8(2):1–10, 2013. ISSN 19326203. doi:10.1371/journal.pone.0057923.

[76] C. Yuan, J. Lei, J. Cole, and Y. Sun. Reconstructing 16S rRNA genes in metagenomic data. *Bioinformatics*, 31(12):i35–i43, 2015. ISSN 14602059. doi:10.1093/bioinformatics/btv231.

[77] M. Tessler, J. S. Neumann, E. Afshinnekoo, M. Pineda, R. Hersch, L. F. M. Velho, B. T. Segovia, F. A. Lansac-Toha, M. Lemke, R. Desalle, C. E. Mason, and M. R. Brugler. Large-scale differences in microbial biodiversity discovery between 16S amplicon and shotgun sequencing. *Scientific Reports*, 7(1):1–14, 2017. ISSN 20452322. doi:10.1038/s41598-017-06665-3.

[78] S. K. Ames, D. A. Hysom, S. N. Gardner, G. S. Lloyd, M. B. Gokhale, and J. E. Allen. Scalable metagenomic taxonomy classification using a reference genome database. *Bioinformatics*, 29(18):2253–2260, 2013. ISSN 13674803. doi:10.1093/bioinformatics/btt389.

[79] D. T. Truong, E. A. Franzosa, T. L. Tickle, M. Scholz, G. Weingart, E. Pasolli, A. Tett, C. Huttenhower, and N. Segata. MetaPhlAn2 for enhanced metagenomic taxonomic profiling. *Nature Methods*, 12(10):902–903, 2015. ISSN 15487105. doi:10.1038/nmeth.3589.

[80] R. J. Sengwa, A. Madhvi, and S. Sankhla. Study of dielectric relaxation and dipole moment of some hydrogen bonded solvent binary mixtures in 1,4-dioxane. *Indian Journal of Pure and Applied Physics*, 44(12):943–952, 2006. ISSN 00195596. doi:10.1093/nar/gkv180.

[81] S. Sunagawa, D. R. Mende, G. Zeller, F. Izquierdo-Carrasco, S. A. Berger, J. R. Kultima, L. P. Coelho, M. Arumugam, J. Tap, H. B. Nielsen, S. Rasmussen, S. Brunak, O. Pedersen, F. Guarner, W. M. De Vos, J. Wang, J. Li, J. Doré, S. Dusko Ehrlich,

A. Stamatakis, and P. Bork. Metagenomic species profiling using universal phylogenetic marker genes. *Nature Methods*, 10(12):1196–1199, 2013. ISSN 15487091. doi:10.1038/nmeth.2693.

[82] D. E. Wood and S. L. Salzberg. Kraken: Ultrafast metagenomic sequence classification using exact alignments. *Genome Biology*, 15(3):R46, 2014. ISSN 1474760X. doi:10.1186/gb-2014-15-3-r46.

[83] T. H. Dadi, B. Y. Renard, L. H. Wieler, T. Semmler, and K. Reinert. SLIMM: species level identification of microorganisms from metagenomes. *PeerJ*, 5(3):e3138, 2017. ISSN 2167-8359. doi:10.7717/peerj.3138.

[84] J. Lu, F. P. Breitwieser, P. Thielen, and S. L. Salzberg. Bracken: estimating species abundance in metagenomics data. *PeerJ Computer Science*, 3:e104, 2017. ISSN 2376-5992. doi:10.7717/peerj-cs.104.

[85] R. Ounit, S. Wanamaker, T. J. Close, and S. Lonardi. CLARK: fast and accurate classification of metagenomic and genomic sequences using discriminative k-mers. *BMC Genomics*, 16(1):236, 2015. ISSN 14712164. doi:10.1186/s12864-015-1419-2.

[86] D. H. Huson, A. F. Auch, J. Qi, and S. C. Schuster. MEGAN analysis of metagenomic data. *Genome Research*, 17(3):377–386, 2007. ISSN 10889051. doi:10.1101/gr.5969107.

