SORBONNE UNIVERSITÉ

Analysis of human pangenome graphs

And other *k*-mer based applications

Francesco Andreace

Thesis submitted for the degree of Philosophiae Doctor

Ecole Doctorale Informatique, Télécommunications et Electronique EDITE (ED130)
Sorbonne Université

Members of the jury :

Dr. Francois Sabot	Université de Montpellier	Reviewer
Dr. Matthias Zytnicki	INRAE	Reviewer
Dr. Paola Bonizzoni	Università degli Studi di Milano Bicocca	Examiner
Dr. Camille Marchet	CNRS, Université de Lille	Examiner
Dr. Pierre Peterlongo	IRISA, Université de Rennes	Examiner
Dr. Rayan Chikhi	Institut Pasteur	Supervisor
Dr. Yoann Dufresne	Institut Pasteur	Supervisor

2025

INSTITUT PASTEUR

Series of dissertations submitted to the
Faculté d'Informatique, Sorbonne Université

Print production: Insitut Pasteur.

Alla mia famiglia, sempre presente nei momenti che contano.
A mi Belsy, por hacer de este tiempo algo inolvidable.

Preface

This thesis is submitted in partial fulfillment of the requirements for the degree of *Philosophiae Doctor* at Sorbonne Université. The research presented here was conducted at the Institut Pasteur, under the supervision of Dr. Rayan Chikhi and Dr. Yoann Dufresne. This work is part of the Marie Skłodowska-Curie ITN ALPACA project that has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grants agreements No 956 229 and 872 539.

This thesis is a collection of some of the different projects I worked on during my stay at Institut Pasteur. I begin with an small foreword of the research output of my PhD, and a gentle introduction of the scientific background needed to contextualize the work proposed in the manuscript. In the first part I present the published paper I am first author of, together with other unpublished work I lead or independently developed. In the second part I present other results of my scientific production, with novel elaboration of the work that appeared in the other papers I am co-author and presentation of projects that have or will be submitted to revision. The common theme is human pangenomics and computational methods used to generate and use such models to infer relevant information. This essay ends with a chapter showcasing future perspectives and conclusions.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to Rayan for his constant support, patience, encouragement, and understanding throughout this journey. His ability to bring value to our work, his admirable work ethic, and the scientific intelligence he demonstrated in all the fields we discussed have been a true source of inspiration. I am truly fortunate to have had the opportunity to learn from him.

I would also like to extend my deepest gratitude to Yoann, who has been a constant source of motivation and inspiration throughout this journey. His passion for his work has been truly contagious, fueling my own interest in this field. Yoann not only helped me significantly improve my computer science skills but also became a valued person in my life in Paris. Whether it was inviting me to events at La Treso, patiently teaching me French alongside Pierre, explaining me the technicalities of the game he invents, talking about politics or inviting me for runs, his presence enriched both my personal and professional experiences.

A heartfelt thanks goes to François Sabot and Matthias Zytnicki for kindly agreeing to review this manuscript and for devoting their time and effort to being part of the thesis jury. I understand the significant commitment involved,

and I deeply appreciate their contributions.
I extend my sincere thanks to Paola Bonizzoni, Camille Marchet, and Pierre Peterlongo for serving as members of the jury. Each one of them has played a special role in the trajectory of my thesis for particular reasons.
Additionally, I wish to express my appreciation to Pierre Peterlongo and Martin Weigt for being part of my Comité de Thèse, offering valuable feedback that greatly contributed to the successful completion of this project.
To the SeqBio group, I would like to extend a special thank you. The group is composed of remarkable individuals, and I am lucky to have been part of it. In particular, I want to thank Camila for always being there to talk things through and for offering invaluable advice. I am equally grateful to Yoshihiro for his kindness and willingness to help in any circumstance and to Mélanie, whose support during every "chat noir" situation ensured I could finish this PhD.
I want to express deep gratitude to the Italian and Spanish community in Pasteur, who made me feel at home in every moment of these 40 months.
Lastly, I would like to thank the Alpacas, a distinguished group of young scientists, but even more importantly, incredible people. It has been such a pleasure to work and spend time with all of you.

Francesco Andreace
Paris, February 2025

Summary

English summary

For over two decades, the human reference genome has laid the ground for human genomic research. However, its power to provide insights has been constrained by the presence of gaps and simulated sequences. In 2022, the Telomere-to-Telomere consortium achieved an important milestone by releasing the first full sequence of an haploid human genome (T2T-CHM13), empowering a new discovery on the previously missing regions of the genome. Nevertheless, a single genome cannot adequately represent the entire genetic diversity within the human population, in particular large *structural* variants.

To address the inherent reference bias of using a single genome as mean of comparison, the scientific community is transitioning towards pangenomes: these are models that encapsulate multiple alleles from a collection of genomes. The field of computational pangenomics aims at finding new and more efficient pangenome models that can improve the results of reference-based analyses. Among others, the most common pangenome representation is based on graphs.

This dissertation presents two primary contributions to computational pangenomics. The first is a comparative analysis of pangenome graph representations, based on the construction of the largest pangenome graph to that date. This analysis compares different graph models, using five state-of-the-art tools, shedding light on key differences between the representation, particularly on how they capture genetic variation in complex loci.

The second contribution focuses on advanced data structures for k-mer sets representation. In particular, on three novel data structures that focus on improving metadata association, downstream analysis accessibility and scalability. These k-mer-based methods aim at facilitating genomic and pangenomic analyses.

This dissertation present my contributions to the ongoing evolution of computational pangenomics research.

Résumé en français

Depuis plus de vingt ans, le génome Humain de référence a jeté les bases de la recherche en génomique Humaine. Toutefois, la quantité d'informations extraite de cette référence est limitée par sa non complétude et ses morceaux issus de simulations. En 2022, le consortium Telomere-to-Telomere a franchi une étape importante en publiant la première séquence complète d'un haploïde Humain (T2T-CHM13), ce qui a permis de faire de nouvelles découvertes sur les régions du génome qui étaient auparavant manquantes. Néanmoins, un seul génome ne peut pas représenter de manière adéquate l'ensemble de la diversité génétique au sein de la population Humaine, en particulier les grands variants *structuraux*. Pour remédier au biais de référence inhérent à l'utilisation d'un seul génome comme moyen de comparaison, la communauté scientifique s'oriente vers les pangénomes : il s'agit de modèles qui englobent de multiples allèles provenant d'une collection de génomes. Le domaine de la pangénomique computationnelle vise à trouver de nouveaux modèles de pangénomes plus efficaces qui peuvent améliorer les résultats des analyses basées sur les références. Entre autres, la représentation la plus courante du pangénome est basée sur les graphes.

Cette thèse présente deux contributions principales à la pangénomique computationnelle. La première est une analyse comparative des représentations graphiques du pangénome, basée sur la construction du plus grand graphique du pangénome à ce jour. Cette analyse compare différents modèles de graphes, en utilisant cinq outils de pointe, mettant en lumière les principales différences entre les représentations, en particulier sur la façon dont elles capturent la variation génétique dans les loci complexes.

La deuxième contribution porte sur les structures de données avancées pour la représentation des ensembles de k-mer. En particulier, trois nouvelles structures de données visent à améliorer l'association des métadonnées, l'accessibilité des analyses en aval et la scalabilité. Ces méthodes basées sur les kmer visent à faciliter les analyses génomiques et pangénomiques.

Cette thèse présente ma contribution à l'évolution en cours de la recherche en pangénomique computationnelle.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As sequencing technology becomes increasingly accessible and accurate, the production of high-quality genome assemblies for organisms of the same species is accelerating. The contrast in processing time is stark: while the Human Genome Project took 13 years to produce its results in 2003, hundreds of genomes of similar quality are now being produced within a few years. Large-scale initiatives such as the UK Biobank, which has sequenced the genomes of 100,000 individuals, present new challenges in efficiently analyzing genetic data from relatively similar populations.

Traditionally, genomics software has been developed under the assumption that only one or a few high-quality reference sequences of a single species were available for analyzing limited amounts of new data. Consequently, the wealth of newly available public information is not being fully utilized to enhance the quality of current studies. Furthermore, the velocity and heterogeneity of new sequences deposited in public databases like ENA or SRA render current algorithms inadequate for rapidly analyzing population data that can be inferred from them.

To address these limitations, a transformative approach known as pangenomics is emerging. Pangenomics aims to reduce the observational bias inherent in current genomic analyses by capturing the entire genetic diversity within a single population, species, or group of similar organisms into a complex representation called a pangenome. This approach necessitates the development of novel computational methods capable of processing thousands of large and complex genomes.

As the field of computational pangenomics is relatively new, there is currently no one-size-fits-all solution that satisfies all requirements and enables straightforward downstream analysis for all genomic applications. Different models are being explored to address various challenges in representing, indexing, compressing, and analyzing pangenomic data.

One model gaining significant attention is an improvement on the sequence graph, called a variation graph, which models relationships between shared parts and differences in genomes using nodes and edges. Alternatively, an established method for efficiently representing genomic data involves decomposing variable-length sequences, known as reads, into fixed-size tokens called k-mers. The term k-mer derives from the use of an arbitrary value, k, which represents the fixed length of these tokens.

The k-mer model has demonstrated its effectiveness in numerous bioinformatics techniques, particularly in assembling genomes from reads of a specific species. More recently, k-mers have gained popularity by enabling superior processing of complex data such as ancient DNA or metagenomes from marine samples

compared to other methods. Various data structures have been proposed to represent sequencing data as sets of k-mers, each with its own strengths and limitations.
As the field of pangenomics continues to evolve, researchers are exploring and refining these different approaches to develop more comprehensive and efficient methods for analyzing the vast amounts of genomic data being generated. The ultimate goal is to create tools and techniques that can fully leverage the wealth of genomic information now available, leading to more accurate and insightful analyses of genetic diversity within and across species.

1.1 Research Output

In this dissertation I present the result of my work on analyzing, developing and applying computational methods, mostly k-mer based, for pan-genomics. In the 40 months I spent at the Institut Pasteur under the supervision of Rayan Chikhi and Yoann Dufresne, I have contributed to several projects on k-mer based data structures and pursued my own research direction on pangenomics.
This has resulted in several publications, including one as first author and two as second author. At the moment of the writing of this manuscript I also expect to be author of other articles based on the work presented here.
In these years I have been involved in oral and poster presentations, mainly at events related to the Marie Skłodowska-Curie Actions (MSCA) Innovative Training Network (ITN) I have been part of.
Finally, by being part of the student association at Pasteur, I have been involved in several outreach events, as the European Researcher Night in 2022 and 2023, to broadly present the bioinformatics and genomics research fields to the general public in Paris.

1.2 General Outline

The manuscript is organized as follows:

Chapter 2 An introduction to the subject of the thesis and outline of the manuscript.

Chapter 3 An introduction of the background concepts of Genomics and Pangenomics.

Chapter 4 This chapter is structured into two main sections. The first section presents my work on human pangenome graph representation, detailing the methodologies employed and the insights gained from this research. This work contributes to the current efforts to create more comprehensive and accurate representations of human genetic diversity.
The second section addresses the challenges associated with constructing pangenomes that represent cross-chromosome events in yeast. It explores the complexities of capturing genetic variations that span multiple chromosomes, a particularly challenging aspect of pangenome graphs construction.

Chapter 5 Presentation of three k-mer based methods to which I contributed. The chapter begins with a concise overview of the current state of the art. It is then divided into three separate sections, each providing a detailed discussion of the respective method, including its underlying principles, implementation details, and the specific contributions.

Chapter 6 Perspectives.

Chapter 7 Conclusions.

Chapter 2

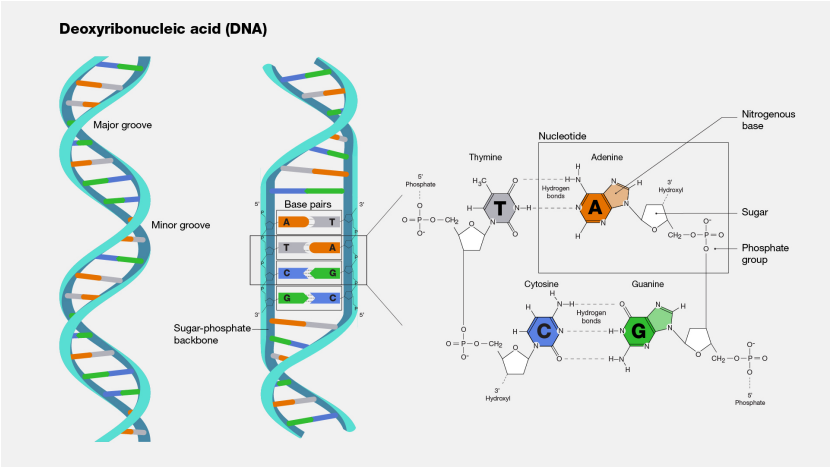
Background

A fundamental understanding of the data produced by the sequencing of biological organisms is essential for comprehending the research outlined in this manuscript. If you are already familiar with DNA sequences, how they are obtained, and how they differ between species or individuals, you may proceed to Section 2.3 *From reads to* k*-mers*.

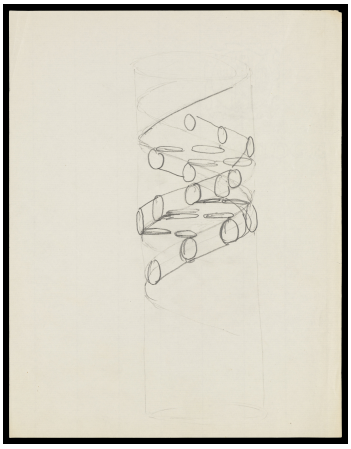
2.1 DNA, genome variation and sequencing data

DNA (Deoxyribonucleic Acid) is a complex molecule with a double helix structure that carries the genetic information of an organism. Although its discovery was the result of the cumulative work of many scientists over nearly 90 years, the currently accepted model was first correctly described through the work of James Watson and Francis Crick, along with Maurice Wilkins and Rosalind Franklin, between 1951 and 1953 in Cambridge, UK [1].
The information encoded in DNA provides the instructions for an organism to develop, survive in its environment, and reproduce. These instructions are stored as a sequence of monomers (from Greek *mónos-single* and *méros-part*) called nucleotides. Each nucleotide consists of a sugar, a phosphate group, and one of four nucleobases: cytosine, guanine, adenine, and thymine. Nucleotides are typically represented by the first letter of their respective nucleobase: A, C, G, and T. In RNA (a different molecule that acts as transcription of DNA), thymine is replaced by uracil (U). Adenine and guanine have similar fused-ring molecular structure and are called purine bases while cytosine, thymine and uracil have simple ring molecular structure and are called pyrimidines.
The nucleotides are linked by a sugar-phosphate backbone, and hydrogen bonds between complementary nucleotides stabilize the molecule's double-stranded structure: A pairs with T, and C pairs with G. The purine-pyrimidine pair, called base complement, is crucial for both DNA replication and protein synthesis. Figure 2.1 illustrates the structure of the DNA molecule and its nucleotides, as initially drawn by Francis Crick in 1953.

To fit within the cell nucleus, DNA is organized into highly compact structures. First, it is coiled around proteins called histones, forming a compact structure known as chromatin. The chromatin further forms loops, which are held in place by other molecules to create the structure of a chromosome. Chromosomes are inherited by offspring through sexual or asexual reproduction. Humans are diploid organisms, meaning they contain two copies of each chromosome, one inherited from the mother (via the egg) and one from the father (via the sperm). Both the egg and the sperm (collectively called gametes) contain one copy of each chromosome. While mammals are typically diploid, other organisms can



(a) The DNA molecule and the structure of the nucleotides, the basic piece of information of the DNA. Figure from NIH glossary [2].



(b) The DNA molecule model draw by Francis Crick in 1953.

Figure 2.1: The DNA molecule.

be haploid (containing a single copy of each chromosome, like male ants) or polyploid, meaning they possess more than two copies of each chromosome. For example, sugarcane, the world's most harvested crop by tonnage, can have more than eight copies of a chromosome, reaching up to twelve [3]. The complete set of genetic material present in a cell is the genome. In humans, each nucleus of non-reproductive cells contains 23 pairs of chromosomes. Of these, 22 pairs are autosomes, which are chromosomes that are not involved in determining sex and are common to both sexes. The 23rd pair is the sex chromosomes, which consist of either two copies of the X chromosome for females or one X and one Y chromosome for males. The telomere forms the end part of the

chromosome, while the centromere is located in the central region. Telomeres protect chromosome ends by blocking DNA damage repair mechanisms. In humans, telomeres are composed of consecutive repeats of the sequence `TTAGGG`, which span 5 to 15 thousand bases. As the ends of chromosomes cannot be fully replicated during cell division, telomeres naturally shorten with each division, contributing to the aging process, though the repeated sequence itself does not change. Abnormal telomere length can lead to genetic defects and diseases [4]. In contrast to the relatively stable structure of the telomere, the centromere is one of the most rapidly mutating regions of the human genome. Its sequence is organized into Higher Order Repeats (HOR), which consist of consecutive copies of large sections containing multiple repetitions of smaller sub-sequences. Centromeres are the most challenging regions to reconstruct from sequencing data [5].

Finally, there is also the mitochondrial genome, which is located outside the nucleus. It has a circular structure and is primarily maternally inherited. Analysis of mitochondrial DNA, together with the Y chromosome, has been used to infer matrilocal or patrilocal marriage mechanisms in human population studies [6].

2.1.1 DNA sequencing

In many biological disciplines, studying an organism's genetic information contained in its DNA is essential. Over the years, researchers have developed various methods and techniques to sequentially read nucleotides from cellular DNA; these techniques are collectively referred to as genome sequencing. The result of these processes is a collection of sequence reads, often simply called "reads," which represent the nucleotide sequences observed in the input DNA molecules. By sequencing DNA fragments and computationally assembling them, we can observe genomes [7]. Genome assembly is a computational process used to reconstruct the complete genome sequence of an organism from a set of reads generated by one or more sequencing techniques.
Sequencing typically involves three main steps. Below, I describe them with significant simplifications to provide a brief overview:

Library preparation The first step requires several hours of nontrivial biological manipulation of samples to extract and purify DNA from the cell nucleus without causing damage. This process isolates the genetic material from other cellular components, such as RNA and proteins. The DNA molecules are then fragmented into pieces of varying lengths, followed by ligation of "5'" and "3'" adapters. Some technologies require PCR amplification of fragments, while others do not.

Sequencing Next, specialized machines detect the sequence of nucleotides that make up the extracted DNA fragments. These processes, referred to as sequencing, employ various—often proprietary—technologies to determine the precise order of nucleotides (A, C, G, and T) in a DNA molecule. The raw data generated by these machines consist

of sequences of characters, commonly referred to as sequencing reads or simply reads.

Analysis In this step, quality control (QC) is usually performed to remove adapters and reads that are too short or of low quality. Typically, the first step after QC is either the assembly of the sequences or their input into a workflow specific to the intended application.

The landscape of DNA sequencing has evolved significantly since its inception. In 1977, Frederick Sanger and his colleagues introduced the first widely adopted sequencing method, known as chain termination sequencing, or Sanger sequencing [8]. This technique enabled the reliable and reproducible determination of nucleotide sequences in a DNA molecule for the first time. It was the method used to achieve the first sequencing of mitochondrial DNA and the first (almost) complete human genome in 2001 [9, 10]. Sanger sequencing, through gel-based techniques, produced the first reference genomes for several important organisms. Although Sanger sequencing revolutionized genetic research, it has largely been replaced by more advanced technologies. These newer methods fall into two main categories: Next Generation Sequencing (NGS) and Third Generation Sequencing. These technologies offer significant improvements in terms of speed, cost-effectiveness, and data output compared to Sanger sequencing.

2.1.1.1 Next Generation Sequencing

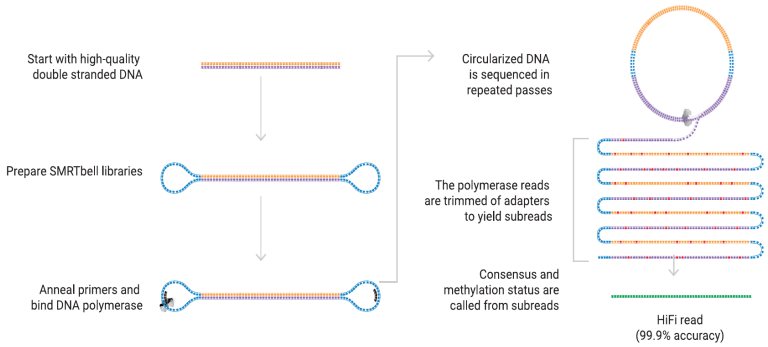
(NGS) derives its name from initiating a so-called "next generation" by revolutionizing sequencing through massive parallelization. This technology has continuously improved since 2005, currently yielding up to 8 terabases per single sequencing run in a maximum of two days, and has reduced the cost to sequence an individual's genome to nearly 100 dollars [11]. The advancement is primarily due to the ability to run many reactions and analyses in parallel, producing millions to billions of reads with lengths varying between 150 and 300 bases. These reads are known as short reads. A significant advantage of this sequencing method is its low error rate, with at least 80% of the bases with fewer than 1 error per 1000 bases (i.e. 99.9% accuracy). This technology is largely dominated by the California biotechnology company Illumina.
However, the main drawback of NGS is the short length of the reads. Due to their brevity, short reads are insufficient for assembling a high-quality *de-novo* complete genome and are therefore predominantly used for *re-sequencing*, where the reads are mapped to a reference genome to infer variations. This makes them valuable for studies of population variation, for example. Additionally, the short length of the fragments often impedes the mapping of sequences from genome regions that are either absent from the reference or are complex and/or repetitive, resulting in the loss of information from such regions.
This limitation has been partially addressed by the introduction of paired-end sequencing, a technology now integrated into all Next Generation sequencing machines. Paired-end sequencing reads both ends of a DNA fragment and

associates the two reads to provide more long-range information. Although this method is still not sufficient to resolve all complex variations, it significantly improves the retention of reads that would otherwise be discarded and has found relevant applications in other fields such as metagenomics. In fact, I utilized this feature of paired-end reads in a method I developed prior to my PhD to improve the estimation of different species in environmental samples sequenced with NGS [12].
Finally, NGS technology has also enabled other types of sequencing, such as RNA-seq, ATAC-seq, ChIP-seq, and others, which allow for the assaying of regulatory activity within the genome.

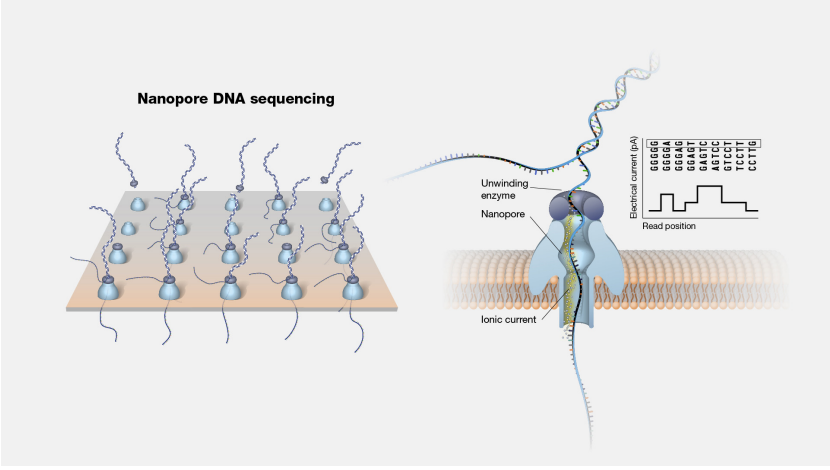
2.1.1.2 Third Generation Sequencing

(TGS) is the latest technology, offering alternative approaches to NGS to address the challenges posed by the short read lengths. The primary distinction is that, while NGS amplifies small fragments of DNA using PCR before sequencing, these new technologies directly sequence nucleic acids in their native form, which is why they are referred to as *single molecule technologies*.
There are two prominent technologies that are provided by the leading companies in this market: Pacific Biosciences (PacBio) and Oxford Nanopore Technologies (ONT), which are two biotechnology companies, based in California and United Kingdom, respectively.
PacBio offers "Hi-Fi sequencing" which produces reads up to 25,000 bases in length, with an accuracy comparable to that of NGS. This is achieved by first obtaining a circularized DNA from high-quality double stranded DNA and then using a DNA polymerase enzyme to read multiple times the same molecule to produce a final consensus sequence with accuracy of around 99.9%. These are long and accurate reads that enable ultra-fast assembly of human genomes [13] at a cost around $1000 per sequencing reagents kit for a 30X coverage of a human genome.
Oxford Nanopore machines instead generate *ultra-long reads*, which are on average longer than the PacBio HiFi ones and can reach up to the megabase scale (i.e. ~ 100 times longer). The sequencing is done by passing a single-strand DNA molecule trough a tiny nanopore. Each pore is associated to an electrode and a sensor that measure the current that is passing through the pore. As the DNA goes through the pore, the current changes and, thanks to a Machine Leaning based basecalling algorithm, it is possible to detect the nucleotides by the change in the current. This process is done in parallel across $\sim 800 - 1500$ pores. An issue with this kind of sequencing are homopolymers, i.e., stretches of DNA that repeat the same nucleobases. As there is no large change in the current it is difficult to estimate the number of bases that pass through the pore.
Thanks to the length of the sequence they produce, both technologies enable the detection of all types of genetic variations, including both small and large-scale variations, and can resolve large repetitive regions by spanning thousands of bases. Additionally, both methods allow for the direct detection of DNA methylation directly from the sequenced molecule. DNA methylation is a

chemical modification that occurs on the DNA molecule and regulates gene expression by recruiting proteins involved in gene repression or by inhibiting the binding of transcription factors to DNA [14].
Figure 2.2 provides basic schematics illustrating how these two technologies work.



(a) Pacific Biosciences Hi-Fi reads generations scheme. Image from PacBio website [15].



(b) An array of pores sequences multiple molecules in parallel. A double stranded DNA (dsDNA) molecule is split by the helicase enzyme and then a single stranded DNA (ssDNA) sequence slowly gets through the pore for sequencing. Changes in the ionic current is used by a Deep Learning algorithm to infer the nucleotides of the sequence. Image from NIH website [16]

Figure 2.2: Third generation sequencing technologies.

2.2 From reads to *k*-mers and beyond

The sequences produced by any of the aforementioned technologies can be considered as text strings, i.e. successions of characters, like the phrases of this manuscript, in which each character correspond to a nucleotide. These sequences can therefore being stored in plain text formats, like FASTQ, that preserve basecalling quality information or in others, like FASTA, that retains only the actual sequence. In order to use less space and take advantage of redundancy in the sequencing data, these files are often compressed, using one of the many tools publicly available like `gzip` or `zstd`, by Facebook.
As paired-end short-reads have different features than ultra-long reads or Hi-Fi long reads, most of the tools focus on providing applications for just one single sequencing type. In cases like assembling a genome from the reads or detecting variations in the sequenced genome compared to the reference, the information from different sources can be combined to provide superior results. In order to generate high-quality genome assemblies, for example, many consortia, like the Human Pangenome Reference Consortium (HPRC) or the Telomere to Telomere consortium (T2T), use Hi-Fi long reads as bases for assembly plus ultra-long reads as scaffolds to chain together the assemblies into sequences that span from telomere to telomere of a chromosome.
In the work presented in this manuscript, most of the tools will ingest as input or raw sequences (both NGS or TGS) or high-quality, near telomere-to-telomere assemblies. Some of the tools that I have used and all of the ones I have developed or co-developed transform the input sequences or assemblies into k-mers to produce the desired output.

DNA alphabet The DNA alphabet, denoted by $\sum$ consists of the four characters representing the first letter of the nucleobases: $\sum = \{A, C, G, T\}$,

sequence a biological sequence from $\sum$ is defined as $S \in \sum^l$, with $|S| = l$, with length l of arbitrary length.

sequence read a read is a biological sequence obtained from sequencing. Its length, denoted as z, is typically fixed, if generated by NGS, or variable, if produced by TGS.

***k*-mer** a k-mer of S, denoted as x, is defined as $x \in \sum^k$, with $|x| = k$ i.e. any valid sub-string of S of length k.

As shown in table 2.1, from any sequence S, it is possible to obtain its constituent k-mers. To efficiently extract all k-mers from a sequence, the best approach is to employ a sliding window technique. This is done by identifying the first k-mer at the start of the string and then iteratively shifting the window one position at a time, appending the newly encountered character to the right while removing the leftmost character.
The length k of a k-mer is an arbitrary value, that is usually chosen depending on the kind of sequences used (cannot have $k > n$, with n the length of the

read from which it is derived), the characteristics of the data that is used (is it from a single organism, a collection of the same species, a collection of different organisms) and on the disk or memory space that is available for computation or storage (as in table 2.1, the longer the k, the more space is used by repeating the characters of the same underlying sequence in multiple k-mers). A more detailed explanation of these considerations will be provided in section 4.5.
As it is possible to retrieve k-mers from a single read, it is trivial to extend this property to any set of reads, for example produced by a single sequencing run of a sample. This does not directly mean that a set of k-mers is not properly equivalent to the set of reads it is obtained from. In order to characterize this transformation as lossless, i.e. without any loss of information, an association from each k-mer to the read(s) it comes from would be needed. In most of the cases this is not useful and k-mers are obtained from reads without remembering from which reads they come from. In other, specific, applications it might instead be needed to know in which reads there are certain k-mers [17]. As presented in section 2.1 the DNA is double-stranded, with A bases are paired with T ones, while C bases are paired with G ones, also called complements. If a k-mer appears in a sequence, in the other strand of the molecule there would be what is called its reverse complement. This is the spelling of the k-mer from the end to the beginning, substituting each base with its complement. For example if in one strand there appear the sequence $ATGC$, on the other strand would spell $GCAT$.
When enumerating k-mers from a sequence or when storing them, only "canonical" k-mers are kept: this means that for each k-mer produced from a sequence, its reverse-complement is computed and only the one that is considered smaller by a certain property is kept. For example, if the lexicographic order is used, the k-mer (with $k = 4$) $ATGC$ is lexicographically smaller than $GCAT$ so when either of the two is seen, only the first is kept.
A classic operation that is done when enumerating k-mers from sequences is to keep track of how many times each canonical k-mer appears in the set of sequences. This is called k-mer counting and finds important applications in many genomic disciplines like metagenomics or transcriptomics.
k-mers are being used in lots of applications based on NGS short reads while they are less applied on methods for error-prone long reads because using k-mers on one side destroys the long range information provided by reads that span thousands of bases, on the other hand, error-rates higher than NGS would produce too many erroneous k-mers that would be very difficult to correct if not with very deep sequencing, providing additional cost bottlenecks. With Hi-Fi reads and improved quality of Nanopore basecalling, it is possible to overcome the error limitation and use k-mers for long reads. One example that uses advanced concepts based on k-mers is the tool `mdbg` that drastically improved assembly of Hi-Fi reads.

2.2.1 *k*-mer based objects

Unitigs correspond to the string spelled by a non-branching path in a de Bruijn graph (dBG), which is a graph representing the overlap between all unique k-mers from a set of sequences. In almost all applications unitigs are considered as maximal, i.e. the result of the maximum non-branching path in the graph. Non-branching paths are lists of nodes that have in-degree and out-degree of 1. The dBG is more formally defined in section 2.4.4.
Minimizers are strings of fixed length that are used to subsample k-mers from sequences, since consecutive k-mers are overlapping and contain redundancy. The sampling is done by a pre-defined optimization function, with guarantees that the sampling will produce similar outcome from similar sequences. The function depends on the application and can be for example lexicographic order or the minimum value of a transformation (hence the word *minimizer*). There are multiple declinations of minimizers. The first distinction is on the way they are chosen: on a sliding window or the 'universe'. When minimizers are chose on a sliding window of length w, every k-mer is evaluated against the chosen function and each l character the one with the best value is retained. Universal minimizer are instead chosen independently of any spacing in the sequence and are the ones whose output value of the defined function is, for example, smaller than a threshold.
Minimizers can be also used as smaller subsequences of length $m < k$ inside k-mers to help group together k-mers based on their corresponding minimizer. In this case from each k-mer the *m-mer* that minimizes the function will represent it.
Super*k*-mers are strings produced by concatenating adjacent k-mers sharing the same minimizer. They reduce the redundancy of consecutive k-mers and decrease the amount of data needed to represent a set of k-mers.

Position	1	2	3	4	5	6	7	8	9	10
Sequence S	C	T	G	A	A	C	T	A	C	A
$3-mers$	C	T	G							
		T	G	A						
			G	A	A					
				A	A	C				
					A	C	T			
						C	T	A		
							T	A	C	
								A	C	A

Position	1	2	3	4	5	6	7	8	9	10
Sequence S	C	T	G	A	A	C	T	A	C	A
$4-mers$	C	T	G	A						
		T	G	A	A					
			G	A	A	C				
				A	A	C	T			
					A	C	T	A		
						C	T	A	C	
							T	A	C	A

Table 2.1: k-mers with $k = (3, 4)$ being computed from the sequence `S = CTGAACTACA`. $l - k + 1$ k-mers are generated for a total of $(l - k + 1) * k$ bases. While with $k = 3$ the total bases are $8 * 3 = 24$, with $k = 4$ they are instead 28, as larger k encodes more information redundancy.

Sequence id	sequence
seq1	ACATCA
seq2	CTTCAG
seq3	TACAGC
seq4	GCTTAC

Sequence id	seq1	seq2	seq3	seq4
k-mers	ACA (TGT)	CTT (AAG)	TAC (GTA)	GCT (AGC)
	CAT (ATG)	TTC (GAA)	ACA (TGT)	CTT (AAG)
	ATC (GAT)	TCA (TGA)	CAG (CTG)	TTA (TAA)
	TCA (TGA)	CAG (CTG)	AGC (GCT)	TAC (GTA)

ordered canonical *k*-mer	count
AAG	2
ACA	2
AGC	2
ATG	1
ATC	1
CAG	2
GAA	1
GTA	2
TAA	1
TCA	2

Table 2.2: Example of canonical *k*-mers enumeration and count. Given a set of sequences, *k*-mers are computed in a stream. For each of them, on the fly, the reverse complement is computed. Then the ones that are considered canonical are passed and counted.
Reverse complements are between parentheses and the canonical between the two (by lexicographic order) is underlined.

2.3 Genetic diversity: focus on humans.

The Human genome contains more than 3 billions base pairs and includes 42,611 genes, of which 20,352 are presumed protein-coding genes, i.e. specific sections of DNA that serve as blueprints for proteins. The remaining 22,259 are non-coding: they produce RNA that serves a biological function but that it is not translated into proteins [18]. The rest of DNA consists of regions that function as regulatory element, like enhancers, promoters and silencers or as other conserved, functional element. Variations in specific regions cause phenotypic changes that can increase, decrease or not affect the fitness of an individual.
Genetic diversity is the variability that exists between organisms at the genetic level, i.e. differences in the information enclosed in their DNA. It is the raw material for biological evolution as, without heritable genetic differences between us, we would not be able to biologically evolve. Here what I will present is valid for humans, as the large part of my work has been with human DNA sequences. Most genetic changes have no effect on the individuals carrying them but some can result in phenotypic differences.

2.3.1 Causes and drivers of genetic diversity in humans

There are two main mechanisms of genetic diversity: the rise of new mutations and the reshuffling of already present genetic material trough recombination and duplications. Mutations can arise through two processes. First, if physical or chemical damage, such as caused by UV radiation, occurs prior to cell division and is not repaired, a wrong base can be integrated in the DNA of the cell. Second, errors during the DNA replication in cell division can lead to mutations. When this occurs in germinal cells they are transmitted to the offspring, while when happening in a somatic cell (not reproductive), the mutation is not transmitted but can instead be responsible for certain types of cancer. In humans, it is estimated that a newborn carries on average 70 point mutations (one nucleotide substitute with another), 15 from the mother and 55 from the father. The amount of mutations is proportional with the age of the person and, more than induced by replication, it is due damaging of the DNA molecule that is not corrected [19].
On top of the mutations, chunks of chromosomes from the mother and the father chromosomes are shuffled to produce new combinations. The effect of this random process produces the differences between siblings with the same biological parents. Recombination is heterogeneous in the DNA and depends on some motifs that promote higher recombination. Finally, recombination is also influenced by the age, mostly of the mother, as older mothers tend to produce offspring with more misplaced recombinations, also causing the well known trisomy 21.
Without diverging too deep into population genetics, it is also important to understand how new variations are conserved, lost or fixed (become prevalent)

in a population. These outcomes are driven by two main factors: genetic drift and natural selection.
Genetic drift is a process, given by the randomness in the individuals that reproduce in a specific population. This can contribute to the loss or fixation of some variants just because of randomness and not because they provide an advantage to the individual. Specifically, in populations with small number of reproductive individuals, this can fixate detrimental variants, while in large populations, the large number of individuals buffers the event.
Natural selection, on the other hand, is a mechanism that explains evolution: as genetic variations causes the gain or loss of specific phenotypic traits, these traits can confer positive or negative advantage compared to the rest of the individual in a population (fitness). This phenomenon can contribute selecting certain variations in a population by either contribute to the fixation or the loss of a variant. This mechanism explains our species adaptation to nutritional resources, climate and pathogens: in 10 thousands years a mutation in a gene that conferred the ability to digest milk as adults, the lactase persistence, applied such a strong selective pressure that has almost reached fixation in some human populations (mostly of North-European or African ancestry) [20]. Selection on certain genes explains better adaptations to cold or high altitudes and selection in some alleles (HbS or DARC genes) has helped humans adapt and survive malaria infections [19].

2.3.2 Human Genomic variation: types of variants

There are various types of genomic variants: from the shortest, the single-nucleotide variants (SNV) or Single Nucleotide Polymorphism (SNP) when it is present in at least 1% of the population, is the difference of one nucleobase between two individuals. For example, in a specific part of the genome one person can have instead of a cytosine (C) a thymine (T), like for the SNP located 14 thousands bases upstream of the lactase gene that enables the lactase persistence mentioned earlier [21]. A second group of small variants is made of insertions and deletions (called together *indels*): these are events in which a group of less than 50 nucleotides is added or removed in a sequence compared to another one (e.g. a reference sequence). The number of 50 nucleotides is an arbitrary threshold used to better separate them from other kind of variations. Specific types of indels are the tandem repeats that, as the names suggests, are insertions or deletions of small repeated sequences of DNA. These repetitions usually are one after the other with no other sequence in between [22].

