SORBONNE UNIVERSITÉ

Thesis Manuscript Draft

Analysis of human pangenome graphs using *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	University of 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

2024

INSTITUT PASTEUR

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

Print production: Insitut Pasteur.

To Sofia, my sweet old cat that lives so well without overthinking. (This is a placeholder)

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 Insitut Pasteur, under the supervision of Dr. Rayan Chikhi and Dr. Yoann Dufresne. This work was supported by the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No 956 229 (+Panagaia +Pasteur + INCEPTION).

The thesis is a collection of the different projects I worked on during my stay at Insitut Pasteur. I begin with an small foreword of the research output of my PhD, and a gentle introduction of the scientific concepts needed to understand the rest of the manuscript. In the first section I present the published paper I am first author of, together with other unpublished work I lead or independently developed. In the second section 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 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

TODO

Francesco Andreace
Paris, October 2024

Contents

List of Figures

List of Tables

Chapter 1

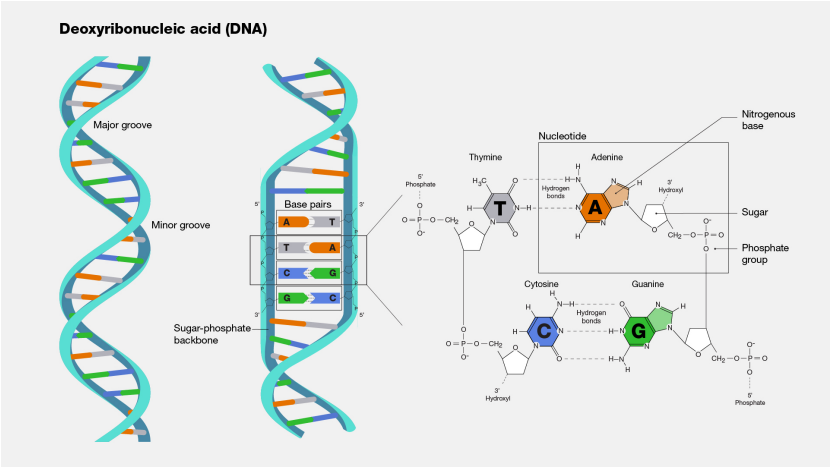
Background

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 1.3 *From reads to* k*-mers*.

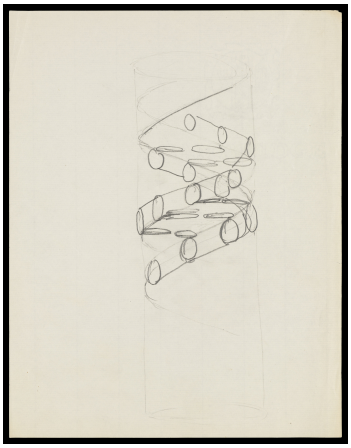
1.1 DNA, genome variation and sequencing data

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 [**franklin**].
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 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.
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. This pairing is crucial for both DNA replication and protein synthesis. Figure 1.1 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 be haploid (containing a single copy of each chromosome) 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 [**sugarcane**]. The complete set of



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



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

Figure 1.1: The DNA molecule.

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` s, 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 [**telomeres**]. 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 [**centromeres**].

Finally, there is also the mitochondrial genome, which is located outside the nucleus. It has a circular structure and is primarily maternally inherited.

1.1.1 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 [**garrison_pangenome**]. 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 [**sanger_sequencing**]. 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 [**mitochondrialDNA**, **first_human_genome**]. 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.

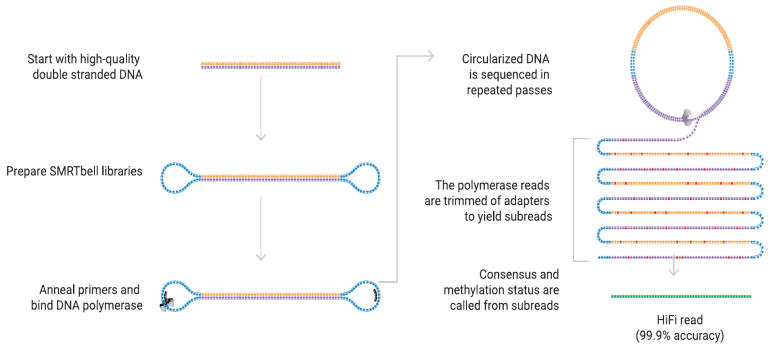
1.1.1.1 Next Generation Sequencing

(NGS) erives 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 [**100dollars**]. 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 Ph.D. to improve the estimation of different species in environmental samples sequenced

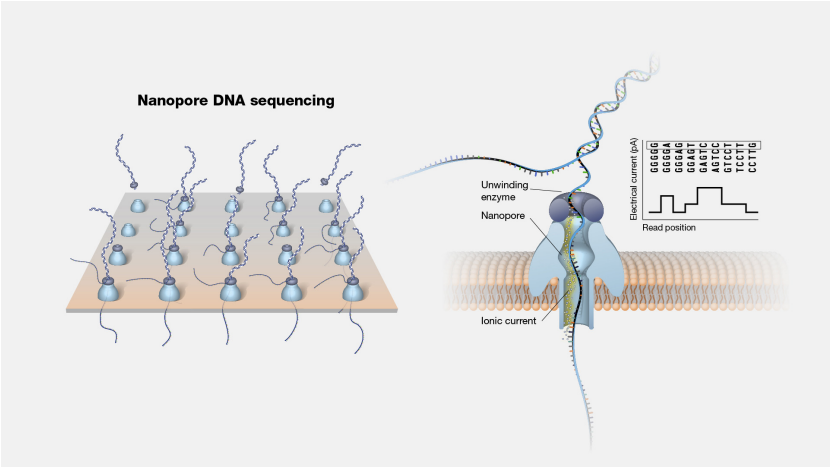
with NGS [**metaprob2**].
Finally, NGS technology has also enabled other types of sequencing, such as RNA-seq, ATAC-seq, ChIP-seq, and others, which allow for the assaying of regulatory activity within the genome.

1.1.1.2 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*.
In this section, I will describe the two most prominent technologies, 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 [**mdbg**] 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 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.
It is important to emphasize that 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. 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 [**methylation**].
Figure 1.2 provides basic schematics illustrating how these two technologies work.



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



(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 machine learning algorithm to infer the nucleotides of the sequence.

Figure 1.2: Third generation sequencing technologies.

1.2 From reads to *k*-mers and beyond

The sequences produced by any of the aforementioned technologies are 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 type. In cases like assembling a genome from the reads or calling the variant of the sequenced genome compared to one 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, 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: A,C, G and T: $\sum = A, C, G, T$,

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 tipically 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 $|k - mer| = k$ i.e. any valid sub-string of S of length k.

As shown in table 1.1, 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 a 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 1.1, 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 do they come from. In other, specific, applications it might instead be needed to know in which reads there are certain k-mers [**back_to_sequences**]. As presented in section 1.1 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 $k = 4$) $ACGT$ is lexicographically smaller than $TGCA$ 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 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.

1.2.1 *k*-mer based objects

Unitigs correspond to the string spelled by concatenating a non-branching path in a 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 concatenations of nodes that have in-degree and out-degree of 1. The dBG is more formally defined in section 1.4.4
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 1.1: 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 1.2: Example of canonical k-mers enumeration and count. Given a set of sequences, for each of them 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.

1.3 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 [**chess**]. 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.

1.3.1 Causes and drivers of genetic diversity in humans

There are two main mechanisms of genetic diversity: the arise 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 type 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 to not corrected damage.
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 human 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 thousand 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) [**lactase**]. Selection on certain genes explains better adaptations to cold or high altitudes and selection in HbS or DARC alleles has helped humans adapt and survive malaria infections[**genome_diversity_quintana**].

1.3.2 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. 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[**lactase_persistance**]. A second group of small variants is made of insertions and deletions (called together *indels*): these are events in which it is present or missing a group of less than 50 nucleotides. The number of 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 [**nih_variation**].