[87] R. Overbeek, T. Begley, R. M. Butler, J. V. Choudhuri, H. Y. Chuang, M. Cohoon, V. de Crécy-Lagard, N. Diaz, T. Disz, R. Edwards, M. Fonstein, E. D. Frank, S. Gerdes, E. M. Glass, A. Goesmann, A. Hanson, D. Iwata-Reuyl, R. Jensen, N. Jamshidi, L. Krause, M. Kubal, N. Larsen, B. Linke, A. C. McHardy, F. Meyer, H. Neuweger, G. Olsen, R. Olson, A. Osterman, V. Portnoy, G. D. Pusch, D. A. Rodionov, C. Rül;ckert, J. Steiner, R. Stevens, I. Thiele, O. Vassieva, Y. Ye, O. Zagnitko, and V. Vonstein. The subsystems approach to genome annotation and its use in the project to annotate 1000 genomes. *Nucleic Acids Research*, 33(17): 5691–5702, 2005. ISSN 03051048. doi:10.1093/nar/gki866.

[88] N. Segata, L. Waldron, A. Ballarini, V. Narasimhan, O. Jousson, and C. Huttenhower. Metagenomic microbial community profiling using unique clade-specific marker genes. *Nature Methods*, 9(8):811–814, 2012. ISSN 15487091. doi:10.1038/nmeth.2066.

[89] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology*, 10(3), 2009. ISSN 14747596. doi:10.1186/gb-2009-10-3-r25.

[90] E. M. Glass and F. Meyer. The Metagenomics RAST Server: A Public Resource for the Automatic Phylogenetic and Functional Analysis of Metagenomes. *Handbook of Molecular Microbial Ecology I: Metagenomics and Complementary Approaches*, 8:325–331, 2011. ISSN 14712105. doi:10.1002/9781118010518.ch37.

[91] J. R. Cole, B. Chai, R. J. Farris, Q. Wang, A. S. Kulam-Syed-Mohideen, D. M. McGarrell, A. M. Bandela, E. Cardenas, G. M. Garrity, and J. M. Tiedje. The ribosomal database project (RDP-II): Introducing myRDP space and quality controlled public data. *Nucleic Acids Research*, 35(SUPPL. 1):169–172, 2007. ISSN 03051048. doi:10.1093/nar/gkl889.

[92] J. Wuyts. The European database on small subunit ribosomal RNA. *Nucleic Acids Research*, 30(1):183–185, 2002. ISSN 13624962. doi:10.1093/nar/30.1.183.

[93] A. L. Mitchell, M. Scheremetjew, H. Denise, S. Potter, A. Tarkowska, M. Qureshi, G. A. Salazar, S. Pesseat, M. A. Boland, F. M. Hunter, P. Ten Hoopen, B. Alako, C. Amid, D. J. Wilkinson, T. P. Curtis, G. Cochrane, and R. D. Finn. EBI Metagenomics in 2017: Enriching the analysis of microbial communities, from sequence reads to assemblies. *Nucleic Acids Research*, 46(D1):D726–D735, 2018. ISSN 13624962. doi:10.1093/nar/gkx967.

[94] J. Gregory Caporaso, J. Kuczynski, J. Stombaugh, K. Bittinger, F. D. Bushman, E. K. Costello, N. Fierer, A. G. Peña, J. K. Goodrich, J. I. Gordon, G. a. Huttley, S. T. Kelley, D. Knights, J. E. Koenig, R. E. Ley, C. a. Lozupone, D. Mcdonald, B. D. Muegge, M. Pirrung, J. Reeder, J. R. Sevinsky, P. J. Turnbaugh, W. a. Walters, J. Widmann, T. Yatsunenko, J. Zaneveld, and R. Knight. correspondence QIIME allows analysis of high- throughput community sequencing data Intensity normalization improves color calling in SOLiD sequencing. *Nature Publishing Group*, 7(5):335–336, 2010. ISSN 1548-7091. doi:10.1038/nmeth0510-335.

[95] M. Kanehisa, S. Goto, Y. Sato, M. Furumichi, and M. Tanabe. KEGG for integration and interpretation of large-scale molecular data sets. *Nucleic Acids Research*, 40 (D1):109–114, 2012. ISSN 03051048. doi:10.1093/nar/gkr988.