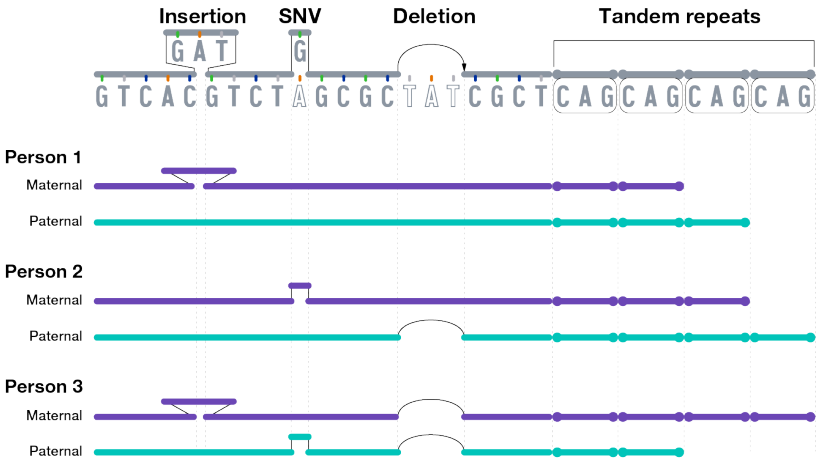


Figure 2.3: Graphic showing the types of small genomic variants. Individuals have different genomes, and these differences are encoded as variations in their DNA sequences. The most common type of variation between individuals is a Single Nucleotide Variation (SNV), in which a base at a specific position is different (e.g., A and G). Another common variation is the presence of extra or missing nucleotide(s) in a specific part of the DNA: when this difference involves fewer than 50 nucleotides, it is called an insertion or deletion (INDEL). Finally, tandem repeats are contiguous repetitions of a small stretch of nucleotides.
In this example Person 1 has an insertion in the Maternal haplotype and a different number of repetitions of `CAG` in the tandem repeats. These are referred to as heterozygous variations because they differ in their form between the two chromosome copies.
Person 2 has an SNV in the maternal haplotype, where the base G appears instead of the more common A. In the paternal haplotype, the `TAT` sequence is deleted, while the number of `CAG` repeat copies is different: with four on the paternal haplotype and three on the maternal chromosome.
In the case of In the case of Person 3, a deletion is present in both haplotypes, while they differ on the number of tandem repeats and presence of Indels or SNVs. Graphic taken from [22].

These groups of small variants, shown in figure 2.3 are the most described, studied and associated with diseases as they were the only one consistently detectable with NGS sequencing. For these reasons, studies that tried to associate genomic variation with diseases commonly used only these kind of variants.
The other kind of variations are the ones that stretch at least 50 nucleobases and that can reach the dimension of large chunks of the chromosomes: they are called structural variations (SV). These can be indels or tandem repeats with

the repeated section longer than 50 nucleotides, accounting for nearly half of all SVs [22], that take the name of Copy Number Variants (CNV). Moreover, there are also inversions, in which a chunk of DNA is inverted compared to another and translocations in which pieces of two different chromosomes trade places [22].

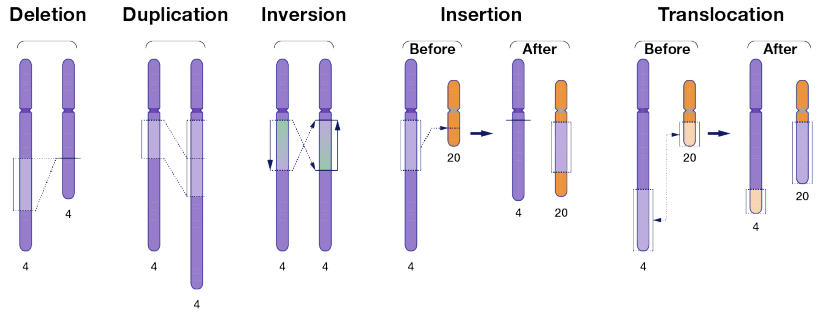


Figure 2.4: Graphic showing the types of large genomic variants, also named structural variants. Deletion or insertions of more than 50 base pairs are considered structural variants.
Duplications are when large segments are copied one or multiple times. They tend to have a nested and modular structure. The copies can span non-coding regions or genes. Different copies of genes can alter the expression of the associated protein. This is not the case for gene TBC1D3, a primate-specific gene associated to increase of the prefrontal cortex. The high variability in the human population for such an important gene has been recently explained by the fact that the expression is limited to a subset of copies [23].
An inversion is a segment of a chromosome that is present in the reverse orientation as a result of breaking off and errors in the reattachment. Inversion can be cause diseases, like hemophilia A, or increase the risk of further mutations that can cause disease, like for the microdeletion syndrome [24].
A translocation occur when a chromosome breaks and a portion is attached to another chromosome: this event can cause diseases like leukemia [25]. Graphic taken from [22].

Finally, these kind of variations can be present in just one haplotype (copy of the chromosome) or in both. An heterozygous allele is referred to having inherited from the mother and father a different version of a specific part of the genome.

2.3.3 The importance of studying genomic diversity in populations context

DNA differs between individuals of the same population (inter-individual) and between different populations of the same species (inter-population): figure 2.5 shows the percentage of inter-individual variation for four close primates. Different species may vary in the amount of genetic diversity present between individuals within a population, as seen in humans, or between populations, which accounts for a significant portion of genetic variation in orangutans. As discussed before, differences in DNA are given by having a different nucleotide at the same place (SNV), indels and large and complex variations, up to megabases, that can produce different counts of copies or different ordering of a same region. On average, each human carries around 10 thousands amino-acid altering mutations, 300-400 gene disruption events (like stop, splice and indels) affecting 200-300 genes and is heterozygous at 50-100 mutations associated with an inherited disorder [19]. Finally, even when close species share a large portion of genetic material, structural changes that rearrange the same material in different order or invert it, contribute to meaningful changes. In figure 2.6 it is shown how the chromosome 7 and 16 of some primates, even if very similar, differs in terms of organization. These large structure rearrangements are thus fundamental to understand the biology of organisms.
Moreover, genetic diversity is driven by two main factors: genetic drift and natural selection. Genomic duplication followed by adaptive mutation is considered one of the primary forces for evolution of new functions [23].

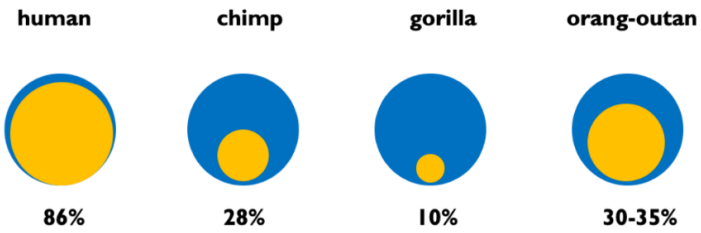


Figure 2.5: Share of inter-individual (yellow) and inter-population (blue) diversity for four different primates. While for humans the majority of the diversity is within populations, for other primates it is between populations. This shows how Humans are more mixed than other primates. Percentage shows the inter-individual variation share. Graphic taken from [19].

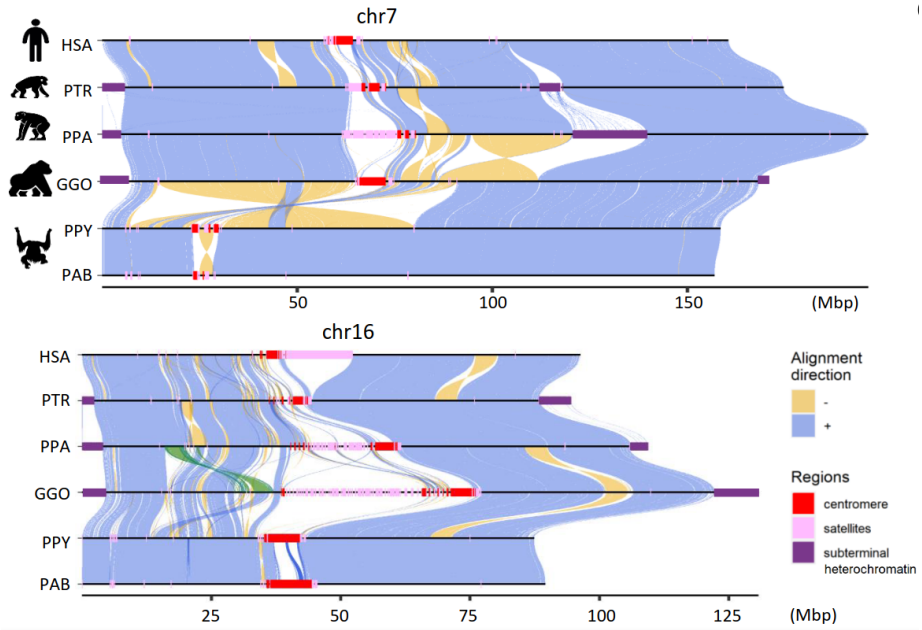


Figure 2.6: A comparative ape alignment of human (HSA) chromosomes 7 and 16 with chimpanzee (PTR), bonobo (PPA), gorilla (GGO), Bornean and Sumatran orangutans (PPY and PAB). The image on the top shows most of the chromosome 7 is conserved except for large inversions happening between the species. The image below shows complex inversions in chromosome 16. Image taken from "Complete sequencing of ape genomes" [26].

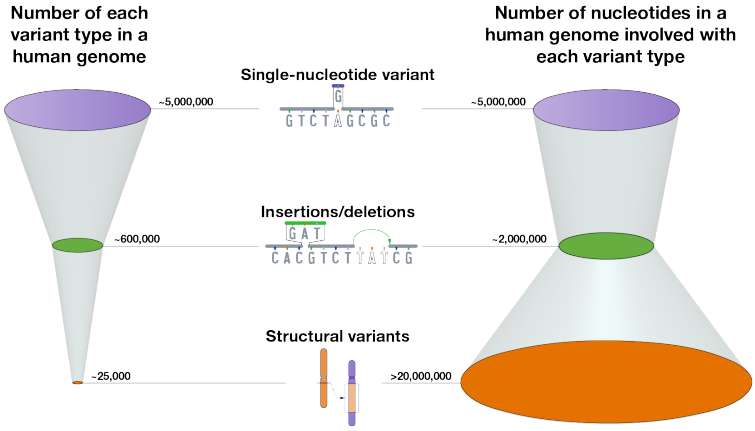


Figure 2.7: Spectrum of Human Genetic Variation. While SNPs are the most common variation event, their impact in the total amount of bases in a genome is ~ 4 times smaller than the one of Structural Variations, that are 200 times less frequent. This shows the great need to consider SVs in genomic analysis and not to stop at the SNP/indel level. Graphic taken from [22]

2.4 Pangenomics, pangenomes and pangenome graphs

2.4.1 The premises for Pangenomics

There are a number of factors that must be taken into consideration to understand on one side the need for a new paradigm and on the other side the conditions that lead to its development. Here I will briefly expose some of them before diving into pangenomics approaches and methods.

2.4.1.1 A single linear genome for all analyses

Since the first complete genome sequences have been available in the late '90s, all analyses based on sequencing data depended upon the use of a single linear reference genome, i.e. the best version of the genome available for any species. This reference sequence can originate from the genome of a single organism or be a patch and consensus of multiple available genomes of the same species. Its purpose is to be used to infer information from newly, less refined, genomes that are being sequenced. We now know that this approach is suboptimal in a wide range of applications as a lot of genetic material of the species cannot be present in a single linear representation: this is valid for eukaryotes and even more for bacteria, that tend to be very diverse even in the same strain. The goal would therefore be to find a representation that provides more genetic material of a

single species by intelligently combine the information from genomes of multiple organisms and their differences.

2.4.1.2 A quantity and quality revolution

In the last few years we are witnessing a new revolution in sequencing. As the price of sequencing is lowering more than 2x per year, from \$1/basepair to $\$10^{-7}$/basepairs [27], new scientific discoveries and technological advances are leading to a remarkable increase of quality, in term of per-base error rate, and throughput of TGS. This means that in the near future we will dispose of a rich wealth of high quality sequencing information to produce hundreds or thousands of new first grade assemblies of large eukaryotic genomes.
For example, the history of complete human genome assemblies clearly exposes how much more high quality genomes it is now possible to generate. The Human Genome Project took 13 years to produce its result [28] and the absence of long reads with low error rate made it impossible to automatically resolve repetitive regions like telomeres and centromeres [29], producing a reference only 92% complete [30]. This problem was only solved in 2022 with a new, reference genome that did not have any gaps or unresolved regions, from the telomere to the other telomere of each chromosome [30]. Now, many consortia are producing increasingly more genomes to a level comparable to the one produced in 2022. For example, the HPRC released 47 new human genomes (92 haplotypes) in 2021 and has recently released other 153 genomes to a total of 400 haplotypes of very high quality.
Finally, it is important to understand the quantity of biological information produced. As shown in table 2.3, the number of base pairs sequenced has more than doubled each year since 1995. As this is faster than the famous Moore's law on computing power, it is becoming evident that a new paradigm is needed to store and analyze such wealth of data. Public repositories, like Sequence Read Archive (SRA) and European Nucleotide Archive (ENA), are rapidly increasing the number of samples being sequenced and rendered publicly available to everyone, with tens of billions of millions of basepairs from genomic samples, as shown in figure 2.8. Other repositories of genomic data with associated medical metadata, like the UK biobank that comprises around 500 thousands individuals, are also emerging. These conditions are pushing the adoption of novel methods to process and analyze genomes.

2.4.1.3 The need to better understand difference between genomes

The ability to produce such good data is the main enabler of increasing efforts from the scientific community to propose new methods to analyze genomes: not anymore by comparing new genomes against a single good reference sequence but by comparing it in a comprehensive representation of the species.
Moreover, as new high quality sequences and assembled genomes are available, complex and/or highly repetitive regions can be now represented also for new genomes therefore enabling comparison between the ones of different genomes.

year	genome(s)	base pairs
1995	Bacterium	$2 * 10^6$
2001	Human	$3 * 10^9$
2012	Tomato	$9 * 10^8$
2013	2500 humans	$7.5 * 10^{12}$
2018	Wheat	$17 * 10^9$
2021	1M genomes	$3 * 10^{15}$

Table 2.3: Scale of DNA sequencing achievements over the years, with underlined the first reference genomes. Sequenced base pairs are now 10^9 times compared to 30 years ago. The amount of data available more than doubled each year in the last ~ 30 years. The rate was derived from $\log_2(10^9) = 29.9 \implies 10^9 \approx 2^{30}$.[27].

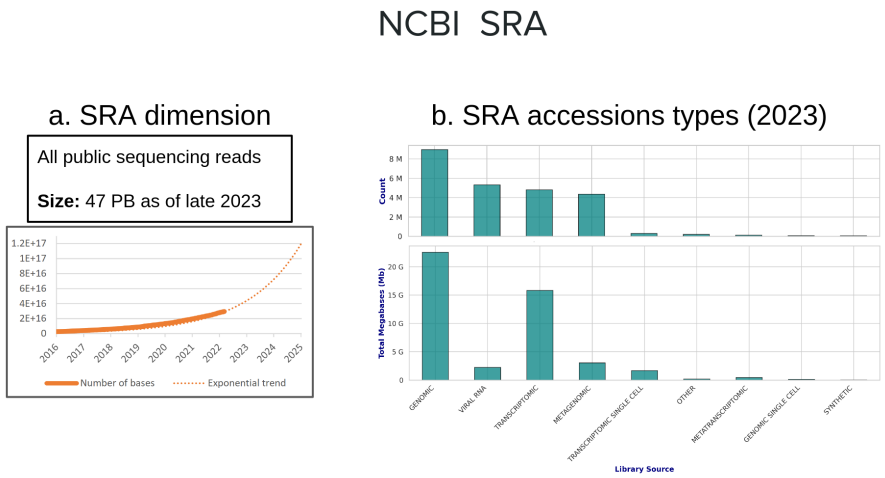


Figure 2.8: a) The size in petabases (peta = 10^{15}) of the SRA archive; b) The type of data in the SRA database shows the vast amount of genomic data available. Image made from Rayan Chikhi's slides.

This is very important as up until these improvement in sequencing and assemblies arrived, analyses were mostly blind to large, complex and/or repetitive structural variations. As we now know that these are the ones responsible for most of the difference between human genomes, new proposed approaches should provide new and better tools to understand, represent and analyze such variations.
Finally, as a single reference sequence cannot enclose all the possible structural variations of a population into a linear model, the need for a change in data structure for genomics arises.

2.4.2 Pangenomics

Pangenomics is a rapidly evolving field in genomics that aims to capture and analyze the full genetic diversity within a species or a group of closely related species [31, 32, 33]. Unlike traditional approaches that rely on a single, linear reference genome, pangenomics compares genomes to a collection of others, representing all genetic variations and structural differences across a set of genomes [32, 33]. It leverages large collections of high-quality assemblies from many individuals or species to overcome the observational bias inherent in using a single haplotype as a reference for an entire population. As illustrated in Figure 2.9, the pangenome model strives to represent all variations among a group of complete genomes by describing their direct relationships, whereas the linear model compares each genome only to a reference. While traditionally, in genomics, genomes were indirectly compared through their differences relative to a single linear reference, pangenomics enables direct comparisons between genomes. When a new genome is added to the collection, traditional genomics compares it solely to the reference genome, whereas in pangenomics, it is compared to all genomes in the model [32]. It was first conceptualized for bacterial genomes, and at gene level, without considering non coding regions [35, 36]. This was mostly due to the fact that bacteria share genes between each other, generating high diversity in the gene repertoire between organisms of the same species or strain [36]. The first proposed pangenome model had a subdivision between a core genome, made by genes present in all individuals of a species, and a dispensable or accessory genome, with genes present in some, but not all, individuals.
This definition would then extend to a more general model that would consider variations at the nucleotide level to contain all variations in a set of genomes.

2.4.2.1 Pangenomes

A pangenome can therefore be considered as any collection of genomic sequences to be analyzed jointly or to be used as reference [32, 33]. This definition provides two important concepts for the rest of the studies provided in this manuscript:

Model The pangenome is not a well-defined structure or model; it can range from a simple collection of sequences to more complex data structures. As a result, various approaches have been developed and are employed depending on the specific application or research focus.

Scope a pangenome can be either used as:

- A new reference for a specific species to be used for analyses in a similar way as linear genome. This means that a large consortium would be producing a representation that is accepted as new standard. For the Human genome this is done by the HPRC consortium as the T2T consortium produced the best-quality linear reference genome [30].
- A different model that can be used to study a set of genomes, without needing *a priori* to use a reference. This model can find applications in population variation studies.

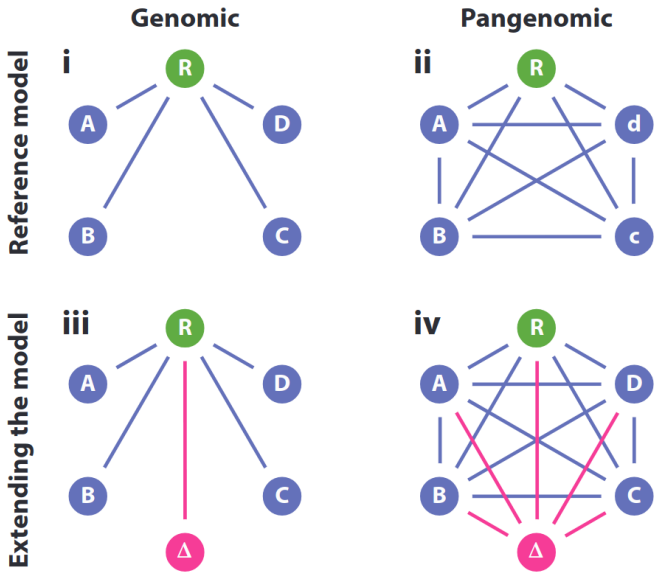


Figure 2.9: The genomic vs pangenomic model. i) In the genomic model each genome is compared only to the reference sequence. Any comparison between a pair of genomes is done indirectly via their difference with the reference. ii) In the pangenomics, variations are described in a relative way for any genome. Any pair of genomes can be compared directly. iii) In the genomic model, to add a new genome in the collection it has to be compared to the reference. iv) In pangenomics, each new genomes added to the model is automatically compared to all the ones in the collection. Figure from [34]

A pangenome can also be an unaligned set of sequences [32]. This is the most basic case, with no processing of the data but that conserves the full information from the assembly without introducing any bias or error. In this sense, a group of complete genomes of a family, species or genera can be considered a basic form of pangenome. They can be used together to infer direct relationships between each other, via alignment. For example, as the T2T consortium has fully resolved the centromeres of 2 human genomes, when considered together, it is possible to detect small-scale and large-scale centromere variations, something that was not possible before [5]. By having high quality assemblies of various apes, it is possible to reconstruct complex and large variations and rearrangements in chromosomes between them and the human genome that could not be detected before [**apes_genomes**].
A multiple sequence alignment (MSA) of haplotype-resolved complete genomes can be considered a pangenome. This data structure originated from complex and costly alignment operations is the basis of many computational approaches,

also in pangenomics, like the founder graphs. These models are limited in scope as it is impractical as it does not work well when genomes are too large, have complex variations or are very divergent.
Pangenomes can be also represented as sets of k-mers. This approach has several advantages: it scales very well to large collections of genomes, accepts as input raw reads to complete assemblies and is unbiased. The drawbacks mostly consist on the right choice of the k-mer length and the loss of positional information that is naturally encoded in reads or assemblies.

2.4.2.2 Pangenome Graphs

Graphs are a natural way of directly representing information between a group of objects that share some properties: they provide a human interface to a set of relationships.
A graph is a mathematical structure used to represent associations between abstract entities. It consists of two main components:

Nodes (vertices) are the entities of the graph that possess some properties (like a value or label);

Edges are the connections between the nodes that represents the relationships between the nodes (shared property or difference).

Graphs are widely used in a lot of applications, mainly to describe and interpret complex structures in social, transportation or computer networks.
Graphical models are largely adopted to represent pangenomes. They differ in the property associated to the vertices and therefore in the information provided by the edges.

De Bruijn graphs have nodes with labels representing k-mers and overlap relationship between k-mers is expressed using edges; k-mers and their reverse complements are represented by the same node, so the graph is bidirected, with edges connecting a strand of a node label to another strand of a node label.

Directed genome graphs have nodes labels representing a sequence and edges signal adjacency of two sequences in at least one genome. A sequence and its reverse complement are assigned to two different nodes, as the only information represented by the graph is the contiguity of pairs of sequences.

Bidirected genome graphs have nodes with a label and two sides, i.e. the start and the end of the label. Edges connect one side of a label to the side of another, to provide the starting point of the sequence spelled. If a node is traversed from left to right, it is the forward strand of the sequence, if done right to left, it is the reverse strand of the DNA [37].

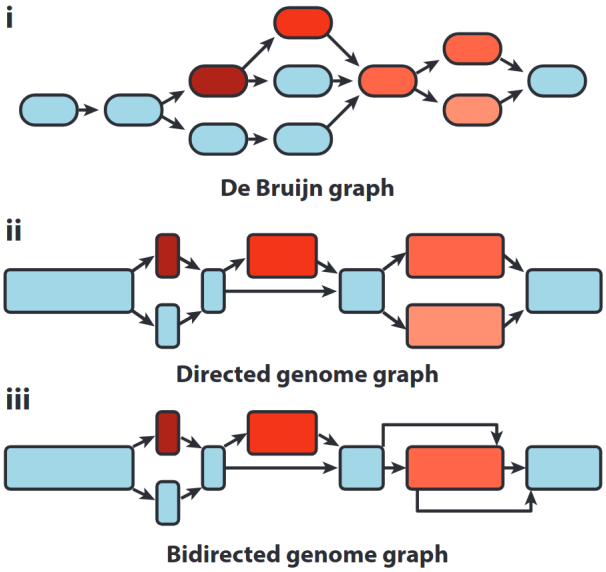


Figure 2.10: The three main kinds of graphs used to represent pangenomes. Figure from [34]

The choice of a particular model depends largely on the intended application, as there is no one-size-fits-all solution due to trade-offs between different desirable features. For instance, a model that facilitates effective visualization may not be suitable for handling large collections of genomes. Similarly, models that support the addition of new genomes without requiring complete recomputation—often referred to as dynamic updates—may not be the most efficient in terms of compression.

In the context of genomic variations represented in graphs, "bubbles" depict alternative paths between a source node and an end node. Different types of variations lead to distinct bubble structures, and different models generate bubbles with varying patterns.

Below, I introduce the two most widely used models for constructing pangenome graphs: Variation Graphs and De Bruijn Graphs. An example of how de Bruijn graphs and variation graphs represent variations in genomes is shown in Figure 2.11.

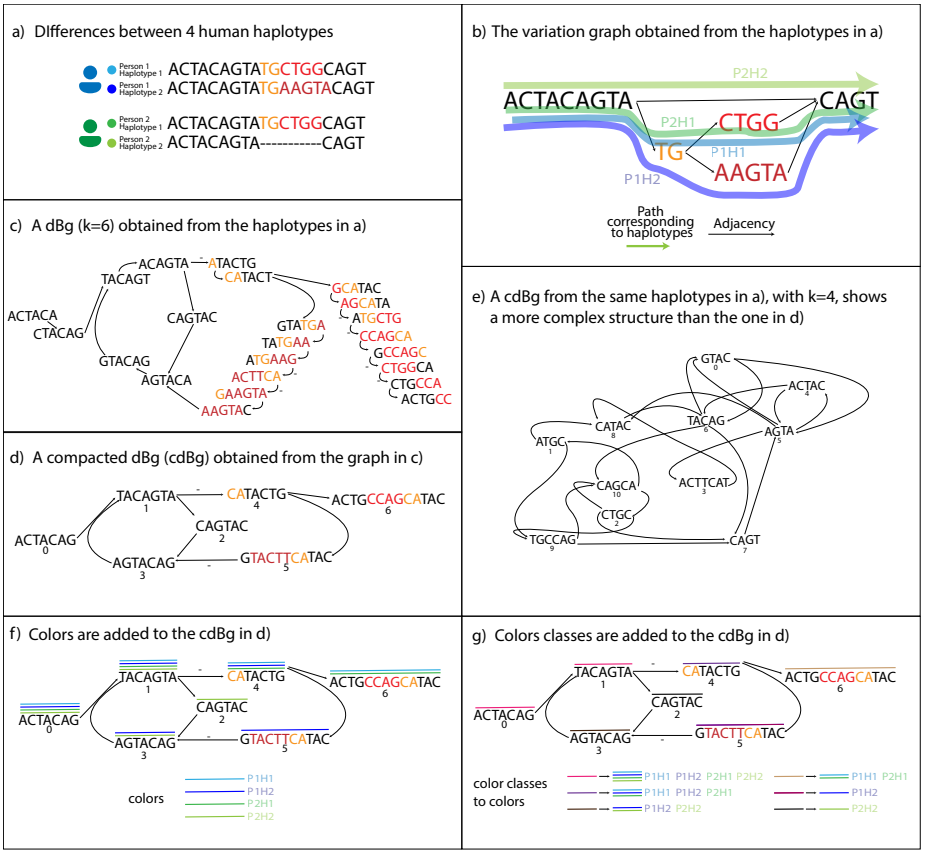


Figure 2.11: Example of variation and de Bruijn graphs models. a) The 4 haplotypes of two individuals with their variations highlighted; b) The variation graph obtained from the haplotypes is a), with paths spelling the haplotypes; c) the dBG ($k = 6$) obtained from the haplotypes in *a)*. For simplicity, nodes with a '-' flag when the overlap is between a k-mer and the reverse complement of the other. d) The compacted dBG from *c)* shows the reduction in redundancy, and the increased ease of visualization. Still, it does not provide direct visualization insights like the variation graph, as redundant part of the genome make the graph not linear. e) The compacted dBG of the same haplotypes in *a)*, with $k = 4$. This shows the increased complexity in the graph with smaller values of k. f) The color compacted dBG from *d)* with colors. Colors are assigned to k-mers and displayed in the figure on top of unitigs, for simplicity. Each color represents an haplotype. g) The color compacted dBG from *d)*, using color sets instead of colors. A map from color sets to colors is provided. This shows the difference between the two representations (colors and color sets).

2.4.3 Variation Graphs

Variation graphs are an enhancement of bidirected genome graphs with paths. Paths correspond to walks in the graph that visit nodes in an assigned orientation to reproduce sequences provided as input, as shown in figure 2.11. It therefore consists of a bidirected genome graph constructed from the sequences in the input genomes plus a list of paths that spell such sequences inside the graph. This data structure has been first proposed to represent textual variations. As shown in figure 2.12, the variation graphs represent the conservation and variation in a system: it is therefore a very good model to represent the direct relationships between a group of genomes.

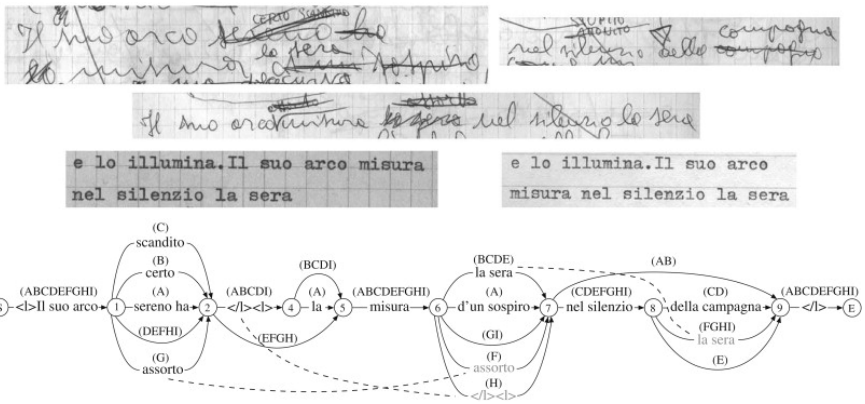


Figure 2.12: Instead of having to perform all the pairwise comparisons of the nine version of Valerio Magrelli's "Campagna Romana" poem from 1981, the variation graph structure describes the differences between them. It also removes the high redundancy in the versions of the poem [38, 7].

Variation graphs of genomes can now be constructed thanks to the advent of TGS and complete high quality assembly pipelines. As already discussed, genomes are full of repeats, making assembly is hard, especially if the only information available is reads shorter than the repeat sequences, as with NGS. Variation graphs can be generated in a direct, all-v-all unbiased way or in a iterative, reference driven manner.

2.4.4 De Bruijn Graphs

Similar to variation graphs, dBGs were not originally developed for pangenomics. They are a well-known data structure that has been widely used in genome assembly, particularly in the context of Next Generation Sequencing. A dBG represents a collection of input sequences as a set of k-mers. By storing each unique k-mer only once, the structure reduces redundancy in the input data. In the graph, nodes are labeled with k-mers, and directed edges between nodes represent an overlap of $k-1$ bases between the two k-mers. dBGs are also

bidirected, as each k-mer includes its reverse complement; the version present in the node label is usually the canonical one. Bidirected edges thus encode the strand orientation of both the starting and ending k-mers.

The choice of k depends on multiple factors and in fact is a trade-off between:

Specificity The larger the k, the sparser is the set of k-mers of the dBG in the space. While smaller k (21-31) is more general and suitable for most applications, larger values of k (61-100) provide greater specificity. The k-mers in genomes are not random and follow a skewed heavy-tail distribution [39]. Using larger k value reduces the probability that two genomes share the same k-mer by chance.

Variation resolution While variation is always encoded in the dBG, using larger values of k results in a sparser graph, where there are fewer nodes representing k-mers that occur multiple times in the genome. This implies that, when visualizing a specific region of the graph, it becomes easier to detect local variations without being confounded by nodes and edges coming from other parts of the genome that share the same k-mers as the region of interest.

Space As shown in table 2.1, larger values of k produces more redundancy and a greater number of basepairs with the same input sequences. If the collection to study is large, smaller k can provide beneficial features for disk storage or computational resources needed when using it.

2.4.4.1 Colored and Compacted De Bruijn Graphs

In order to produce a more compact and informative representation, in pangenomics it is used a particular version of the dBG, called ccdBG: colored compacted dBG. The main characteristics of this data structure are:

- Paths that do not contain any branches or bubbles are compacted into a single node. Given a chain of nodes in the graph, if the internal nodes have an in-degree of 1 and an out-degree of 1, the starting node has an out-degree of 1, and the final node has an in-degree of 1, the entire chain can be compacted into a single node. The resulting label is the extension of the first k-mer with the last nucleotide of the labels of the subsequent nodes. These compacted labels are no longer of length k and are thus not k-mers; they are referred to as unitigs, providing a more succinct representation of the k-mers in the de Bruijn graph, as shown in Figure 2.11.

- The genome of origin for each k-mer is recorded, and this information is referred to as its color, indicating which genomes the k-mer is present in. The color of each k-mer in the colored compacted de Bruijn graph is stored in a highly compressed format to minimize space usage. This color encoding enables more advanced analyses by allowing us to not only

examine the presence or absence of a k-mer in the de Bruijn graph, but also to determine in which genomes the k-mer was observed.

While k-mer compaction into unitigs can be performed as a post-processing step starting from a de Bruijn graph, many modern methods compute unitigs directly, bypassing the need for constructing the full dBG.
The term color can be somewhat confusing, as it is used in two distinct contexts: it may either refer to the dataset of origin for a k-mer, or it can represent any combination of datasets in which a k-mer is observed (commonly referred to as color sets) [40]. The color set model is the most widely used, as it assigns an integer identifier to every unique combination of datasets in which a k-mer is found, instead of storing the precise dataset affiliations for each k-mer. This approach greatly reduces space usage, as illustrated in Figure 2.11.

2.5 Main contributions outline

In the work presented below, we investigate the graphical pangenome representation on the features presented in the sections above. We focused mainly on the construction of such models, their underlying data structures and the downstream applications they enable.
The contributions presented in this manuscript are the following:

1. **An analysis of pangenome construction methods and their applications**. Even if the variation graph model has been devised around 15 years ago, its application for pangenomics is very recent. dBGs are instead a known model that has been extensively used for genome assembly and their application to pangenomics is relatively straightforward. We used all the state-of-the-art tools to produce a pangenome graph based on these two representations using a large collection of complete human genomes and tested computational resources, variation representation and applications.

2. **A novel construction of pathogenic yeast strains to discover chromosomal translocations.** Pangenome graphs can be used to investigate complex structural variations in genomes. In this case we sequenced, assembled and analyzed a group of 11 samples. We modified a variation graph construction pipeline to detect cross-chromosome events and discussed differences in the final representation.

3. **A unitig matrix construction pipeline for presence/absence or counts.** We proposed a small pipeline, based on already published tools and a novel method, `kmat_tool`, a pipeline to build unitig matrices with abundances from a set of genomes via k-mer matrix and to directly generate a presence absence unitig matrix using ccdBGs.

4. **A novel super$k$-mers enumeration and sorting method.** We propose a novel way to encode and store super$k$-mers that preserves locality for

the ones sharing the same minimizer to improve sequence queries. We also propose a tool to enumerate and sort super$k$-mers from a set of sequences using this model.

The next chapters present this work and discuss the possible directions for future improvements.

Bibliography

[1] Cobb, M. and Comfort, N. "What Rosalind Franklin truly contributed to the discovery of DNA's structure." In: *Nature* (Apr. 2023). DOI: 10.1038/d41586-023-01313-5. URL: https://www.nature.com/articles/d41586-023-01313-5.

[2] NIH. *NIH glossary on DNA*. 2024. URL: https://www.genome.gov/genetics-glossary/Deoxyribonucleic-Acid.

[3] Healey, A. L. et al. "The complex polyploid genome architecture of sugarcane". In: *Nature* vol. 628, no. 8009 (Apr. 2024), pp. 804–810. ISSN: 1476-4687. DOI: 10.1038/s41586-024-07231-4. URL: https://doi.org/10.1038/s41586-024-07231-4.

[4] Revy, P., Kannengiesser, C., and Bertuch, A. A. "Genetics of human telomere biology disorders". In: *Nature Reviews Genetics* vol. 24, no. 2 (Feb. 2023), pp. 86–108. ISSN: 1471-0064. DOI: 10.1038/s41576-022-00527-z. URL: https://doi.org/10.1038/s41576-022-00527-z.

[5] Logsdon, G. A. et al. "The variation and evolution of complete human centromeres". In: *Nature* vol. 629, no. 8010 (May 2024), pp. 136–145. ISSN: 1476-4687. DOI: 10.1038/s41586-024-07278-3. URL: https://doi.org/10.1038/s41586-024-07278-3.

[6] Gunnarsdóttir, E. D., Nandineni, M. R., Li, M., Myles, S., Gil, D., Pakendorf, B., and Stoneking, M. "Larger mitochondrial DNA than Y-chromosome differences between matrilocal and patrilocal groups from Sumatra". In: *Nature Communications* vol. 2, no. 1 (Mar. 2011), p. 228. ISSN: 2041-1723. DOI: 10.1038/ncomms1235. URL: https://doi.org/10.1038/ncomms1235.

[7] Garrison, E. *Building and Understanding the Human Pangenome*. 2023. URL: https://www.youtube.com/watch?v=ukopTzkfPVk.

[8] Sanger, F., Nicklen, S., and Coulson, A. R. "DNA sequencing with chain-terminating inhibitors". In: *Proceedings of the National Academy of Sciences* vol. 74, no. 12 (1977), pp. 5463–5467. DOI: 10.1073/pnas.74.12.5463. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.74.12.5463. URL: https://www.pnas.org/doi/abs/10.1073/pnas.74.12.5463.

[9] Anderson, S. et al. "Sequence and organization of the human mitochondrial genome". In: *Nature* vol. 290, no. 5806 (Apr. 1981), pp. 457–465. ISSN: 1476-4687. DOI: 10.1038/290457a0. URL: https://doi.org/10.1038/290457a0.

[10] Venter, J. C. et al. “The Sequence of the Human Genome”. In: *Science* vol. 291, no. 5507 (2001), pp. 1304–1351. DOI: 10.1126/science.1058040. eprint: https://www.science.org/doi/pdf/10.1126/science.1058040. URL: https://www.science.org/doi/abs/10.1126/science.1058040.

[11] Pennisi, E. *A $100 genome? New DNA sequencers could be a ‘game changer’ for biology, medicine.* URL: https://www.science.org/content/article/100-genome - new - dna - sequencers - could - be - game - changer - biology - medicine.