These groups of small variants, shown in figure1.3 are the most described, studied and associated with diseases as they were the only one consistently detectable with NGS sequencing. For these reason, 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 (SVs). These can be indels or tandem repeats with the repeated section longer than 50 nucleotides, accounting for nearly half of all SVs, that take the name of Copy Number Variants (CNVs). Moreover, there are also inversions, in which a chunk of DNA is inverted compared to

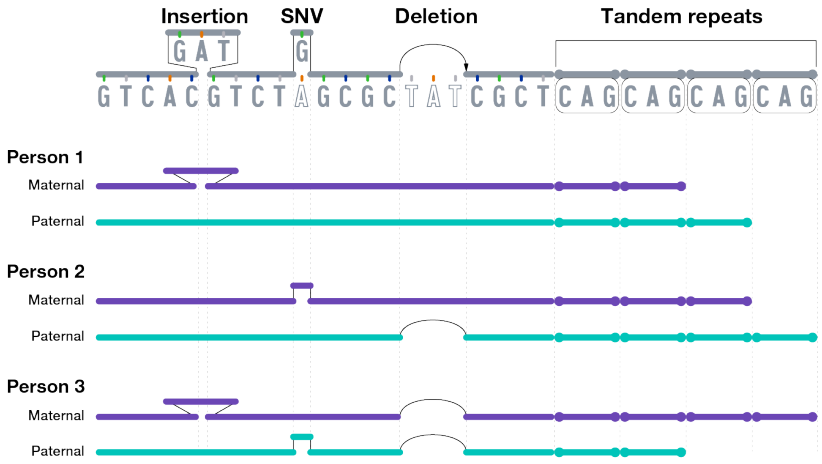


Figure 1.3: Graphic showing the types of small genomic variants [**nih_variation**]. Individuals have different genomes, and these differences are encoded as variations in their DNA sequences. The most common type of variation between individuals is a Single Nucleotide Variation (SNV), in which a base 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 haplotypes.

another and translocations in which pieces of two different chromosomes trade places [**nih_variation**].

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

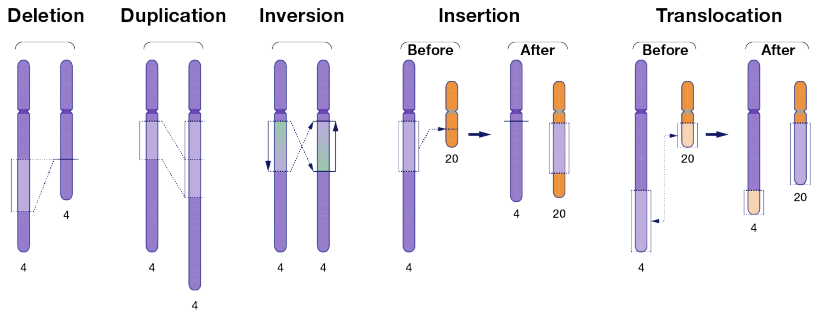


Figure 1.4: Graphic showing the types of large genomic variants, also named structural variants [**nih_variation**]. Deletion or insertions of more than 50 base pairs are considered structural variants.
Duplications are when large segments are copied one or multiple times. They tend to have a nested 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 [**tbc1d3**].
An inversion is a segment of a chromosome that preset 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 [**inversions_disease**].
A translocation occur when a chromosome breaks and a portion is attached to another chromosome: this event can cause diseases like leukemia [**leukemia**].

1.3.3 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 1.5 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 [**genome_diversity_quintana**]. 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 1.6 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 [**tbc1d3**].

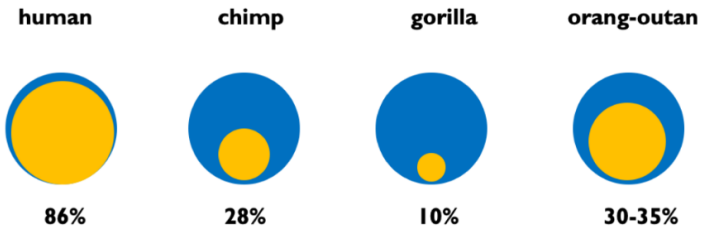


Figure 1.5: Share of inter-individual (yellow) and inter-population (blue) diversity for four different primates. While for humans the majority of the diversity is within populations, for other primates it is between populations. This shows how Humans are more mixed than other primates. Percentage shows the inter-individual variation share [**genome_diversity_quintana**].

1.4 Pangenomics, pangenomes and pangenome graphs

1.4.1 The premises for Pangenomics

There are a number of factors that must be taken into consideration to understand 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.

1.4.1.1 A single linear genome for all analyses

Since the first complete genome sequences have been available in the late '90, 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

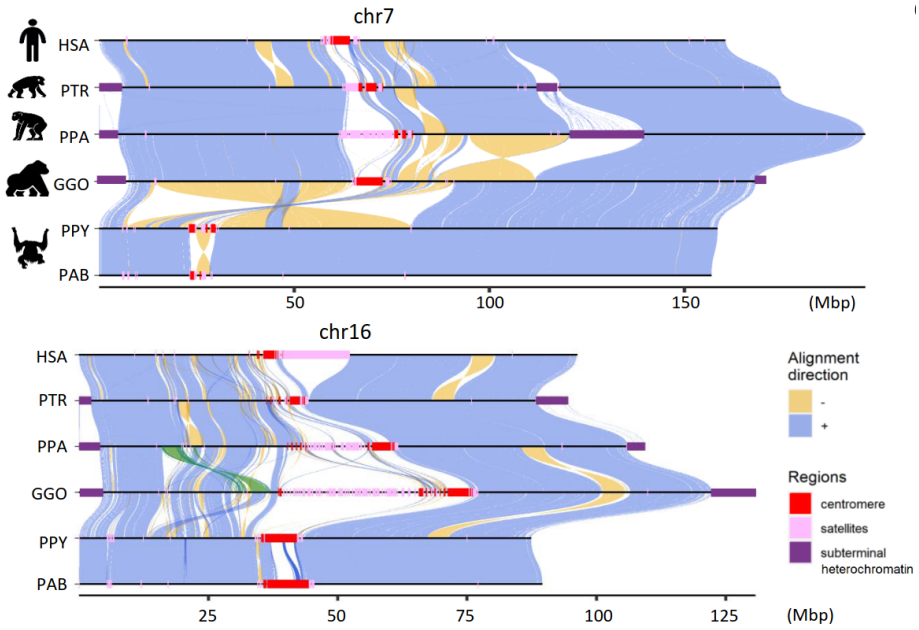


Figure 1.6: A comparative ape alignment of human (HSA) chromosomes 7 and 16 with chimpanzee (PTR), bonobo (PPA), gorilla (GGO), Bornean and Sumatran orangutans (PPY and PAB). The image on the top shows most of the chromosome 7 is conserved except for large inversions happening between the species. The image below shows complex inversions in chromosome 16. Image taken from "Complete sequencing of ape genomes" [**ape_genomes**].

purpose is to use it 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 genome of multiple organisms and their differences.

1.4.1.2 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[**durbin_recomb**], 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 next 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.

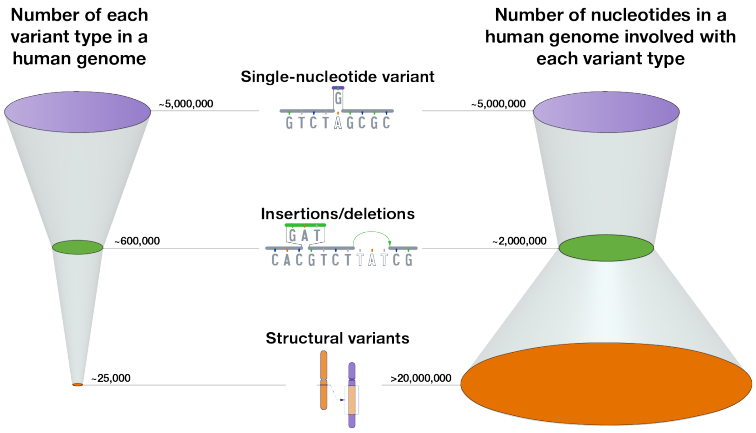


Figure 1.7: Spectrum of Human Genetic Variation [**nih_variation**]. While SNPs are the most common variation event, their impact in the total amount of bases in a genome is ~ 4 times smaller than the one of Structural Variations, that are 200 times less frequent. This shows the great need to consider SVs in genomic analysis and not to stop at the SNP/indel level.

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 [**humangenomeproject**] and the absence of long reads with low error rate made it impossible to automatically resolve repetitive regions like telomeres and centromeres [**human-pangenomics-era**], producing a reference only 92% complete [**t2t**]. 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 [**t2t**]. Now, many consortia are producing increasingly more genomes to a level comparable to the one produced in 2022. For example, the HPRC, i.e. the Human Pangenome Reference Consortium, 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 1.3, 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 1.8. 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.

year	genome(s)	base pairs
1995	Bacterium	$2 * 10^6$
2001	Mammal	$3 * 10^9$
2013	2500 humans	$7.5 * 10^12$
2021	1M genomes	$3 * 10^{15}$

Table 1.3: Scale of DNA data increase over the years. 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 derive from $\log_2(10^9) = 29.9 \implies 10^9 \approx 2^{30}$.[**durbin_recomb**].