[96] J. T. Roehr, C. Dieterich, and K. Reinert. Flexbar 3.0 - SIMD and multicore parallelization. *Bioinformatics*, 33(18):2941–2942, 2017. ISSN 14602059. doi:10.1093/bioinformatics/btx330.

[97] Ł. Grześkowiak, B. Martínez-Vallespín, T. H. Dadi, J. Radloff, S. Amasheh, F. A. Heinsen, A. Franke, K. Reinert, W. Vahjen, J. Zentek, and R. Pieper. Formula feeding predisposes neonatal piglets to clostridium difficile gut infection. *Journal of Infectious Diseases*, 217(9):1442–1452, 2018. ISSN 15376613. doi:10.1093/infdis/jix567.

[98] M. Shakya, C. Quince, J. H. Campbell, Z. K. Yang, C. W. Schadt, and M. Podar. Comparative metagenomic and rRNA microbial diversity characterization using archaeal and bacterial synthetic communities. *Environmental Microbiology*, 15 (6):1882–1899, 2013. ISSN 14622912. doi:10.1111/1462-2920.12086.

[99] B. Jia, L. Xuan, K. Cai, Z. Hu, L. Ma, and C. Wei. NeSSM: A Next-Generation Sequencing Simulator for Metagenomics. *PLoS ONE*, 8(10), 2013. ISSN 19326203. doi:10.1371/journal.pone.0075448.

[100] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970. doi:10.1145/362686.362692.

[101] E. Siragusa, D. Weese, and K. Reinert. Fast and accurate read mapping with approximate seeds and multiple backtracking. *Nucleic Acids Research*, 41(7), 2013. ISSN 03051048. doi:10.1093/nar/gkt005.

[102] H. Hauswedell, J. Singer, and K. Reinert. Lambda: The local aligner for massive biological data. *Bioinformatics*, 30(17):i349–i355, 2014. ISSN 14602059. doi:10.1093/bioinformatics/btu439.

[103] M. J. Bauer, A. J. Cox, and G. Rosone. Lightweight BWT construction for very large string collections. In R. Giancarlo and G. Manzini, editors, *Lecture Notes in*

Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), volume 6661 LNCS, pages 219–231, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 9783642214578. doi:10.1007/978-3-642-21458-5_20.

[104] J. Sirén. Compressed Suffix Arrays for Massive Data. In J. Karlgren, J. Tarhio, and H. Hyyrö, editors, *String Processing and Information Retrieval*, pages 63–74, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-03784-9.

[105] H. Mohamadi, B. P. Vandervalk, A. Raymond, S. D. Jackman, J. Chu, C. P. Breshears, and I. Birol. DIDA: Distributed indexing dispatched alignment. *PLoS ONE*, 10(4):1–14, 2015. ISSN 19326203. doi:10.1371/journal.pone.0126409.

[106] G. N. Samuel and B. Farsides. The UK's 100,000 Genomes Project: manifesting policymakers' expectations. *New Genetics and Society*, 36(4):336–353, 2017. ISSN 14699915. doi:10.1080/14636778.2017.1370671.

[107] B. Codenotti, G. De Marco, M. Leoncini, M. Montangero, and M. Santini. Approximation algorithms for a hierarchically structured bin packing problem. *Information Processing Letters*, 89(5):215–221, 2004. ISSN 00200190. doi:10.1016/j.ipl.2003.12.001.

[108] R. Rahn, D. Weese, and K. Reinert. Journaled string tree-a scalable data structure for analyzing thousands of similar genomes on your laptop. *Bioinformatics*, 30 (24):3499–3505, 2014. ISSN 14602059. doi:10.1093/bioinformatics/btu438.

[109] K. Schneeberger, J. Hagmann, S. Ossowski, N. Warthmann, S. Gesing, O. Kohlbacher, and D. Weigel. Simultaneous alignment of short reads against multiple genomes. *Genome Biology*, 10(9), 2009. ISSN 14747596. doi:10.1186/gb-2009-10-9-r98.