[12] Andreace, F., Pizzi, C., and Comin, M. “MetaProb 2: Metagenomic Reads Binning Based on Assembly Using Minimizers and K-Mers Statistics”. In: *Journal of Computational Biology* vol. 28, no. 11 (2021). PMID: 34448593, pp. 1052–1062. DOI: 10.1089/cmb.2021.0270. eprint: https://doi.org/10.1089/cmb.2021.0270. URL: https://doi.org/10.1089/cmb.2021.0270.

[13] Ekim, B., Berger, B., and Chikhi, R. “Minimizer-space de Bruijn graphs: Whole-genome assembly of long reads in minutes on a personal computer”. In: *Cell Systems* vol. 12, no. 10 (2021), 958–968.e6. ISSN: 2405-4712. DOI: https://doi.org/10.1016/j.cels.2021.08.009. URL: https://www.sciencedirect.com/science/article/pii/S240547122100332X.

[14] Moore, L. D., Le, T., and Fan, G. “DNA Methylation and Its Basic Function”. In: *Neuropsychopharmacology* vol. 38, no. 1 (Jan. 2013), pp. 23–38. ISSN: 1740-634X. DOI: 10.1038/npp.2012.112. URL: https://doi.org/10.1038/npp.2012.112.

[15] NIH. *Pacbio sequencing website.* 2025. URL: https://www.pacb.com/technology/hifi-sequencing/how-it-works/.

[16] NIH. *NIH glossary on Nanopore DNA sequencing.* 2024. URL: https://www.genome.gov/genetics-glossary/Nanopore-DNA-Sequencing.

[17] Baire, A., Marijon, P., Andreace, F., and Peterlongo, P. “Back to sequences: Find the origin of k-mers”. In: *Journal of Open Source Software* vol. 9, no. 101 (2024), p. 7066. DOI: 10.21105/joss.07066. URL: https://doi.org/10.21105/joss.07066.

[18] Pertea, M., Shumate, A., Pertea, G., Varabyou, A., Breitwieser, F. P., Chang, Y.-C., Madugundu, A. K., Pandey, A., and Salzberg, S. L. “CHESS: a new human gene catalog curated from thousands of large-scale RNA sequencing experiments reveals extensive transcriptional noise”. In: *Genome Biology* vol. 19, no. 1 (Nov. 2018), p. 208. ISSN: 1474-760X. DOI: 10.1186/s13059-018-1590-2. URL: https://doi.org/10.1186/s13059-018-1590-2.

[19] Quintana-Murci, L., Patin, E., Ségurel, L., and Verdu, P. *Diversity of the human genome.* 2024. URL: https://lms.fun-mooc.fr/courses/course-v1:pasteur+96014+session03/info.

[20] Itan, Y., Jones, B. L., Ingram, C. J. E., Swallow, D. M., and Thomas, M. G. “A worldwide correlation of lactase persistence phenotype and genotypes”. en. In: *BMC Evol Biol* vol. 10 (Feb. 2010), p. 36.

[21] Guimarães Alves, A. C. et al. “Tracing the Distribution of European Lactase Persistence Genotypes Along the Americas”. In: *Frontiers in Genetics* vol. 12 (2021). ISSN: 1664-8021. DOI: 10.3389/fgene.2021.671079. URL: https://www.frontiersin.org/journals/genetics/articles/10.3389/fgene.2021.671079.

[22] NIH. *NIH fact sheets about human genomic variation.* 2024. URL: https://www.genome.gov/about-genomics/educational-resources/fact-sheets/human-genomic-variation.

[23] Guitart, X. et al. “Independent expansion, selection and hypervariability of the TBC1D3 gene family in humans”. In: *Genome Research* (2024). DOI: 10.1101/gr.279299.124. eprint: http://genome.cshlp.org/content/early/2024/08/05/gr.279299.124.full.pdf+html. URL: http://genome.cshlp.org/content/early/2024/08/05/gr.279299.124.abstract.

[24] Feuk, L. “Inversion variants in the human genome: role in disease and genome architecture”. In: *Genome Medicine* vol. 2, no. 2 (Feb. 2010), p. 11. ISSN: 1756-994X. DOI: 10.1186/gm132. URL: https://doi.org/10.1186/gm132.

[25] Sampaio, M. M. et al. “Chronic myeloid leukemia-from the Philadelphia chromosome to specific target drugs: A literature review”. In: *World journal of clinical oncology* vol. 12, no. 2 (Feb. 2021), pp. 69–94. ISSN: 2218-4333. DOI: 10.5306/wjco.v12.i2.69. URL: https://europepmc.org/articles/PMC7918527.

[26] Yoo, D. et al. “Complete sequencing of ape genomes”. In: *bioRxiv* (2024). DOI: 10.1101/2024.07.31.605654. eprint: https://www.biorxiv.org/content/early/2024/07/31/2024.07.31.605654.full.pdf. URL: https://www.biorxiv.org/content/early/2024/07/31/2024.07.31.605654.

[27] Durbin, R. *Assembly and analysis of geneome sequences across the tree of life.* 2023. URL: https://www.youtube.com/watch?app=desktop&v=E9ItgcnecAs.

[28] National Human Genome Research Institute. *Human Genome Project Timeline.* https://www.genome.gov/human-genome-project/timeline. Last access on 2024-8-05. 2024.

[29] Sherman R.M., S. S. “Pan-genomics in the human genome era.” In: *Nat Rev* vol. Genet, no. 21 (2020), pp. 243–254. DOI: https://doi.org/10.1038/s41576-020-0210-7.

[30] Nurk, S. et al. “The complete sequence of a human genome”. In: *Science* vol. 376, no. 6588 (2022), pp. 44–53. DOI: 10.1126/science.abj6987. eprint: https://www.science.org/doi/pdf/10.1126/science.abj6987. URL: https://www.science.org/doi/abs/10.1126/science.abj6987.

[31] Eisenstein, M. “Every base everywhere all at once: pangenomics comes of age.” In: *Nature* (). URL: https://www.nature.com/articles/d41586-023-01300-w.

[32] The Computational Pan-Genomics Consortium. "Computational pan-genomics: status, promises and challenges". In: *Briefings in Bioinformatics* vol. 19, no. 1 (Oct. 2016), pp. 118–135. ISSN: 1477-4054. DOI: 10.1093/bib/bbw089. eprint: https://academic.oup.com/bib/article-pdf/19/1/118/25406834/bbw089.pdf. URL: https://doi.org/10.1093/bib/bbw089.

[33] Liao, W.-W. et al. "A draft human pangenome reference". In: *Nature* vol. 617, no. 7960 (May 2023), pp. 312–324. ISSN: 1476-4687. DOI: 10.1038/s41586-023-05896-x. URL: https://doi.org/10.1038/s41586-023-05896-x.

[34] Eizenga, J. M. et al. "Pangenome Graphs". In: *Annual Review of Genomics and Human Genetics* vol. 21, no. Volume 21, 2020 (2020), pp. 139–162. ISSN: 1545-293X. DOI: https://doi.org/10.1146/annurev-genom-120219-080406. URL: https://www.annualreviews.org/content/journals/10.1146/annurev-genom-120219-080406.

[35] Rouli, L., Merhej, V., Fournier, P.-E., and Raoult, D. "The bacterial pangenome as a new tool for analysing pathogenic bacteria". In: *New Microbes New Infect.* vol. 7 (Sept. 2015).

[36] Colquhoun, R. M. et al. "Pandora: nucleotide-resolution bacterial pan-genomics with reference graphs". In: *Genome Biology* vol. 22, no. 1 (Sept. 2021), p. 267. ISSN: 1474-760X. DOI: 10.1186/s13059-021-02473-1. URL: https://doi.org/10.1186/s13059-021-02473-1.

[37] Guarracino, A., Heumos, S., Nahnsen, S., Prins, P., and Garrison, E. "ODGI: understanding pangenome graphs". In: *Bioinformatics* (May 2022). btac308. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btac308. eprint: https://academic.oup.com/bioinformatics/advance-article-pdf/doi/10.1093/bioinformatics/btac308/43882774/btac308.pdf. URL: https://doi.org/10.1093/bioinformatics/btac308.

[38] Schmidt, D. and Colomb, R. "A data structure for representing multi-version texts online". In: *International Journal of Human-Computer Studies* vol. 67, no. 6 (2009), pp. 497–514. ISSN: 1071-5819. DOI: https://doi.org/10.1016/j.ijhcs.2009.02.001. URL: https://www.sciencedirect.com/science/article/pii/S1071581909000214.

[39] Chor, B., Horn, D., Goldman, N., Levy, Y., and Massingham, T. "Genomic DNA k-mer spectra: models and modalities". In: *Genome biology* vol. 10 (2009), pp. 1–10.

[40] Marchet, C. *Advances in colored k-mer sets: essentials for the curious.* 2024. arXiv: 2409.05214. URL: https://arxiv.org/abs/2409.05214.

Chapter 3

Pushing the limit of pan-genome construction methods

In this first chapter I present and discuss two projects in which I have pushed the current limit of pan-genome construction methods to provide useful insight of their features, of their relative advantages and of the improved capabilities compared to current genomic approaches.
In the first one I have analyzed the current state of the art methods available at the moment and stress-tested them by also generating what was, to the best of my knowledge, the largest human pan-genome produced at the time.
The second one is the generation of a yeast pan-genome reference for the species *Lodderomyces elongisporus*, with some of the tools used in the aforementioned work, to demonstrate how pan-genomes are a superior representation to investigate cross-chromosomal rearrangements compared to the linear reference used by commonly used genomic tools. In order to best capture this large rearrangement between three chromosomes, I had to modify and customize one of the most used variation graph pangenome construction pipeline and compared to other two graphs. Here I present the challenges faced, discuss which data structure to use to achieve biological correctness, genome variation resolution, scalability or downstream application usability and show how a pan-genome enables improved analysis of inter-chromosome rearrangements.

Motivation

The paper that follows this section originates from a discussion early in my PhD journey on which were the best tools suited for large cohort pan-genome construction, specifically for large Eukaryote genomes, like primates. As pointed out in the introduction, there is no one-fits-all solution and most of the tools, at the time of the analysis, were freshly released or distributed under development. It was therefore important for the community of developers and users of such new tools and models to understand the limitations and the potential of the new pan-genomic methods.
In order to perform a thorough assessment of the best available methods, we tried to mimic the conditions that they could face in the near future. We therefore decided to test on the largest collection of high quality human data as it is paramount to understand how pangenomes can be used and adapted to be the platform of future large genomic studies.
As introduced in section 2.3, there are multiple ways of representing a group of genomes to be analyzed or used jointly. One that took traction in the last few years has been graphs. Graphs can represent the sequences as labels of nodes, relationship between them (adjacency or overlap) as edges and infer difference

in the genomes as different set of nodes in them.
We specifically focused our attention on the most used graph models: variation graphs and De Bruijn Graphs. In variation graph edges represent adjacencies, i.e. the genome is spelled by a walk on nodes connected by an edge. In De Bruijn Graphs they represent overlaps, i.e. the suffix of a node is the prefix of the next node connected to it: this implies that edges can exist between nodes that are not adjacent in the genome. As discussed in the next sessions, this distinction implies several differences in how these graphs can be used for downstream analysis.
In this article, we surveyed the the methods and tools that build such graphs, then tested them on different dataset sizes and permutations, and finally analyzed the resulting representation's features. The outcome is a small guide on which are the best applications for each of these tools and an analysis of how they represent variations in genomes.

3.1 Comparing methods for constructing and representing human pangenome graphs

Francesco Andreace, Pierre Lechat, Yoann Dufresne, Rayan Chikhi

Published in *Genome Biology*, November 2023, volume 24, issue 1, article number 274. DOI: 10.1186/s13059-023-03098-2.

Abstract

Background: As a single reference genome cannot possibly represent all the variation present across human individuals, pangenome graphs have been introduced to incorporate population diversity within a wide range of genomic analyses. Several data structures have been proposed for representing collections of genomes as pangenomes, in particular graphs.
Results: In this work we collect all publicly available high-quality human haplotypes and construct the largest human pangenome graphs to date, incorporating 52 individuals in addition to two synthetic references (CHM13 and GRCh38). We build variation graphs and de Bruijn graphs of this collection using five of the state-of-the-art tools: `Bifrost`, `mdbg`, `Minigraph`, `Minigraph-Cactus` and `pggb`. We examine differences in the way each of these tools represents variations between input sequences, both in terms of overall graph structure and representation of specific genetic loci.
Conclusion: This work sheds light on key differences between pangenome graph representations, informing end-users on how to select the most appropriate graph type for their application.

3.1.1 Introduction

In recent years, the majority of studies on human genetics have been conducted on the basis of comparing new samples against a single, standard reference sequence. This reference sequence is a linear succession of nucleotides that acts as a blueprint of the human genome. It is routinely used to align raw sequencing data to it in order to find variations between genomes, e.g. single-nucleotide polymorphisms (SNPs), insertions or deletions (indels). It also is the backbone of the UCSC Genome Browser [1] which enables inspection of genomic and epigenomic features. Despite updates that have improved the quality of the human reference sequence in the last two decades, its linear form severely limits the ability to capture population genetic diversity. For instance the locations of frequently observed structural variations cannot be easily integrated into a linear reference. To see this, consider the difficulty of designing a suitable coordinate system in the presence of (possibly nested) structural variants. Having a single genome as reference sequence also introduces an observational bias towards the chosen alleles that were integrated into that sequence, negatively impacting

many primary analyses such as reads mapping, variant calling, genotyping and haplotype phasing. As a result our ability to precisely characterize structural variants, SNPs and small indels is limited [2, 3, 4]. The GRCh38 human reference genome is estimated to miss up to 10% of our species genetic information [5]. Improvements in sequencing data quality and length, as well as genome assembly methods, are providing a fast expanding collection of haplotype-resolved human genome assemblies. If adequately combined together, these high-quality individual genomes may offer an powerful alternative to the linear reference. There now is an active line of research on pangenomes, i.e. data structures that represent a collection of genomic sequences to be analyzed jointly or to form a reference [3, 6]. Pangenome-based approaches have been shown to improve biological analyses. Pangenomes are at the basis of bioinformatics tools that perform high-quality short read mapping [4], genotyping of SNPs, indels and SVs [7], RNA-seq mapping [8]; de novo variant calling [2]; to store, compress and retrieve high quality genomes [9]; to condensate all the information from a high number of genomes to then visualize specific regions or perform ad-hoc analysis, particularly on complex loci, SVs and tandem repeats [8]. These results pave the way for new applications, e.g. genome-wide association studies, where more precise identification of variants can improve the scope of genetic studies in aging, human diseases, and cancer [3, 6]. Several pangenomic data structures have been proposed: multiple sequence alignments, de Bruijn graphs, cyclic and acyclic variation graphs, and haplotype-centric models that use the Burrows-Wheeler transform [3]. Each of these approaches aim to represent a collection of genomic sequences in an efficient way, to store, visualize, and retrieve differences of interest between the considered genomes. Graph-based pangenome data structures, such as the de Bruijn graph and the variation graph, appear so far to be the most advanced in their ability to handle large amounts of input data. They are capable of representing tens to hundreds of human haplotypes simultaneously. Variations graphs use a sequence graph and a list of paths to store input haplotypes, while de Bruijn graphs store all haplotype k-mers annotated by their haplotype(s) of origin. Scaling pangenome graph data structures to store hundreds of genomes is a challenge that requires significant computational resources and engineering efforts. Many software tools have been created, here we briefly describe major ones. Pantools [10] and Bifrost [11] are two methods that have been developed to generate pangenomes for analysis on large collections of genomes, mostly for applications in phylogenetics and bacterial genomics. The PanGenome Graph Builder (`pggb`) [12], `Minigraph-Cactus` and `TwoPaCo` [13] are methods for building general-purpose pangenome graphs. `Minigraph` [14] builds a particular type of pangenome graph by aligning sequences in an iterative way to a reference template. Minimizer-space de Bruijn graphs (`mdbg`) [15] are variants of de Bruijn graphs that can efficiently represent very large collections of bacterial pangenomes (e.g. 600,000 bacteria). `vg` [2] builds variation graphs from a reference sequence and a variant calling file (vcf) that contains a list of variations from it. Many human pangenomes have been generated, e.g. using Pantools [10] (7 genomes), `Minigraph` [14] (94 haplotypes), `Minigraph-Cactus` [16, 17] and `pggb` [8] (94 single chromosomes), and `TwoPaCo` [13] (100 simulated genomes).

Lastly, a draft version of a human reference pangenome constructed using `pggb` and the `Minigraph-Cactus` pipeline has appeared in a very recent article from the Human Pangenome Reference Consortium [8]. These pangenomes are still limited by some factors: at the present moment, the number of high-quality haplotype assemblies is still low, even if it is expected to grow in the future; the vcf files containing variation are limited in term of bias, type of variation or number of samples; the population representation, even if opened up in recent years to more ethnicities, is still affected by sampling bias.

3.1.2 Results

In this article we provide a comprehensive view of whole-genome human pangenomics through the lens of five methods that each implement a different graph data structure: `Bifrost`, `mdbg`, `Minigraph`, `Minigraph-Cactus` and `pggb`. We examine several features of pangenome graphs, in particular their scalability and how they represent genetic diversity. To this end we collected all publicly available high-quality human haplotypes and attempted to construct pangenomes of various complexity with each selected tool. Although `vg` has been widely used at the basis of relevant pangenome-based discoveries, for example on fast and accurate short read mapping [4], we decided to not consider it in our analysis for two main reasons: the bias introduced by the reference sequence that is used as the backbone of the graph (and associated to the vcf) together with the limited capacity of this method to integrate structural variations from many genomes. We believe both aspects are drivers of the use of pangenome graphs.

Scalability and characteristics of pangenome graph construction tools

We ran the above five tools on three datasets consisting of 2, 10, and 104 human haplotypes respectively (Table 3.3). We compared the computational performance of construction algorithms as well as characteristics of the produced pangenome graphs. The goal is to assess the ability of each method to scale to data available in the near future, i.e. thousands or even millions of human genomes [5].
The performance of each tool is evaluated in terms of running time, peak memory, disk space required by the output data structure (graph and annotations). We also compared the number of nodes, edges and connected components as indicators of the complexity of the graph. Results are displayed in Table 3.1.

In terms of running time, `mdbg` is two orders of magnitude faster than other tools on all considered datasets, taking around two minutes on the H2 dataset and half an hour on H104. `Bifrost` is the second fastest on H104 (18 hours), and `Minigraph` is the second fastest on H2 (8 minutes). `Minigraph-Cactus` takes one order of magnitude more time than `Minigraph`. We could not obtain graphs for `pggb` and `Minigraph-Cactus` on H104 as for the first execution did not finish after 2 weeks and the second returns an error.

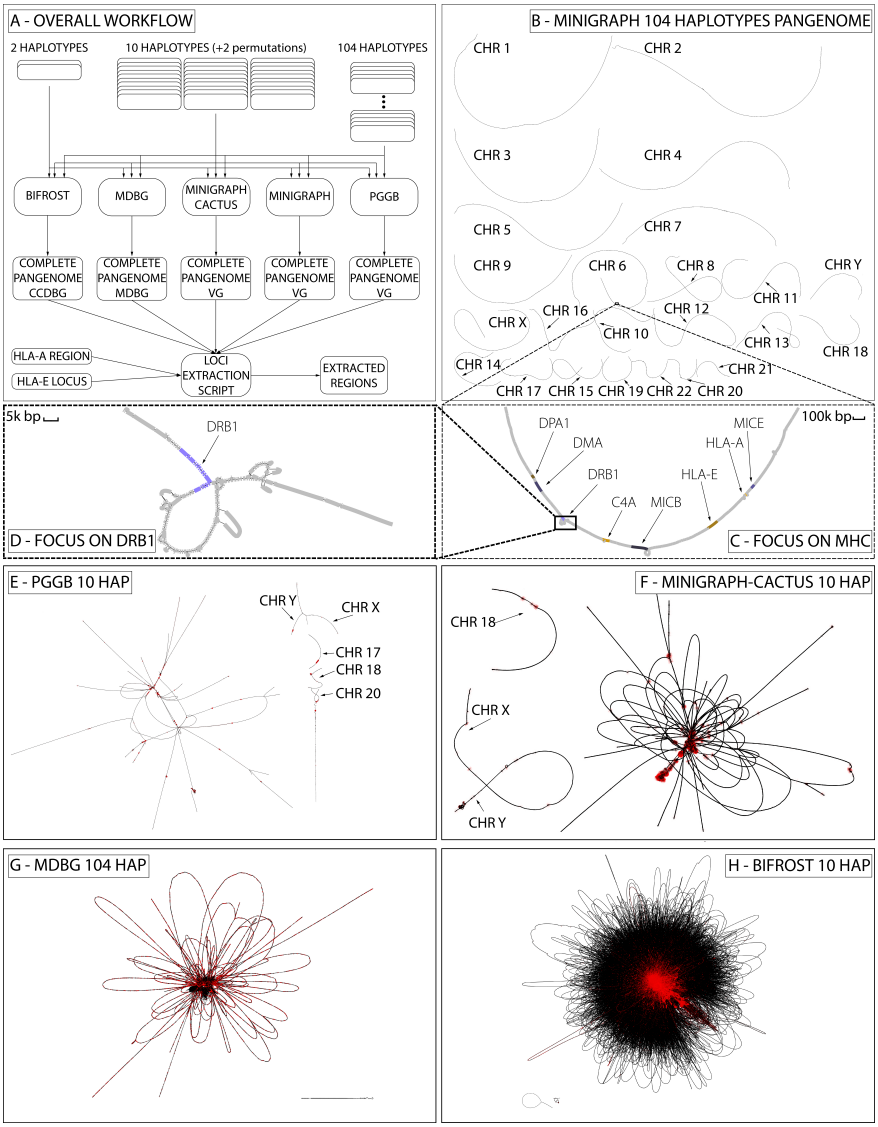


Figure 3.1: **The complete pangenome construction scheme and visualization. A**, The overall workflow, using 5 different tools on 3 different datasets; **B**, complete 104 haplotypes variation graph built by `Minigraph`; **C**, focus on part of HLA (MHC) region in chromosome 6 from panel B; **D**, focus on DRB1-5 locus of HLA from panel C; **E**, complete 10 haplotypes variation graph built with `pggb`; **F**, 10 haplotypes variation graph built with `Minigraph-Cactus`; **G**, 104 haplotypes pangenome `mdbg`; **H**, 10 haplotypes `Bifrost` dBG. All graphs except those produced by `Minigraph` have been simplified using gfatools and rendered using `Bandage`. VG is for variation graph.

In terms of memory usage, `mdbg` consistently uses less than half the memory of other tools (31 GB on H104), followed by `Minigraph` (61 GB on H104). On H2 all tools used between 8 and 66 GB of memory.

All tools used reasonable disk space to store the resulting graph, $\leq$ 12 GB for H10 and $\leq$ 38 GB for H104. Although `Minigraph-Cactus` and `pggb` retain all variations and are the only two tools able to reconstruct the input haplotypes directly from the graph, they are the second and third most efficient in term of disk space (for `Minigraph-Cactus`, 3.6 GB on H2 and 7 GB on H10). While `Bifrost` and `Minigraph` perform all computation in memory, `pggb`, `Minigraph-Cactus`, and `mdbg` store intermediate files on disk, taking comparable space to the input size (up to 3x for `Minigraph-Cactus`).

Different tools yield different pangenome graphs topologies

Graph metrics such as the number of nodes, edges and connected components provide useful insights on the level of detail of the represented variations and on the complexity and accessibility of the information inside the pangenome.

The number of graph nodes varies between 17,000 and 11 millions for the H2 dataset across all tools. In all cases, the number of nodes is at least 3 orders of magnitude smaller than the number of bases in the haplotypes, indicating that pangenome graphs are effective at compressing linear parts of the haplotypes. Tools which discard variations (`Minigraph` and `mdbg`) yield in the order of 10^4–10^5 nodes across all datasets, while tools which retain all variation (`Bifrost`, `Minigraph-Cactus` and `pggb`) yield in the order of 10^6–10^7 nodes. In all cases going from the H10 dataset to the H104 dataset increases the number of nodes by 5x, indicating that graph complexity grows sublinearly with the number of added haplotypes.

The number of connected components varies between 2 and 1402 across all methods and datasets, and the number of large components (i.e. those with more than 1% of total base pairs) varies between 1 and 30. If chromosomes were separated perfectly, pangenome graphs should contain exactly 24 connected components (one per nuclear chromosome, excluding mitochondria). `Minigraph` produces 24 large connected components as the number of chromosomes in the reference CHM13 v2.0 (25 including mitochondria). `Bifrost` and `Minigraph-Cactus` yield graphs with less than 25 connected components while `mdbg` and `pggb` have more than 25. In the `Bifrost` dBG, the vast majority of sequences (>99.99%) are in a single giant component, as chromosomes are joined because they share common k-mers. In `mdbg` such joining does not occur on dataset H2, which has 24 large enough components (each containing > 1% of bases), possibly due to the absence of long and similar enough regions between chromosomes. `Minigraph` does not map any mitochondrial sequence from the input haplotypes to the reference, while they do get included into `Minigraph-Cactus` graphs.

Even if it is common practice to analyze pangenomes chromosome by chromosome [8, 17], in this analysis we purposely used entire genomes as input

Table 3.1: Time, memory, final disk space, nodes, edges, total connected components and connected components with more than 1% of base pairs comparison of `Bifrost`, `mdbg`, `pggb`, `Minigraph` and `Minigraph-Cactus` for different number of haplotypes in input. `Minigraph-Cactus` times include the `Minigraph` graph construction step. `pggb` was not able to complete its execution on the largest dataset in more than 2 weeks thus it is not considered. `Minigraph-Cactus` failed to compute the 104 HAP dataset.

Haplotypes	Metric	`Bifrost`	`pggb`	`Minigraph`	`Minigraph-Cactus`	`mdbg`
2	time (hh:mm:ss)	1:21:25	15:45:30	00:08:33	3:11:59	00:02:38
	memory (GB)	53	24	38	66	8
	disk space (GB)	4.8	4.3	2.9	3.6	4.4
	nodes	9,482 k	8,492 k	34 k	10,851 k	17 k
	edges	13,108 k	11,503 k	48 k	14,702 k	23 k
	conn comp	2	1402	25	4	174
	conn comp > 1% bp	1	30	24	4	24
10	time (hh:mm:ss)	2:27:29	117:08:09	2:03:29	15:57:05	00:05:46
	memory (GB)	102	71	49	154	18
	disk space (GB)	12	7.6	2.9	7	9.7
	nodes	27,468 k	29,315 k	133 k	37,767 k	67 k
	edges	37,584 k	40,282 k	190 k	51,595 k	93 k
	conn comp	3	864	25	3	40
	conn comp > 1% bp	1	5	24	3	1
104	time (hh:mm:ss)	18:38:28	—	46:22:00	—	00:31:38
	memory (GB)	122	—	61	—	39
	disk space (GB)	29.4	—	3.2	—	38
	nodes	106,339 k	—	632 k	—	270 k
	edges	293,839 k	—	912 k	—	396 k
	conn comp	17	—	25	—	1097
	conn comp > 1% bp	1	—	24	—	1

instead. This was done for two reasons: i) to highlight the scalability of the tools, and ii) because separating chromosomes prevents the identification of inter-chromosomal inversions, translocations, and transposable elements, even if most of the generated inter-chromosomal events are probably alignment artifacts. The effects of this choice can be seen in the `pggb` and the `Minigraph-Cactus` H10 variation graphs of Figure 3.1. In the `pggb` graph 19 chromosomes are linked into a single giant component, while chromosomes 17, 18, 20, X, and Y are in other large components. This giant component consists of 25 M nodes that contain 83% of the total basepairs. The remaining 859 components represent only 4.7% of the total bases due to small sequences in the input haplotypes. In the `Minigraph-Cactus` graph all chromosomes are linked into a single giant component except chromosome 18 that is in a separate component, and the sexual chromosomes (X and Y) that are connected together into another component.

Interpretation of variation in pangenome graphs: focus on two HLA loci

The ability to detect and annotate variations among input haplotypes defines the scope of each pangenome graph construction method. Previous work [18] recommends to build graphs on a specific loci rather than the entire genome for the purpose of i) identifying genomic diversity and ii) mapping raw reads

to divergent regions, specifically difficult-to-map repeats. Here we evaluate how pangenomes built from entire haplotypes represent specific biologically relevant loci.

Extraction of HLA-E and a complex HLA region from complete pangenome graphs We extracted from complete pangenomes the regions corresponding to two loci of the Human Leukocyte Antigen complex, also known as HLA. These regions are highly medically relevant as they contain many disease-associated variants [19]. The first locus is the HLA-E gene, that is part of the nonclassical class I region genes, spanning 4.8 kbp and is relatively conserved across populations. It has been shown to have significant association with hospitalization and ICU admission as a result of COVID-19 infection [20]. The second is an HLA complex region comprising the HLA-A gene, part of the classical, highly polymorphic class I region. It is around 58 kbp long and contains the HLA-U, HLA-K, HLA-H, and HCG4B genes. We extracted these two regions from pangenome graphs using a custom script that yields a subgraph corresponding to a given set of sequences and their variation. The script uses a different recommended method for each of the pangenome graph types. In a nutshell, we extracted regions using exact coordinates when possible, and resorted to sequence-to-graph alignment otherwise (see Appendix Section "Loci extraction method" for details). While on variation graphs and mDBGs nearby nodes of an aligned region correspond to variations of the locus, this is not always true for standard dBGs. Extracting accurate and complete loci representation is an unsolved challenge for dBGs.

HLA-E: a low complexity region Figure 3.2 shows how the different tools represent HLA-E over datasets H2, H10 and H104. As expected, `Minigraph` does not detect any variation, since the SNPs that characterize the region are too small to be considered in the construction steps of their algorithm. `pggb`, on the contrary, has 2 SNPs in H2 and 3 in H10. `Bifrost` detects the same SNPs as `pggb` in H2 and H10. Both of them represent the exact same variations and render the same haplotypes paths. `mdbg` captures the heterozygosity of a large region containing the HLA-E locus as the number of samples grows. As the `mdbg` graph is built in minimizer space, nodes represent long genomic segments (in the order of hundreds of thousand of base-pairs). In H10 and H104, the minimizer-space representations of the haplotypes are identical; however, differences in flanking regions of the graph create variations that are captured in extra nodes that are also extracted in this region. On H2, `Minigraph-Cactus` detects 3 variations as the dataset used is different, containing the CHM13 reference and just one haplotype of HG006 (as in `Minigraph`), as discussed in Section "Datasets and haplotypes collection".

Figure 3.2 also illustrates how pangenome complexity grows with the number of genomes: the `Bifrost` H104 subgraph has the most variations across all methods, highlighting that dBGs represent variations exhaustively in large

graphs. On the other hand, `pggb` has the most straightforward method for extracting subgraphs, and also represents variants exhaustively in datasets H2 and H10, but could not scale to the H104 dataset.

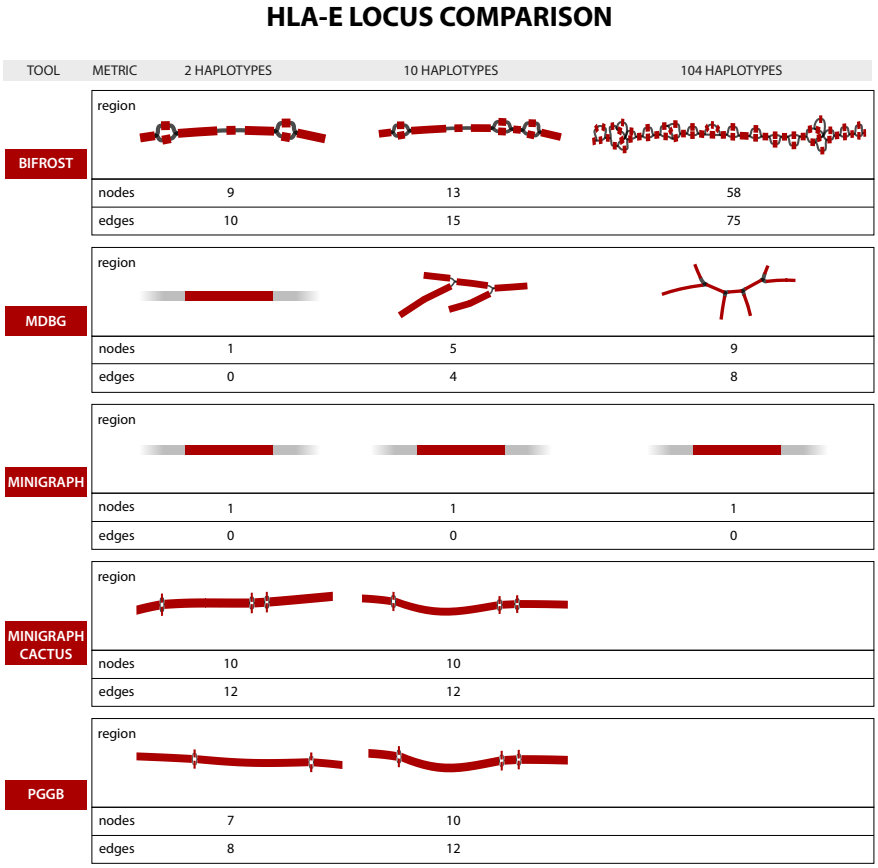


Figure 3.2: Representations of the HLA-E locus by five graph construction methods over three increasing large human pangenomes. Nodes highlighted in red contain part of the locus sequence. The numbers of nodes and edges displayed below each graph concerns the whole subgraph (both red and grey nodes). `Minigraph`, on H2, H10 and H104, and `mdbg`, on H2, have only a portion of one node highlighted since the 4.8k bp region is contained inside a single, long node.

HLA complex locus: high complexity region Figure 3.3 is the counterpart of Figure 3.2 for the complex locus part. In this case the overall interpretability of the region is more challenging, as the number and the structure of the variations is different than in HLA-E. It is also more difficult to compare across tools. Base-level variations, e.g. SNPs, are not visually recognizable in Figure 3.3 in

the methods that retain them (i.e. `pggb`, `Minigraph-Cactus` and `Bifrost`) due to the large sizes of graphs.

There are notable differences in how tools represent the variation, which is well-illustrated in the H2 dataset. While `Minigraph` renders H2 as a single sequence plus a large structural variant (SV) of $\approx$ 52k bp, `pggb` separates it into two paths that differ by $\approx$ 54k bp in length. `Bifrost` represents a detailed bubble that contains many variations inside each path. In `mdbg`, even extracting the complete locus is a challenge as many of the subgraph nodes were not selected by our procedure. `Minigraph-Cactus` adds base level divergences between haplotypes on top of `Minigraph` SV graph.

These differences between representations are further accentuated in the H10 dataset. For it, `pggb` tends to separate the haplotypes into different paths, `Bifrost` renders consistently the same compacted representation and `Minigraph` neglects most of the small differences but is able to display accurately the general picture, and `Minigraph-Cactus`, as in H2, adds small variations on top of `Minigraph` structure.

Uncovering characteristics of graphical pangenome tools

The data structures generated by pangenome building tools are expected to facilitate comparisons between the input genomes. In addition pangenome graphs should be stored in such a way to be easily used by downstream applications. We identify 8 important features for pangenome graph construction tools: i) stability, ii) editability, iii) accessibility by downstream applications, iv) haplotype compression performance v) ease of visualization, vi) quality of metadata and annotation. Two other but important features, scalability and interpretability of produced graphs, were already discussed in Sections "Scalability and characteristics of pangenome graph construction tools" and "Interpretation of variation in pangenome graphs: focus on two HLA loci". Table 3.2 summarizes some of the following considerations on the relative strength of the tools.

Editability and dynamic updates As more high quality assemblies will be generated in the near future, haplotypes may be added to a pangenome, or replaced by improved versions. Updating an existing data structure instead of rebuilding it from scratch is both computationally and energetically efficient. However, many succinct data structures currently used in pangenome representation are static, i.e. cannot be updated. Some methods allow a restricted set of editing operations. `Minigraph` allows to add new haplotypes on top of an already built graph. `Bifrost` provides C++ APIs to add or remove (sub-)sequences, k-mers and colors from the ccdBG. `pggb`, using `odgi` [21], allows specific operations that delete and modify nodes and edges and add and modify paths through the graph. As `Minigraph-Cactus` can be opened with `odgi`, it supports the same operations as `pggb`. The current `mdbg` implementation uses a dynamic hash table, but does not expose an interface that supports updates.

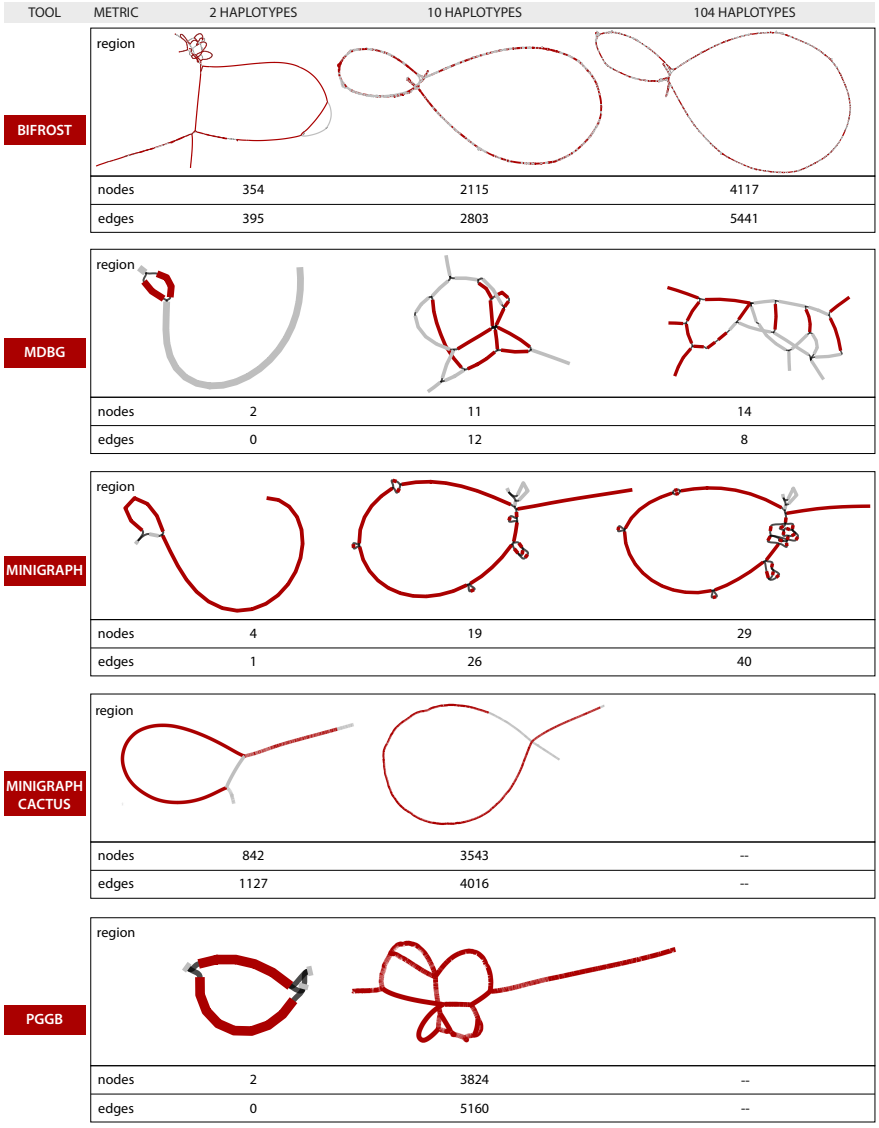


Figure 3.3: Representations of the complex HLA region by five graph construction methods over three increasing large human pangenomes. Nodes highlighted in red are the ones that contain part of the locus sequence. The number of nodes and edges displayed represent the ones directly or indirectly attributed to the sequences that represent the locus.