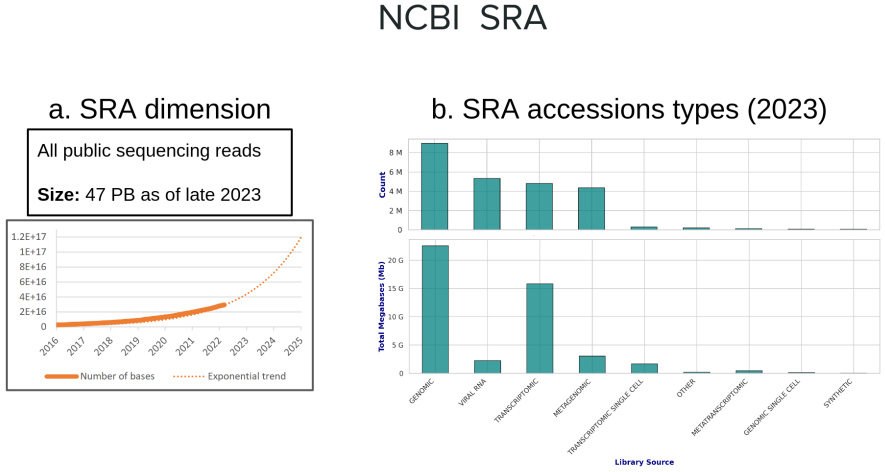


Figure 1.8: a) The size in petabases (peta = 10^{15}) of the SRA archive; b) The type of data in the SRA database shows the vast amount of genomic data available. Image made from Rayan Chikhi's slides.

1.4.1.3 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. 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.

1.4.2 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. 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. 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 1.9, 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. It was first conceptualized for bacterial genomes, and at gene level, without considering non coding regions. 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. 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.

1.4.2.1 Pangenomes

A pangenome can therefore be considered as any collection of genomic sequences to be analyzed jointly or to be used as reference. 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.

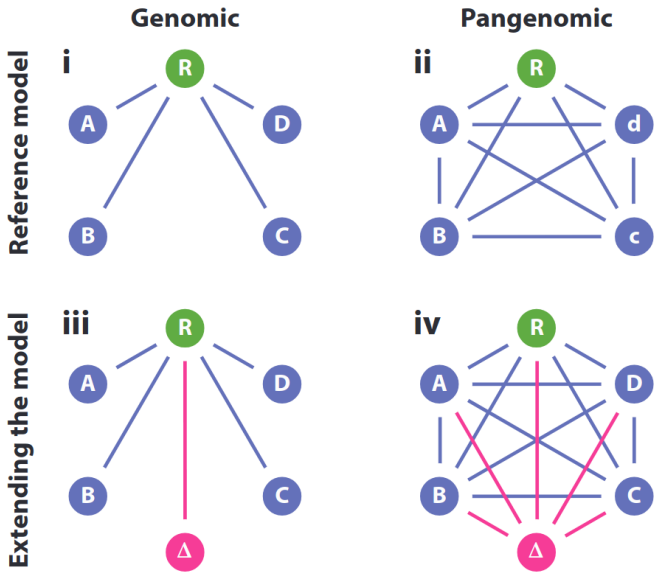


Figure 1.9: The genomic vs pangenomic model i) In the genomic model each genome is compared only to the reference sequence. Any comparison between a pair of genomes is done indirectly via their difference with the reference. ii) In the pangenomics, variations are described in a relative way for any genome. 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 [**eizenga**]

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 [**t2t**].
- 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.

A pangenome can also be an unaligned set of sequences. 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 [**centromeres_eichler**]. 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 from 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.

1.4.2.2 Pangenome Graphs

Graphs are a natural way of directly representing information between a group of object 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 [**odgi**].

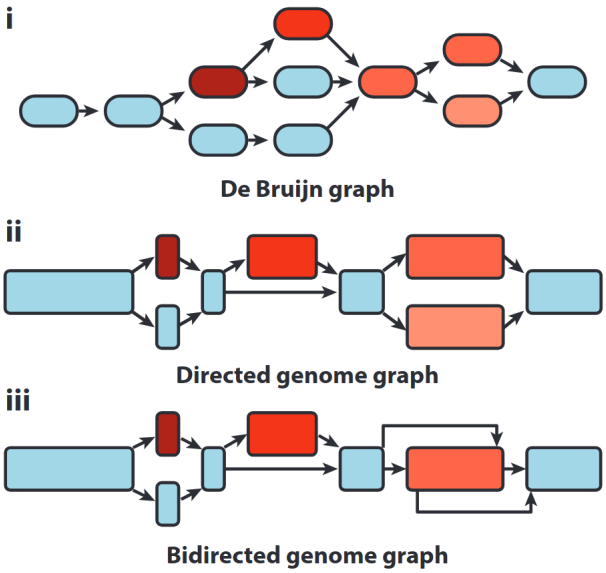


Figure 1.10: The three main kind of graphs used to represent pangenomes. Figure from [**eizenga**]

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. Examples of such bubbles for de Bruijn graphs and genome graphs are shown in Figures 1.11 and 1.13.

Below, I introduce the two most widely used models for constructing pangenome graphs: Variation Graphs and De Bruijn Graphs.

1.4.3 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 1.11. 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.

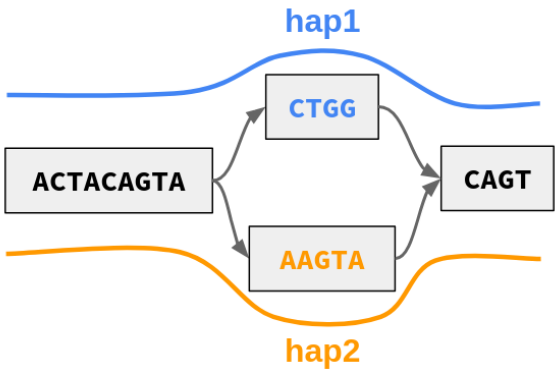


Figure 1.11: An example of variation graph, in which haplotype1 spells the sequence *ACTACAGTACTGGCAGT*, while haplotype 2 spells the sequence *ACTACAGTAAAGTACAGT* [**garrison_pangenome**].

As shown in figure 1.12, 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. 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.

1.4.4 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.

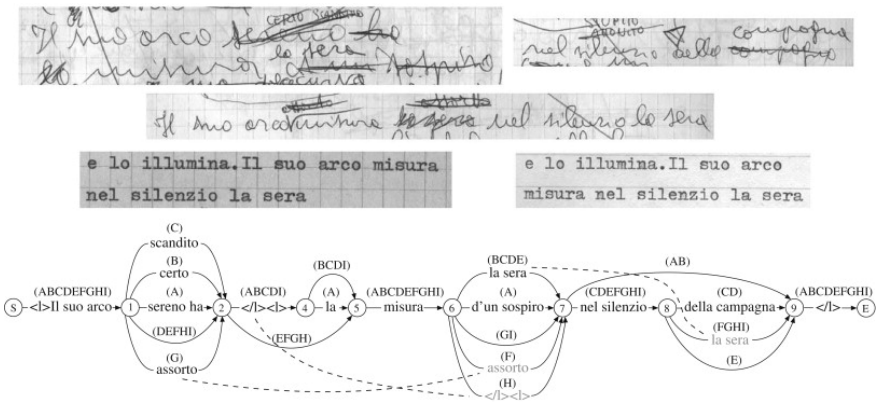


Figure 1.12: Instead of having to preform all the pairwise comparisons of the nine version of Valerio Magrelli's "Campagna Romana" poem from 1981, the variation graph structure describes the differences between them. It also removes the high redundancy in the versions of the poem [**variant_graph**, **garrison_pangenome**].

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.

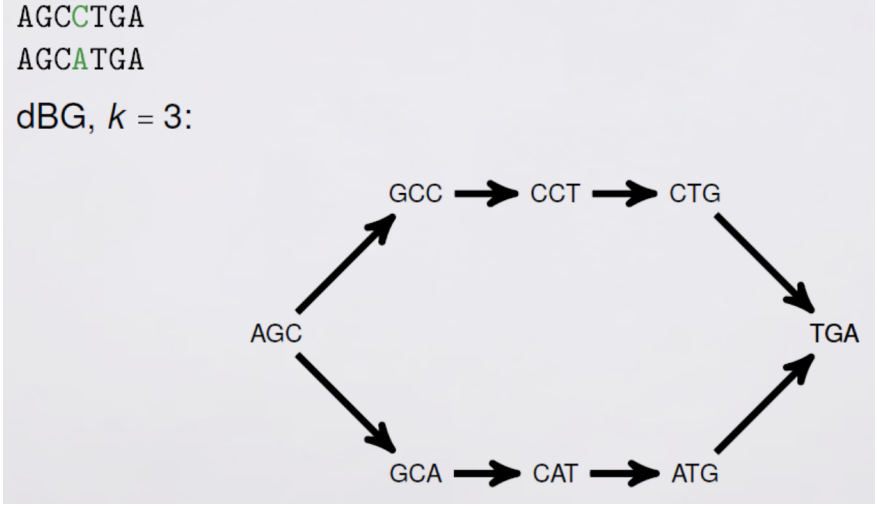


Figure 1.13: Example of dBG for $k = 3$ with a bubble representing a SNP. Image made by Rayan Chikhi.