[110] T. H. Dadi, E. Siragusa, V. C. Piro, A. Andrusch, E. Seiler, B. Y. Renard, and K. Reinert. DREAM-Yara: an exact read mapper for very large databases with short update time. *Bioinformatics*, 34(17):i766–i772, 2018. ISSN 1367-4803. doi:10.1093/bioinformatics/bty567.

[111] P. Bradley, H. den Bakker, E. Rocha, G. McVean, and Z. Iqbal. Real-time search of all bacterial and viral genomic data. *bioRxiv preprint*, page 234955, 2017. doi:10.1101/234955.

[112] K. Reinert, T. H. Dadi, M. Ehrhardt, H. Hauswedell, S. Mehringer, R. Rahn, J. Kim, C. Pockrandt, J. Winkler, E. Siragusa, G. Urgese, and D. Weese. The SeqAn C++ template library for efficient sequence analysis: A resource for programmers. *Journal of Biotechnology*, 261(September):157–168, 2017. ISSN 18734863. doi:10.1016/j.jbiotec.2017.07.017.

[113] D. Weese, M. Holtgrewe, and K. Reinert. RazerS 3: Faster, fully sensitive read mapping. *Bioinformatics*, 28(20):2592–2599, 2012. ISSN 13674803. doi:10.1093/bioinformatics/bts505.

[114] M. Holtgrewe, A. K. Emde, D. Weese, and K. Reinert. A novel and well-defined benchmarking method for second generation read mapping. *BMC Bioinformatics*, 12, 2011. ISSN 14712105. doi:10.1186/1471-2105-12-210.

[115] F. P. Breitwieser, D. N. Baker, and S. L. Salzberg. KrakenUniq: confident and fast metagenomics classification using unique k-mer counts. *Genome Biology*, 19(1): 198, 2018. ISSN 1474-760X. doi:10.1186/s13059-018-1568-0.

[116] D. J. Nasko, S. Koren, A. M. Phillippy, and T. J. Treangen. RefSeq database growth influences the accuracy of k-mer-based lowest common ancestor species identification. *Genome Biology*, 19(1):165, 2018. ISSN 1474-760X. doi:10.1186/s13059-018-1554-6.

[117] W. Zhou, N. Gay, and J. Oh. ReprDB and panDB: minimalist databases with maximal microbial representation. *Microbiome*, 6(1):15, 2018. ISSN 2049-2618. doi:10.1186/s40168-018-0399-2.

[118] W. Li and A. Godzik. Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*, 22(13):1658–1659, 2006. doi:10.1093/bioinformatics/btl158.

[119] B. T. James, B. B. Luczak, and H. Z. Girgis. MeShClust: an intelligent tool for clustering DNA sequences. *Nucleic Acids Research*, 46(14):e83, 2018. doi:10.1093/nar/gky315. URL http://dx.doi.org/10.1093/nar/gky315.

[120] V. C. Piro, T. H. Dadi, E. Seiler, K. Reinert, and B. Y. Renard. ganon: continuously up-to-date with database growth for precise short read classification in metagenomics. *bioRxiv*, 2018. doi:10.1101/406017. URL https://www.biorxiv.org/content/early/2018/08/31/406017.

List of Figures

List of Tables

Acronyms

AMD acid mine drainage 11

CRT cyclic reversible termination 20

DNA deoxyribonucleic acid 9

DREAM index Dynamic seaRchablE pArallel coMpressed index 1

EBR effective binning ratio 74

emPCR emulsion PCR 19

ENA European Nucleotide Archive 43

FDR false discovery rate 41

IBF interleaved Bloom filters 1

kbp kilobase pairs 17

LCA lowest common ancestor 38

mbp megabase pairs 17

NGS next-generation sequencing 46

PCR polymerase chain reaction 15

RNA ribonucleic acid 9

SLIMM species level identification of microorganisms from metagenomes 45

SNA single nucleotide addition 20

WGS whole genome shotgun sequencing 1

WGMS whole metagenome shotgun sequencing 21