Stability Counter-intuitively, a pangenome graph construction tool may in some cases generate different outputs when executed multiple times with the same haplotypes as input. This *unstability* could be due to a permutation in the order of the sequences given as input, or non-determinism in the construction algorithm. Yet in order to facilitate the reproducibility of results, pangenome building tools should generate an unchanged output from the same set of input sequences, independently of the particular run or the order in which these are given. We performed two tests to evaluate tool stability: i) we run the tools 3 times using as input the same H10 dataset and ii) we run the tools twice on shuffled input sequences, i.e. changing the order of the haplotypes of H10.

`Bifrost` and `mdbg` constructed exactly the same pangenome on every test, as by definition, de Bruijn graphs are stable. `Minigraph` generates identical graphs on identical inputs, but generates slightly different graphs when the input is permuted. Indeed the construction algorithm of `Minigraph` is order-sensitive as it augments the existing graph structure by aligning the next given haplotype to it and adding divergent sequences. `Minigraph-Cactus` generates slightly different graphs on identical input. `pggb` generated slightly different graphs while maintaining the same haplotype sequences in the paths. The overall representation of the input genomes is therefore preserved, while the topology of the variation graph varies. The first two of the three phases of the `pggb` pipeline (all-vs-all alignment and graph imputation) produce the same result on different runs with the same input but differences arise when the order of the input haplotypes changes. Most of the differences in the graph topology are thus due to the final smoothing steps.

Accessibility by downstream applications To facililate their adoption, pangenome representations should be easily processed by downstream analyses. De Bruijn graphs are challenging to analyze due to their high number of nodes, edges, and redundancy (the $k-1$-overlaps between nodes). Though De Bruijn graph representations usually support queries of presence/absence on nodes (as in `Bifrost`), they lack tools able to perform more elaborate analyses such as those discussed in Section "Interpretation of variation in pangenome graphs: focus on two HLA loci", e.g. incorporating haplotype information at the k-mer level. On the other hand, variations graphs with paths provide more flexibility, i.e. as in `pggb` and `Minigraph-Cactus` with the `odgi` visualization toolkit. Finally in `Minigraph`, which considers a narrower spectrum of variants, the absence of path information prevents haplotype-level analysis; haplotypes would need to be manually mapped back to the graph. The choice of the pangenome building tool depends on the envisioned application. `pggb` and `Minigraph-Cactus` graphs have been shown to outperform linear references for short read mapping, genotyping and RNA sequencing mapping [8]. As these two methods are complex pipelines based on multiple tools where parameters have been carefully set, they can be more challenging to install and run than single integrated tools. `Minigraph` alone can also be used if the focus is on structural variation instead of SNPs or small indels, and to quickly produce a

pangenome graph for complex loci visualization and interpretation. The dBG-based approaches show that, for example with `Bifrost`, they retain the same base-level information as more computational-heavy variation graph approaches, but the lack of tools to use them for analysis limits their adoption.

Haplotype compression Building a graph pangenome can be seen also as a way to store, compact and retrieve the input haplotypes. As the number of new assemblies is growing faster than the data storing capacity, pangenomes can potentially help save storage space. This is highlighted by the disk space reported in Table 3.1, which is consistently smaller than the sum of haplotype sizes for all methods and datasets.

In order to losslessly retrieve the input genomes from a pangenome, the representation has to store all variations from the original haplotype sequences as paths in the graph. `pggb` and `Minigraph-Cactus` fall into this category while the other three considered tools do not store paths, or do not consider all variations, thus they are lossy.

Of note, the GBZ tool [9] enables graph pangenomes that store paths in the GFA file format to be stored in a lossless compressed form. It uses a Graph Burrows-Wheeler transformation to compress the graph in a more efficient way than using gzip [9]. Using GBZ, the pangenomes generated by `pggb` and `Minigraph-Cactus` are losslessly compressed with space gains of 3.5-5x.

Ease of Visualization Visualizing large graphs which exceed hundreds of thousands of node is a challenge that exceeds the scope of pangenomics. The H104 pangenomes are difficult to visualize. Among the visualization tools considered by the Human Pangenome Reference consortium [6], only `Bandage` is able to display the `Minigraph` or `mdbg` H104 graphs, which contains a few million nodes. We reduced the number of nodes and edges of `pggb`, `Minigraph-Cactus` and `Bifrost` H10 graphs by collapsing isolated subgraphs representing SNPs or indels up to 10k bp (using the command `gfatools asm -b 10000 -u`).

Quality of Metadata and Annotation Augmenting pangenome structures with information from other omics data would increase pangenome relevance in biological discoveries. As biobanks are rapidly growing, more data is available on regulatory regions, transcriptomics, CNVs and other medically relevant traits [22, 23]. Pangenome data structures could leverage such information, and some of the considered tools offer basic functionality in this sense. `Bifrost` provides a function to link data to graph vertices through C++ APIs. `pggb` and `Minigraph-Cactus`, using `odgi`, offer annotation capabilities through insertion of paths or BED records. `Minigraph` and `mdbg` do not offer any annotation feature. Specifically, in order to enhance a pangenome graph with metadata (for example with genes and regulatory regions known variants), it is desirable to maintain compatibility with methods and data formats that use a linear reference. One could conceivably project data from a graph to a reference genome to continue downstream analyses using linear coordinates. A simple

Table 3.2: **Relative strengths of five pangenome graph construction tools**
Explanation of rows: 1) efficacy of construction algorithm, measuring wall-clock time; 2) degree to which variants (e.g. SNPs) are retained; 3) ability of a tool to perform well on large datasets, both in comparison to other tools but also compared to smaller datasets; 4) ability to modify the produced data structure to add or remove haplotypes; 5) property of producing the same result irrespective of perturbations, such as permutation of the input order, and repeating the same run; 6) existence of tools (and operations) that can be applied to the resulting graphs; 7) whether input haplotypes information is retained by the tools, and if so, its space efficiency; 8) whether the entire graph can be directly visualized and interpreted; 9) easiness of 'zooming in' a specific genomic region and interpret variants; 10) summarizes the functionalities provided by the tools to annotate the pangenomes with genomic data; 11) ability to share information between the graph and a linear reference.

Metric	`Bifrost`	`pggb`	`Minigraph-Cactus`	`Minigraph`	`mdbg`
1) Construction speed	●●○	●○○	●○○	●●○	●●●
2) Variations	●●●	●●●	●●●	●●○	●●○
3) Scalablilty	●●●	●○○	●○○	●●○	●●●
4) Editability	●●●	●●○	●○○	●●○	●○○
5) Stability	●●●	●○○	●○○	●●○	●●●
6) Accessibility by downstream applications	●○○	●●●	●●●	●●○	●○○
7) Haplotype compression performance	●●○	●●●	●●●	●○○	●○○
8) Ease of visualization	●○○	●●○	●●○	●●●	●●●
9) Loci visualization and interpretability	●○○	●●○	●●●	●●○	●○○
10) Metadata and annotation	●●○	●●●	●●○	●○○	●○○
11) Compatibility with a linear reference coordinates	●○○	●●●	●●●	●●○	●○○

method to achieve this compatibility, in our view, is to store the reference genome of interest inside the graph pangenome that supports retrieving such a reference. Variation graphs built using `pggb` or `Minigraph-Cactus`, due to their locally acyclical and directed construction and their use of haplotype paths, store all the coordinates needed for such a task. Haplotype paths play an important role as they avoid additional mapping to the graph, by using the `odgi` tool to extract or inject the required information. `Minigraph` does not store haplotype paths and requires mapping sequences to the graph to restore haplotype information. On the other hand, De Bruijn graphs, using associated color data, can record the membership of k-mers to a reference sequence, yet one cannot fully reconstruct a haplotype unless k-mers positions are also stored.

3.1.3 Discussion

Five state-of-the-art pangenome graphs construction tools were compared on the representation of up to 104 human haplotypes. The approaches significantly differ in terms of speed, graph size, and representation of variations. We find that it remains computationally prohibitive to generate human pangenome graphs for hundreds of haplotypes, especially while retaining all variations. Each approach has its own set of strengths, and ultimately the choice of the method depends

on the downstream application. In addition, several takeaway points emerged from our analysis.

First, our focused analysis of HLA loci revealed that de Bruijn graphs and variation graphs represent genomic variations equally well as pangenomes. This is of particular importance as also shown by the draft human pangenome references [8]: pangenomes are pivotal to trace complex and clinically relevant loci. While de Bruijn graphs are faster to construct, more stable, and scale better in terms of input size, the resulting graphs are challenging to interpret downstream. Variations graphs on the other hand are more practical to analyze at the expense of a less efficient construction step. Their visualization are more straightforward to interpret, mostly due to not having cycles, and provide insights into loci differences.

Second, we can highlight two categories of pangenomic methods that have distinct application domains. `pggb`, `Minigraph-Cactus` and `Bifrost` store all possible variations, and keep haplotype information as paths or colors. They provide a complete picture of the set of variations in the input genomes which makes them difficult to analyze. They can be used for a large variety of genomic analysis, as shown for `pggb` and `Minigraph-Cactus` [8]. `Minigraph` and `mdbg` generate 'sketched' pangenome graphs that consider only large variants, ignoring smaller differences, and are more efficient to construct and visualize. They can be used for large scale characterization of variation in population, as proven for bacteria [15].

Third, every tool possesses an exclusive set of features. `pggb` facilitates downstream analyses using the companion tool `odgi`. It allows to precisely extract and browse any locus of interest. It is the only tool that generates variation graphs without a reference. It also keeps a lossless representation of the input sequences. `Minigraph` generates a pangenome graph based on a reference sequence taken as a backbone. Its shines in the representation of complex structural variations, but does not include small or inter-chromosomal variations. The pipeline `Minigraph-Cactus`, that uses the `Cactus` base aligner, can be used to add small level variations on top of the `Minigraph` graph and to keep a lossless representation of the input sequences. `Bifrost` illustrates that classical de Bruijn graphs are scalable, stable, dynamic, and store all variations. However, extracting information from them remains a challenge. Lastly, `mdbg` is the fastest construction method which generates an approximate representation of differences between haplotypes. As discussed in Section "Accessibility by downstream applications", these features enable different genomic analyses and downstream applications.

3.1.4 Conclusions

In conclusion, our results highligh the strengths and weaknesses of current pagnenome construction tools for human applications, on how do they represent specific loci of medical relevance. We also provide insights on the features they possess and point out their best application domains. In our view, future directions for human pangenomes building tools should focus on tackling

efficiency bottlenecks, aiming to represent hundreds to thousands of haplotypes. Representations should further be lossless and represent the input haplotypes as paths in the graph. Such features would unlock many other applications such as lossless compression of haplotypes and cancer copy number variant analysis. Finally, we recognize the need for more user-friendly tools that can be used by biologists and that can translate complicated questions into graph queries. While `odgi` begins to address these questions in variation graphs, other approaches have not yet provided user-friendly interfaces. A package similar to `odgi` for de Bruijn graphs would help fully realize their potential.

3.1.5 Methods

Datasets and haplotypes collection

In order to evaluate the state of current human pangenome representations, we sought to build a human pangenome that contains all publicly available high-quality human haplotypes. We collected from two different sources 102 different haplotypes from the genome of 51 individuals, and also used the two reference genomes, GRCh38 from the Genome Reference Consortium (GRC) [24] and CHM13 v2.0 cell line of the T2T Consortium [25]. Five haplotypes correspond to Google Brain Genomics DeepConsensus [26] assembly dataset: they are hifiasm assemblies of PacBio Hi-Fi reads corrected with DeepConsensus. The average of their N50 is 37.99 Mbp. The remaining haplotype assemblies as well as the T2T reference are from the Human Pangenome Reference Consortium (HPRC) year-1 freeze [6], and GRCh38 is from the GRC. Their average N50 is 40.3 Mbp. Since HG002 is contained in the DeepConsensus data, the HPRC HG002 haplotypes were not used. The origin and the sex of the individuals are diverse and provide a fair representation of the diversity in human population: out of 51 total individuals, 21 are males and 30 are females and they represent 14 different ethnic groups, from US to Africa and Asia. We did not perform any additional selection, regarding sex and ethnicity, on these public datasets as our main goal was to use as many genomes as possible. However, the HPRC stated that the genomes were selected to represent genetic diversity in humans [8].

To evaluate the scalability of pangenome construction tools, we created three datasets of increasing size: 1) 2 haplotypes from the same individual, HG006, 2) 10 haplotypes from 5 different individuals (HG002, HG003, HG004, HG006 and HG00735) and finally 3) all of the 104 haplotypes. To test whether the order of the input sequences matters, we considered various random orderings for the 10 haplotypes in Dataset 2. Since `Minigraph` needs a reference sequence as fist haplotype in order to correctly build the graph, we generated specific 2 and 10 haplotypes datasets with the first haplotype replaced by the reference genome CHM13. This was applied to the `Minigraph-Cactus` pipeline as well as it uses `Minigraph` variation graphs.

Table 3.3: **Description of the three datasets generated to test the scalability of the tools**
Data sources: [1] Google Brain Genomics [27]; [2] Human Pangenome Reference Consortium [28]; [3] 1000 Genomes Project [28]; [4] Telomere to Telomere Consortium [28].

Haplotypes	Project	Bases
2	Google[1]	5.9 Gbp
10	Google, HPRC[2]	30 Gbp
104	Google, HPRC, 1KG[3], T2T[4]	313.6 Gbp

Pangenome graph construction tools

We evaluated tools that generate graph pangenomes as variation graphs and colored compacted de Bruijn graphs. Variation graphs are generally locally acyclic while de Bruijn graphs have cycles. In variation graphs, nodes represent sequences and edges represent immediate sequence adjacency without overlap. Variation graphs are generally easier to visualize and to interpret while challenging to construct at scale and, apart from `pggb`, require a reference genome. In de Bruijn graphs (dBG), nodes are k-mers (string of length k) and edges are (k-1)-overlaps between nodes. In practice, dBGs are represented in a compact way where all nodes along unbranching paths are compacted into *unitigs*. The resulting graph is called compacted De Bruijn Graph, where nodes are unitigs and edges represent (k-1)-overlaps. As shown in Figure 3.1, de Bruijn graphs result in large graphs that pose visualization and interpretation challenges, in particular as there is no alignment to a reference.

- `Bifrost` constructs dynamic, coloured compacted de Bruijn Graphs (*ccdBG*). It first generates a standard dBG using an efficient variant of Bloom Filters and then computes the compacted dBG from it. Colors, i.e. identifiers representing the sample origin of each k-mer are added by storing an array per k-mer. A human genome ccdBG typically consists of a single large connected component, as common k-mers are shared between chromosomes. This pangenome representation contains all the variations present in input sequences.

- `mdbg` builds a variant of de Bruijn graphs called a minimizer-space de Bruijn Graph (`mdbg`), which is efficient to construct as it only considers a small fraction of the input nucleotides. Color information is currently not supported in the implementation. Similarly to Bifrost, a `mdbg` also typically represents a human genome as a single large connected component, albeit with orders of magnitude less nodes. Minimizer-space de Bruijn graphs mostly discard small variants, yet are sensitive to heterozygosity which creates branches in the graph.

Table 3.4: **URL, version, pangenome representation and parameters of the three analyzed tools.**
pggb/0.2.0 used wfmash v0.7.0, seqwish v0.7.3 and smoothxg v0.6.1.

Tool	Github repository	Graph type	Version	Parameters
`Bifrost`	pmelsted/Bifrost	De Bruijn graph	1.0.5	-k100 -c
`pggb`	pangenome/pggb	variation graph	0.2.0	-p 98 -s 10000 -k 311 -G 13033,13117 -O 0.03 -v -t 8 -T 8 -A -Z
`Minigraph`	lh3/Minigraph	variation graph	0.18	-cxggs
`Minigraph-Cactus`	ComparativeGenomics Toolkit/cactus	variation graph	2.2.3	–maxLen 10000 –delFilter 10000000
`mdbg`	ekimb/rust-mdbg	De Bruijn graph	1.0.1	-k 10 -d 0.0001 –minabund 1 –reference

- `Minigraph` constructs a directed, bidirected and acyclic variation graph iteratively by mapping new haplotypes using a combination of the minimap2 tool and the graph wavefront alignment algorithm. The first input sequence acts as a backbone for the whole representation. The sample(s) of each node are stored in a rGFA output file. `Minigraph` considers only variations longer than 50 bps hence it is oblivious to isolated SNPs and small indels: even if it produces base-level alignment for contigs, the graphs are not a base-level resolution. The resulting graph is divided into connected components that correspond to the chromosomes present in the first given input genome.

- `Minigraph-Cactus` is a variation graph construction pipeline that combines `Minigraph` to generate a structural variations graph and `Cactus` base aligner to generate base-level pangenome graphs of a set of input assemblies and embed haplotype paths. `Cactus` [16] is a highly accurate and scalable reference-free multiple whole-genome alignment tool, that in this pipeline considers the reference sequence used by `Minigraph` to ensure that the resulting variation graph is acyclic. The final graph is further normalized using GFAffix[29]. The pipeline allows to generate multiple graphs, one for each chromosome, or produce a single graph that includes inter-chromosomal variants.

- `pggb` is a directed acyclic variation graph construction pipeline rather than a single tool. It calls three different tools: pairwise base-level alignment of haplotypes using wfmash [30], graph construction from the alignments with seqwish [31], graph sorting and normalization with smoothxg and GFAffix [32, 29]. The resulting variation graph represents variations of all lengths present in the input sequences.

Supplementary Information

Benchmark infrastructure

Running time of pangenome construction tools was measured as wall clock time and peak memory as maximum resident set size using the `time` command.

Other metrics were obtained with custom Python scripts. All benchmarks were performed on a Supermicro Superserver SYS-2049U-TR4, with 3 TB RAM and 4 Intel SKL 6132 14-cores @ 2.6 GHz, using 8 cores.

TwoPaCo

We did not consider `TwoPaCo` as it is redundant with `Bifrost`. Both methods construct the same de Bruijn graphs. `TwoPaCo` is a method for constructing ccdBGs by finding junction k-mers at the boundaries of unitigs or in branching nodes. It consists of two main steps in which it approximates the dBG with a Bloom filter in order to reduce the size of the problem and then runs a two pass highly parallel algorithm on it. It constructs ccdBGs similarly to `Bifrost`. `Bifrost` is faster, supports edit operations, and accepts also reads other than assemblies as input. We tested both tools on NCBI datasets from three different known human variation regions part of the human leukocyte antigen (HLA) complex: HLA-A, MICB and TAP1. These loci have different number of sequences and have complexity and length. The resulting graphs have exactly the same k-mer content and substantially equal topology. The difference is that `TwoPaCo` considers sequences with IUPAC 'N' bases while `Bifrost` does not and that in some cases `TwoPaCo` renders some unitigs split in two or more consecutive nodes.

Loci extraction method

For `Bifrost` and `mdbg` graphs, nodes corresponding to the input sequences are identified with `GraphAligner` [33] and the subgraph is extracted using the `Bandage` *reduce* function. As the aligned nodes are not expected to represent the full diversity of the population in the pangenomes, the considered portion of the graph contains also nodes up to a certain distance from the aligned ones: 1 for `mdbg` and 3 for `Bifrost`. This number is based on the size of the sequences spelled by the nodes and on the considered variations. Artifacts, mostly tips, that are not part of the locus of interest are removed with a custom python script. For `Minigraph` generated graphs, the `Minigraph` own alignment function has been used to identify the nodes and then `Bandage` is used to extract the subgraph. For `pggb`, first we generate a bed file of the position of the region of interest in every haplotype used to construct the graph. The ranges are derived from aligning them to the locus sequence(s) using minimap2 [34]. The graph corresponding to the region is then extracted using the `odgi` extract and `odgi` view functions. For `Minigraph-Cactus` we use the same strategy as `pggb`, with the difference that the bed file is only for the reference CHM13, present in the graph.
The annotation of the specific loci in the subgraph is done using nodes from the alignment with `Minigraph` or `GraphAligner` to the subgraph. This makes it possible to highlight multiple sections in the region, as, for example, genes and pseudogenes of interest.

Availability of data and materials

The scripts used to generate and analyse the pangenomes can be found at [35, 36] under MIT license. Google Brain Genomic assemblies can be found at [27]. HPRC assemblies, CHM13 and GRCh38 can be found at [28].

Funding

R.C was supported by ANR Full-RNA, SeqDigger, Inception and PRAIRIE grants (ANR-22-CE45-0007, ANR-19-CE45-0008, PIA/ANR16-CONV-0005, ANR-19-P3IA-0001). This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grants agreements No. 872539 and 956229.

Author's contributions

FA, YD and RC conceived and designed the project. FA implemented the scripts. FA and PL ran the experiments. FA, YD, PL and RC wrote the paper. The authors read and approved the final manuscript.

Authors' affiliations

Francesco Andreace Institut Pasteur, Université Paris Cité, G5 Sequence Bioinformatics
Sorbonne Université, Collège doctoral
25-28 Rue du Dr Roux, 75015 Paris

Pierre Lechat Bioinformatics and Biostatistics Hub, Institut Pasteur, Université de Paris Cité
25-28 Rue du Dr Roux, 75015 Paris

Yoann Dufresne Institut Pasteur, Université Paris Cité, G5 Sequence Bioinformatics
Bioinformatics and Biostatistics Hub, Institut Pasteur, Université de Paris Cité
25-28 Rue du Dr Roux, 75015 Paris

Rayan Chikhi Institut Pasteur, Université Paris Cité, G5 Sequence Bioinformatics
25-28 Rue du Dr Roux, 75015 Paris

3.1.6 Perspectives

The results of the work outlined in this section suggest several directions for future investigation and software development: they can be divided into a few axes. On one side, the scripts and software that I developed to produce the analysis of this paper could have been extended into an automatic reproducible pipeline. By integrating the code into platforms like nextflow or Snakemake, this could become a benchmark for current and future pangenome tools, as, at the moment, there is no other way to independently compare the output of tools that produce variation graphs and dBGs.
On another side, there are many more features that could be added to the analysis workflow to produce a more in-depth and accurate description of the resulting pangenomes, like node (sequence) length distribution, node degree distribution, count of SNPs and indels. It is important to stress that detection of specific patterns of variation is non-trivial in data-structures like dBGs. Finally, another experiment that could be done is to convert, using path information, the variation graph into a ccdBG, to compare against a ccdBG built directly from the input sequences: this would allow to verify if the information stored into the two data structures is equivalent.
Another very interesting avenue would be to develop a tool that enables complex information extraction from ccdBGs. Construction tools usually offer commands or APIs to perform simple absence/presence queries, that are mostly useful when they are used for metagenomic purposes but offer less actionable insight in pangenomics analysis. An idea that I have started exploring during this PhD is to extract a subgraph of a ccdBG that represent a locus or gene of interest in the whole dataset provided as input.
Finally, it would be very useful to define a ccdBG output color format that every tool should also use to output the colors, like gfa is used to output graphs. As of now, each tool implements its own binary format for color storage, discouraging downstream analysis software development, as it would be bound to specific construction tools and not the common data structure used.

3.2 Building a *Lodderomyces elongisporus* pangenome reference: overcoming current limitations.

We present another example of extending variation graph pangenome building strategies, specifically in producing a graph that can represent inter-chromosomal events, such as rearrangements, for the medically relevant yeast strain of *Lodderomyces elongisporus*. We also demonstrate how a pangenome-based approach can provide more comprehensive insights into such events compared to a linear genome-based approach.
The work presented here is partially a collaborative effort with other PhD students and researchers, co-led by Daniel Doerr and myself during a winter school organized by our consortium. I would like to acknowledge Simon Heumos for our discussions on the construction of such graphs, and Nicola Rizzo. The majority of the results and findings reported here are primarily my own subsequent work.
The following sections will explain the process required to produce custom, biologically significant, and biologically driven pangenome graphs of a yeast strain using current state-of-the-art tools. We will offer insights into workarounds that can be employed in cases with special requirements, where current tool features limit the analysis that can be performed. This will also demonstrate how pangenome models can offer more powerful tools for analyzing specific variations in a group of similar genomes compared to linear reference-based models.
In this specific case, the goal was to produce a variation graph that represents the long chromosomal translocations observed by the team that produced the assemblies from short and long reads. Chromosome-crossing syntenies for multiple contigs in alignment hits of chromosomes C, G, and H were detected, suggesting that they could correspond to a single recombination group. This finding was consistent with an independent SNP-based genomic study performed on isolates from a fungemia outbreak in a neonatal intensive care unit in Delhi between 2021 and 2022. The study noted that these translocation events are more frequent in hospital and patient-associated populations than in fruit-associated populations [37].
In summary, the work here presented will be organized as follows:

1. Brief characterization of *Lodderomyces elongisporus* genome and relevance;
2. Pangenome graphs constructed and their utility;
    a) Determining the chromosomal communities from the assemblies to generate a variation graph of interest;
    b) Producing a variation graph with `pggb`;
    c) Producing a variation graph with `Minigraph-Cactus` by customizing the pipeline and the data;
3. Representation of translocation events in variation graphs.

3.2.1 *Lodderomyces elongisporus*: genetic characteristichs, interest and used data

Lodderomyces elongisporus is a diploid yeast that has been isolated from, among many sources, humans and it is recently emerging as pathogenic. It is phylogenetically placed in the Candida clade and the size of its genome is usually between 15 and 16 Mb, 2 orders of magnitude smaller than a human genome [38]. Its DNA is organized into 8 chromosomes, that here will be referred in alphabetical order and decreasing size A to H, from around 3.5Mbp of chr A to 800 Kbp of chr H, plus a 35 Kbp mitochondrial DNA. Our analysis shows that it has a stable core genome of 13Mbp, as shown in section 3.2.4. Increasing reports of (mostly bloodstream) infection in mainly immunosuppressed adults makes it an increasingly important subject of studies [38, 39, 40, 41]: it also got recent attention when an outbreak was reported occurring in a neonatal ICU in Dheli, India from September 2021 to February 2022 with 1 death[37].
11 samples sequenced by us were assigned names with one letter in alphabetical order from A to K followed by a number greater or equal than 0 that denoted the quality of the assembly (with 0 draft to 2 maually curated). More in depth statistics like the number of sequences, N50 and others are shown in table 3.5. Moreover, one sample, B2, for which a member of the consortium produced a high quality assembly after intensive manual curation, was used as relative reference in the cohort. Another sample, J2, was also fully resolved into chromosomes while the others were assembled into contigs.

3.2.2 Building ad hoc pangenome reference

We produced a dBG from all 11 assemblies described in table 3.5 using `Bifrost`, but their usefulness remains limited for visual analysis of complex biological events interpretation and study.
Figure 3.4 shows the visualization of the dBG for k-mer length equal to 25: repetitions make the visualization of the graph challenging. This is the same phenomenon previously described for human genomes, as better insight can be acquired when just a small region is visualized. De Bruijn Graphs can instead be useful to produce quite straightforward whole-genome alignment-free and reference-free phylogenetic analysis in a fraction of time required by competitor methods that use all vs. all alignment. The tool `SANS serif` [43] can process directly a `Bifrost` generated ccdBG to estimate the phylogenetic splits between the genomes contained in the graph. Figure 3.5a shows the visualization of the phylogenetic network produced by `SANS serif` using the tool `SplitsTree` [44]. By adding another genome from a close species it is possible control that the dBG-based analysis provides results compatible with expectations. Figure 3.5b shows the phylogenetic network when an assembly of *Lodderomyces Beijingensis* is added to the graph: the new genome is very separated compared to the other ones from the same species. This results shows how dBGs are a powerful model when applied to specific applications.

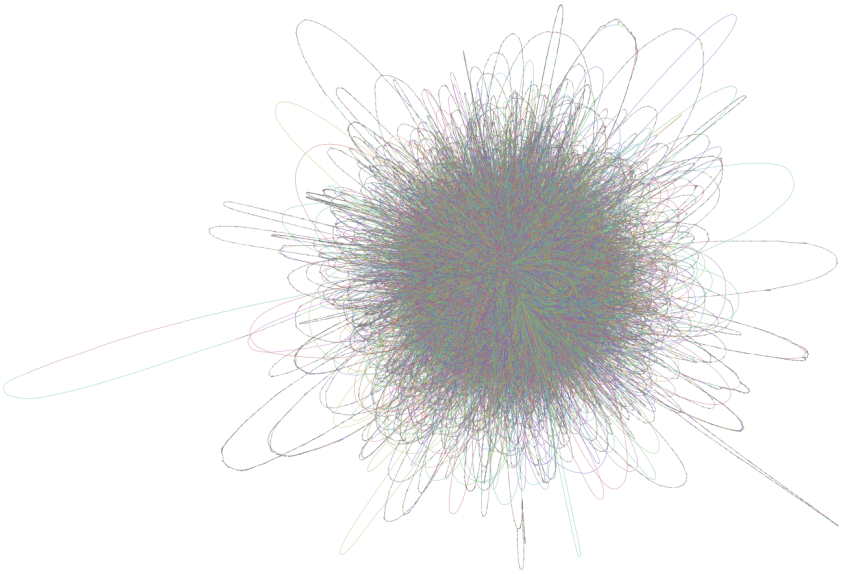
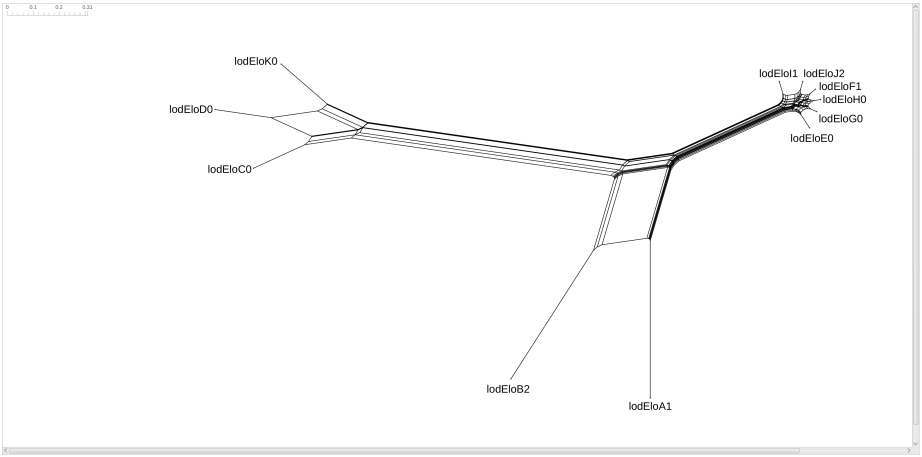


Figure 3.4: `Bandage` visualization of the pangenome dBG of the cohort of 11 *Lodderomyces elongisporus* strains. As for human genomes, visualization of the whole data offers no particular insight, apart from the large variations visible on the rounded parts away from the dense part.

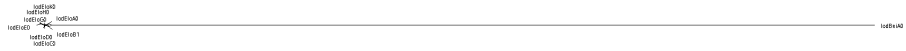
Given high quality assemblies generated by the sequences of these 11 samples, we decided to also build a pangenome graph with small variant resolution using `pggb` and `Minigraph-Cactus`, in a similar way to what has been done with the Human Draft Pangenome Reference [8]. In order to produce a graph with a structure that resembles biologically expected variation structures, several rounds of parameter tuning and manual curation are needed, with knowledge which is out of reach from inexperienced users.
The first step to build such pangenomes is to divide the genome assemblies into communities of sequences belonging to chromosomes.

3.2.2.1 Determining chromosomal communities

Variation graphs construction pipelines use mapping or alignment between the input set of genomes to infer graphs. Their first step consists in grouping the sequences from the assemblies into communities representing a chromosome in order to run a single computation instance and produce a separate graph for each of them. The final graph is then obtained by joining together the output of each group. This means that without any pre-processing, no inter-chromosomal event can be detected.



(a) `SplitsTree` visualization of the phylogeny network generated using `SANS serif` from the ccdBG constructed using `Bifrost`.



(b) Phylogeny network of the 11 *Lodderomyces elongisporus* strains plus one genome of *Lodderomyces Beijingensis* shows the ladder separated on the right while the group of subfigure b compacted on the left. As they are two different species, although close in the Candida/Lodderomyces clade [45], this result provides positive control for the phylogeny network generated using `SANS serif` from the ccdBG constructed using `Bifrost`. This image was produced by Nicola Rizzo.

Figure 3.5: Visualization of the ccdBG representation and phylogeny analysis of the *Lodderomyces elongisporus* pangenome.

In this specific use-case, in order to identify inter-chromosomal events, contigs associated to any of the 3 chromosomes conjectured to be part of the rearrangement had to fall into the same community and be provided together in input to the pipeline. This would ensure that, if such rearrangement exists, it would produce a feature in the graph that would show as a tangle between the chromosomes.
As the rest of the assemblies, with the exception of the J2 sample, were not resolved into single chromosomes, each of them was aligned to the reference B2 using `wfmash` alignment segment size of 10k and 95% sequence identity (and lower segment size, 90% sequence identity if unmapped). The identity scores of

the alignment of the genomes to the reference B2 assembly is shown in figure 3.6. From this alignment, contigs were assigned to the chromosomal community to which they best mapped. Finally, contigs assigned to chromosomes C,G,H were grouped manually into a single community.

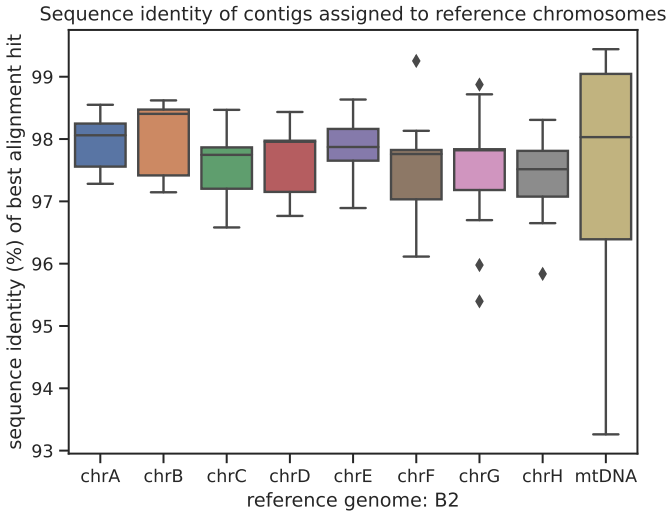


Figure 3.6: Sequence Identity of contigs assigned to reference chromosomes. Image produced using a pipeline developed by Simon Heumos.

3.2.2.2 Producing a variation graph using pggb

As `pggb` uses all-vs-all alignment of a collection of sequences as first step to infer the graph, it enables the representation of recombination among chromosomes placed inside the same community, as seen also for human acrocentric chromosomes [46].
The `nextflow/pangenome` (`pggb`) pipeline was run for each of the identified chromosomal communities to produce a first pangenome representation of the 11 yeast strains. The tangle visible in figure 3.7 clearly shows the recombination happening between the three chromosomes. This work was mainly done by Simon Heumos and the graph shown in figure 3.7 is the result of more than 25 rounds of parameters tuning. The detection of the event with a variation graph using `pggb` encouraged the effort to produce a similar representation with `Minigraph-Cactus`, a pipeline that is not designed to construct grouped chromosomes pangenomes. The goal was to double check with another pangenome construction software the structure of the translocation in the pangenome graph.

3.2.2.3 Overcoming `Minigraph-Cactus` limitation by modifying both the data and the pipeline

In order to produce a graph that represents variation between multiple chromosomes with `Minigraph-Cactus`, a custom pipeline has to be used. `Minigraph-Cactus` communities are implicitly inferred by the first step, performed by `Minigraph`. For this tool, the chromosomes present in the first reference genome given as input are used as communities and backbone for the whole graph and no sequence can be both assigned to different chromosome. This is an intrinsic characteristic of `Minigraph` and cannot be changed with input parameters: it means that there is no feature to have chromosome C,G and H considered together in input.
To try to overcome this limitation of the approach we tried to produce a graph that respected the condition of having the three chromosome inside the same connected component, at the cost of producing a representation that was not biologically correct. We therefore produced a chimeric contig consisting of the concatenation of the three chromosomes assemblies of the B2 sample. The rationale was to provide the 3 chromosomes chained together as a single backbone in the `Minigraph` construction step. This would allow sequences to be mapped to any of chr C, G and H to be considered together in the subsequent steps of alignment and graph the `Minigraph-Cactus` pipeline. The expectation was to therefore produce a graph that showed the recombination from the mapping of the contigs of the other genomes.
We generated a graph using `Minigraph` with the chimeric chromosome CGH and all the contigs of the other genomes assigned to chromsome C, G and H. As shown in figure 3.8a, it does not represent any recombination event. This was somewhat expected, as it is known that `Minigraph` does not also consider inversion between genomes. When the pipeline of `Minigraph-Cactus` is run using by chromosome communities, with the chimeric chromosome CGH and its assigned contigs as a single community, it is possible to see the tangle between the chromosomes, as shown in figures 3.8b 3.11.