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 [**chor2009genomic**]. Using larger k value reduces the probability that two genomes share the same k-mer by chance.

ariation 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 in table 1.1, larger value 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.

1.4.4.1 Colored and Compacted De Bruijn Graphs

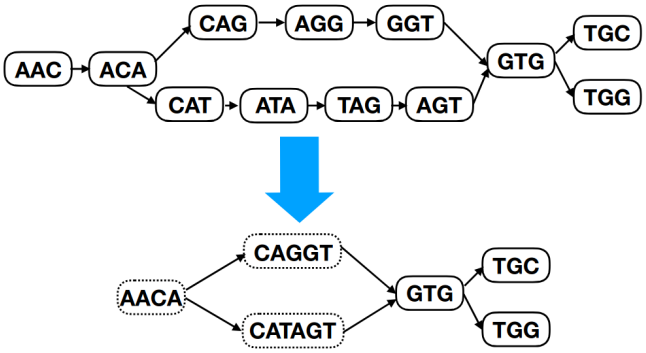
In order to produce a more compact and informative representation, in pangenomics it is used a particular version of the dBG, called ccdBG: colored compacted dBG. The main characteristics of this data structure are:

- 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 1.14.

- 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) [**marchet_kmersets**]. 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 1.14.



(a) Compacting a dBG from the two sequences `AACAGGTGC` and `AACATAGTGG` into a cdBGs reduces paths of nodes with in-degree and out-degree of 1 into a single node. Figure from [**embedding_dbg**]

(b) The two kind of colors that can be used on ccdBG: colors and color sets.

Figure 1.14: Compaction and colors: the two main characteristic of a ccdBG compared to a dBG. Figure from [**marchet_kmersets**]

1.5 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 work based on it.

Chapter 2

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 constructing 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 downstram application usability and show how a pan-genome enables improved analysis of inter-chromosome rearrangements.

2.1 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 1.3, 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 exists 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.

Chapter 3

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.

Contents

3.1 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 [**ucsc**] 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 [**vg**, **computational_pangenomics**, **giraffe**]. The GRCh38 human reference genome is estimated to miss up to 10% of our species genetic information [**human-pangenomics-era**]. 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 [**computational_pangenomics**, **hpp**]. 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 [**giraffe**], genotyping of SNPs, indels and SVs [**pangenie**], RNA-seq mapping [**hdpr**]; de novo variant calling [**vg**]; to store, compress and retrieve high quality genomes [**gbz**]; 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 [**hdpr**]. 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 [**computational_pangenomics**, **hpp**]. 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 [**computational_pangenomics**]. 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 [**pantools**] and Bifrost [**bifrost**] 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`) [**pggb**], `Minigraph-Cactus` and `TwoPaCo` [**twopaco**] are methods for building general-purpose pangenome graphs. `Minigraph` [**minigraph**] builds a particular type of pangenome graph by aligning sequences in an iterative way to a reference template. Minimizer-space de Bruijn graphs (`mdbg`) [**mdbg**] are variants of de Bruijn graphs that can efficiently represent very large collections of bacterial pangenomes (e.g. 600,000 bacteria). `vg` [**vg**] 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 [**pantools**] (7 genomes), `Minigraph` [**minigraph**] (94 haplotypes), `Minigraph-Cactus` [**cactus**, **mcactus**] and `pggb` [**hdpr**] (94 single chromosomes), and `TwoPaCo` [**twopaco**] (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 [**hdpr**]. 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.

3.2 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 [**giraffe**], 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 [**human-pangenomics-era**].
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.

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,

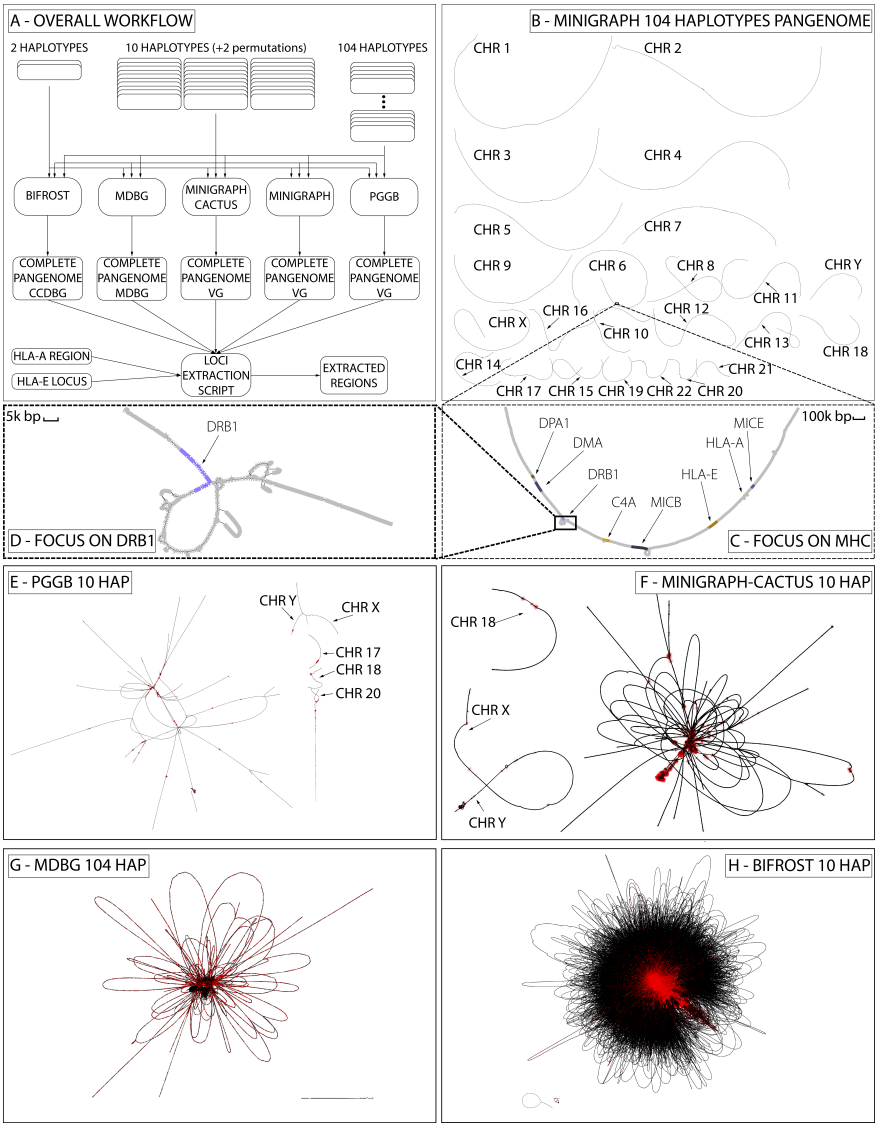


Figure 3.1: **The complete pangenome construction scheme and visualization. A**, The overall workflow, using 5 different tools on 3 different datasets; **B**, complete 104 haplotypes variation graph built by `Minigraph`; **C**, focus on part of HLA (MHC) region in chromosome 6 from panel B; **D**, focus 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.

`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 [**hdpr**, **mcactus**], in this analysis we purposely used entire genomes as input 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

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

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 [**chin**] 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 [**HLA-nature**]. 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 [**hla-e-covid**]. 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 variation 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.

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.