3.2.3 Representing translocation events from groups of genomes

Applying simple community separation on the all vs all alignment of the contigs, like the one suggested in the manual of `pggb`, does not help confirming the hypothesis, mainly because of segment length selection. Figure 3.9 shows community detection using Louvain algorithm on the contig network inferred by all vs all alignment. This inter-chromosomal rearrangement is instead detectable using linear whole-genome assembly based tools. By aligning J2 to the relative reference genomes B2 with `wfmash` and then looking for syntenies and rearrangements with `SyRI`, it is possible to detect syntenic path (longest set of co-linear regions), structural rearrangements (inversions, translocations, and duplications) [47]. Figure 3.10 shows the detected rearrangements and duplications between the 3 chromosomes using `plotsr` [48]. While figure 3.10 shows in a clear way the kind of inter-chromosomal variation between the 2

strains, SyRI does work only with genomes resolved to chromosome level. This means that such analysis is not possible on the whole cohort using standard linear reference tools. The only way to see the rearrangement for those assemblies is through a variation graph built with pggb and Minigraph-Cactus. This result shows the power of the pangenome model to analyze a set of genomes.

3.2.4 Estimating core genome and pangenome growth

Finally, pangenome graphs are also useful to quantify the part of the genome that is shared between genomes (what is called core-genome) and the parts that are mostly shared or private to each one. It is also interesting to estimate its growth, i.e. to measure how much the total genomic content grows with the increase of the size of the sample. These metrics can be obtained by using the tool panacus [49]. Panacus is a tool that calculates coverage distributions of countable elements in variation graphs: it uses paths to detect how many genomes are associated to any node, edge or basepair of the pangenome graph. From these distributions it computes the pangenome growth and core curves as function of the number of genomes.
Calculating these metrics can be used to validate that the two variation graphs built with pggb and Minigraph-Cactus agree on the underlying genetic distribution of the input sample.
Figure 3.13 shows very concordant metrics for the two variation graphs. First, they show similar basepair coverage histogram, that highlight a great portion of genome shared by all the samples, and a non-negligible share of sequences that are private to each genome ($\sim$ 1.5 Mbp in total). Secondly, both growth curves show a similar pattern of a plateau. This is also highlighted by the fraction of new base pairs introduced by each sample, that decreases from $\sim$ 700k new basepairs with the new sample to less than $\sim$ 142k base pairs in the 11th genome. Finally, the core pangenome can be estimated by the basepairs that are spelled by all the genomes in the graph: the computed value for the 11 samples is $\sim 13,2$ Mbp for pggb and $\sim 13,6$ Mbp for Minigraph-Cactus. The variability in the results is expected and due to different graph construction methods.

3.3 Conclusion and Perspectives

The work presented above shows how pangenomes can serve as analysis platforms of samples from same-strain yeast.
dBG-based tools currently offer a limited range of possibilities, especially for lower sample sizes. They best serve as fast and memory-efficient container for large amounts of data that require simple interrogations like absence-presence queries. This model can nevertheless help answer simple biological questions on such small samples, like phylogenetic analysis shown in figure 3.5a.
Variation Graphs on the other side are powerful and very useful on few genomes analysis as they can be built quickly enough and provide a well-established platform for downstream analysis tools. On the other side, there is still a need

for very labor-intensive manual revision of the output graphs to find the input parameters that produce the best result, as, for example, it takes more than 25 rounds to find the best `pggb` parameters to produce the graph shown in figure 3.7. As they are produced on heuristics and not on mathematically defined concepts, each variation graphs-construction tool produces different results: in figure 3.12, it is possible to see the difference of the small chromosome E between `pggb` and `Minigraph-Cactus` using a one dimensional representation. Finally, in order to detect the inter-chromosomal rearrangement with `Minigraph-Cactus` as in figure 3.11, I had to rewrite the pipeline and modify the input data.
Apart from the aforementioned limitations, that show how such methods, although established, could be more robust, this analysis shows how much potential there is to improve the current state-of-the art in genomes analysis. Linear-sequences tools are based on well-established genome-to-genome comparison methods that fail to adapt to heterogeneous data, like different levels of assembly quality. As `SyRI` fails to detect rearrangements on genomes that are not assembled to the chromosome level, variation graphs are able to show the variation among all samples, even if the majority of the genomes contain contigs.
This work is another display of the great potential pangenome approaches have. In the future it would be very interesting to build pipelines or develop tools that enable visualizations and straightforward analysis from variation graphs or ccdBGs to the same level as the current linear-genomes tools. As I have already conceptualized some possible approaches for ccdBGs, in the future it would be useful to develop simple prototypes and test how fast and precise these would be.
In the future it would be very informative to produce a more comprehensive report of the work done by my group and me, together with the other groups that worked on analyzing these novel *Lodderomyces elongisporus* samples, to offer a comprehensive view of how specific pangenomes can be built and used to provide improved genomic analysis capabilities.

sample	tot length (bp)	sequences	mean length	longest seq	shortest seq	N count	Gaps	N50	N50n
A1	15699113	25	627964.52	2595744	3907	0	0	1354781	5
B2	15485469	9	1720607.67	3516991	35166	0	0	2266654	3
C0	15532065	20	776603.25	3532941	810	0	0	1331232	4
D0	15507665	17	912215.59	3532853	5008	0	0	2160581	3
E0	15332588	18	851810.44	3544471	3228	0	0	1992182	3
F1	15664073	21	745908.24	3548518	1282	0	0	2165076	3
G0	15636520	19	822974.74	3548910	550	0	0	1697956	4
H0	15601346	21	742921.24	3549008	6842	0	0	2170489	3
I1	15639882	30	521329.40	3622524	536	0	0	1999295	3
J2	15425942	9	1713993.56	3543738	35442	0	0	2157297	3
K0	15752744	16	984546.50	3534745	19835	0	0	1631500	4

Table 3.5: *Lodderomyces elongisporus* samples assembly statistics. N50n represents the number of sequences that contain 50% of the assembly.
There are no unknown nucleotides or gaps (any arbitrary stretch of Ns - i.e. unknown nucleotide). Metrics obtained using assemblies stats software from Sanger Institute [42].

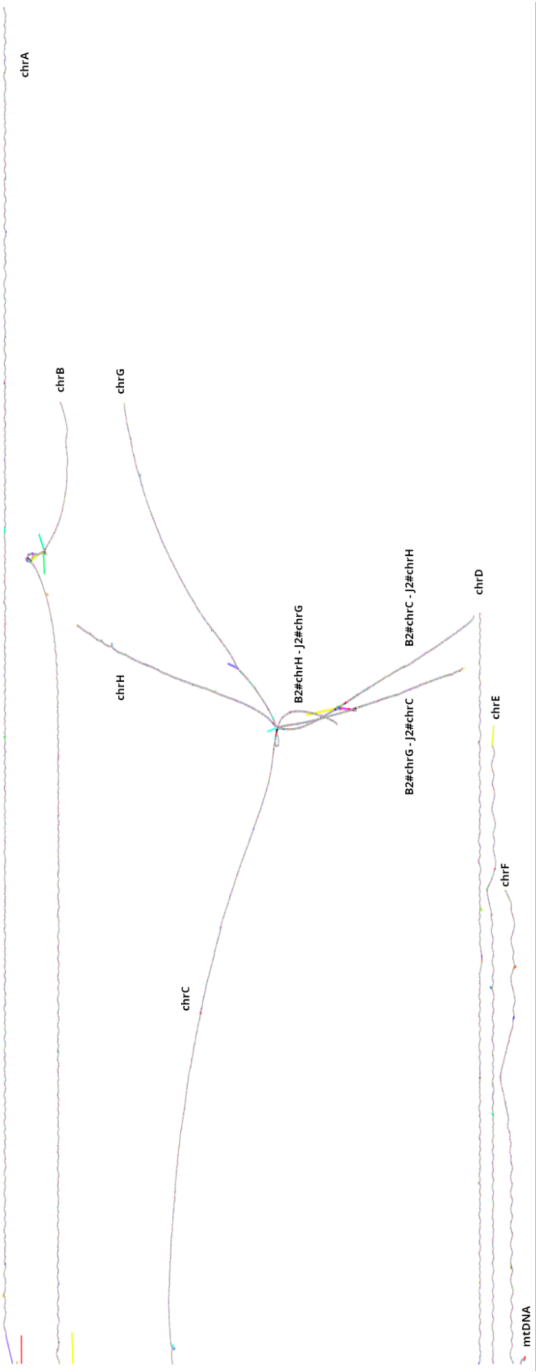
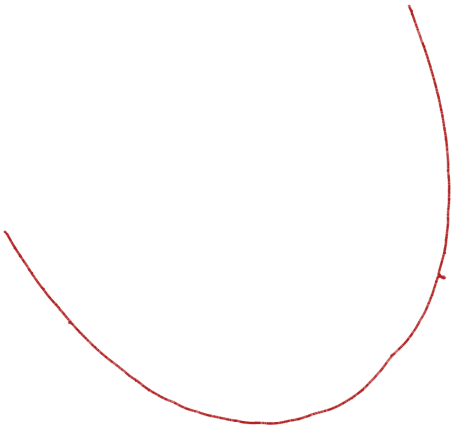
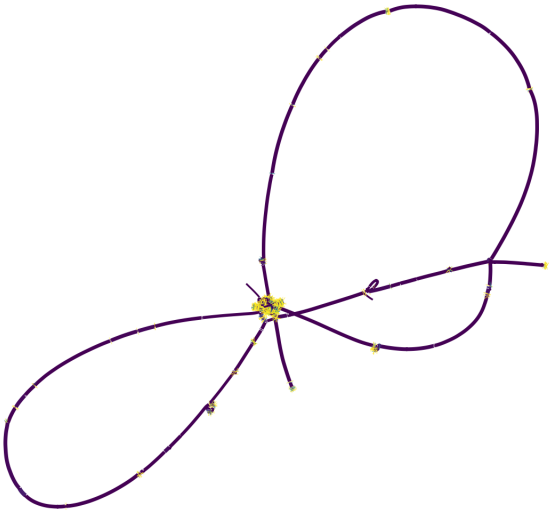


Figure 3.7: `gfaestus` visualization of the `pggb` variation graph of the 11 *Lodderomyces elongisporus* samples. Image produced by Simon Heumos.



(a) Graph of chimeric chromosome CGH from sample B2 and all the contigs of the other genomes aligning to it produced with `Minigraph`. The graph is linear and no inter-chromosomal event is visible.



(b) The graph after all the other steps of the `Minigraph-Cactus` pipeline, colored by depth, after simplification of variants $<$ 1kbp using the command `gfatools asm -b 1000 -u`. The large recombination event is now visible.

Figure 3.8: Difference in output between `Minigraph` and `Minigraph-Cactus` of the chimeric graph produced to visualize the inter-chromosomal event between C,G and H.

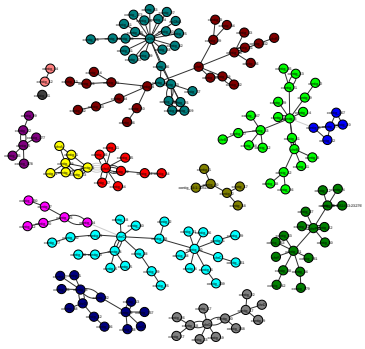


Figure 3.9: Community partition of the contigs based on all-vs-all alignment scores.

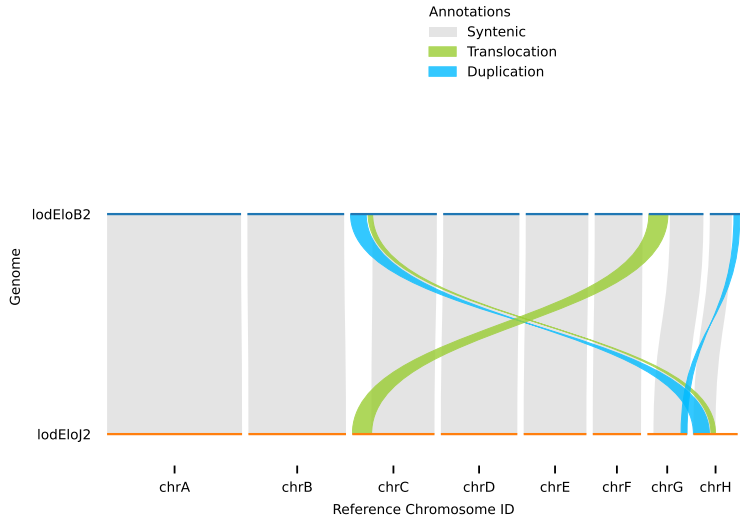


Figure 3.10: `plotsr` visualization of the inter-chromosomal recombination detected using `wfmash` and `SyRI`.

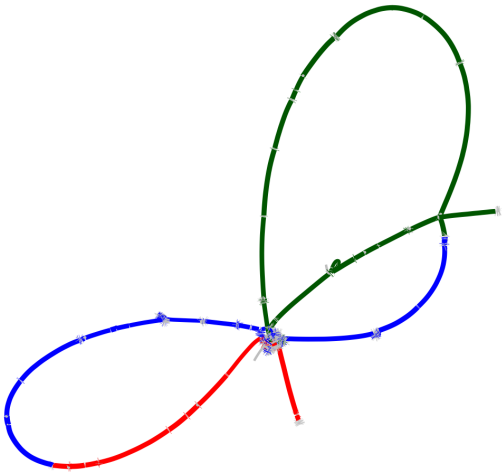
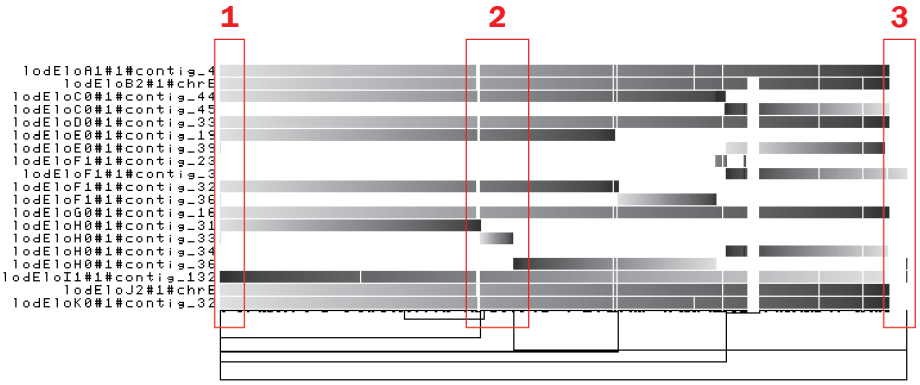
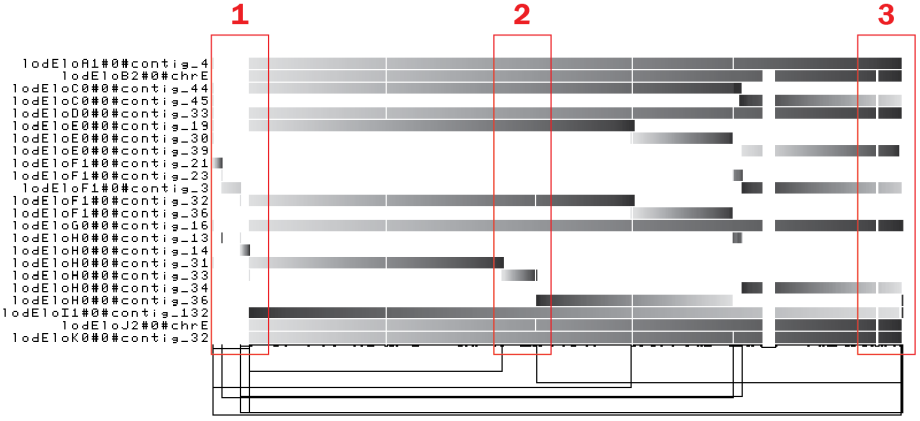


Figure 3.11: `Bandage` visualization of the tangle of chromosomes C, G and H in the `Minigraph-Cactus` variation graph. Nodes are colored based on `Minigraph` alignment of chromosome C (dark green), G (blue) and H (red) of the reference assembly B2. The three chromosomes are bound together because of the construction.



(a) One dimensional visualization of the variation graph built with `pggb`, containing 19 contigs from the assemblies, selected before construction using all vs all alignment data.



(b) One dimensional visualization of the variation graph built with `Minigraph-Cactus`, containing 23 contigs from the assemblies, selected automatically by the pipeline.

Figure 3.12: One dimensional visualization of chromosome E variation graphs of `pggb` and `Minigraph-Cactus` using `odgi`. Differences are highlighted by the three red boxes.

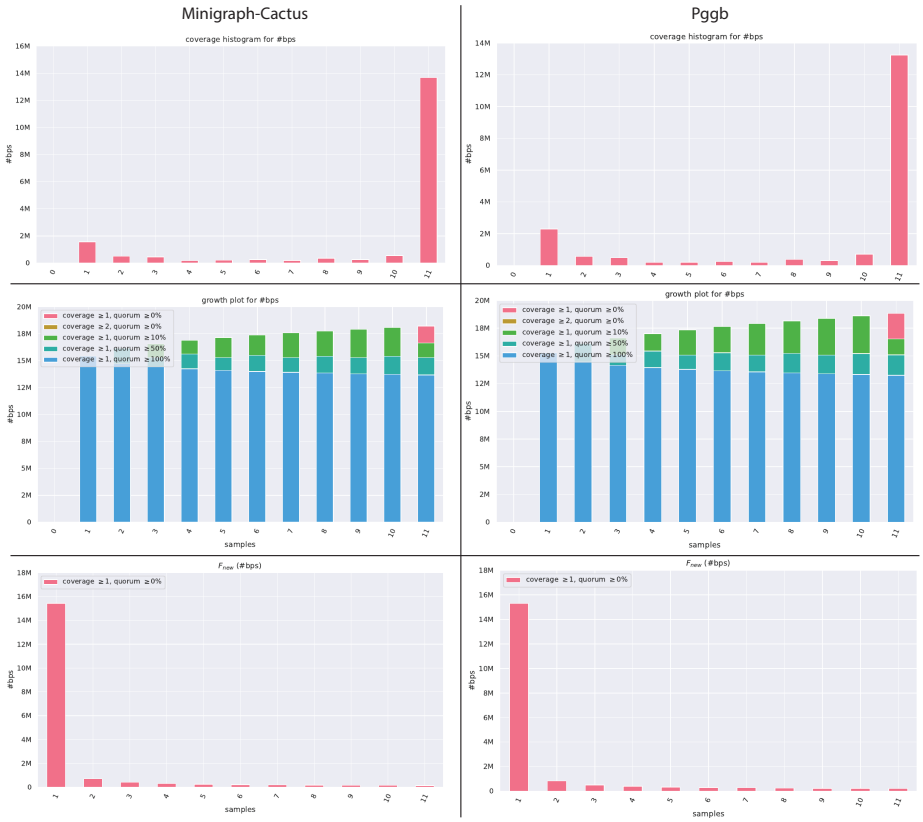


Figure 3.13: Pangenome core and growth of `pggb` and `Minigraph-Cactus` variation graphs. The top figures show the coverage histogram in number of basepairs, the middle ones show the evolution of the pangenome growth and core by increasing sample size and the bottom ones the new basepairs added by each new genome in the sample. The two graph exhibit concordant metrics.

Bibliography

[1] Haeussler, M., Zweig, A. S., Tyner, C., Speir, M. L., Rosenbloom, K. R., Raney, B. J., Lee, C. M., Lee, B. T., Hinrichs, A. S., Gonzalez, J. N., et al. "The UCSC genome browser database: 2019 update". In: *Nucleic acids research* vol. 47, no. D1 (2019), pp. D853–D858.

[2] Garrison, E. et al. "Variation graph toolkit improves read mapping by representing genetic variation in the reference". In: *Nature Biotechnology* vol. 36, no. 9 (Oct. 2018), pp. 875–879.

[3] Consortium, T. C. P.-G. "Computational pan-genomics: status, promises and challenges". In: *Briefings in Bioinformatics* vol. 19, no. 1 (Oct. 2016), pp. 118–135. ISSN: 1477-4054. DOI: 10.1093/bib/bbw089. eprint: https://academic.oup.com/bib/article-pdf/19/1/118/25406834/bbw089.pdf. URL: https://doi.org/10.1093/bib/bbw089.

[4] Sirén, J. et al. "Pangenomics enables genotyping of known structural variants in 5202 diverse genomes". In: *Science* vol. 374, no. 6574 (2021), abg8871. DOI: 10.1126/science.abg8871. eprint: https://www.science.org/doi/pdf/10.1126/science.abg8871. URL: https://www.science.org/doi/abs/10.1126/science.abg8871.

[5] Sherman R.M., S. S. "Pan-genomics in the human genome era." In: *Nat Rev* vol. Genet, no. 21 (2020), pp. 243–254. DOI: https://doi.org/10.1038/s41576-020-0210-7.

[6] Wang, T. et al. "The Human Pangenome Project: a global resource to map genomic diversity". In: *Nature* vol. 604, no. 7906 (Apr. 2022), pp. 437–446. ISSN: 1476-4687. DOI: 10.1038/s41586-022-04601-8. URL: https://doi.org/10.1038/s41586-022-04601-8.

[7] Ebler, J. et al. "Pangenome-based genome inference allows efficient and accurate genotyping across a wide spectrum of variant classes". In: *Nature Genetics* vol. 54, no. 4 (Apr. 2022), pp. 518–525. ISSN: 1546-1718. DOI: 10.1038/s41588-022-01043-w. URL: https://doi.org/10.1038/s41588-022-01043-w.

[8] Liao, W.-W. et al. "A draft human pangenome reference". In: *Nature* vol. 617, no. 7960 (May 2023), pp. 312–324. ISSN: 1476-4687. DOI: 10.1038/s41586-023-05896-x. URL: https://doi.org/10.1038/s41586-023-05896-x.

[9] Sirén, J. and Paten, B. "GBZ file format for pangenome graphs". In: *Bioinformatics* vol. 38, no. 22 (Sept. 2022), pp. 5012–5018. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btac656. eprint: https://academic.oup.com/bioinformatics/article-pdf/38/22/5012/47153721/btac656.pdf. URL: https://doi.org/10.1093/bioinformatics/btac656.

[10] Sheikhizadeh, S., Schranz, M. E., Akdel, M., Ridder, D. de, and Smit, S. "PanTools: representation, storage and exploration of pan-genomic data". In: *Bioinformatics* vol. 32, no. 17 (Sept. 2016), pp. i487–i493.

[11] Holley G., M. P. "Bifrost: highly parallel construction and indexing of colored and compacted de Bruijn graphs." In: *Genome Biol*, no. 21 (2020), p. 249. DOI: https://doi.org/10.1186/s13059-020-02135-8.

[12] Garrison, E. et al. "Building pangenome graphs". In: *bioRxiv* (2023). DOI: 10.1101/2023.04.05.535718. eprint: https://www.biorxiv.org/content/early/2023/04/06/2023.04.05.535718.full.pdf. URL: https://www.biorxiv.org/content/early/2023/04/06/2023.04.05.535718.

[13] Minkin, I., Pham, S., and Medvedev, P. "TwoPaCo: an efficient algorithm to build the compacted de Bruijn graph from many complete genomes". In: *Bioinformatics* vol. 33, no. 24 (Sept. 2016), pp. 4024–4032. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btw609. eprint: https://academic.oup.com/bioinformatics/article-pdf/33/24/4024/25168506/btw609.pdf. URL: https://doi.org/10.1093/bioinformatics/btw609.

[14] Li, H., Feng, X., and Chu, C. "The design and construction of reference pangenome graphs with minigraph." In: *Genome Biol*, no. 21 (2020), p. 265. DOI: https://doi.org/10.1186/s13059-020-02168-z.

[15] Ekim, B., Berger, B., and Chikhi, R. "Minimizer-space de Bruijn graphs: Whole-genome assembly of long reads in minutes on a personal computer". In: *Cell Systems* vol. 12, no. 10 (2021), 958–968.e6. ISSN: 2405-4712. DOI: https://doi.org/10.1016/j.cels.2021.08.009. URL: https://www.sciencedirect.com/science/article/pii/S240547122100332X.

[16] Armstrong, J. et al. "Progressive Cactus is a multiple-genome aligner for the thousand-genome era". In: *Nature* vol. 587, no. 7833 (Nov. 2020), pp. 246–251. ISSN: 1476-4687. DOI: 10.1038/s41586-020-2871-y. URL: https://doi.org/10.1038/s41586-020-2871-y.

[17] Hickey, G. et al. "Pangenome graph construction from genome alignments with Minigraph-Cactus". In: *Nature Biotechnology* (May 2023). ISSN: 1546-1696. DOI: 10.1038/s41587-023-01793-w. URL: https://doi.org/10.1038/s41587-023-01793-w.

[18] Chin, C.-S., Behera, S., Metcalf, G., Gibbs, R. A., Boerwinkle, E., and Sedlazeck, F. J. "A pan-genome approach to decipher variants in the highly complex tandem repeat of LPA". In: *bioRxiv* (2022). DOI: 10.1101/2022.06.08.495395. eprint: https://www.biorxiv.org/content/early/2022/06/10/2022.06.08.495395.full.pdf. URL: https://www.biorxiv.org/content/early/2022/06/10/2022.06.08.495395.

[19] Dendrou, C. A., Petersen, J., Rossjohn, J., and Fugger, L. "HLA variation and disease". In: *Nature Reviews Immunology* vol. 18, no. 5 (May 2018), pp. 325–339. ISSN: 1474-1741. DOI: 10.1038/nri.2017.143. URL: https://doi.org/10.1038/nri.2017.143.

[20] Vietzen, H., Zoufaly, A., and Traugott, M. e. a. “Deletion of the NKG2C receptor encoding KLRC2 gene and HLA-E variants are risk factors for severe COVID-19.” In: *Genet Med* vol. 23 (2021), pp. 963–967. DOI: 10.1038/s41436-020-01077-7.

[21] Guarracino, A., Heumos, S., Nahnsen, S., Prins, P., and Garrison, E. “ODGI: understanding pangenome graphs”. In: *Bioinformatics* (May 2022). btac308. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btac308. eprint: https://academic.oup.com/bioinformatics/advance-article-pdf/doi/10.1093/bioinformatics/btac308/43882774/btac308.pdf. URL: https://doi.org/10.1093/bioinformatics/btac308.

[22] “100,000 Genomes Pilot on Rare-Disease Diagnosis in Health Care — Preliminary Report”. In: *New England Journal of Medicine* vol. 385, no. 20 (2021). PMID: 34758253, pp. 1868–1880. DOI: 10.1056/NEJMoa2035790. eprint: https://doi.org/10.1056/NEJMoa2035790. URL: https://doi.org/10.1056/NEJMoa2035790.

[23] Johnson, R. et al. “Leveraging genomic diversity for discovery in an electronic health record linked biobank: the UCLA ATLAS Community Health Initiative”. In: *Genome Medicine* vol. 14, no. 1 (Sept. 2022), p. 104. ISSN: 1756-994X. DOI: 10.1186/s13073-022-01106-x. URL: https://doi.org/10.1186/s13073-022-01106-x.

[24] Schneider, V. A. et al. “Evaluation of GRCh38 and de novo haploid genome assemblies demonstrates the enduring quality of the reference assembly”. en. In: *Genome Res* vol. 27, no. 5 (Apr. 2017), pp. 849–864.

[25] Nurk, S. et al. “The complete sequence of a human genome”. In: *Science* vol. 376, no. 6588 (2022), pp. 44–53. DOI: 10.1126/science.abj6987. eprint: https://www.science.org/doi/pdf/10.1126/science.abj6987. URL: https://www.science.org/doi/abs/10.1126/science.abj6987.

[26] Baid, G. et al. “DeepConsensus improves the accuracy of sequences with a gap-aware sequence transformer”. In: *Nature Biotechnology* (Sept. 2022). ISSN: 1546-1696. DOI: 10.1038/s41587-022-01435-7. URL: https://doi.org/10.1038/s41587-022-01435-7.

[27] Baid, G. et al. *Dataset. Google Brain Assemblies.* 2023. URL: https://console.cloud.google.com/storage/browser/brain-genomics-public/research/deepconsensus/publication/analysis/genome_assembly.

[28] Liao, W.-W. et al. *Dataset. Human Pangenome Reference Consortium Asseblies.* 2023. URL: https://s3-us-west-2.amazonaws.com/human-pangenomics/index.html?prefix=working/.

[29] Doerr, D. “Gfaffix identifies walk-preserving shared affixes in variation graphs and collapses them into a non- redundant graph structure.” In: (2021). URL: https://github.com/marschall-lab/GFAffix.

[30] Guarracino, A., Mwaniki, N., Marco-Sola, S., and Garrison, E. *wfmash: whole-chromosome pairwise alignment using the hierarchical wavefront algorithm.* Sept. 2021. URL: https://github.com/ekg/wfmash.

[31] Garrison, E. and Guarracino, A. "Unbiased pangenome graphs". In: *Bioinformatics* vol. 39, no. 1 (Nov. 2022), btac743. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btac743. eprint: https://academic.oup.com/bioinformatics/article-pdf/39/1/btac743/48448986/btac743.pdf. URL: https://doi.org/10.1093/bioinformatics/btac743.

[32] Guarracino, A. and Garrison, E. *smoothxg: local reconstruction of variation graphs using partial order alignment*. 2021. URL: https://github.com/pangenome/smoothxg.

[33] Rautiainen, M. and Marschall, T. "GraphAligner: rapid and versatile sequence-to-graph alignment". In: *Genome Biology* vol. 21, no. 1 (Sept. 2020), p. 253. ISSN: 1474-760X. DOI: 10.1186/s13059-020-02157-2. URL: https://doi.org/10.1186/s13059-020-02157-2.

[34] Li, H. "Minimap2: pairwise alignment for nucleotide sequences". In: *Bioinformatics* vol. 34, no. 18 (May 2018), pp. 3094–3100. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bty191. eprint: https://academic.oup.com/bioinformatics/article-pdf/34/18/3094/25731859/bty191.pdf. URL: https://doi.org/10.1093/bioinformatics/bty191.

[35] Andreace, F. *Github sourcecode*. 2023. URL: https://github.com/frankandreace/CRHPG.

[36] Andreace, F. *Zenodo sourcecode*. 2023. DOI: 10.5281/zenodo.8370336.

[37] Yadav, A., Jain, P., Jain, K., Wang, Y., Singh, A., Singh, A., Xu, J., and Chowdhary, A. "Genomic Analyses of a Fungemia Outbreak Caused by Lodderomyces elongisporus in a Neonatal Intensive Care Unit in Delhi, India". In: *mBio* vol. 14, no. 3 (2023), e00636–23. DOI: 10.1128/mbio.00636-23. eprint: https://journals.asm.org/doi/pdf/10.1128/mbio.00636-23. URL: https://journals.asm.org/doi/abs/10.1128/mbio.00636-23.

[38] Wang, Y. and Xu, J. "Lodderomyces elongisporus: An emerging human fungal pathogen". In: *PLOS Pathogens* vol. 19, no. 9 (Sept. 2023), pp. 1–9. DOI: 10.1371/journal.ppat.1011613. URL: https://doi.org/10.1371/journal.ppat.1011613.

[39] Asadzadeh, M., Al-Sweih, N., Ahmad, S., Khan, S., Alfouzan, W., and Joseph, L. "Fatal Lodderomyces elongisporus Fungemia in a Premature, Extremely Low-Birth-Weight Neonate". In: *Journal of Fungi* vol. 8, no. 9 (2022). ISSN: 2309-608X. DOI: 10.3390/jof8090906. URL: https://www.mdpi.com/2309-608X/8/9/906.

[40] Dear, T., Joe Yu, Y., Pandey, S., Fuller, J., and Devlin, M. K. "The first described case of Lodderomyces elongisporus meningitis". In: *Journal of the Association of Medical Microbiology and Infectious Disease Canada* vol. 6, no. 3 (2021), pp. 221–228. DOI: 10.3138/jammi-2021-0006. eprint: https://doi.org/10.3138/jammi-2021-0006. URL: https://doi.org/10.3138/jammi-2021-0006.

[41] Koh, B., Halliday, C., and Chan, R. “Concurrent bloodstream infection with Lodderomyces elongisporus and Candida parapsilosis”. In: *Medical Mycology Case Reports* vol. 28 (2020), pp. 23–25. ISSN: 2211-7539. DOI: https://doi.org/10.1016/j.mmcr.2020.03.007. URL: https://www.sciencedirect.com/science/article/pii/S221175392030021X.

[42] *Asseblies stats software by Sanger Institute*. https://github.com/sanger-pathogens/assembly-stats. Last access on 2024-8-05. 2024.

[43] Rempel, A. and Wittler, R. “SANS serif: alignment-free, whole-genome-based phylogenetic reconstruction”. In: *Bioinformatics* vol. 37, no. 24 (June 2021), pp. 4868–4870. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btab444. eprint: https://academic.oup.com/bioinformatics/article-pdf/37/24/4868/50334826/btab444.pdf. URL: https://doi.org/10.1093/bioinformatics/btab444.

[44] Huson, D. H. and Bryant, D. “The SplitsTree App: interactive analysis and visualization using phylogenetic trees and networks”. In: *Nature Methods* (Sept. 2024). ISSN: 1548-7105. DOI: 10.1038/s41592-024-02406-3. URL: https://doi.org/10.1038/s41592-024-02406-3.

[45] Brejová, B., Hodorová, V., Mutalová, S., Cillingová, A., Tomáška, Ľ., Vinař, T., and Nosek, J. “Chromosome-level genome assembly of the yeast Lodderomyces beijingensis reveals the genetic nature of metabolic adaptations and identifies subtelomeres as hotspots for amplification of mating type loci”. In: *DNA Research* vol. 31, no. 3 (Apr. 2024), dsae014. ISSN: 1756-1663. DOI: 10.1093/dnares/dsae014. eprint: https://academic.oup.com/dnaresearch/article-pdf/31/3/dsae014/57725551/dsae014_suppl_supplementary_figures_s1-s9_tables_s1-s2.pdf. URL: https://doi.org/10.1093/dnares/dsae014.

[46] Guarracino, A. et al. “Recombination between heterologous human acrocentric chromosomes”. In: *Nature* vol. 617, no. 7960 (May 2023), pp. 335–343. ISSN: 1476-4687. DOI: 10.1038/s41586-023-05976-y. URL: https://doi.org/10.1038/s41586-023-05976-y.

[47] Goel, M., Sun, H., Jiao, W.-B., and Schneeberger, K. “SyRI: finding genomic rearrangements and local sequence differences from whole-genome assemblies”. In: *Genome Biology* vol. 20, no. 1 (Dec. 2019), p. 277. ISSN: 1474-760X. DOI: 10.1186/s13059-019-1911-0. URL: https://doi.org/10.1186/s13059-019-1911-0.

[48] Goel, M. and Schneeberger, K. “plotsr: visualizing structural similarities and rearrangements between multiple genomes”. In: *Bioinformatics* vol. 38, no. 10 (Apr. 2022), pp. 2922–2926. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btac196. eprint: https://academic.oup.com/bioinformatics/article-pdf/38/10/2922/49010748/btac196.pdf. URL: https://doi.org/10.1093/bioinformatics/btac196.

[49] Parmigiani, L., Garrison, E., Stoye, J., Marschall, T., and Doerr, D. "Panacus: fast and exact pangenome growth and core size estimation". In: *bioRxiv* (2024). DOI: 10.1101/2024.06.11.598418. eprint: https://www.biorxiv.org/content/early/2024/06/12/2024.06.11.598418.full.pdf. URL: https://www.biorxiv.org/content/early/2024/06/12/2024.06.11.598418.

Chapter 4

Exploring new *k*-mer based methods for Pangenomics

4.1 Introduction: using *k*-mer sets in pangenomics

As discussed in the previous sections of this manuscript, the construction of pangenome as variation graphs is based on an alignment step that is known to be accurate but computationally expensive. Even if recent advances on alignment algorithms and tools, as the wavefront algorithm [1] or full-text indexes as the r-index [2] and move index [3] have provided improvement in construction time or query performance.

The variation graph represents an effective methodology for conducting thorough analyses of a curated set of samples. As an illustrative example, at the time of this manuscript's composition, the Human Pangenome Reference Consortium is in the process of releasing an additional cohort comprising approximately 220 high-fidelity human genome sequences. These genomes are intended for integration into the development of an updated reference pangenome for the human species.

The alignment step implicitly requires high-quality complete assemblies to produce reasonably connected graphs. While it is anticipated that the availability of such high-quality genomes will continue to grow in the coming years, there is currently a large quantity of raw (or minimally processed) data that can be utilized in pangenomics applications [4, 5] but cannot be harnessed by variation graph models.

For this reason, k-mer-based approaches provide a robust alternative. As the k-mer length used is typically relatively small (from 21 to 100), these approaches can be applied to more fragmented assemblies or directly to raw sequencing reads, and their scalability has been proven to be orders of magnitude superior to variation graphs. Moreover, they can be used to build representations from data of varying quality, such as phased assemblies from one cohort and unitigs from another. These tools usually employ different data structures to represent a de Bruijn graph model internally and efficiently. The main challenge of these data structures is primarily the amount of space used to represent the k-mers versus the time used to query elements (single k-mers or sequences). Consequently, implementation decisions are often bound to optimization compromises made to achieve specific goals: disk compression to produce small-sized indexes from large collections, fast query time of a given sequence, or time/memory trade-offs. The computational resources necessary for generating compacted colored de Bruijn graphs from a set of input genomes are, as shown in the preceding chapter, reduced in comparison to those required for variation graph construction.

Furthermore, the tools employed in this approach demonstrate superior scalability, allowing for the processing of substantially larger genomic collections.
As *k*-mer-based methods present valid alternatives for pangenomic studies, I focused part of my PhD on studying and developing data structures that could find useful applications in pangenomics. Here, I will present three projects that yielded relevant outcomes. In two of these projects, I had the opportunity to collaborate with other researchers, both within my unit and from other groups. For these projects, my work was included as a component in a larger framework. While I cannot claim to have carried each of these projects as a whole, I nevertheless brought significant input to each of them.

The chapter is organized as follows:

- Introduction on *k*-mer sets and metadata representation: why it is needed and how;
- Overview of our contributions and my part on them;
- `muset`: from graph to matrices for downstream analysis;
- Prototyping dynamic data structures for *k*-mer counting:
  - Re-implementation of a Quotient Filter as a base for multiple applications;
  - Explore dynamicity without indexing: super-*k*-mer sorting;
- Summary and conclusions.

4.2 Introduction: sets of *k*-mers and metadata association

k-mer-based data structures for representing sets of input genomes inevitably involve trade-offs to support useful applications in pangenomics. The key characteristics of such structures are often in competition with one another, requiring a balance between the following:

- Efficient storage of the data;
- Fast querying of a *k*-mer or string to retrieve metadata;

The process of optimizing data storage for quick lookups is referred to as indexing, whereas the processes of retrieving the metadata and determining membership are called querying. Ideally, these data structures should be capable of performing both tasks efficiently, especially as datasets grow in size—as discussed in the introduction of this chapter. Given that genomic data is expanding at near-exponential rates, minimizing both storage requirements and query times for *k*-mer sets has become increasingly crucial.
A simple but very effective analogy of indexing and querying can be done with books and words. Imagine I recall having a book in my library featuring a protagonist named Ricardo whose narrative unfolds also in Paris. Without a

systematic organization of my collection, I might need to sequentially examine each book's content from beginning to end. This is an incredibly inefficient process, especially in the case of a large library. A more effective approach for me would be to maintain a comprehensive index of all words contained within my library's volumes. This system would allow me to rapidly identify books containing specific terms, such as "Ricardo" or its diminutive form, "Ricardito". Furthermore, if I were to implement an additional index associating geographical locations with books featuring scenes set in those places, I could cross-reference the occurrence of the name "Ricardo" with Paris as a location and, without even needing to open a single book, I would find that the book I'm recalling is *Travesuras de la niña mala* of the Nobel Laureate Mario Vargas Llosa.
This is an example of how, indexing sequencing data, *the words* and their metadata, *the places*, one can rapidly check which samples, *the books*, contain a requested value, without having to look at the raw data (the content of the book).
As the dBG is a model for a *k*-mer set representation, under the hood there are different data structures that can be used to store the *k*-mers and index them for efficient retrieval. These data structures can be divided into exact and inexact data structures. In the rest of this section I will present the characteristics of such data structures and briefly mention some propaedeutical to the prototypes we developed.

4.2.1 Hashing *k*-mers

As described in section 2.2, a *k*-mer is a sub-string of length k of a biological sequence. In order to reduce the space used to store them, the text string is converted, or hashed, into a binary string that can therefore be interpreted as an integer number. The use of the equivalence of a binary string with an integer is the basis of a great part of *k*-mer-based data structures: from here on I will consider methods for *k*-mers representation as binary string using hash functions. Methods that consider *k*-mer as text string won't be therefore considered.

A hash function is any function that maps data from one set (usually text but not only) to another (usually fixed-size machine-word-length integers). The ingested value is called key and the output is usually called hash value or simply hash. They are used in a lot of applications such as basic computer science models such as dictionaries, cryptography or bioinformatics.

Hash functions should optimize on some of these properties:

Uniformity In most cases input data should be mapped in an uniform way in the output space: in the *k*-mer case, lexicographically similar *k*-mers should be mapped to different hashes. This is not true when data locality matters: the property of similar *k*-mers pointing to similar hashes can be used by some data structures to avoid many cache misses, improving the computation time.

Speed the fastest it is possible to compute a hash from a key, the better it is. Speed depends on the number and latency of the operations executed computation;

Collision avoidance collisions, i.e. mapping different keys into the same hash, should be infrequent. The collision rate is correlated to the size of the hash space and therefore to the space that can be used to store the hash. This trade-off will be explored better in the next section.

Moreover, *k*-mer length impacts the time/space trade-off stated in the previous section: as larger *k*-mer offer greater specificity, they largely increase the amount of space needed to store them (because the hash will probably be larger) as also shown for plain-text representation in section 2.2.

4.2.2 Minimum set of operations and metadata

Given a set of sequencing samples, the data structure should support the addition of *k*-mers from each sample through an *insert()* operation during its construction. In some data structures, the order of the input data doesn't matter, while others require a specific order for insertion to function correctly. As we'll discuss in more detail in Section 4.2.5, post-construction insertion isn't always a guaranteed feature in certain implementations.
Normally these data structures can return two kind of information: the presence absence of an element, using a *memb()* operation and metadata associated with the specific *k*-mer. There are some implementations in which:

- only the presence or absence can be retrieved, using the *memb()* operation. Bloom-filters, that will be presented in section 4.2.4.1, fall under this category.
- only metadata can be retrieved. Minimal Perfect Hash Functions (MPHF), that won't be discussed in this manuscript, are maps of a static set of known keys to metadata that do not support *memb()* operations [6].
- both presence and metadata can be queried, as in color compacted dBG, presented in section 2.4.4.1.

Metadata is a broad keyword that I will use to identify information associated to the *k*-mers represented in the data structure itself. Presence or absence of a *k*-mer in the set is usually, with some exception such as MPHF, directly encoded in the insertion of the elements in the data structure and do not require additional bits. Data structures that report only absence or presence are considered to not support metadata. The ones that do support different kind of metadata are considered associative, i.e. associate metadata to the *k*-mers.
A data structure may or may not retain internal information after executing a membership query. For instance, when searching for an element in a dictionary, the CPU ends up accessing a chunk of memory that includes the queried key-value pair along with other closely located data. Some data structures explicitly store

variables to remember exactly where in their internal representation the most recent *memb()* operation occurred. This becomes especially important when sequences, not individual k-mers, are queried, as sequential *memb()* operations are performed on k-mers that typically overlap. Efficiently leveraging these properties can significantly impact the scalability and performance of these data structures.

4.2.2.1 Metadata types

In the most fundamental case, the data structure represents the presence or absence of an element within it, utilizing a binary *memb()* operation (denoted as 0 for absence and 1 for presence). Other metadata that can be useful in pangenomics can store:

Counts The data structure contains the number of times a k-mer appears in it. Counts are useful to discern copy variants number in different DNA samples, the expression of genes in RNA samples and for species abundance estimation in metagenomic samples.

Colors It stores in which samples a k-mer has been seen and returns a list of containing samples for each queried k-mer.

Id In applications in which the graph structure is relevant (for example in visualization), it is useful to know in which k-mers (in the case of dBG) or unitigs (in the case of cdBGs) of the graph are represented the k-mers of a queried sequence. This case is relevant for dBG based models.

Text Text data can be used to associate k-mers to genetic information as genes, regulatory elements, flags to discern pathogenic variants from non-pathogenic ones and so on.

Of these metadata, the first two are the ones that are usually taken in consideration for query by recently developed data structures. Text data would impose a significant space requirement for the data structure and could be mimicked by assigning numerical labels to text. Just in the case of text metadata query an additional map associating numbers to text would be used. The id information is quite overlooked by, to my knowledge, all implementations.
We will now turn out attention to sets of k-mer sets, where each input sample is considered as a different set. This is done by using colors, that make possible to retrieve from the data structure in which sample a k-mer can be found. The color terminology can be confusing as it is often used to refer to two different concepts. From here on, the term color is used to refer to the data set from which a set of k-mer comes from. A k-mer x will have colors 1,2,3 if it comes from dataset 1,2,3. Sometimes k-mers are associated to color sets, that represent a specific group of colors. For example the color set A will contain colors 1,2,3 so in the data structure k-mer x is associated to the identifier A and the dataset of origins can be retrieved by remembering the map from color-set to colors.

The use of color sets instead of directly associate a *k*-mer with its colors allows efficient storage [7].

4.2.2.2 Metadata: why it is important

Metadata is important to enable different kind of applications that need more information than just the presence or absence of a *k*-mer in a set.
In some applications, it's important to understand how many times a particular genetic sequence is repeated, and *k*-mer counting can serve as a useful proxy for that. For instance, the abundance of specific RNA molecules in a cell can help differentiate between normal and cancerous activity [8, 9]. Similarly, variations in the copy number of specific DNA regions can distinguish between phenotypes, revealing important differences between samples in a pangenome. In dBG models, counting *k*-mers is essential to capture the multiplicity of repeated sequences, whereas in variation graphs, copy number variants are implicitly represented within the paths themselves.
In some applications it is important to discern between the different samples used to fill the data structure, hence representing sets of *k*-mer sets [10]. Colors are vastly used in pangenomics, as they allow to keep track of the genomes associated to variations and the ones that are part of the core genome, both in bacteria and in eukaryotes.
Remembering the dBG overlap structure is also important in many applications that rely on visualization. This would enable fast subgraph identification for loci of interest and enable specific genomic applications for dBG based methods. For example `ssHash`, an indexing data structure for unitigs, would be suited for this scope [11].
Finally, part or all of these metadata might be useful to be stored at the same time for many applications, including pangenomics. For example, *k*-mer counts and colors are useful to differentiate copy number repeats between different genomes in a population. Counts and genome positions can be used for lossless reconstruction genomes.

4.2.3 Basic data structures: sorted list and hash table

The most basic data structure used in computer science to maintain an ordered collection of elements for efficient searching is a sorted list. By ordering the entire enumeration of the set of *k*-mers in each sample, one can use a binary search algorithm to find a requested *k*-mer in time $O(k\log n)$, using $O(kn)$ space, with n the cardinality of the set. The use of sorted lists for *k*-mer storage and retrieval is feasible for very basic cases involving small sets of *k*-mers. However, this approach becomes intractable for the large-scale genomic applications previously mentioned, due to poor scaling in both query time and storage requirements. Nevertheless, sorted list can be used in case the number of elements is greatly reduced (by using compacted *k*-mer representations for example) and to avoid costly indexing. More on sorted lists is going to be discussed in section 4.6.
Hash-tables, a well known implementation of dictionaries (or maps) in computer

science, associate keys and values using hash functions. Hash tables solve the problem of the query time, bringing it to $O(k)$ or $O(1)$, depending on the particular hash function used. As they still require $O(kn)$ space, general hash table implementations can be used when dynamicity is required and the space (bit per k-mer) is not a major bottleneck: Rust HashSet implementation has shown to be useful in particular genomic use-cases, as to find in which reads are present k-mers belonging to a specific set. I provided a small contribution on this particular subject by helping with the development of Back to Sequences, a tool written in Rust that uses an hash table to find in a set of samples S the reads r containing k-mers present in a set K. The tool outputs the sequences containing the k-mers, using both a minimum and maximum threshold given as input [12].

4.2.4 Approximate membership and filters

Approximate Membership Query (AMQ) data structures offer a trade-off between storage space (in memory or on disk) and query accuracy to enhance space efficiency. Unlike sorted lists or hash tables that always return correct information, AMQ structures respond with a non-zero probability of false positives (i.e., reporting the presence of an element that is not actually in the dataset). Due to this probabilistic nature, they are often referred to as probabilistic data structures.
Filters are a subset of AMQs frequently used in genomic applications. They allow fast answers on negative queries. Their basic value can be either a single bit (hence bitvectors) or any amount of bits that ensure optimal space-efficiency. These 'slots' that contain data can be smaller than a machine word or a single byte, using low-level implementation operations.

4.2.4.1 Bloom Filters

Bloom filters represent the most widely utilized probabilistic data structures, finding applications in diverse fields. In genomics, they are employed for tasks such as the elimination of contaminant-related reads from ancient DNA samples [13]. Beyond genomics, their utility extends to various computational domains. For instance, they are used to reduce disk lookups for non-existent rows or columns in databases, and they are part of Google Chrome's protection mechanism against malicious URLs [14, 15]. In genomics they are used to provide a very space-efficient representation of a set of k-mers by using a bitvector and multiple different hash functions. When a k-mer is inserted, multiple different hashes are generated and the position in the bitvector corresponding to the hashes are set to 1. When an element is queried, the same hash functions are applied and if all positions in the bitvector are set to 1 the element is considered present. If at least one position is set to 0 it means that the element is not present, thus preventing false negatives. As collision can happen, especially when using multiple hash functions, it is instead possible that a position associated to

the output of a hash function of a *k*-mer was set to 1 by the output of another hash of another *k*-mer, leading to false positives, i.e. reporting a *k*-mer is inside the data structure while it is not present.
Counting Bloom filters store counts instead of presence/absence in the vector positions. The value they return depends in the way elements are inserted. A *k*-mer count gets updated if a value larger than the stored one gets inserted. When a sequence is queried, they return the minimum value of the counts associated to the composing *k*-mers.
Interleaved Bloom filters are instead made of several Bloom filters, as each one is filled with *k*-mers from a specific sample. The filters are divided into chunks and each specific chunk of all filters can be found in a contiguous region of memory. This speeds up color queries, as the lookup is done in a specific constrained region, limiting cache loading or misses.
Multiple implementation and optimization techniques, such as the blocked-Bloom filters used to speed query and insert operations, are used to maximize the potential of this data structure won't be addressed here but are thoroughly explained in these reviews [16, 17, 18].

4.2.4.2 Quotient Filters

Quotient filters are another data structure that is based on the idea of filling a vector with metadata but it does so in a different way compared to the Bloom filter. The hash computed from the *k*-mer get separated into two parts: the quotient (leftmost or rightmost bits, depending on the implementation) and the remainder (the rest). The size of the quotient depends on the amount of data that is being stored. Instead of filling the vector with the metadata at the position associated to the whole hash, it fills the slot at the position associated to the quotient with the bits of the remainder. In order to avoid collision when hashes with the same quotient occur, the remainders of a quotient are stored in order in successive slots, also called runs, to preserve the information and enable fast queries. This is done by using companion data structures that are used to trace where the run of a quotient is in the vector. When metadata such as counts has to be stored, multiple slots can be used to encode the count of a single remainder. This is done in the Counting Quotient Filter. Another possibility is to reserve some bits of the slots to store the count, as in the Backpack Quotient Filter. These filters enable collision resolution by using slots in a more flexible way. More about this data structure will be discussed in section 4.5.

4.2.5 Static vs Dynamic data structures

Another important feature of *k*-mer data structures is whether they can be modified after construction, which brings us to the distinction between static and dynamic data structures.

- **Static data structures** cannot be altered once they have been built. Addition or removal of elements from the original set is not supported. To

modify the underlying set, the entire data structure has to be recreated. Their advantage is efficiency, as they usually allow for greater compression and use less space making them ideal for applications where a stable reference set is used, and there isn't a need for frequent modifications. For example, they're well-suited for comparing new datasets against a reference genome that remains constant.

- **Dynamic data structures** allow updates after their initial construction. The types of modifications that can be made depend on the specific implementation and the use case. The most common operation is the insertion of new k-mers, which effectively functions as an union of two sets (for presence or colors). In the case of counting structures, usually a count of an existing k-mer is updated if last inserted has a greater count. Other possible updates include deleting k-mers or modifying the metadata associated with them.

While most methods are static, dynamic structures that allow efficient insertion and, less frequently, deletion of k-mers are being developed in recent years [16].

4.3 Our contributions: an outline

The three projects span different topics and can be divided at 2 different levels of engineering. The first is mainly organizing a pipeline with some already developed bioinformatics tools and contributing to the development of a tool for k-mer information manipulation. The other two are developments of a tool from scratch.
They can be presented as follow:

`muset` is a pipeline to construct plain text unitig matrices from input sample. It enables to build an abundance matrix in which the unitig count in each sample is the average of the counts of its constituent k-mers and a presence/absence matrix that report an unitig as present in a sample if its constituent k-mers are present in the sample over a given threshold.

A **Quotient filter** implementation that allows dynamic updates, resizing, and a framework to develop different specific data structures on top of it. It is the building block of a novel data structure, the Backpack Quotient Filter that has been recently published. I also redeveloped a Counting Quotient filter that uses a special scheme that uses less space by mimicking large k-mers by storing smaller ones.

A **super-k-mer sorting** implementation to explore a different data structure to store k-mers without indexing and allowing queries.

Some of the research and development I did, mostly in the second and third projects, can be labeled as exploration and prototyping as the result is not

intended to be a novel tool to be widely adopted by the community but as first step in possible route of research in this domain.

4.4 `muset`: building unitig matrices for downstream analyses

In this section I will present the work that I have been doing on building unitig abundance matrices.

4.4.1 Rationale

Recent advancements in genomic sequencing technologies have led to the generation of massive datasets from large-scale projects. Some of them very functional to human pangenomics, such as the 1K Genomes Project and the HPRC, while others focus on related genomic areas like transcriptomics (e.g., GEUVADIS [19]) and metagenomics (e.g., MetaSub [20], Tara Ocean [21]).
In pangenomics, these large datasets present significant challenges for traditional variation graph analyses due to their size and complexity. *k*-mer-based methods offer an alternative approach to study this data, using techniques such as *k*-mer counting and matrix representation. These methods can lead to improved accuracy in estimating the abundance of loci across multiple samples, enabling more comprehensive analyses of complex genomic datasets. A potential application of these methods could allow GWAS studies to be conducted on all possible variations within genomes, expanding beyond the current focus on SNPs and small indels. Additionally, *k*-mer abundance matrices could serve as training data for Deep Learning models to, potentially uncovering traits that distinguish healthy from non-healthy populations for specific diseases.
Given the usefulness of these applications, we propose a novel method for constructing plain text abundance unitig matrices. These matrices can be directly utilized in various downstream applications, offering a valuable resource for researchers in genomics and related fields.

4.4.2 Related work

Cutting-edge tools that compute a cdBGs or ccdBGs from input samples have been developed in recent years. While `BCALM` and `Cuttlefish` generate cdBGs (hence a *k*-mer set) that do not record the sample of origin, `Bifrost` and `ggcat` do build ccdBGs that use colors to trace the source of the *k*-mers. While ccdBGs are an implicit representation of an unitig matrix, as they contain the same information (unitigs and origin of *k*-mers) but represented in a different way, tools that build them do not produce a matrix as output nor provide any APIs or scripts to do so.
Recently, `kmtricks` [22], a very fast tool to build a *k*-mer abundance matrix from a set of samples has been proposed but the cardinality of the *k*-mer set obtained from input data renders these matrices poorly tractable for the aforementioned downstream applications. This is not a limitation of the tool but a feature of the *k*-mer spectrum of the datasets.

Remembering that unitigs, as described in section 2.2.1, are a more succinct representation for k-mers, we propose a pipeline that mix the strength of both cdBGs tools to build unitigs and `kmtricks` to represent k-mer color and abundance to produce unitig matrices that are more tractable for analysis. We also propose a simple pipeline to build presence absence unitig matrices from samples using a script that renders in a digestible text format the implicit representation of a ccdBG.

4.4.3 From sequencing data to unitig matrices

4.4.3.1 Main features and differences of unitig matrices

The construction of an abundance unitig matrix is based on two key observations: first, k-mer matrices can now be constructed with high efficiency, and second, unitigs can also be built efficiently. Therefore we can create a more compact and manageable representation that preserves the high-level information crucial for genomic variation diversity studies, loci variation visualization, and input for machine learning libraries.
The main parts of the process to generate an abundance unitig matrix from an abundance k-mer matrix involves compacting k-mers into unitigs and estimating unitig abundance by averaging k-mer counts. In this way we obtain a more condensed, lossy representation of the data. As single k-mer counts are lost, the essential information is preserved by the average count of the unitig. Given that k-mer counts typically exhibit minimal variation within a unitig, this representation achieves compaction into a computationally manageable form, albeit at the expense of a minor loss of information.
In cases where abundance information is not required, presence-absence unitig matrices can be used instead. They still capture sequence variation between individuals and are valuable for diversity studies. Figure 4.1 provides a schematic of the workflow used to generate the matrix.
A key difference between the proposed model for abundance or presence-absence matrix construction process is a filtering step that retains only k-mers reflecting differences between samples. This targeted approach focuses on the most informative genomic regions and is applied on the abundance matrix construction. The presence-absence matrix includes the entire set of k-mers from the input genomes, providing a comprehensive view of the input data.

4.4.3.2 Matrix construction overview

Formally, given an unitig u searched in a sample S, the k-mer presence ratio is defined as follow:

$$f(u, S) = \frac{\sum_{i=1}^{N} x_i}{N} \tag{4.1}$$

while the average abundance of a unitig u with respect to a sample S is defined as:

$$A(u, S) = \frac{\sum_{i=1}^{N} c_i}{N} \tag{4.2}$$

In the equations N is the number of k-mers in u, and x_i is a binary variable that is 1 when the i-th k-mer is present in sample S and 0 otherwise, while c_i is a non-negative integer count.
Figure 4.1 shows the main steps of the `muset` pipeline. To produce an abundance matrix, the main steps are:

1. A k-mer abundance matrix is built from FASTA/FASTQ files using `kmtricks`.

2. k-mers that are present in at least 10% of the samples and absent in at least 10% of them are retained, while the others are discarded. The thresholds are customizable.

3. Unitigs are created from this set of retained k-mers in order to compress the representation. Unitigs shorter than a certain value are discarded. While this variable can be modified by the user, our recommendation is to keep it as $2k-1$ with k the length of the k-mer. This value is the minimum value to observe a SNP in the set as a unitig representing a SNP would have 2 times $k-1$ bases as overlap to the unitigs representing the adjacent bases in the genome and 1 base for the variation.

4. The abundance unitig matrix is therefore constructed. This is done by

   a) creating a dictionary, using `ssHash`, to link k-mers to the unitig in which they have been compacted;

   b) each unitig abundance score is computed by summing the count of its constituent k-mer set divided by the cardinality of the set. This is done independently for each sample to retain the color information of the k-mer matrix.

To generate a presence-absence the main steps of the pipeline are:

1. the ccdBG is built using `ggcat`. Unitigs are in FASTA format while colors are stored in a compressed representation accessible only via `ggcat` CLI or APIs.

2. unitigs are filtered by length like in step 3 of the abundance pipeline;

3. filtered unitigs are queried against the `ggcat` color index and for each sample in which at least 1 k-mer of the unitig was present, the presence ratio is reported. If no k-mer was present in the sample, the sample is not reported.

4. the unitig query is then parsed (from JSONL) and the presence (1) or absence (0) is reported for every unitig in every sample in form of a matrix. The presence is determined when the fraction of present k-mers in the sample is above a pre-defined, although modifiable, threshold. It is also possible to produce a matrix that does report the presence ratio instead of a binary value. The JSONL (or JSON Lines) format is a text format for storing structured data to be processed one line at a time. When querying `ggcat`, the presence/absence information for each queried sequence is represented as one dictionary per line. This dictionary associates the queried sequence with each sample in the graph that contains at least 1 k-mer of the queried sequence. The ratio of k-mers present for each sample is paired with the sample name. In order to generate a matrix, `muset` parses each line of the JSONL file and generates a vector containing the ratio for present samples and 0 for non-present ones. Moreover, basic filtering of sequences by minimum ratio to be considered or minimum number of samples to be found to keep a queried sequence (as for the abundance matrix) can be applied to filter the matrix for specific use-cases.

Only the abundance pipeline has been tested against the most similar state-of-the art tool that is in fact `ggcat`, that produces, as mentioned, an implicit presence/absence matrix. Even without the `muset` script to produce an explicit one, the abundance matrix script is faster than `ggcat` when run on a large collection of 360 ancient oral samples, as shown in table 4.1.
It's important to note the differences in output formats and functionality between our tool and `ggcat`. `ggcat` utilizes a JSONL (JSON Lines) format, where each line represents the result of a sequence query. In its color query mode, it only outputs colors (representing samples) with more than one k-mer present in the sequence, providing the ratio between present k-mers and total k-mers in the sequence. While this output can serve as a useful input for developing new applications, it is not sufficiently comprehensive on its own for many analysis applications. Recognizing this limitation, we developed the additional `muset` script to transform `ggcat`'s output and produce a practical and widely applicable unitig presence/absence matrix, as described in the steps just outlined. This matrix provides a clear, binary representation of unitig occurrences across all samples, which is crucial for various downstream analyses in comparative genomics and pangenomics. The `kmat_tool` function that transforms the JSONL into a CSV with or the presence ratio for all samples or a binary 1/0 - presence/absence value (using a given threshold), is also useful for other applications that use `ggcat`. No computational resources test has been done on human genomes as it was out of the scope of the pure demonstration of the usability and efficiency of the method.

Method	Wall-clock time	Peak memory	Disk usage
`muset`	**9h 43m 12s**	**19 GB**	1.5 TB
`ggcat`	24h 20m 40s	167 GB	641 **GB**

Table 4.1: Comparison of running time, peak memory, and disk usage between `muset` (filtered unitig matrix) and `ggcat` (implicit matrix and unfiltered unitigs) on 360 ancient oral samples.

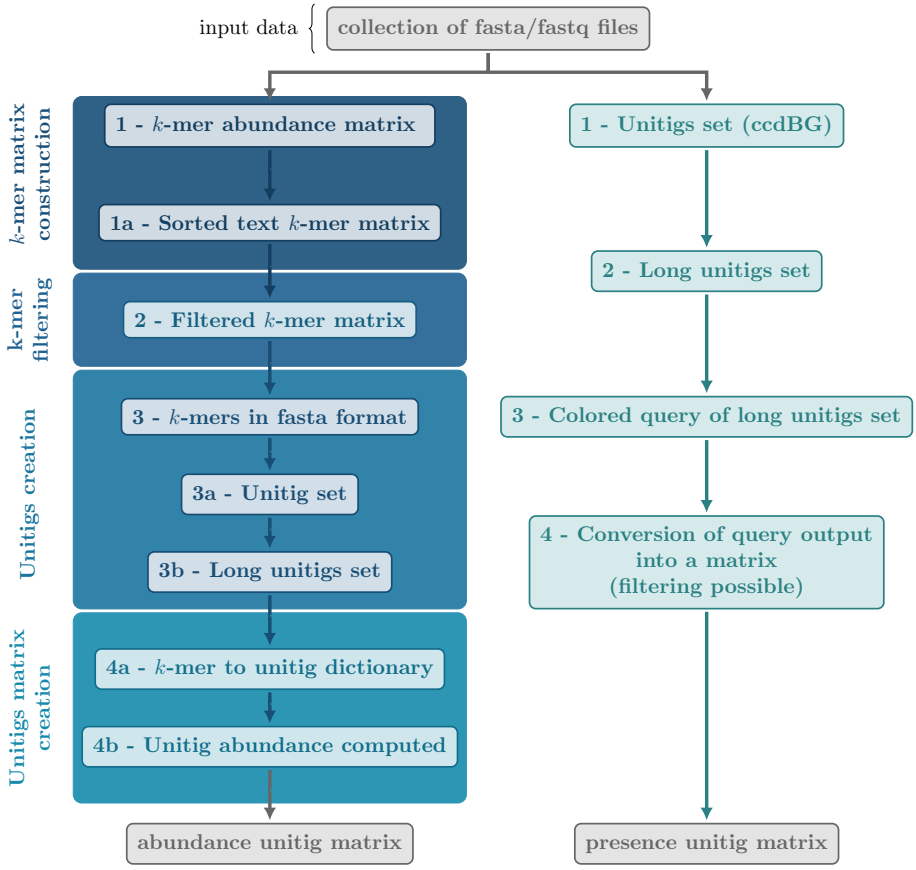


Figure 4.1: Scheme representing the main steps of the `muset` pipeline.

4.4.4 Conclusions and perspectives

Unitig matrices are pangenomes

Matrices, although often overlooked in the pangenome community, offer a valid and valuable representation of pangenomes. Each genome can be conceptualized as a binary vector within a matrix that encodes the alleles of a pangenome graph. Both variation graphs and ccdBGs can be used to infer presence-absence

matrices, where rows represent node IDs and columns represent input samples. While plain text matrices may not provide the same visual representation of genome variations as graphs do, they offer distinct advantages:

1. They are a more accessible starting point for downstream applications that dBGs;
2. bio-statistical methods and population genetics can be easily applied to this model;
3. provide a standardized format that can be compatible with various computational tools and libraries, providing a more reliable framework for researchers to build upon. This is in distinct contrast to color representation for ccdBGs that depend on the particular implementation.

The significance of `muset` lies in its novel approach to generate plain text unitig matrices. Prior to the development of the `muset` pipeline, abundance matrices could be theoretically generated only from paths in variation graphs. Our tool makes now possible to obtain a sufficiently precise and compact representation that enables new kind of analysis in pangenomics, transcriptomics, paleogenomics and metagenomics.

4.4.5 Improving the method

`muset` is the first pipeline that integrates various existing and novel tools to produce unitig matrix representations of pangenomes. The `kmat_tool` software has been specifically developed for the pipeline: it implements filtering, count averaging, presence-absence steps and handles the various input/outputs. Most of the tools incorporated in `muset` were instead not originally designed for this particular application.
While `muset`'s main contribution is in proposing for the first time a method to build such matrices, the current implementation has room for improvement and optimization. The pipeline involves multiple steps in which data is written and read to disk, not for a specific design choice to ease the burden from in-memory processing, but because of subsequent steps are done by different software. Other operations are also slowing down the pipeline. Two of these that are easy to spot are: building unitigs from a file containing one k-mer per sequence in the abundance pipeline and transforming the JSONL output of `ggcat` to a CSV instead of using `ggcat` API to get presence values to avoid two I/O operations and dumping directly to disk the CSV matrix in the presence-absence implementation.
Addressing these bottlenecks could yield a more optimized method to generate abundance unitig matrices that could scale to larger datasets and allow matrix-based analysis on them.

4.5 Prototyping Dynamic Data structures for *k*-mer counting: a Rank Select Quotient Filter

While abundance unitig matrices are valuable for specific genomics applications, they may not be the most efficient representation for all use-cases. In particular, some applications require rapid determination of presence or absence of specific sequences in datasets of interest. By indexing the dataset, the information is organized in data structures that enable fast data queries.
Sequence query in *k*-mer-based data structures is typically achieved through pseudo-alignment, a technique that involves tokenizing sequences into *k*-mers and querying each *k*-mer subsequently. Some data structures exploit the difference of just 1 character between each subsequent *k*-mer to organize the data so these kind of successive queries can be answered with less operations that a single *k*-mer query. While cdBGs or ccdBG can be used as a starting base for indexing, the graph construction is a major bottleneck and requires explicit association between each *k*-mer and its abundance.

4.5.1 Brief filter data structures overview

k-mer indexing data structures imply mainly two strategies to efficiently store and retrieve entries: representing *k*-mers as string or as hashes[16]. In this work we will describe a method that stores a set of *k*-mer as hashes and in particular uses structures with false positive (returning that an entry is present while it is not) rate greater than zero: this means that while false negatives (returning that an entry is not present when in fact it is). Hash-based data structures without false positives are static or dynamic hash tables [11].
Approximate Membership Query (AMQ) data structures, such as Bloom filters, quotient filters, and cuckoo filters, offer a more space-efficient representation of a set or multi set of elements 4.2.4.2. These data structures have become essential tools not only in computational biology but also in other domains like databases, storage systems and networks. They are termed approximate because they allow queries to return a controlled false positive value at a rate δ. This means that while they always confirm the presence of an inserted item, they might erroneously return true for non-inserted items with a probability of δ. This trade-off provides space savings.
Recent developments in the *k*-mer research community yielded improved version of data structures that allow to store the count of elements in a dataset, instead of just reporting its presence or absence [23, 24, 25, 26, 11]. AMQs that provide counting feature are termed counting AMQ or cAMQ. In the case of cAMQ, false positive values report a greater and not inferior count compared to the real ones: this avoids underestimating the abundance of an entry in the dataset. Counting AMQs are particularly useful in many bioinformatics applications, from pangenomics to transcriptomics and metagenomics.
At the present moment, the main areas of improvement of such data structures are:

1. latency of *memb()* operations. It determines the range of applications they can be used for (e.g., computer networks) and the volume of data that can be queried within a reasonable amount of time (e.g., bioinformatics). The latency is mainly influenced by:
    - data locality design. Developing an architecture that reduces cache misses by storing data that might be frequently accessed together in slots that fit within the largest process cache.
    - implementation efficiency, by carefully engineering the software for code branches, SIMD operations and multiprocessing. This is aspect is particularly important, as theoretically less efficient data structures can outperform others in practice due to optimization.
2. space efficiency, as it poses constraints on the computer architectures that can utilize it. Despite RAM cost declining, the rapid growth of data requires efficient and parsimonious encoding of data to the bit level.
3. supported features. The possibility to modify the data contained after initialization. The most useful operations are:
    - incrementing elements count, or add new ones;
    - decrementing elements count, or removing them when their count reaches zero;
    - enumerating elements with their respective count;
    - automatically resize the data structure when it reaches maximum capacity, avoiding the need for a priori cardinality estimation and allowing for addition of new elements.

A data structure that addresses efficiently all these aspects is the Counting Quotient Filter or CQF [25]. It is based on the Rank-Select quotient filter and it uses a counting scheme to efficiently encode the count of inserted elements in the slots of the filter [25]. This method offers reduced memory usage and faster lookup speed compared to inserting multiple times elements in a Quotient Filter to remember the counts.

4.5.1.1 Filter data structures

AMQs are often termed with the name of filter. Here I briefly describe the most common ones.

The **Bloom filter** is a well-known AMQ, which uses hash functions to map inserted items into a bit vector. Despite its space efficiency (one to two bytes per element for common δ values like 1/50 to 1/1000), a major drawback is it cannot be resized and doesn't support deletions. Counting Bloom filters (CBF) extend Bloom filters by using saturating counters instead of bits, enabling deletions at the cost of increased space. Scalable Bloom filters, on the other hand,

maintain a low false-positive rate even when the number of items is unknown by employing multiple Bloom filters.

The **Quotient filter** uses hashed fingerprints to manage table slots. It supports a range of operations like insertion, deletion, and resizing. It is more cache-efficient and faster than Bloom filters—though less space-efficient than CBFs—making it suitable for systems like SSDs. One downside is that its performance degrades significantly once the table exceeds 60% occupancy.

The **Cuckoo filter** uses cuckoo hashing. It uses two potential slots for storing each item and moves items between their alternate locations if needed, causing a cascade of movements (kicks) until a stable arrangement is achieved. This filter is fast for lookups but can suffer from poor cache performance if many kicks occur, especially when the structure becomes full.

All these implementations provide a fast lookup table where information can be stored in fixed-size slots.
I will focus on the implementation of the Quotient Filter, as it forms the basis of our implementation.

4.5.2 Quotient Filter structure: Rank and Select

The Rank-Select quotient filter, or RSQF, works by hashing items into a p-bit fingerprint x and then dividing the bits into two parts: the quotient $h0(x)$ of q bits and the remainder $h1(x)$ of $r = p - q$ bits. The RSQF has an array of 2^q r-bit slots that stores the remainder of each item. When inserting an item, the process begins by attempting to place the remainder in the slot determined by the quotient. If the home slot is occupied, a linear probing technique is used to locate the next available slot. To track and manage the runs (collections of subsequent slots containing remainders of a specific quotient), the RSQF uses two metadata bitvectors:

`occupieds` indicates for which quotients there is an entry stored in the filter slots;

`runends` indicates the end of each set of consecutive entries associated to the same quotient (also denoted as "run") in the filter.

The combination of these two bit vectors allows the RSQF to efficiently locate and manage inserted items by using rank and select operations. Specifically, the `RANK[x]` function on the `occupied` metadata bitvector counts the number of `occupied` slots up to a `x`, and `SELECT[y]` operation on the `runends` metadata bitvector identifies where in the filter the `y`-th run ends. The combination of these two functions allows efficient lookup, insertion, and enumeration of data in the filter.
Moreover, to store data in a cache-friendly manner, the filter is structured into blocks of 64 slots. To minimize operations requiring the scan of multiple blocks

when the filter is relatively full, an offsets array tracks the distance from the start of a run to where it ends, every 64 slot. Computing offsets involves scanning only small sections of the metadata (64 bits or fewer) per operation, making the filter significantly faster.
Each block fragments contains one offset value, 64 `occupieds` bits, 64 `runends` bits, and 64 slots for remainders data. By grouping these elements together, the system minimizes memory access and enhances cache efficiency. This structure is optimized for rapid traversal and manipulation of the filter content to enable fast membership queries and updates.
A scheme of how the quotient filter works is shown in figure 4.2. For the sake of simplicity the metadata vectors are not shown.

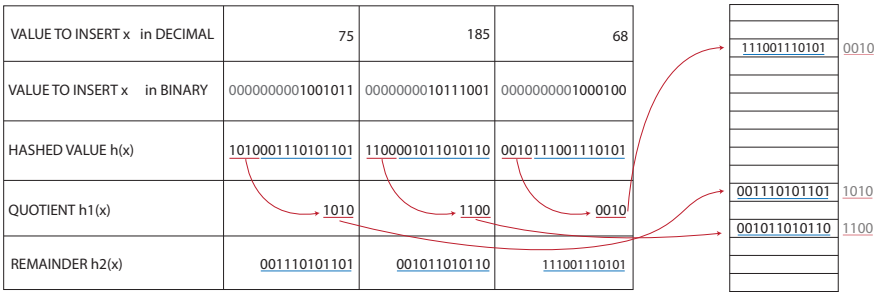


Figure 4.2: Example of a quotient filter. An entry is codified as an integer number, represented in decimal form, can be expressed as a binary sequence. An unsigned 16-bit integer is composed of 16 binary digits (bits) that can store a value from 0 to $2^{16} - 1 = 65535$. To construct a quotient filter, a hash value is generated by applying some binary operations on the entry integer. From the computed hash value, the leftmost 4 bits are considered the quotient (although, depending on the implementation, the rightmost bits could be used instead). The remaining bits from the hash form the remainder. The remainder is stored in the filter at the position indicated by the quotient.

Counting

Counting data structures can store either exact counts or order of magnitude, depending on the application. In fields like human genomics, where count numbers are expected to be relatively low and precision is crucial, exact counts are preferred. Differently, in fields like metagenomics, where skewed abundance is more probable, an estimate of the count is often sufficient.
Counts can be stored using three different strategies. In all cases, the remainders associated with a certain quotient are stored in monotonic order within the data structure.

1. a count can be encoded as the number of times the same remainder is inserted into consecutive slots. This is the most basic implementation and in most cases the least space-efficient one. The data structure occupation

is directly related to the total counts in it, making poor use of the bits in the slots.

2. a count N of a remainder R can be encoded into multiple consecutive slots as follows:

   - if $1 <= N <= 2$, then the remainder R is inserted N times;
   - else $K = N-2$ can be encoded into a sequence of slots. The boundaries are flagged by two remainders R. The slots in between store the count K. To signal that the slots are used as counter, the count elements break the monotonicity just after the first remainder.