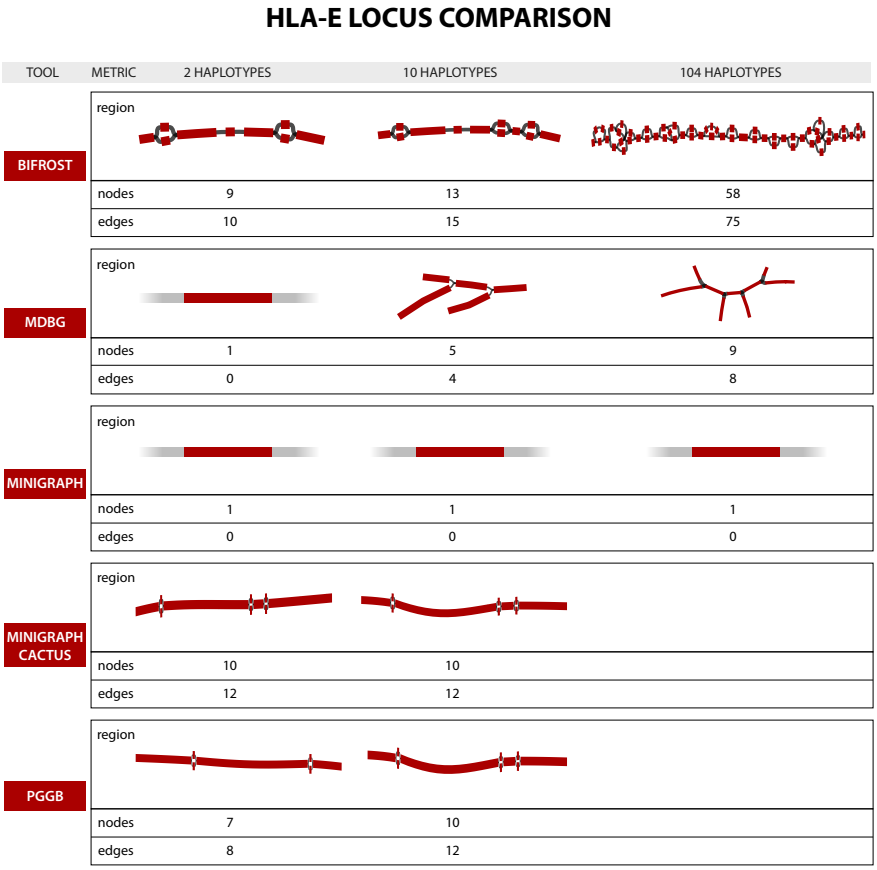


Figure 3.2: Representations of the HLA-E locus by five graph construction methods over three increasing large human pangenomes. Nodes highlighted in red contain part of the locus sequence. The numbers of nodes and edges displayed below each graph concerns the whole subgraph (both red and grey nodes). `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.

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` [**odgi**], 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.

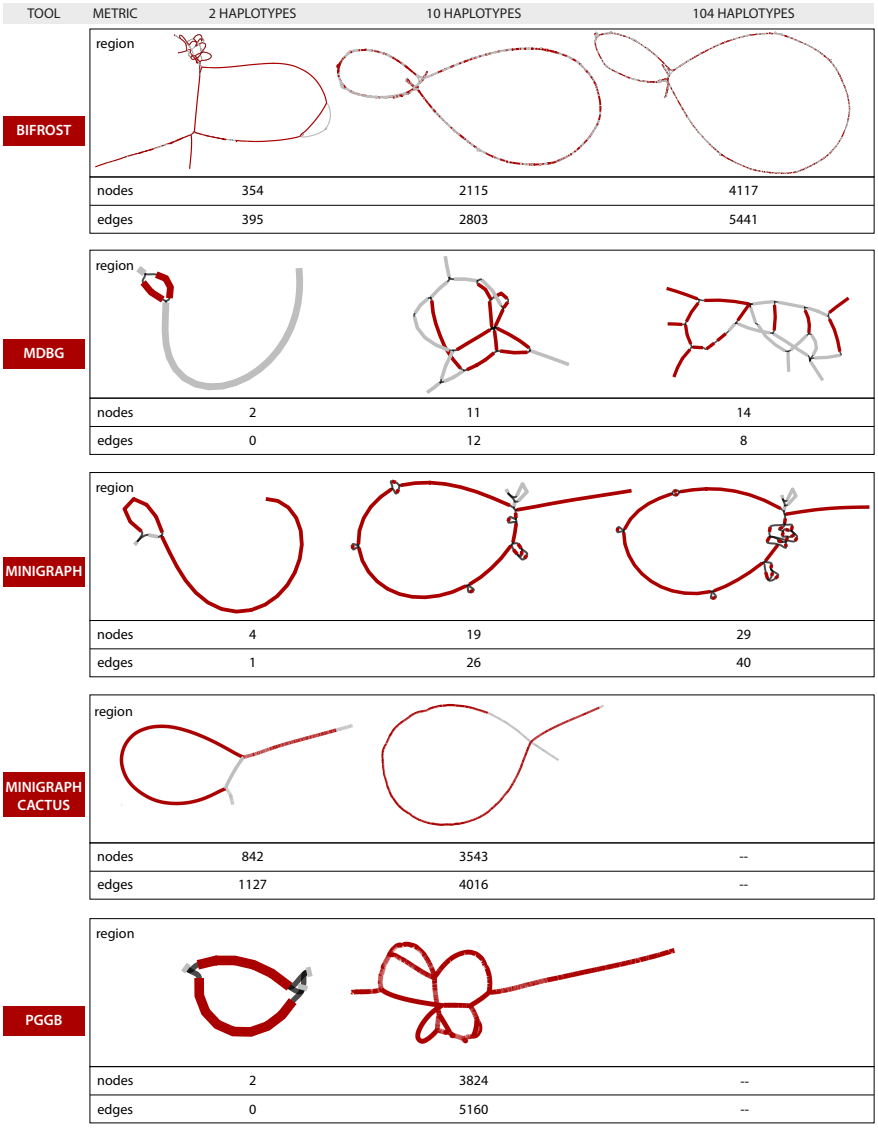


Figure 3.3: Representations of the complex HLA region by five graph construction methods over three increasing large human pangenomes. See caption of Fig. 3.2 for details.

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 [**hdpr**]. 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 [**gbz**] 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 [**gbz**]. 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 [**hpp**], 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 [**100000genomes**, **ucla**]. 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.

3.3 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 [**hdpr**]: 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` [**hdpr**]. `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 [**mdbg**].

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 remain 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.

3.4 Conclusions

In conclusion, our results highligh the strengths and weaknesses of current pagnenome construction tools for human applications, with specific focus 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.

3.5 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) [**grc**] and CHM13 v2.0 cell line of the T2T Consortium [**t2t**]. Five haplotypes correspond to Google Brain Genomics DeepConsensus [**deepconsensus**] 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 [**hpp**], 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 [**hdpr**].

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 [**google-assemblies**]; [2] Human Pangenome Reference Consortium [**HPRC-haplotypes**]; [3] 1000 Genomes Project [**HPRC-haplotypes**]; [4] Telomere to Telomere Consortium [**HPRC-haplotypes**].

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 embg: The definition of as changed! Check your packages! Replacing it with the kernel definition on input line 145.ed haplotypes paths. `Cactus` [**cactus**] 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[**gfaffix**]. 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 [**wfmash**], graph construction from the alignments with seqwish [**seqwish**], graph sorting and normalization with smoothxg and GFAffix [**smoothxg**, **gfaffix**]. 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` [**graphaligner**] 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 [**minimap2**]. 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 [**source-code-github**][**source-code-zenodo**] under MIT license. Google Brain Genomic assemblies can be found at [**google-assemblies**].
HPRC assemblies, CHM13 and GRCh38 can be found at [**HPRC-haplotypes**].

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.

Series of dissertations submitted to the Insitut Pasteur Paris, Universite' de Paris Cite, University of Oslo No. 1234 ISSN 1234-5678

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

3.6 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 axis. 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 pattern 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.

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

Here we present another example of pushing the boundaries of variation graph pangenome building strategies, precisely in producing a graph that can represent inter-chromosomal events, like rearrangements, for the medically interesting yeast strain of *Lodderomyces elongisporus*. We show also how a pangenome based approach can help produce a more comprehensive insight of such events than a linear genome based one.
The work presented here is also in part a team effort with other PhD students and researcher I was co-leading with Daniel Doerr during a winter-school organized by my consortium: Simon Heumos, which I would like to thank for the discussion we had on the construction of such graphs and Nicola Rizzo. Most of the results and findings reported here are moslty my own, subsequent, work.
The following sections will explain the process needed to produce custom, biologically-significant and biologically-driven pangenome graphs of a yeast strain with the current state-of-the-art tools and offer insights on workarounds that can be used in cases with special needs, where current tools features limit the analysis that can be done. This will also show how pangenome models can offer more powerful tools to analyze specific variation in a group of similar genomes compared to linear reference based ones.
In this specific case, the goal was to produce a variation graph that considered, and showed, the long chromosomal translocations that were noticed 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 singular same recombination group. This finding was consistent with an independent SNP-based genomic study. It was performed on the isolates of a fungemia outbreak in a neonatal intensive care unit in Delhi between 2021 and 2022 and it noticed that this translocation events are more frequent in hospital and patient associated populations than in fruit ones [**lodelo_india**].
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;

3.7.1 *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 [**Lodderomyces**]. 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 3.7.4. Increasing reports of (mostly bloodstream) infection in mainly immunosuppressed adults makes it an increasingly important subject of studies[**lodelo_pathogen**, **lodelo_fatal**, **lodelo_meningitis**, **lodelo_bloodstream**]: 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[**lodelo_india**].
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.