   This approach uses at least 3 slots for $n > 2$ and, while more efficient than the first, is still inefficient for low count values.

3. another way of storing the counter value is reserving m extra bits for every slot to encode in it the count associated to the remainder. As this method adds $2^q * c$ bits the choice of c should be calibrated for the suited application.

4.5.3 Developing a new library for a Quotient Filter

The existing implementation of the Counting Quotient Filter (CQF), as described in [25] and available at `https://github.com/splatlab/cqf`, presents several limitations that restrict its versatility and applicability. First, the generated quotient filter is constructed as a static object, meaning that once created, its structure and content cannot be modified. This constrains the ability to dynamically adjust or update the filter after its initial creation, which is often the case in many applications. Second, the implementation is specifically focused on the CQF, which limits the potential for experimenting with and developing variations based on the RSQF or the CQF itself that can add new functionalities or improve performances in specific cases. Furthermore, the existing implementation does not explicitly address the concept of "toricity" in the filter. With the term toricity, I refer to the circular logic of the filter, where elements can be moved around from the *end* to the *beginning* of the data structure, if needed. This property is crucial to properly handle the insertion of hashes with highest quotient value and when the filter is filled near capacity.
To address these limitations, we have re-implemented the data structure with these features:

- it can be exact (when no space trade-off is chosen) or approximate in the k-mer space;
- it handles exact or approximate counts;
- it handles different count encoding.

The data structure is layered as follows:

Low Level comprises agnostic operations in the data structure such as

- setting a specific slot to a certain value. The value can be a remainder, count or combination of the two. It can also be any metadata as it only imposes bits to a certain region of memory corresponding to a slot;
- clearing of a slot to zeros for removals of elements from the filter and shifting operations;
- reading the value from specific slots;
- shifting slots by a certain amount z. Slots starting at position x to y are moved between positions $x + z$ and $y + z$ modulo the length of the filter. This operation is needed when a new element has to be inserted in an already occupied position. By shifting right of z slots the already set slots, the position x to $x + z - 1$ are therefore free to accommodate new elements or counts;
- metadata operations for setting to 1 or to 0 the `occupieds` and `runends` bits in their bitvectors;
- updating the offset vector to track runs.

Medium Level it comprises operations to insert, remove and query elements without explicitly handling bit-level operations. They implement a RSQF data structure.

- addition of an element to the data structure;
- removal of an element from the data structure;
- query of an element from the data structure;
- metadata handling in each of the aforementioned operations.

High Level it comprises operations built on top of the RSQF layer and describes a CQF.

- extended addition and removal to allow for multiple slots altogether;
- functions to encode and decode counters, as described in point 2 of section 4.5.2;
- query operations using specific linear probe, recognizing the start and end of a counter.

Application Specific functions specific for handling input, output and resize.

- k-mer hashing.
- initialization of the data structure;
- enumeration of elements within the data structure;

– dynamic resizing of the data structure, including element enumeration, filter size doubling, and element reinsertion.

This implementation has been developed as a library and as standalone software, offering a basic binary ready to be used and a toolset that can use the filter for a specific application. It is available at `https://github.com/frankandreace/cqf_implementation/`.

4.5.3.1 Allowing multiple types of counts: CQF and BQF

While the primary focus was on re-implementing a Counting Quotient Filter (CQF) from the Rank-Select Quotient Filter (RSQF) implementation, the basic structure with low to mid-level operations and application operations can be used to build other models on top of the RSQF. Victor Levallois has successfully implemented another data structure called the Backpack Quotient Filter (BQF) [27], which uses the count encoding presented in point 3 of the previous section on RSQF counting methods 4.5.2. Instead of using other slots to encode the counter, it reserves a pre-defined amount of bits in the remainder to store the approximate count: this is done at the expense of some precision.
The successful implementation of the BQF demonstrates the flexibility and extensibility of the proposed architecture in order to handle different kind of counts.

4.5.3.2 Handling toricity

A crucial property of the filter, overlooked in the original CQF paper [25], is the handling of specific edge cases related to its toroidal structure. Runs of remainders (or counts or combinations of both) associated with the same quotient are stored contiguously in monotonic order. This storage must be done in increasing slot ID order. When two or more elements with the same quotient are added to an empty filter, the slots used are the ones associated with the quotient and *on the right of it* (i.e., the slots associated with the next quotients), and so on.
A challenge arises in the part of the filter having the greatest quotient values. When new elements are added, it's likely that at some point, a slot should be pushed to the next element after the final slot. To address this issue, the filter is implemented with a toroidal structure, meaning the slot following the last slot (by ID number) is the first slot of the filter. This property eliminates problems associated with filling the filter in the final slots by imposing equal properties to all slots in the filter.
Implementing the filter with this characteristic is non-trivial, as the toroidal property requires different handling of all comparisons within functions and shifting operations. It necessitates careful consideration of edge cases and boundary conditions to ensure correct behavior when operations wrap around from the end to the beginning of the filter.
This toroidal implementation enhances the robustness and efficiency of the filter, allowing for more uniform utilization of the entire data structure and eliminating

potential bottlenecks that could occur at the boundaries of a linear structure. Figure 4.3 shows how the toroidal property works in a case of insertion where the final part of the remainder vector is already full.

4.5.3.3 Dynamic resizing strategy

This implementation of the filter eliminates the need for a priori computation of the *k*-mer set cardinality (and the slots used by the counts), allowing for direct data insertion. To prevent failure when the number of inserted elements exceeds capacity, a dynamic resize and reallocation strategy is implemented. A specific counter keeps track of the number of slots in use and it is incremented each time an entry is inserted in the filter.When the slot occupancy ratio reaches 95%, the insertion process is paused and the filter automatically doubles its capacity. This process operates as follows:

1. Enumeration: all stored hashes (with their counts) are enumerated. Hash values are temporarily stored in a hash table, with the element as the key and the count as the value. The enumeration is done by linear probing the entire filter. Each `occupied` metadata bit is sequentially scanned. When a bit is set to one, the probe looks for the corresponding run of remainders in the filters and proceeds to enumerating all the elements (composed of the combination of quotient and remainder) with their counts. When the entire `occupieds` bitvector is scanned, the enumeration ends.

2. Parameters update: the size of the quotient is increased of one and the size of the remainder is decreased of the same amount. By adding 1 bit to the quotient, the number of index-able and insert-able elements doubles. All other internal values depending on quotient or remainder are updated. The occupancy counter is set to 0.

3. Re-instantiation of the vectors: new remainder and metadata vectors are initialized with double size.

4. Insertion of elements in the new filter. The elements are fetched from the hash table and inserted back in the filter.

The dynamic resizing allows for flexible use of the data structure without prior knowledge of data size and ensures efficient space utilization by growing only when necessary. However, this process produces a temporary spike in memory, of $O(N)$ - with $N = 2^q$ slots, by having at the same time the map containing the enumeration and the newly doubled filter. Finally, this process is $O(n)$ in time, as all the elements have to be enumerated, inserted into the hash table and re-inserted in the new filter. As this operation happens 1 time each N insertions, its amortized cost for each insertion is proportional to a constant. Nevertheless, when the resize happens, the latency of the insertion operation will be extremely high. This is not a problem for bioinformatics applications.

4.5.3.4 Using the Fimpera scheme to reduce space

The size of a RSQF data structure is given by $2^q * (r + m)$ bits, where $m \sim 2.125$ metadata bits (along with some other relatively negligible overhead) for the metadata bitvectors. This suggests that if there were a way to store nearly the same input information while reducing r, it would yield substantial space savings and enable the use of the data structure on larger datasets. To achieve this goal, the CQF implementation has been developed also with the Fimpera scheme incorporated [28], like in the BQF.

Fimpera operates by splitting each k-mer into smaller s-mer and storing them in the filter, each with the k-mer count. The k-mer abundance can thus be retrieved through its constituent s-mers, as the presence of a k-mer implies the presence of all its constituent s-mers.
This approach has been demonstrated to significantly reduce the false-positive rate by an order of magnitude without generating false negatives or underestimating k-mer abundance [28]. In this context, Fimpera is used to reduce the dimension of the filter without increasing the false-positive rate, as storing smaller hashes from s-mers results in a smaller r.

To estimate the correct abundance of a k-mer, the smallest count among its constituent s-mers is used. The major drawback of this scheme is the loss of the k-mer enumeration feature, as only the s-mers can be retrieved. However, as the dynamic resizing of the data structure uses directly s-mers, this feature is preserved.

Finally, this approach may introduce false positive k-mers: k-mers that are non present in the original dataset but are composed of s-mers found in other k-mers are going to be reported as present. As this is a joint probability, it is the produce of each independent probability, becoming very low when s is sufficiently large. The s parameter must be chosen as a trade-off between space efficiency and false positive rates. For the BQF, this has been estimated as under 10^{-7} with $s > 20$ when $k = 32$.

This Fimpera-enhanced implementation offers a novel approach to improving the space efficiency of RSQF-based data structures while maintaining acceptable error rates and proves the flexibility of the proposed implementation.

4.5.4 Conclusions and perspectives

The proposed implementation of a prototype quotient filter has demonstrated considerable flexibility in its approach to storing k-mer abundances. This adaptability is a key feature that sets it apart from more rigid data structures and opens up new possibilities for genomic data analysis.
A possible extension of this project would be to incorporate additional metadata, which would extend its functionality beyond a conventional cAMQ. For instance, color information would be greatly useful for specific pangenomics applications. However, this extension is non-trivial due to the memory-intensive nature of color storage and encoding. To address this issue, an interesting direction could use of Shannon coding for colors. When the filter undergoes multiple resizes,

each resize operation can be used to evaluate the distribution of contained colors. The result of this evaluation could be used to inform an adaptive encoding strategy where frequently occurring colors are represented with fewer bits, while less common colors are encoded with into large values. Such an approach would optimize the space-color information trade-off dynamically.

Despite its new features, this implementation has not found its way to a standalone publication. This proposed CQF implementation (without Fimpera) exhibited slower performance compared to the original, more optimized, version. This performance disparity highlights the challenges providing implementations with new features without impacting computational efficiency. The negative results in terms of insertion and query speed on initial testing of the tool together with the positive results of the BQF implementation were the reasons we decided to not move forward with more extensive benchmarking of the tool. Code profiling showed bottlenecks related to the insertion of counts and the encoding and the coding of counts. Based on the same RSQF, the BQF uses less operations as it modifies only one slot at a time, compared to the multiple slots of the CQF. Nevertheless, there is value in prototypes that offer greater customization potential for specific applications, as they provide guidance for the evolution of more flexible and adaptable data structures.

Finally, the BQF implementation, based on this RSQF model, has been presented at RECOMB–seq [29] conference and will be soon published in the associated journal [27].

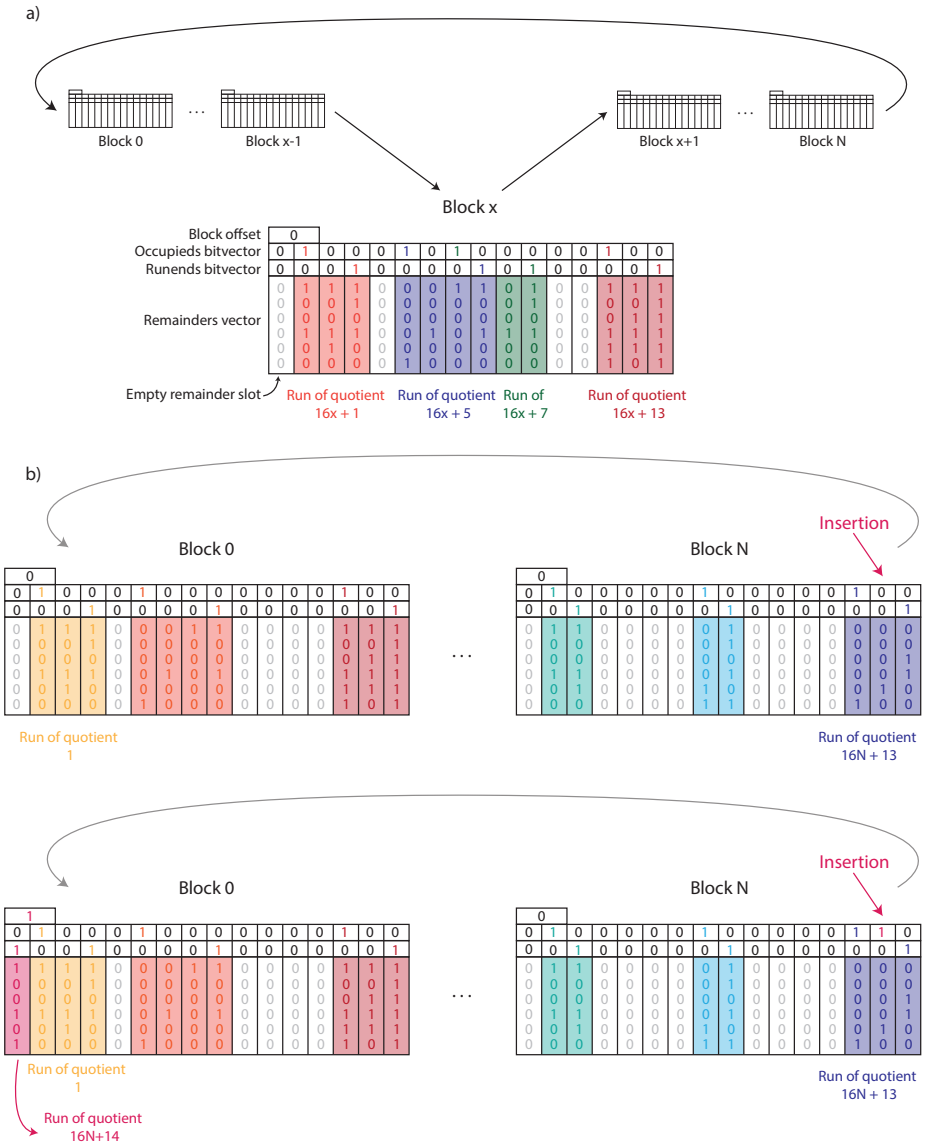


Figure 4.3: a) Scheme of the RSQF data structure. Along with the remainders vector, each block consists of the correspondent block of `occupieds` and `runends` bitvectors and one offset value. When elements associated to a quotient (here $16 * x + 7$) cannot be stored in the relative slot because it is already used by another run, they are placed into the next contiguous available slots.
b) Toroidal insertion. The remainder associated to quotient $16 * N + 14$ is inserted in the partially filled data structure. Since there are no free slots in the final part of the remainder vector, the next available free slot is at the beginning of the vector. The first slot of the vector, which was free, is then filled with the remainder. Metadata bits are then updated. The `occupied` bit of the quotient $16 * N + 14$ is set to 1. The `runend` at position 0 is then set to 1 and this updates the offset.

4.6 Prototyping Dynamic Data structures for *k*-mer counting: *virtual* super-*k*-mer sorted list

In this chapter, I have examined various examples of k-mer-based methods, demonstrating that there is no universal approach to efficiently represent k-mers. The most appropriate method is largely dependent on the specific application requirements. For instance, when comparing two datasets, one effective approach is to enumerate all k-mers for each set and store them in sorted lists. The difference between these sets can then be quantified using a set metric, such as the Jaccard index, which accounts for k-mer differences in the lists. This process is particularly efficient when working with sorted lists.
Sorted k-mer lists offer a reasonably efficient representation for individual k-mer queries without the need for additional indexing structures. Using binary search, the query time complexity is $O(logN)$, where N represents the cardinality of the set. This logarithmic time complexity ensures relatively fast query operations, especially for moderately sized sets.
While not indexing the data results in slower insertion and query time, using sorted lists offers the advantage of reduced space requirements. A map index achieves $O(1)$ time for each insertion, leading to $O(N)$ complexity when inserting N elements. In contrast, sorting a list, assuming random data without additional hypotheses, requires $O(N \log N)$ time when comparisons are performed. Utilizing a list instead of a more complex data structure, such as the RSQF, provides a more compact representation of the data in memory. The list has no overhead as it does not use empty slots, metadata (`occupieds`, `runends` or offsets) vectors, or additional variables. The size of the data structure is solely determined by the size of the raw data; no external bits of indexing information are added, even though information is gained compared to the input data through the sorting process.

To achieve increased space saving, instead of storing individual k-mers in a list, an encoding can be employed to group overlapping k-mer together. The k-mer research community has explored several loss-less compression techniques by encoding k-mers within string sets, also known as SPSS or Spectrum Preserving String Sets. The fundamental idea is to construct string sets that contain all enumerated k-mers and nothing else. Recent models include `eulertigs`, `simplitigs`, and `super-k-mers` [16]. Our proposed data structure positions in between an SPSS that achieves maximum compaction of k-mers and the logarithmic queries of a sorted k-mer list by compacting k-mers into a *virtual* super-k-mer list that allows logarithmic queries.
The standard definition of super-k-mer is the compaction of successive k-mers present in an input sequence if they share the same minimizer in the sequence. The minimizer is an s-mer, with $s < k$, chosen among all the s-mers in a k-mer, that minimizes a specific function (e.g. lexicographic or minimal hash value). This definition of super-k-mer, that from now on I am going to term as *classical* super-k-mer, requires co-localization of the compacted k-mers inside the input sequence, as they need to share the exact same minimizer (not only the same by

sequence content but also the same by position in the input sequence).
What we instead propose in our method is a *variation* of super-*k*-mer, that I am gonna term from now on as *virtual* super-*k*-mer, whose definition drops the co-localization requirement of the *classical* super-*k*-mer. By doing so, *k*-mers can be compacted into a *virtual* super-*k*-mer if they share the same minimizer by sequence content, independently from where in the input they do come from. We therefore propose a novel data structure that combines the compaction of *k*-mers into *virtual* super-*k*-mers with a sorted list structure for relatively quick non-indexed *k*-mer queries while compacting the *k*-mers. We named this data structure *virtual* super-*k*-mers sorted list and it produces a *k*-mer set representation. Figure 4.4 shows an example of *classical* and *virtual* super-*k*-mers.

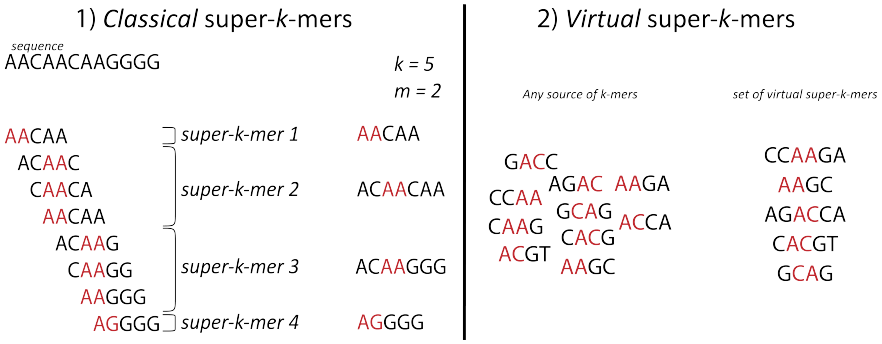


Figure 4.4: *Classical* super-*k*-mer and *virtual* super-*k*-mer definition. 1) *Classical* super-*k*-mers are generated from an input sequence. *k*-mers that share the same minimizer in the input sequence (co-localized) are compacted into the super-*k*-mer. 2) *Virtual* super-*k*-mers are generated by compacting *k*-mers that share the same minimizer, independently of their origin (sequence, set of *k*-mers or any SPSS). Example with *k*-mer length of 5 and minimizer length of 2.

Virtual super-*k*-mers generate a partition of the *k*-mer set, allowing for space savings and parallelization of operations in the set. Although they are a string representation, *virtual* super-*k*-mers are based on hashed minimizers. An advantage of using *virtual* super-*k*-mers in the sorted list is that consecutive *k*-mers, being close to each other, can be queried together, resulting in significant computational time savings. To perform a query, the algorithm locates the *virtual* super-*k*-mer linked to the queried *k*-mer's minimizer and checks for the *k*-mer within that *virtual* super-*k*-mer.
Implementations like `BLight` [30] and `ssHash` [11] use Minimal Perfect Hash Functions (MPHF) to facilitate mapping of minimizers to their corresponding *classical* super-*k*-mers. However, the use of MPHF generates a static dictionary that cannot be updated [31], forcing the data structure to remain static. By employing sorted *virtual* super-*k*-mers lists instead, we can provide dynamic updates, enhancing the flexibility and adaptability of the data structure. This is also due to the relaxation of constraints in the *virtual* super-*k*-mer definition.
The next sections are going to explain in more detail the input/output and

the construction steps of the method we use to generate and query the *virtual* super-k-mer sorted list.
The super-k-mer sorting algorithm we propose is a component of a more comprehensive super-k-mer-based data structure to which I contributed. For the sake of brevity, I will not delve into the full details of this structure nor the way we represent or compare *virtual* super-k-mers under the hood.

4.6.1 Input and output

INPUT Our algorithm takes as input any source of k-mers: sequences, super-k-mers, unitigs, k-mers. The input ingestion produces an enumeration of *virtual* super-k-mers that is used fed to the remaining part of the algorithm.

OUTPUT The method produces a list of *virtual* super-k-mer in which k-mers having the minimizer at the same position are sorted following a given ordering. This is the property that allows efficient lookup of k-mers in the list.

While this method can enumerate the sorted super-k-mer in text format, all the operations, as well as disk storage of the list, are done in binary space after 2-bits encoding the nucleotides. This encoding approach offers dual benefits: it enhances space efficiency for k-mer representation in memory and facilitates fast machine operations for data comparison and modification.

4.6.2 *virtual* super-*k*-mer list and *k*-mer matrix models

To better grasp the rationale of the sorting algorithm steps, it is useful to notice that the *virtual* super-k-mer list enumerated from the input is a list of objects (the *virtual* super-k-mers) that can be partitioned into k-mer by the minimizer position in each k-mer. This defines a k-mer matrix characterized by two key parameters: N, the number of enumerated *virtual* super-k-mers; M, the maximum number of k-mers that can be compacted into a single *virtual* super-k-mer, where $M = k - m + 1$ (k = k-mer length, m = minimizer length). In this matrix model, rows represent distinct *virtual* super-k-mers, while columns identify specific positions within each *virtual* super-k-mer, corresponding to the k-mer inside each *virtual* super-k-mer that contains a minimizer at a defined position: a column of the matrix corresponds to the position of a k-mer within its *virtual* super-k-mer. `Position 0` represents the k-mer in the first possible position of the *virtual* super-k-mer. This position also corresponds to the `suffix size` of the k-mer, where the `suffix` is defined as the distance between the end of the minimizer and the end of the k-mer.
All the steps of the sorting algorithm can be conceptualized as operations on or between the matrix columns, i.e. by considering all the k-mers that have the minimizer at the same position.
In the next sections I am going to use the matrix representation to provide a clearer and simplified explanation of the algorithmic operations. Figure 4.5

shows an example of ingested *virtual* super-*k*-mers and their matrix model representation.

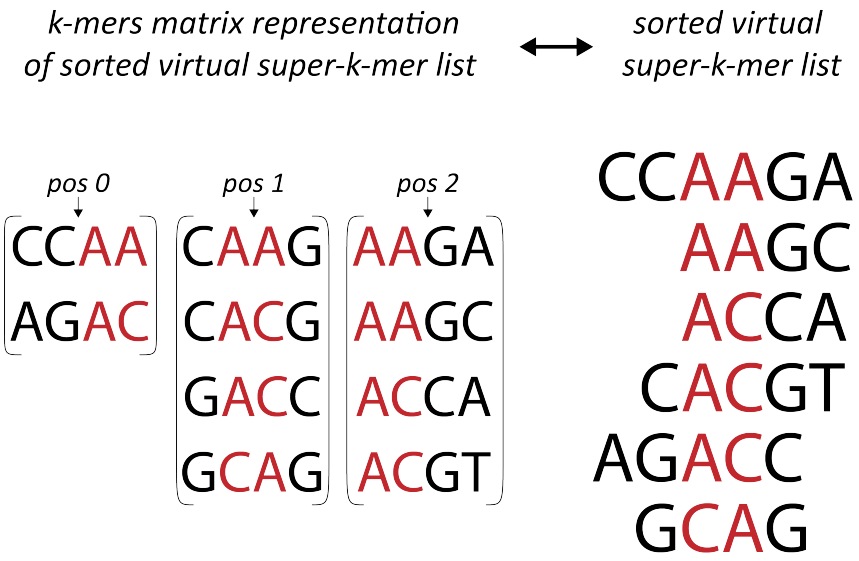


Figure 4.5: The matrix model for a *virtual* super-*k*-mer list. When *k*-mers are ingested in input as an enumeration of *virtual* super-*k*-mer, *k*-mers can be considered as organized in a matrix model. Each row corresponds to an enumerated *virtual* super-*k*-mer and each column corresponds to a *k*-mer inside a *virtual* super-*k*-mer. While rows partition the *k*-mers by input ordering, columns partition *k*-mers by minimizer position (and therefore position of the *k*-mer in the *virtual* super-*k*-mer).

4.6.3 The $\log N$ query model

Unlike a traditional *k*-mer list, the *k*-mers within a *virtual* super-*k*-mer sorted list are not fully ordered. The sorted list has M ordered columns: all *k*-mers part of the same column are sorted by the chosen ordering. This means that a *virtual* super-*k*-mer `A` that is placed before a *virtual* super-*k*-mer `B` in the sorted list will have all the *k*-mers at the same minimizer position with *inferior* ordering value compared to the ones of `B`: for every column a *k*-mer from `A` is going to be smaller than the one in `B`. This does not mean that all the *k*-mers of `A` are smaller than the ones of `B`: only columns preserve the order.
A shown in figure 4.6, this property allows us to transform the classic binary search algorithm as follows:

1. Identify the minimizer position p within the *k*-mer to be queried.

2. Navigate to column p of the matrix, focusing only on the k-mers present in that specific column.

3. Perform a binary search for the target k-mer within the selected column.

Step 1, while potentially requiring up to $k - m + 1$ operations, becomes more efficient when searching for successive k-mers as the cost is amortized over multiple queries. Step 3 maintains the logarithmic time complexity of a binary search, $O(\log N)$, where N is the number of *virtual* super-k-mers. However, in practice, this step tends to be faster than a traditional binary search on a full k-mer list. This improved efficiency is due to the distribution of k-mers among all columns, effectively reducing the search space for each query.

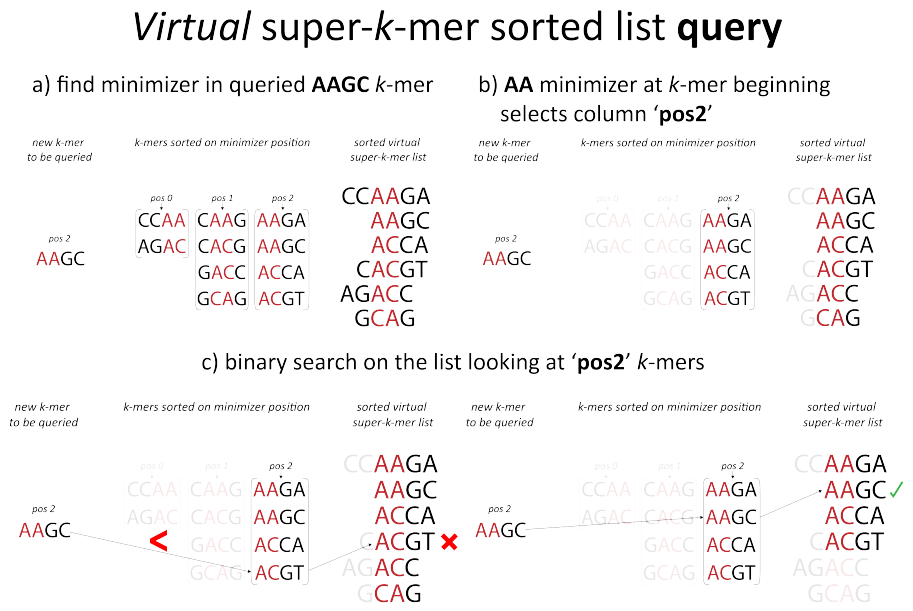


Figure 4.6: Querying the *virtual* super-k-mer sorted list. a)The AAGC k-mer is queried. The first step is to localize the minimizer (in this case AA is at the beginning of the k-mer). b) As the minimizer in AAGC is at the beginning of the k-mer, the query will look only at the last k-mer in each *virtual* super-k-mer in the list (the last k-mer in a *virtual* super-k-mer has the minimizer at the beginning). c) By applying binary search while looking only at the last k-mers of the *virtual* super-k-mers AAGC is found.

4.6.4 1 - Sorting *k*-mers from the same matrix column

As mentioned in section 4.6.2, the operations of the sorting algorithm are performed on or between columns of the k-mer matrix. The initial step involves a column-wise bucketing of the input k-mer source, based on the minimizer

position within each k-mer. This process distributes k-mers across M columns, with M defined as in section 4.6.2. Then, each column is independently sorted based on a minimizer-centric ordering. Once all columns have been individually sorted, the resulting column-sorted matrix is returned as the final output of this phase. The sorting process of each column is completely independent and can be performed in parallel. The result is shown in figure 4.7.

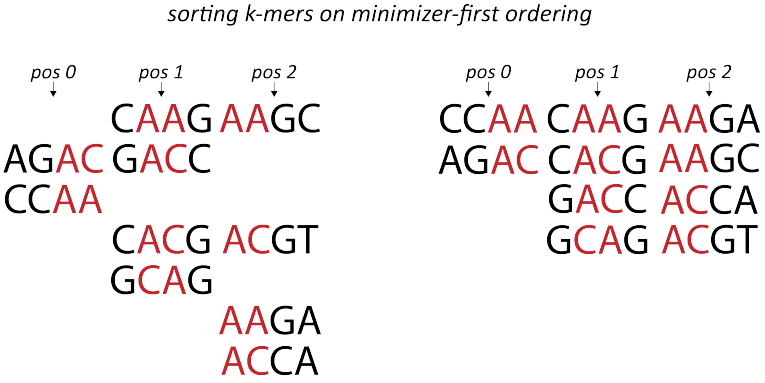


Figure 4.7: On the left side: k-mers are divided on different lists as per position of their minimizer in them. On the right side: Each list is independently sorted based on a minimizer-first ordering.

4.6.5 2 - Construction of overlap lists between consecutive columns

In the next step we detect overlaps between k-mers present in consecutive columns: for all k-mers in column j we find overlaps with the ones in column $j-1$ and $j+1$.
For each pair of adjacent columns, we aim to identify where the $k-1$ suffix of a k-mer in the first column overlaps with the $k-1$ prefix of a k-mer in the subsequent column, as shown in figure 4.8. This process is performed independently for each pair of consecutive columns and can be divided into several steps. First, the algorithm inserts the $k-1$ prefixes of the k-mers from the second column into a hash table, using the prefix as the key and the *virtual* super-k-mer ID as the value. Then for each $k-1$ suffix computed from the first column, it searches for a matching entry in the hash table. When a match is found, the pair of matrix coordinates (one from each column) is inserted into a list of candidate overlaps. A single element in one column may overlap with multiple elements in the adjacent column: in this case all possible overlaps are recorded.

This step identifies overlaps between *k*-mers. The overlap, by column construction, implies that the two *k*-mers share the same minimizer. This step therefore finds all the possible ways to compact the *k*-mers into *virtual* super-*k*-mers, without considering the column ordering.
The independent logic of these operations for each pair of consecutive columns allows for effective parallelization.

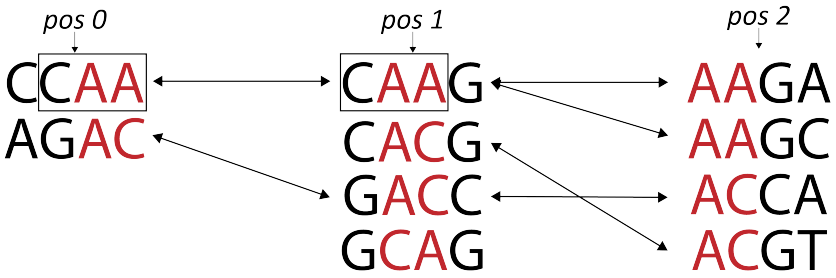


Figure 4.8: Overlap detection between *k*-mers of contiguous columns. For each pair of contiguous columns we detect the *k-1* overlap between *k*-mers. The *k-1* overlap between `CCAA` of column *pos 0* and `CAAG` of *pos 1* is highlighted by a rectangular overlay

4.6.6 3 - Maximal set of overlapping *k*-mers: co-linear chaining

Step 1 ordered the *k*-mers of each column and step 2 computed all possible *k*-mer overlap pairs for compaction. We have therefore all the information we need to produce a *virtual* super-*k*-mer sorted list. However, to produce a list that abides the ordering of *k*-mers on columns, as presented in section 4.6.2, we must compact *k*-mers using a subset of overlaps that respect the ordering of the *k*-mers in the columns. Some overlaps may be incompatible between each others due to two possible conditions:

a a *k*-mer overlaps with multiple *k*-mers in the adjacent column, but can be used just once. The *virtual* super-*k*-mer sorted list represents a *k*-mer sets: any *k*-mer can be present only once in the list.

b two overlaps are incompatible if, by using both of them to compact the *k*-mers into *virtual* super-*k*-mer, they result in a violation of the column ordering in the final sorted list. Given `A`, `B`, `C` and `D` 4 *k*-mers with `A` and `B`

from (sorted) column j, in this order, `C` and `D` from (sorted) column $j+1$, in this order. `A` is overlapping with `D` and `B` is overlapping with `C`. If we use the 2 overlaps to construct 2 *virtual* super-k-mers, we will produce a list [`AD`, `BC`] or a list [`BC`, `AD`]. In the first list `D` appears before `C`, which violates the order of the k-mers from column $j+1$, in the second list `B` is before `A`, which violates the order of column j. For this reason we can use at most 1 of these two overlaps when constructing the final *virtual* super-k-mer sorted list.

In figure 4.9, point c) shows an example of overlap pairs, with some incompatible ones and a maximal set of overlaps (signaled by green edges). It is visually simple to identify incompatible pairs as they cross each other. To select the maximal set of "non-crossing" pairs from those calculated in step 2, we use the classical co-linear chaining algorithm.
Co-linear chaining is a dynamic-programming technique commonly used in alignment algorithms. In the context of pairing shared subsequences between two sequences, it can be summarized as finding pairs of connected elements such that no "crossing" connections occur when visualized.
In read-to-reference sequence alignment, co-linear chaining takes pairs of maximal exact matches (MEMs) as input. It then computes a chain of pairs where the selected pairs' order aligns with their appearance in both strings, while maximizing the number of bases covered by the chain in the read. Recently, this technique has also been applied to sequence-to-graph alignment in pangenomics applications.
In the context of this method, co-linear chaining algorithm is used to select the maximal number of k-mer overlaps in the matrix such that they respect the ordering of the columns and do not fall into the scenarios a) and b) described above. This ensures a consistent ordering on columns while maximizing the number of valid overlaps that can be used to compact k-mers into *virtual* super-k-mers. This step is crucial to produce a coherent and efficiently searchable structure.

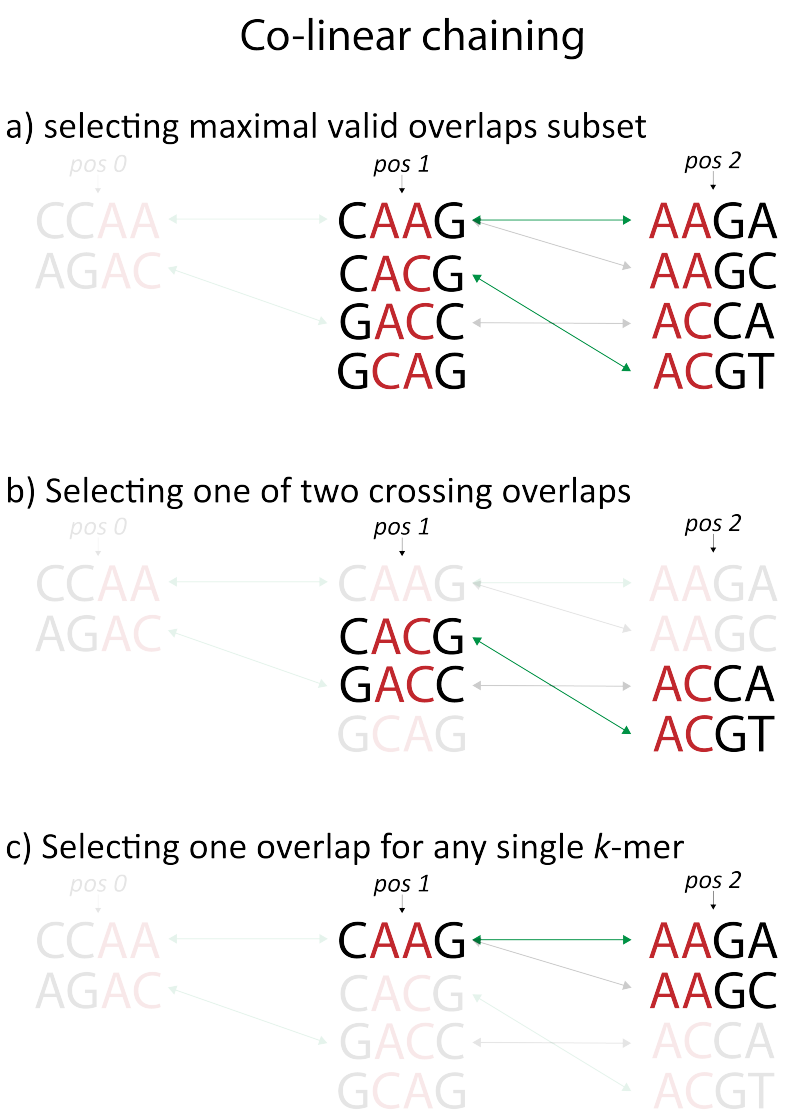


Figure 4.9: Maximal valid overlap set using co-linear chaining. a) example of selection of the maximal valid overlap set between column *pos 1* and *pos 2*: by choosing two out of four overlaps rule a and b described above are maintained when compacting k-mers into super-k-mers. b) Focus on crossing overlaps: as by described in property b above, there cannot be crossing overlaps when compacting k-mers to preserve the column ordering. c) Focus on k-mer set property: as described in a above, each k-mer can be compacted into a super-k-mer just one. For this reason only one of the two overlaps can be used.