3.7.2 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 3.4a 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 for to produce quite straightforward whole-genome alignment- and reference-free phylogenetic analysis in a fraction of time required by competitor methods that use all vs. all alignment. The tool `SANS serif` [**sans**] can process directly a `Bifrost` generated ccdBG to estimate the phylogenetic splits between the genomes contained in the graph. Figure 3.4b shows the visualization of the phylogenetic network produced by `SANS serif` using the tool `SplitsTree` [**splitstree**]. By adding another genome from a close species it is possible control that the dBG-based analysis provides correct results. Figure 3.4c 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.

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 [**hdpr**]. It is important to notice that in order to produce the best biological correct result, several rounds of parameter tuning and manual curation are needed, with knowledge far superior of the one of a first-time user.

The first step to build such pangenomes is to divide the genome assemblies into communities of sequences belonging to chromosomes.

3.7.2.1 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 given by joining together the output of each group. This means that without any pre-processing, no inter-chromosomal event can be detected.

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 3.5. From this alignment, contigs were assigned to the chromosomal community to which they best mapped. Finally, chromosomes C,G,H were grouped manually into a single community.

3.7.2.2 Producing a variation graph using pggb

As `pggb` uses all-vs-all alignment of a collection of sequence 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 [**Guarracino2023**].

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 3.6 clearly shows the recombination happening between the three chromosomes. This work was mainly done by Simon Heumos and the graph shown in figure 3.6 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.

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

In order to produce a graph that represents variation between multiple chromosome 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.
By building a graph using `Minigraph` with the chimeric chromosome CGH and all the contigs of the other genomes assigned to chromsome C, G and H does not represent any recombination event, as can be seen in figure 3.7a. This was somewhat expected, as it is known that `Minigraph` does not also consider inversion between genomes. When the complete modified pipeline of `Minigraph-Cactus` is run, it is possible to see the tangle between the chromosomes, as shown in figures 3.7b 3.10.

3.7.3 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 3.8 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) [**syri**]. Figure 3.9 shows the detected rearrangements and duplications between the 3 chromosomes using `plotsr` [**plotsr**]. While figure 3.9 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.

3.7.4 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` [**panacus**]. `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 distribution 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 **??** 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 (1.5Mbp in total). Secondly, both growth curves show similar pattern, that seems plateauing. This is also highlighted by the fraction of new base pairs introduced by each sample, that decreases from 700k new basepairs with the new sample to less than 142k base pairs in on 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 13,2 Mbp for `pggb` and 13,6 Mbp for `Minigraph-Cactus`. The variability in the results are expected and due to different graph construction methods.

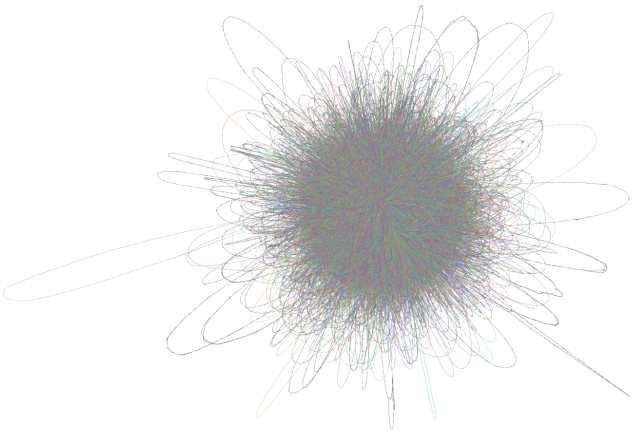
3.8 Conclusion and Perspectives

The work presented above shows how pangenomes can serve as analysis platform 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

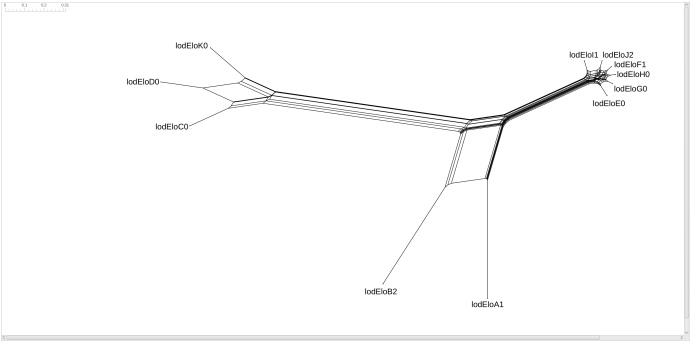
such small samples, like phylogenetic analysis shown in figure 3.4b.
Variation Graphs on the other side are powerful and very useful on few genomes analysis as they can be built quickly enough and provide a well-established platform for downstream analysis tools. On the other side, there is still need to very labor-intensive manual revision of the output graphs to find the input parameters that produce the best result, as it took more than 25 rounds to find the best `pggb` parameters to produce the graph shown in figure 3.6. As they are produced on heuristics and not on mathematically defined concepts, each variation graphs-construction tool produces different results: in figure 3.11, it is possible to see the difference 1d-representation of the small chromosome E between `pggb` and `Minigraph-Cactus`. Finally, in order to detect the inter-chromosomal rearrangement with `Minigraph-Cactus` as in figure 3.10, I had to rewrite the pipeline and modify the input data.
Apart from the aforementioned limitations, that show how such methods are still more prototypes than sound and established tools, 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 assembly stats software from Sanger Institute [**assemblystats**].



(a) `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.



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

(c) 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 figure 3.4b compacted on the left. As they are two different species, although close in the Candida/Lodderomyces clade[**lodelo_genomes**], 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 3.4: Visualization of the ccdBG representation and phylogeny analysis of the *Lodderomyces elongisporus* pangenome.

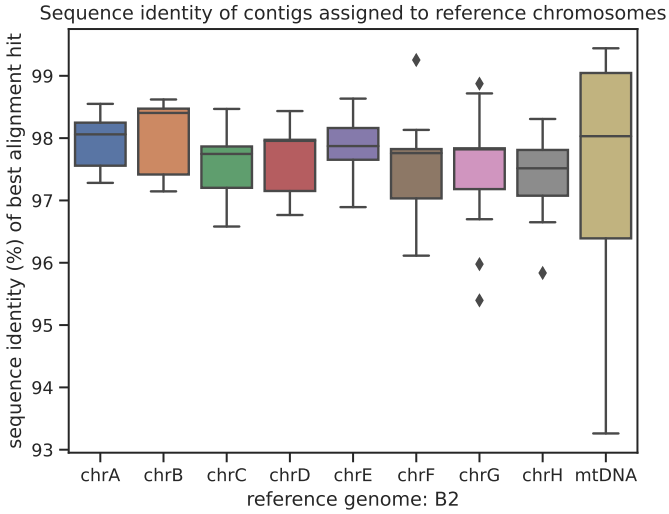


Figure 3.5: Sequence Identity of contigs assigned to reference chromosomes. Image produced using a pipeline developed by Simon Heumos.

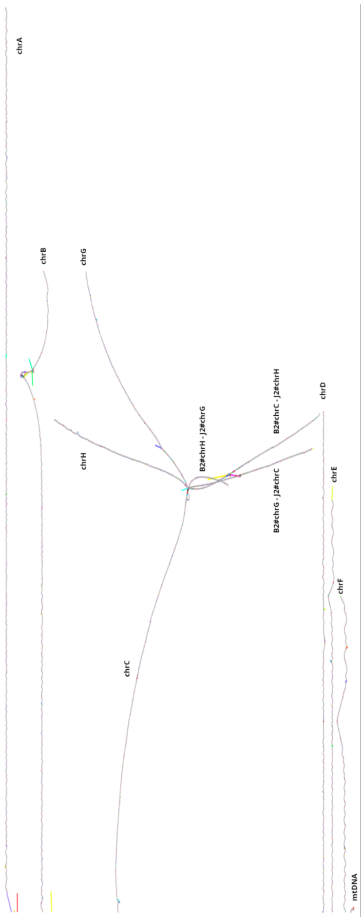
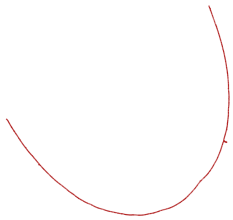
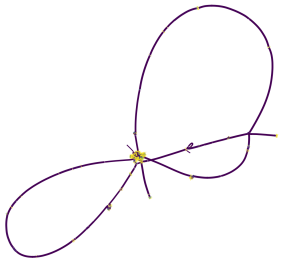


Figure 3.6: `gfaestus` visualization of the `pggb` variation graph of the 11 *Lodderomyces elongisporus* samples. Image produced by Simon Heumos.