Definition 4.6.1 (Co-linear chaining of k-mers in the matrix)**.** Given

- two ordered lists of *virtual* super-k-mer ids of two contiguous columns computed as in section 4.6.4.
  - $A = \{a_1, a_2, ..., a_q\}, |A| = Q$, representing the left column,
  - $B = \{b_1, b_2, ..., b_t\}, |B| = T$, representing the right column;
- a set of tuples $W = \{(v_1, w_1), (v_2, w_2), ..., (v_z, w_z), |W| = Z\}$ such as $v_i \in A$ and $w_i \in B$ and $u_i = (v_i, w_i)$ represents an overlap between k-mers of two contiguous columns `A` and `B`, as computed in point 2, section 4.6.5;

As A and B are sorted, the position of an element in the list gives its relative value: $a_1 \prec a_2 in A \implies a_1 < a_2$. The $\prec$ symbol is define as preceding relative to the position in the sorted list, hence in the ordering of the elements. The objective is to find the maximal list of tuples from U, with $U \subseteq W$ such that

- if $u_i = (v_i, w_i) \prec u_j = (v_j, w_j)$ in $U \implies v_i = a_x \prec v_j = a_y$ in A and $w_i = b_h \prec w_j = b_k$ in B;
- $\forall u_i = (v_i, w_i) \implies \nexists u_j (v_j, w_j) \neq u_i | v_j = v_i \wedge \nexists u_k = (v_k, w_k) \neq u_i | b_k = b_i$.

The first condition implies that the ordering of the column lists A, B is respected in U, while the second implies that each *virtual* super-k-mer ID from A or B can occur only once in the list of tuples U.

The maximal set of overlaps U is computed using the dynamic programming implementation of co-linear chaining [32]. The co-linear chaining process begins by sorting the tuples in $U = \{(v_1, w_1), (v_2, w_2), ..., (v_k, w_k)\}$ based on their order in B. This ensures that no "crossing" occurs in the w ids, as $w_i = b_i : b_i \prec b_j$ will always be processed before $w_j = b_j$. To resolve ties on equal v values, the ordering of $v_i = a_i$ in A is used.
Next, we apply dynamic programming over the A ids to identify the largest set of pairs where the A ids form a non-decreasing sequence. The score is stored for each element v_i in the list of pair. A table C of length M is filled, where index j represents the maximum possible score using tuples from U such that (v, w) has $w \in \{b_1, ..., b_j\}$. For any tuple, we derive a recurrence based on whether it violates or not the aforementioned conditions. The recurrence is calculate as follows:

$$C[j] = \max_{j' : w_{j'} \prec w_j} C[j'] + 1 \tag{4.3}$$

if (v_j, w_j) does not overlap with $U_{j'} \subseteq \{(v_1, w_1), ..., (v_{j-1}, w_{j-1})\}$ which is the non-overlapping subset selected in the previous step.
To optimize the search for the best solutions j' among those already computed, we use a binary search tree. The contains the best score for each value $v \in A_{j-1} \subset A$, as it contains the values up to the previously computed position. As they are maintained sorted based on A ordering, it allows for $O(\log n)$ complexity when querying for the best score. When new scores are computed, the tree can also be updated in $O(\log n)$ time. From this follows that the co-linear chaining costs $O(n \log n)$.

4.6.7 4 - Reconciliation and final output

After computing the maximal lists of valid "non-crossing" overlaps between each pair of columns, the final step is to reconcile the segmented information in each column to compact k-mers into *virtual* super-k-mers using the computed overlaps and fill them in the list using the ordering of the k-mers in each column. The reconciliation process can be divided into two main parts:

Map creation Using the lists of valid non-crossing overlaps, each k-mer in the matrix is added to a map as a key, with its value being the ID of the *virtual* super-k-mer it will be inserted into. When one overlap list returns the pair (x, y) and the subsequent list returns (y, z), k-mers x, y, and z will be compacted into the same *virtual* super-k-mer in the sorted list. Consequently, they are inserted as keys into the map with the same value (e.g., 1). Other *virtual* super-k-mers not to be compacted in the same *virtual* super-k-mer will have different values.

List insertion Each column of the matrix is assigned a pointer that tracks the row containing the next k-mer to be inserted. A k-mer or a *virtual* super-k-mer is inserted into the final sorted list by iterating over these pointers. This part of the process is organized as follows:

a) if a k-mer or *virtual* super-k-mer has all its elements flagged by a pointer, it is inserted into the list;

b) ties are resolved by directly comparing the two *virtual* super-k-mers and inserting the one with the smallest hash first;

c) after an element is inserted, the pointers on top of its k-mers are moved to the next row;

d) this process is repeated by going back to a) until all pointers reach the last element of their column.

By using this approach, we guarantee that the final list is sorted. Part 2 of the process can be done in parallel from the "top" and the "bottom" of the matrix to reduce computation time. Figure 4.10 shows an example of the reconciliation process.

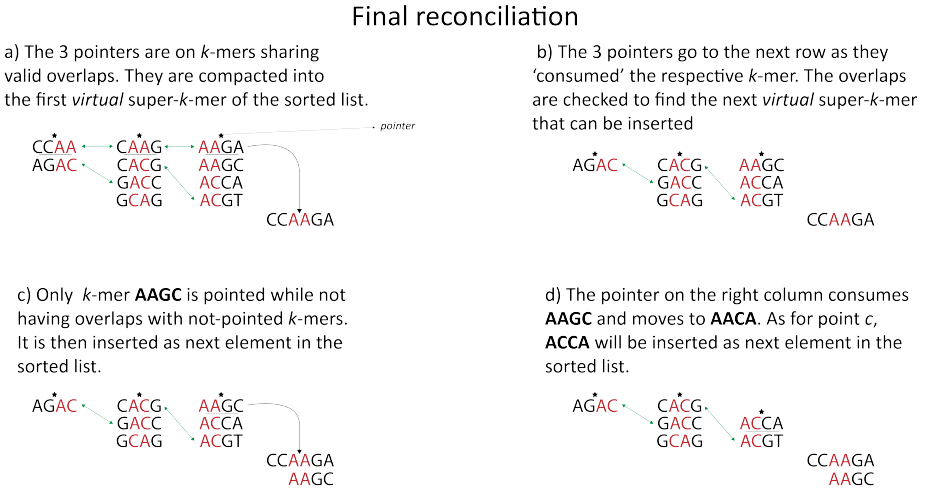


Figure 4.10: Reconciliation of *k*-mers into *virtual* super-*k*-mers. a) Each column has a pointer that selects the next *k*-mer that can be compacted. If all *k*-mers that share an overlap are pointed in their respective column, they are compacted and inserted in the list as *virtual* super-*k*-mers. This procedure ensures that the ordering of the column is maintained. b) After a *k*-mer from a column has been *consumed* by the compaction and inserted in the list, the pointer goes to the next row. After this is done for all columns, the valid overlaps are checked to determine which *k*-mers can be compacted as the next *virtual* super-*k*-mer. c) Only if all *k*-mers with valid overlaps between each other have a pointer on top they can be compacted. In this case only `AAGC` satisfies the condition (no non-pointed overlaps as it is without overlaps). d) In the next step of the reconciliation, like in point *b*, the pointer of the right column goes on `ACCA`, that will be the next element to be inserted.

4.6.8 Conclusion and future work

This project has resulted in a prototype data structure that represents a *k*-mer set as a sorted list of *virtual* super-*k*-mers. The advantages of this representation include:

- unlike some hashing data structures such as Bloom filters, our structure enables the enumeration of *k*-mers within the set;
- by compressing *k*-mers into *virtual* super-*k*-mers, we achieve reduced storage space compared to an ordered list of individual *k*-mers;
- the structure facilitates relatively efficient direct queries without the necessity for indexing;

- it allows for the addition of metadata through a separate, complementary data structure.

While this work is ongoing, our objective is to produce a paper shortly after the defense as a novel approach to organizing dynamic datasets using k-mers. As the method is not fully completed and tested at the time of the writing of this manuscript, we also lack of benchmarking to show the computational performances of such a method.
Furthermore, this prototype could serve as a pre-processing step for storing k-mers in a RSQF. We could use it to produce a sorted and compact k-mer enumeration for direct insertion into the filter. This approach sheds light on the gap between space-efficient representations of k-mers as string sets and query-able structures, offering a balance of compression and functionality. The potential applications extend storage, as indicated by its possible use in RSQF pre-processing. As we continue to work on this prototype, we aim to use it as valuable building block for more complex applications in large-scale genomic data.

4.7 *k*-mer based method exploration: conclusions

In this chapter, I have presented three distinct yet interconnected projects, each utilizing *k*-mer data structures to enhance genomic analysis, with a particular focus on features that can best fit pangenomics applications. These efforts also span different levels of the method development stack.

The `muset` project, whose novel contribution is a pipeline for plain text unitig matrices generation, has had as my main contribution the development of the pipeline and the function of the novel accompanying software, `kmat_tool`, that produces the presence-absence matrix using `ggcat`. My involvement in this project stems from the conviction that there is a substantial need for more downstream-focused *k*-mer tools, as they currently lag behind variation graph-based approaches for genomics analyses. Furthermore, I consider the effort to propose standardized, text-based file formats crucial in helping the community build software upon stable foundations.

The development of the RSQF-based library and tool, while not revolutionary in concept, represents a significant addition to the bioinformatics community. This prototype offers new features such as enumeration and dynamic updates, provides a clearer description and resolution of issues like toricity, and serves as a reproducible and reusable platform for building other prototypes or tools, as exemplified by the BQF.

The super-*k*-mer sorting project is a novel contribution, developed in collaboration with my supervisors. This prototype also uses a novel approach to encoding super-*k*-mers as an alternative development direction for data structures representing *k*-mer sets, specifically optimized for particular applications.

`muset` is an easy recognizable example of my effort in contributing to the development of *k*-mer based data structures. The RSQF and super-*k*-mer sorting are not directly linked specifically to pangenomics, as they can serve more general purposes. Nevertheless they could be used as building blocks of more comprehensive data structures to store, process and analyze pangenomes.

By contributing to these different projects, I have gained a deeper understanding of the strengths and challenges of *k*-mer based methods. I find these experiences invaluable, as they have provided me with a comprehensive view of the field.

Bibliography

[1] Marco-Sola, S., Moure, J. C., Moreto, M., and Espinosa, A. "Fast gap-affine pairwise alignment using the wavefront algorithm". In: *Bioinformatics* vol. 37, no. 4 (Sept. 2020), pp. 456–463. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btaa777. eprint: https://academic.oup.com/bioinformatics/article-pdf/37/4/456/39629847/btaa777.pdf. URL: https://doi.org/10.1093/bioinformatics/btaa777.

[2] Ahmed, O. Y., Rossi, M., Gagie, T., Boucher, C., and Langmead, B. "SPUMONI 2: improved classification using a pangenome index of minimizer digests". In: *Genome Biology* vol. 24, no. 1 (May 2023), p. 122. ISSN: 1474-760X. DOI: 10.1186/s13059-023-02958-1. URL: https://doi.org/10.1186/s13059-023-02958-1.

[3] Zakeri, M., Brown, N. K., Ahmed, O. Y., Gagie, T., and Langmead, B. "Movi: a fast and cache-efficient full-text pangenome index". In: *bioRxiv* (2024). DOI: 10.1101/2023.11.04.565615. URL: https://www.biorxiv.org/content/early/2024/02/15/2023.11.04.565615.

[4] Edgar, R. C. et al. "Petabase-scale sequence alignment catalyses viral discovery". In: *Nature* vol. 602, no. 7895 (Feb. 2022), pp. 142–147. ISSN: 1476-4687. DOI: 10.1038/s41586-021-04332-2. URL: https://doi.org/10.1038/s41586-021-04332-2.

[5] Chikhi, R., Raffestin, B., Korobeynikov, A., Edgar, R., and Babaian, A. "Logan: Planetary-Scale Genome Assembly Surveys Life's Diversity". In: *bioRxiv* (2024). DOI: 10.1101/2024.07.30.605881. eprint: https://www.biorxiv.org/content/early/2024/07/31/2024.07.30.605881.full.pdf. URL: https://www.biorxiv.org/content/early/2024/07/31/2024.07.30.605881.

[6] Pibiri, G. E., Shibuya, Y., and Limasset, A. "Locality-preserving minimal perfect hashing of k-mers". In: *Bioinformatics* vol. 39 (June 2023), pp. i534–i543. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btad219. URL: https://doi.org/10.1093/bioinformatics/btad219.

[7] Cracco, A. and Tomescu, A. I. "Extremely fast construction and querying of compacted and colored de Bruijn graphs with GGCAT". In: *Genome Research* vol. 33, no. 7 (2023), pp. 1198–1207.

[8] Xue, H., Gallopin, M., Marchet, C., Nguyen, H. N., Wang, Y., Laine, A., Bessiere, C., and Gautheret, D. "KaMRaT: a C++ toolkit for k-mer count matrix dimension reduction". In: *Bioinformatics* vol. 40, no. 3 (Mar. 2024), btae090. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btae090. URL: https://doi.org/10.1093/bioinformatics/btae090.

[9] Bessiere, C. et al. “Transipedia.org: k-mer-based exploration of large RNA sequencing datasets and application to cancer data”. In: *Genome Biology* vol. 25, no. 1 (Oct. 2024), p. 266. ISSN: 1474-760X. DOI: 10.1186/s13059-024-03413-5.

[10] Pibiri, G. E., Fan, J., and Patro, R. “Meta-colored Compacted de Bruijn Graphs”. In: *Research in Computational Molecular Biology*. Springer Nature Switzerland, 2024.

[11] Pibiri, G. E. “On weighted k-mer dictionaries”. In: *Algorithms for Molecular Biology* vol. 18, no. 1 (June 2023), p. 3. ISSN: 1748-7188. DOI: 10.1186/s13015-023-00226-2. URL: https://doi.org/10.1186/s13015-023-00226-2.

[12] Baire, A., Marijon, P., Andreace, F., and Peterlongo, P. “Back to sequences: Find the origin of k-mers”. In: *Journal of Open Source Software* vol. 9, no. 101 (2024), p. 7066. DOI: 10.21105/joss.07066. URL: https://doi.org/10.21105/joss.07066.

[13] Duitama Gonzalez, C., Rangavittal, S., Vicedomini, R., Chikhi, R., and Richard, H. “aKmerBroom: Ancient oral DNA decontamination using Bloom filters on k-mer sets”. In: *iScience* vol. 26, no. 11 (2023), p. 108057. ISSN: 2589-0042. DOI: https://doi.org/10.1016/j.isci.2023.108057. URL: https://www.sciencedirect.com/science/article/pii/S258900422302134X.

[14] *Bloom filter for databases*. URL: https://hbase.apache.org/book.html#perf.schema.

[15] *Google Chrome bloom filter*. URL: https://chromiumcodereview.appspot.com/10896048/.

[16] Marchet, C. *Advances in practical k-mer sets: essentials for the curious*. 2024. arXiv: 2409.05210 [q-bio.GN]. URL: https://arxiv.org/abs/2409.05210.

[17] Marchet, C., Boucher, C., Puglisi, S. J., Medvedev, P., Salson, M., and Chikhi, R. “Data structures based on k-mers for querying large collections of sequencing data sets”. In: *Genome Research* vol. 31, no. 1 (2021), pp. 1–12. DOI: 10.1101/gr.260604.119. URL: http://genome.cshlp.org/content/31/1/1.abstract.

[18] Marchet, C. *Advances in colored k-mer sets: essentials for the curious*. 2024. arXiv: 2409.05214 [q-bio.GN]. URL: https://arxiv.org/abs/2409.05214.

[19] Lappalainen, T. et al. “Transcriptome and genome sequencing uncovers functional variation in humans”. In: *Nature* vol. 501, no. 7468 (Sept. 2013), pp. 506–511. ISSN: 1476-4687. DOI: 10.1038/nature12531. URL: https://doi.org/10.1038/nature12531.

[20] Mason, C. et al. “The Metagenomics and Metadesign of the Subways and Urban Biomes (MetaSUB) International Consortium inaugural meeting report”. In: *Microbiome* vol. 4, no. 1 (June 2016), p. 24. ISSN: 2049-2618. DOI: 10.1186/s40168-016-0168-z. URL: https://doi.org/10.1186/s40168-016-0168-z.

[21] Sunagawa, S. et al. “Tara Oceans: towards global ocean ecosystems biology”. In: *Nature Reviews Microbiology* vol. 18, no. 8 (Aug. 2020), pp. 428–445. ISSN: 1740-1534. DOI: 10.1038/s41579-020-0364-5. URL: https://doi.org/10.1038/s41579-020-0364-5.

[22] Lemane, T., Medvedev, P., Chikhi, R., and Peterlongo, P. “kmtricks: efficient and flexible construction of Bloom filters for large sequencing data collections”. In: *Bioinformatics Advances* vol. 2, no. 1 (Apr. 2022), vbac029. ISSN: 2635-0041. DOI: 10.1093/bioadv/vbac029. eprint: https://academic.oup.com/bioinformaticsadvances/article-pdf/2/1/vbac029/47086129/vbac029_supplementary_data.pdf. URL: https://doi.org/10.1093/bioadv/vbac029.

[23] Pandey, P., Bender, M. A., Johnson, R., and Patro, R. “Squeakr: an exact and approximate k-mer counting system”. In: *Bioinformatics* vol. 34, no. 4 (Oct. 2017), pp. 568–575. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btx636. URL: https://doi.org/10.1093/bioinformatics/btx636.

[24] Pandey, P., Bender, M. A., Johnson, R., and Patro, R. “deBGR: an efficient and near-exact representation of the weighted de Bruijn graph”. In: *Bioinformatics* vol. 33, no. 14 (July 2017), pp. i133–i141. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btx261. URL: https://doi.org/10.1093/bioinformatics/btx261.

[25] Pandey, P., Bender, M. A., Johnson, R., and Patro, R. “A General-Purpose Counting Filter: Making Every Bit Count”. In: *Proceedings of the 2017 ACM International Conference on Management of Data*. SIGMOD ’17. Chicago, Illinois, USA: Association for Computing Machinery, 2017, pp. 775–787. ISBN: 9781450341974. DOI: 10.1145/3035918.3035963. URL: https://doi.org/10.1145/3035918.3035963.

[26] Rossignolo, E. and Comin, M. “Enhanced Compression of k-Mer Sets with Counters via de Bruijn Graphs”. In: *Journal of Computational Biology* vol. 31, no. 6 (2024). PMID: 38820168, pp. 524–538. DOI: 10.1089/cmb.2024.0530. URL: https://doi.org/10.1089/cmb.2024.0530.

[27] Levallois, V., Andreace, F., Gal, B. L., Dufresne, Y., and Peterlongo, P. “The Backpack Quotient Filter: a dynamic and space-efficient data structure for querying k-mers with abundance”. In: *bioRxiv* (2024). DOI: 10.1101/2024.02.15.580441. URL: https://www.biorxiv.org/content/early/2024/02/18/2024.02.15.580441.

[28] Robidou, L. and Peterlongo, P. “Fimpera: drastic improvement of Approximate Membership Query data-structures with counts”. In: *Bioinformatics* vol. 39, no. 5 (May 2023), btad305. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btad305. URL: https://doi.org/10.1093/bioinformatics/btad305.

[29] *RECOMB–seq conference website.* URL: https://recomb-seq.github.io.

[30] Marchet, C., Kerbiriou, M., and Limasset, A. "BLight: efficient exact associative structure for k-mers". In: *Bioinformatics* vol. 37, no. 18 (Apr. 2021), pp. 2858–2865. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btab217. URL: https://doi.org/10.1093/bioinformatics/btab217.

[31] Shibuya, Y., Belazzougui, D., and Kucherov, G. "Set-Min Sketch: A Probabilistic Map for Power-Law Distributions with Application to k-Mer Annotation". In: *Journal of Computational Biology* vol. 29, no. 2 (2022). PMID: 35049334, pp. 140–154. DOI: 10.1089/cmb.2021.0429.

[32] Mäkinen, V., Cunial, F., Belazzougui, D., and Tomescu, A. I. *Genome-Scale Algorithm Design, 1st edition. Chapter 15.4.* Helsinki: Cambridge University Press, 2019. URL: https://www.genome-scale.info/1stedition/.

Chapter 5

List of Papers

Comparing methods for constructing and representing human pangenome graphs

Francesco Andreace, Pierre Lechat, Yoann Dufresne, and Rayan Chikhi In: *Genome biology*. Vol. 24, no. 1 (2023), pp. 274. DOI: 10.1186/s13059-023-03098-2.

The Backpack Quotient Filter: a dynamic and space-efficient data structure for querying k-mers with abundance

Victor Levallois, Francesco Andreace, Bertrand Le Gal, Yoann Dufresne, and Pierre Peterlongo. In: *iScience*. Vol. 27, no. 12 (2024), DOI: = 10.1016/j.isci.2024.111435,

MUSET: Set of utilities for the construction of abundance unitig matrices from sequencing data

Riccardo Vicedomini, Francesco Andreace, Yoann Dufresne, Rayan Chikhi, and Camila Duitama Gonzalez. In: *Bioinformatics*. (2025), DOI: 10.1093/bioinformatics/btaf054

Back to Sequences: Find the origin of *k*-mers

Anthony Baire, Pierre Marijon, Francesco Andreace, and Pierre Peterlongo In: *Journal of Open Source Software*. Vol. 9, no. 101 (2024), pp. 7066. DOI: 10.21105/joss.07066.

Analysis of metagenomic data

Liu, S., Rodriguez, J.S., Munteanu, V. et al. In: *Nature Reviews Methods Primers*. Vol. 5, no. 5. (2025) DOI: 10.1038/s43586-024-00376-6

Chapter 6

Perspectives and future work

In this section, I will present several considerations and perspectives derived from the research proposed in this manuscript, from extended discussions with colleagues, supervisors, and other experts in the field over my PhD.

6.1 On human pangenomics: graphs and beyond

The analysis presented in chapter 3.1 provides a basis for understanding the features, limitations, and utility of current software designed for constructing pangenome graphs. These tools use state-of-the-art computational algorithms to be able to produce graphs from large and complex genomes with reasonable amount of computational resources. During the time-frame of my PhD, some tools have been improved and other proposed. As we get closer to a potential paradigm shift from linear reference sequences to pangenomes as reference for analysis, certain aspects of computational pangenomics could be prioritized.

Reproducibility and Computational Stability

In the case of leading general-purpose pangenome reference construction tools, such as `pggb` and `Minigraph-Cactus`, which produce variation graphs, it is important that the graph generated from a set of sequences remain consistent when identical input is provided. Inconsistencies in graph production can hinder reproducibility of analysis based on the same set of input sequences. In contrast, we have confirmed that dBGs do not suffer from this problem as their construction is mathematically deterministic. To partially address and quantify this issue, it would be useful to develop a tool that gets two variation graph as input, generated from a permutation of the same sample of genomes, and outputs the regions of the genomes where the two structures do not coincide. This would be also useful to verify the improvement of subsequent versions of the tools.

Biological Correctness of Pangenome Graphs

As variation graphs rely on sequence alignment to infer the graph structure, difficult-to-map regions and complex loci can lead to confounded graph sections that may not accurately represent the genetic variation in the locus. Highly complex centromeric regions exemplify such areas: in this moment, `Minigraph-Cactus` omits centromeric variation, `pggb` includes it, at the cost increased graph complexity.

Conversely, De Bruijn Graphs are challenging to interpret already at small scale. Extracting a large region from a graph is non-trivial, as distinguishing between sequences belonging to the region and those contiguous to common repetitions is difficult. A valuable future development would be the design of a method evaluating the computational and biological quality of the pangenomic data structures produced. For variation graphs it could be done by using a curated

datasets of known repetitive regions of the genome and verifying the structure of the sub-graphs of that regions by visualization and detection of loops in paths. For dBGs, as I also tried during my PhD, it would be very useful to produce a dBG-based data structure that can be visualized in a quasi-linear way like variation graphs. A possible implementation would require recording paths in the graph by or pseudo-mapping the input genomes to the ccdBG structure or a novel method to construct a cdBGs that directly stores the paths when building the graph. I have tested the first approach using `ggcat` and `ssHash` on human genomes and it is very computationally expensive compared to ccdBG construction, especially in terms of time. Another implementation could record information only for unitigs, i.e., nodes in the graph, that correspond to repetitions in the genome that confound the graph structure. This would help disentangle the graph.

Standardization of dBG Colors

Proposing a common file format or interface to access colors for color dBG based methods would promote the development of new application-specific tools based on k-mers. The current diverse landscape of colored k-mer sets construction tools limits future development to a specific tool. Variation graphs use paths or walks as standard ways to represent haplotype information in the graph. Producing a common interface to query colors in dBGs would serve a similar purpose. To achieve this goal, it would be useful to have a conference of k-mer researchers to agree together to a common interface for color and metadata based tools, starting from examples like the k-mer file format [1] and the k-mer days conference at EBI.

Alternative *k*-mer Based Pangenome Representations

While graphs are the most used representation, other k-mer based methods and data structures can provide a more suitable model for specific use-cases. Unitig (and k-mer) matrices, as produced by `kmtricks` [2] and `muset`, offer valid alternatives to comprehensively represent diversity in populations. Proposing new data structures equivalent to ccdBGs and interfaces to transition between them allows users to choose the best representation for their specific application. While `muset` is a first attempt at producing a unitig matrix representation, it is quite inefficient as it uses multiple tools that are not designed to be used for this scope and passes by various text representations that make it less efficient. In the future it would be useful to implement a direct construction methods that stores a binary representation on disk, loads it in memory for queries and can output a plain text matrix for other downstream applications.

6.2 Exploring *k*-mer data structures

***k*-mer Based Downstream Applications**

While significant advancements are being made in k-mer based methods to efficiently represent samples of varying complexity, further research is required to bridge the gap between these efficient representation methods and more comprehensive tool-sets for downstream analysis. Notable efforts in this direction

provide nucleotide search capabilities for huge collections of data. The Ocean Read Atlas (ORA) [3], uses `kmtricks` to provide real-time queries of 1,393 marine seawater metagenome samples obtained from the Tara Oceans project. Logan [4] and Metagraph [5], that use *k*-mer based methods to represent the entire World genetic diversity, as a snapshot of the ENA or SRA archives. All three projects offer online queries of their data with very useful visualization capabilities.

However, tools that address areas traditionally dominated by alignment-based methods, particularly in genomic analysis, are slowly emerging at scale. `muset` represents a proposal for an alternative platform for genetic variation studies, aims to partially address this gap and offers unbiased genome analysis, potentially enabling discoveries in regions that are not considered in traditional GWAS.

One very useful direction would be to enrich pangenome graphs with other metadata that is not only the sample of origin but also annotations of region of the genome as protein-coding genes, functional elements, chromatin accessibility. This information could in theory be injected in the graph by reproducing the sequences spanned by these regions and adding them to the graph. When a sequence is color-queried, this information would be returned. The drawback is the staggering amount of colors (one associated to each annotation) that would render the color data structure much larger and possibly impractical. Providing a more efficient representation for annotation would render these data structures more useful for other genomic analyses.

Improving methods for metadata

The two prototype *k*-mer based methods proposed in this manuscript offer an example of the great research potential for improving *k*-mer based methods for specific applications, particularly in handling metadata such as colors and counts. As pangenomics continues to evolve and the importance to discern between samples becomes crucial, the proposed data structures, together with other models, could be improved to allow *colored* queries or complex metadata queries. For instance, they could be used to return the abundance of a *k*-mer or sequence in a specific sample. While this type of information is stored in a text matrix produced by `muset`, it would be beneficial to index it to enable fast and programmatic queries. As proposed in the previous point, I think expanding metadata capabilities of *k*-mer based methods to store diverse kind of metadata, like annotations, could be of great use to provide not only information about frequency of a particular sequence in the input genomes but also about its function.

Bibliography

[1] Dufresne, Y., Lemane, T., Marijon, P., Peterlongo, P., Rahman, A., Kokot, M., Medvedev, P., Deorowicz, S., and Chikhi, R. "The K-mer File Format: a standardized and compact disk representation of sets of k-mers". In: *Bioinformatics* vol. 38, no. 18 (July 2022), pp. 4423–4425. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btac528. eprint: https://academic.oup.com/bioinformatics/article-pdf/38/18/4423/49885747/btac528.pdf. URL: https://doi.org/10.1093/bioinformatics/btac528.

[2] Lemane, T., Medvedev, P., Chikhi, R., and Peterlongo, P. "kmtricks: efficient and flexible construction of Bloom filters for large sequencing data collections". In: *Bioinformatics Advances* vol. 2, no. 1 (Apr. 2022), vbac029. ISSN: 2635-0041. DOI: 10.1093/bioadv/vbac029. eprint: https://academic.oup.com/bioinformaticsadvances/article-pdf/2/1/vbac029/47086129/vbac029_supplementary_data.pdf. URL: https://doi.org/10.1093/bioadv/vbac029.

[3] Lemane, T., Lezzoche, N., Lecubin, J., Pelletier, E., Lescot, M., Chikhi, R., and Peterlongo, P. "Indexing and real-time user-friendly queries in terabyte-sized complex genomic datasets with kmindex and ORA". In: *Nature Computational Science* vol. 4, no. 2 (Feb. 2024), pp. 104–109. ISSN: 2662-8457. DOI: 10.1038/s43588-024-00596-6. URL: https://doi.org/10.1038/s43588-024-00596-6.

[4] Chikhi, R., Raffestin, B., Korobeynikov, A., Edgar, R., and Babaian, A. "Logan: Planetary-Scale Genome Assembly Surveys Life's Diversity". In: *bioRxiv* (2024). DOI: 10.1101/2024.07.30.605881. eprint: https://www.biorxiv.org/content/early/2024/07/31/2024.07.30.605881.full.pdf. URL: https://www.biorxiv.org/content/early/2024/07/31/2024.07.30.605881.

[5] Karasikov, M., Mustafa, H., Danciu, D., Zimmermann, M., Barber, C., Rätsch, G., and Kahles, A. "Indexing All Life's Known Biological Sequences". In: *bioRxiv* (2024). DOI: 10.1101/2020.10.01.322164. eprint: https://www.biorxiv.org/content/early/2024/06/07/2020.10.01.322164.full.pdf. URL: https://www.biorxiv.org/content/early/2024/06/07/2020.10.01.322164.

Chapter 7

Conclusions

This manuscript marks the culmination of a 40-month doctoral journey, focused primarily on developing and using data structures to represent sets of human genomes as a pangenome, with a particular attention to k-mer based tools. Every step of this work has been driven by the goal of creating software or analyses capable of addressing relevant biological questions.

The research path was nonlinear, leading to collaborations in various projects that, while not all directly presented in this manuscript, provided invaluable insights. These experiences deepened my understanding of bioinformatics, computer science, and genomics, which is the reason why the scope of this thesis is relatively broad. It begins by analyzing computational pangenomics methods for human genomes representation and concludes with the development of efficient data structures for k-mer handling. This is the result of constant curiosity about sequence bioinformatics problems, a collaborative spirit and reflects a comprehensive and multifaceted approach to the challenges of the field.

A key contribution of this work is an analysis of pangenome graphs constructed from high-quality human haploid assemblies, which was well received by the research community. The findings highlight the strengths and limitations of both the representation models and the methods used to generate them. Specifically, it was demonstrated that different types of graphs serve different purposes: variation graphs are intuitive for downstream analysis and easier to manipulate, whereas dBGs offer better scalability and efficiency in preserving input sequences, particularly for large and different datasets. However, as it is often the case in genomics, there is no "one-size-fits-all" solution. This research also identified areas where k-mer based approaches could become more valuable for genome-wide analyses, a domain still requiring further development. Finally, this showcased that pangenomics holds the key to addressing many of the limitations of current genomics approaches, particularly as it helps better understand structural variations in primate genomes.

The work on k-mer based data structures explored various methods and implementations, often revealing the complexity of finding models that can address multiple problems simultaneously. A significant challenge in pangenomics (and in bioinformatics broadly) is finding the right trade-offs for specific applications: this work spans multiple levels of development—from high-level script design to low-level operational optimizations for more efficient tools.

The research on quotient filters focused on re-developing the Rank Select Quotient filter, aiming to make the underlying data structure suitable for various high-level applications. This led to successful implementations such as the application of the Fimpera scheme on top of a traditional CQF, and the creation of a new data structure, the BQF.

The research on super$k$-mers resulted in a prototype for producing sorted super$k$-mer lists in practical, real-world applications. While it does not represent a standalone tool, it can be used as a building block of a more comprehensive k-mer based tool-set for pangenome sequence manipulation and analysis.

I also contributed to other exciting projects, such as `muset`, a pipeline for creating unitig matrices (both in terms of abundance and presence/absence). This work represents a significant advancement from k-mer matrices, simplifying some of the complexity in downstream genomic analysis while retaining necessary information. Collaborating on this project has been important, as I believe the field of k-mer based tools needs more downstream tools for specific genomic applications.

While tools, pipelines, and data structures presented in this manuscript are not the final solutions to a particular challenge in pangenomics, they represent meaningful progress. Undoubtedly, there's still much work to be done. The promising areas touched by my research are scaling pangenome representations for large collections of eukaryotes, establishing methodologies to validate the biological accuracy of pangenome graphs structures and developing more k-mer-based tools to assist genomics and genetics analyses.

Appendix A

Prototyping Dynamic Data structures for *k*-mer counting: super-*k*-mer sorted list

Searching the list

The sorted super-k-mer list produced by the outlined algorithm enables relatively fast search without indexing k-mers, using binary search. Binary search is an efficient algorithm used to locate a target value within a sorted array: it repeatedly divides the search interval in half and compares the middle element to the target.
In the case of looking in a sorted array with smaller values on the left and larger values on the right, if the target is smaller than the middle element, it continues the search in the remaining the left half, if it's larger, in the right half. The process continues until the element is found or the search interval is empty.
In this section I am going to refer to masks as bit-level masks (or bitmask) as tools that are used to select elements (bits) from a binary representation of some value. In this case k-mers are encoded in 2*k bits and super-k-mers in 2*(2*k-m) bits. To extract a k-mer from a super-k-mer we use a mask to set to zero the 2*(k-m) bits that do not represent it. The k-mer query process in our sorted super-k-mer list operates as follows:

1. the minimizer of the k-mers is computed and the k-mer position in a super-k-mer is determined.
2. a mask associated to that position is selected;
3. the range of the searched list is given by $[x, y]$ and set to $[0, N]$, with N being the length of the list;
4. the binary search algorithm jumps the middle super-k-mer of the range $[x, y]$;
5. if the super-k-mer does not have a k-mer at the searched position, the search moves to on to the next ones until it finds one that does;
6. the mask is applied to the super-k-mer;
7. the resulting binary value is compared to the one of the k-mer. Here one of these 3 situations can occur:
    - the masked super-k-mer value is greater than the k-mer, than $[x, y]$ gets updated to $[x, y] = [(y - x/2), y]$;
    - the masked super-k-mer value is smaller than the k-mer, than $[x, y]$ gets updated to $[x, y] = [x, (y - x/2)]$;

- the item is found, return `FOUND` or `TRUE`;

8. if $x = y$ return `NOT FOUND` or `FALSE`, else go back to point 4;

The advantage of this super-k-mer representation is that it allows the search algorithm to jump elements (super-k-mers) and directly query specific positions (offsets) based on the minimizer position of the queried k-mer. This approach avoids linear probing of entire super-k-mers.
Finally, the time complexity may be worse than binary search due to gaps in the elements of the matrix. Not all super-k-mers are maximal, therefore some queries at specific positions may be impossible due to the absence of a k-mer. An offset is valid if the super-k-mer contains a k-mer in that position, and invalid otherwise. To address this, when a lookup encounters an invalid offset, it performs a linear probe on previous or subsequent elements in the list until it finds a super-k-mer with a valid offset.

Bonus optimizations

We propose three potential optimizations for the presented algorithm:

- Adding information to guide the binary search when looking in a super-k-mer at an invalid offset and having to use linear probing to find a a valid one in the neighbors. This can be achieved by filling the bits of an empty offset in a super-k-mer with "synthetic" nucleotides. These synthetic nucleotides don't serve as genetic information but provide a middle value between the two closest valid offset values, informing the search direction.

- The current lexicographic ordering of super-k-mers is suboptimal due to its bias towards homopolymers and poly-A regions. We propose an improved ordering strategy that gives more weight to central nucleotides, particularly those in or closer to the minimizer. This approach could lead to a more balanced and efficient search structure.

- Instead of using binary search, which compares the searched element with the middle of the remaining search space at each step, we could implement interpolation search to potentially accelerate the process [1]. Interpolation search calculates where in the remaining space the element is likely to be, based on the values at the boundaries of the space and the searched value. If the element isn't found at the calculated position, it uses the same splitting strategy as binary search.

Bibliography

[1] Abrar, M. H. and Medvedev, P. "PLA-complexity of k-mer multisets". In: *bioRxiv* (2024). DOI: 10.1101/2024.02.08.579510. URL: https://www.biorxiv.org/content/early/2024/02/11/2024.02.08.579510.

Acronyms

AMQ Approximate Memmbership Query.

API Application Programming Interface.

BQF Backpack Quotient Filter.

cAMQ counting Approximate Memmbership Query.

CBF Counting Bloom Filter.

CLI Command Line Interface.

CNV Copy Number Variant.

CQF Counting Quotient Filter.

CSV Comma Separated Values.

DNA Deoxyribonucleic acid.

dsDNA double stranded DNA.

ENA European Nucleotide Archive.

GWAS Genome Wide Association Study.

HOR Higher Order Repeat.

HPRC Human Pangenome Reference Consortium.

I/O Input / Output.

JSONL JavaScript Object Notation Lines.

MPHF Minimal Perfect Hash Functions.

NGS Next Generation Sequencing.

NIH National Institute of Health.

ONT Oxford Nanopore Technologies.

PCR Polymerase Chain Reaction.

QC Quality Control.

RNA Ribonucleic acid.

RSQF Rank Select Quotient Filter.

SNP Single Nucleotide Polymorphism.

SNV Single Nucleotide Variant.

SPSS Spectrum Preserving String Set.

SRA Sequence Read Archive.

SSD Soild State Drive.

ssDNA single stranded DNA.

SV Structural Variation.

T2T Telomere to Telomere consortium.

TGS Third Generation Sequencing.