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

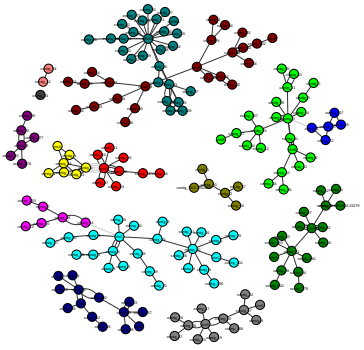


Figure 3.8: Community partition of the contigs based on all-vs-all alignment scores.

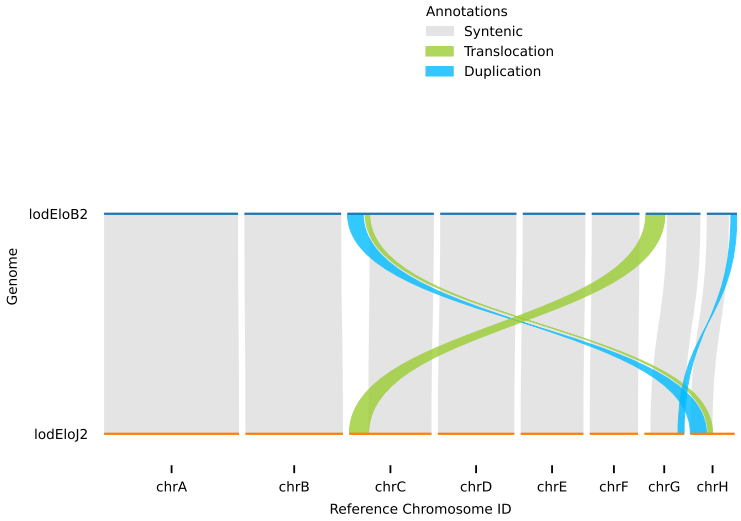


Figure 3.9: `plotsr` visualization of the inter-chromosomal recombination detected using `wfmash` and `SyRI`.

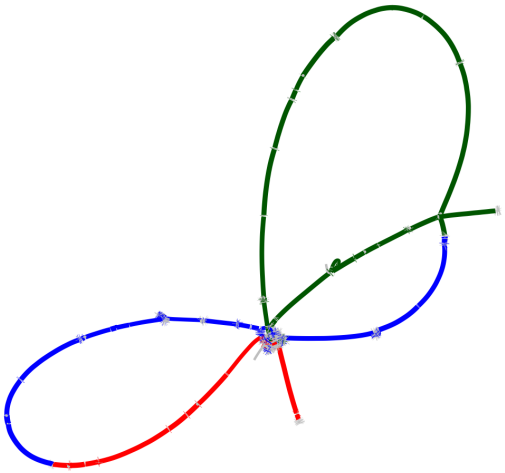
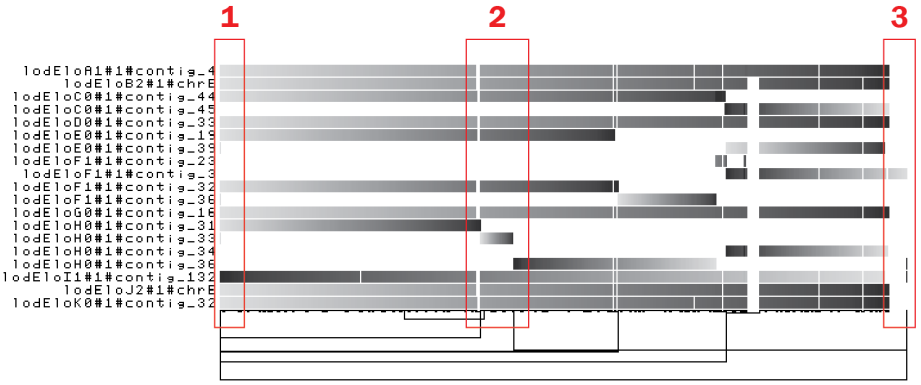
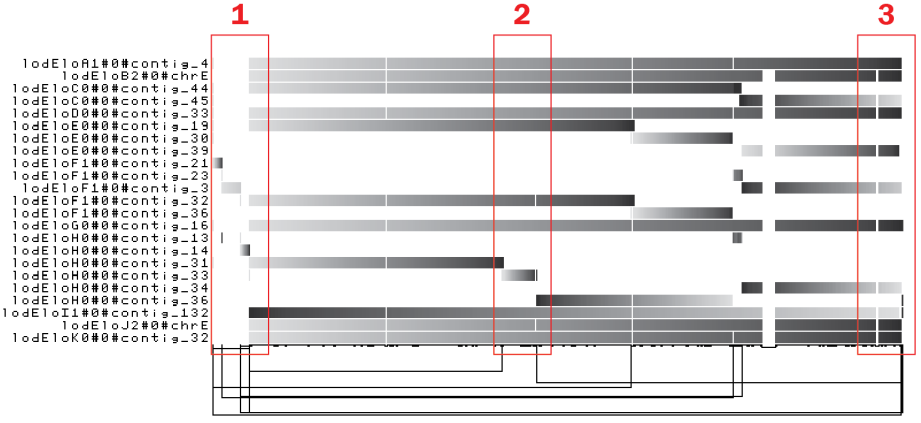


Figure 3.10: `Bandage` visualization of the tangle of chromosomes C, G and H in the `Minigraph-Cactus` variation graph. Nodes are colored based on `Minigraph` alignment of chromosome C (dark green), G (blue) and H (red) of the reference assembly B2. The three chromosomes are bound together because of 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 3.11: One dimensional visualization of chromosome E variation graphs of `pggb` and `Minigraph-Cactus` using `odgi`. Differences are highlighted by the three red boxes.

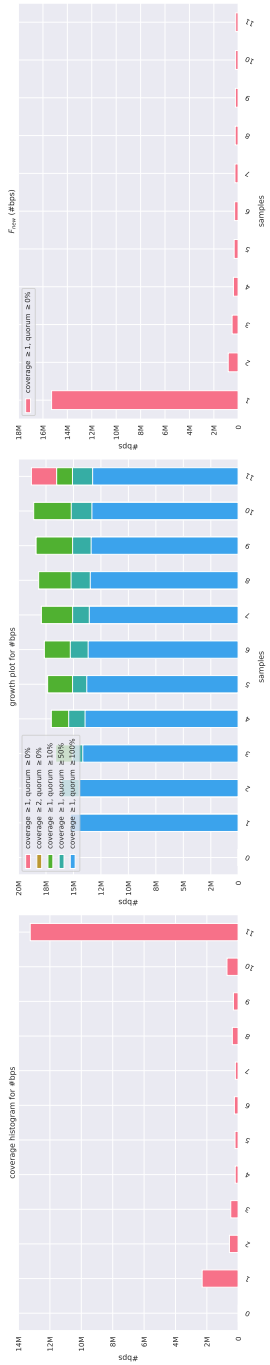


Figure 3.12: Pangenome core and growth of `pggb` variation graph. The bottom figure shows the coverage histogram in number of basepairs, the middle one shows the evolution of the pangenome growth and core by increasing sample size and the top one the new basepairs added by each new genome in the sample.

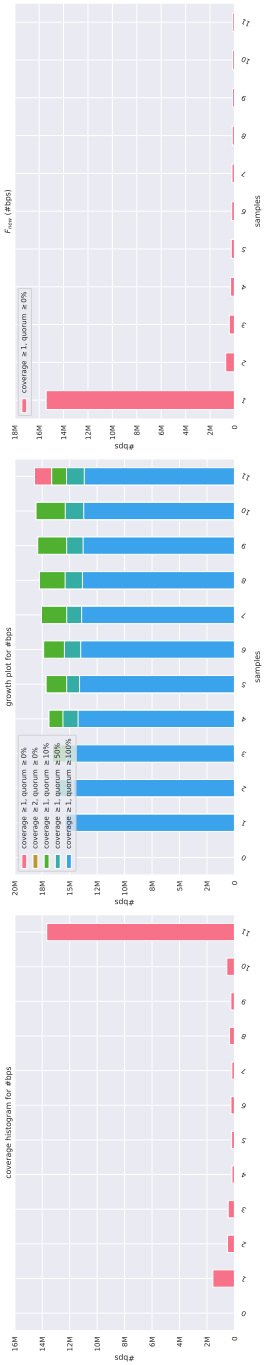


Figure 3.13: Pangenome core and growth of `Minigraph-Cactus` variation graph. The bottom figure shows the coverage histogram in number of basepairs, the middle one shows the evolution of the pangenome growth and core by increasing sample size and the top one the new basepairs added by each new genome in the sample.

Chapter 4

Exploring new *k*-mer based methods for Pangenomics

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

As discussed in the previous section of this manuscript, the construction of pangenome as variation graphs is based on an alignment step that is well known to be accurate but computationally expensive, even if recent advances on alignment algorithms and tools, like the wavefront algorithm [**wavefront**] or full-text indexes like the r-index [**spumoni2**] and move index [**movi**] have provided improvement in construction time or query performance.
The variation graph is a feasible approach for curated analysis of a selected set of samples for large genome organisms: for example, at the time of the writing of this manuscript the Human Pangenome Reference Consortium is releasing a second batch of around 220 high quality human genomes to be used for the construction of a new reference pangenome of the Human species.
Finally, the alignment step implicitly requires high quality complete assemblies to produce reasonably connected graphs. While it is expected that the availability of such high quality genomes will continue growing in the coming years, there is now available a large quantity of raw (or lightly processed) data that can be used in pangenomics applications [**serratus**, **logan**] but cannot be harnessed by variation graph models.
For this reason, *k*-mer based approaches provide a solid alternative: as the used *k*-mer length is usually relatively small (from 21 to 100), they can be used also on more fragmented assemblies or directly on raw sequencing reads and their scalability is proven to be order of magnitude superior than variation graphs. Even more so: they can be used to build representations from data of different quality like phased assemblies from one cohort and unitigs from another.
Such tools usually use different data structures to represent internally and in an efficient way a dBG model. The main challenge of these data structures is mainly the amount of space used to represent the *k*-mers versus the time used to query elements (single *k*-mers or sequences). For this reason, implementations decisions are often bound to optimization compromises made to achieve a specific goal: disk compression to produce small-sized indexes from large collections; fast query time of a novel sequence; time/memory trade-offs. In any case, the computational resources to produce ccdBG from a set of input genomes are, as shown in the previous chapter, quite lower and the tools scale to significantly larger 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 some useful application in pangenomics. Here I will present the three projects that gave birth to some relevant outcomes. On two of them, more than just working on by myself, I mostly collaborated, to different extents, with other researchers both in my unit and in other groups. While I cannot call these project as my personal contribution in the field, I believe I could bring significant input in each of them. The chapter will be 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

Data structures to represent a set of input genomes based on *k*-mers that find useful applications for pangenomics should satisfy these two main characteristics, knowing that they are, to some point, in competition:

- provide efficient storage of the data;
- allow very fast interrogations of a *k*-mer or string to report the associated stored metadata;

These process to store efficiently the input data to enable fast interrogations is called indexing, while the process of the metadata interrogation is called querying. The data structures should also be able to perform this operations for large data collections, as mentioned in the introduction of this chapter. As data repository grow at a quasi-exponential trend, it has become paramount to minimize the storage requirements and query times for *k*-mer sets.
A simple but very effective analogy of indexing and querying can be done with books and words. Let's say I remember I have a book in my library that happens to have as main character someone called Ricardo and that tells about a story that is also based in Paris. Without any organization of my library I might need to sequentially read , in the worst case, all my books from start to finish to then find the one that I was looking for. This is not convenient at all, especially if I posses a lot of books. In case I maintained an index of in which book I can find any of the words from my whole library, I could quite rapidly find the few ones that contain a character that get called Ricardito. Even more, if I had also an

index that associates places with books that have scenes based in them, I could easily triangulate, without even needing to open a single book, that the one I was looking for is *Travesuras de la niña mala* of the Nobel Price 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 1.2, 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 we 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 consider.

An 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 a) basic computer science models like dictionaries; b) cryptography; c) bioinformatics; d) many others.

hash functions should optimize on some of these properties:

Uniformity 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.

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 operation executed in the computation;

collision avoidance collisions, i.e. mapping different keys into the same hash, should be infrequent. The collision rate is proportional 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 1.2.

4.2.2 Minimum set of operations and metadata

Given a set of sequencing samples, the data structure must be able to add *k*-mers from each sample to itself with an *insert()* operation at the moment of generation of the instance of the data structure. As will be detailed in section 4.2.5, insertion after initial construction is not always a guaranteed feature.
The data structure has to be able to return the metadata associated to it, using a *memb()*operation. Metadata is a broad keyword that I will use to identify information associated to the *k*-mers represented in the data structure itself. As presence or absence of a *k*-mer in the set is usually 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.
Finally, the data structure can conserve actively or not its internal state after a membership query. For example when looking in a dictionary if an element is present, the CPU will have inside a chunk of memory containing the queried key-value pair and other ones. Other data structures store explicit variables to remember in which place of their internal representation the *memb()*operation led to. This is important to notice as most of the times, sequences and not *k*-mers are queried to the data structure, meaning that *memb()*operations are done sequentially and on *k*-mers overlapping with each other. Leveraging these properties makes huge differences in the scalability of such data structures.

4.2.2.1 Metadata types

The most trivial case it is presence of the absence of an element inside it, using a binary *memb()*operation (0 for no, 1 for yes). Other metadata that can be useful in pangenomics can be:

count If the data structure contains the number of times a *k*-mer has been seen in the input sample, the *memb()*operation will return 0 if it has never been seen, and a value >= 1 if the value has been seen 1 or multiple times. The count can be exact or represent an order of magnitude of the counts: this is often needed to not saturate the counts as most datasets have skewed *k*-mer count distribution. Counts are useful to discern copy variants number in different samples.

colors In the case it remembers in which samples a *k*-mer has been seen, the *memb()*operation will return 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 it is contained the 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 and could be mimicked by assigning numerical labels to text and use an additional map to report the text for the *memb()*operation. The id information is quite overlooked by, to my knowledge, all implementations.
Finally, the main difference between representation of *k*-mer sets and sets of *k*-mer sets is that in the second 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.

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 is useful to understand how many copies of a particular genetic sequence is repeated, hence its *k*-mer count can function as a proxy of that. The abundance of specific RNA in the cell can for example be a discriminating factor between normal and cancerous activity. The presence of a different copy number in a specific region of the DNA can discriminate between multiple phenotypes, hence highlighting differences in the samples in a pangenome: counting *k*-mers this is the only way to allow dBG models to identify the multiplicity of repetitions, while in variation graphs they are implicitly encoded in the paths.
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. 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.
Finally, part or all of these 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 necessary at the same time to enable lossless encoding of genomes in a dBG model (but they are not sufficient).

4.2.3 Basic data structures: sorted list and hash table

The most simple data structure used in computer science to maintain an ordered collection of elements to be searched in less than linear time is a sorted list of elements. By ordering the whole enumeration of the set of k-mers in each sample, one can use a binary search to find a requested k-mer in time $O(k\log n)$, using $O(kn)$ space. This is feasible for very basic cases with small set of k-mers but it is intractable for the aforementioned use-cases, as both time to query a single element or store the dataset scale too poorly. 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 this in section 4.6.
Hash-tables, a well known implementation of dictionaries (or maps) in computer science, solve the problem of the query time, bringing it to $O(k)$ or $O(1)$, depending on the particular hash function used. They still require $O(kn)$ space that makes them still unusable for large collections of data.

4.2.4 Approximate membership and filters

Approximate membership data structure offer a trade-off between the space (in memory or disk) used to store the ingested information and the probability of returning a correct answer to improve the space efficiency. While a sorted list or a hash table return always the correct information to a query, these data structures answer with a non-zero probability of false-positive (i.e. reporting a k-mer present in the raw data when in fact it is not) and zero false-negative rates (i.e. reporting a k-mer as not present while it was present). They take the name of probabilistic data structures. Finally, the filters are data structures that resemble vectors, whose basic element (also named slots) can be single bits (hence bitvectors) or any amount of bits that ensure optimal space-efficiency and that can be smaller than a machine word or a single byte using low-level implementation operations.

4.2.4.1 Bloom Filters

Bloom filters are the most used probabilistic data structures and are used in a multitude of genomic applications, like removing from ancient DNA [**akmerbroom**] or non-genomic applications (non-genomic bloom filter). 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 an 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 and return an averaged value when queried.
n which sample a Interleaved bloom filters instead are made by several bloom filters chunked together to report sample origin queries, when each filter si filled with *k*-mers from a sample.
Multiple implementation and optimization techniques, like 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 [**marchet2024kmersets**, **marchet2021kmer**, **marchet2024coloredkmersets**].

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 bits) 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 with the quotient with 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 that is not absence or presence has to be stored, like counts, multiple slots can be used to encode the count of a single remainder, like for the Counting Quotient Filter or some bits of the slot might be reserved to store the count, like 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 characteristic of data structures that represent *k*-mer sets is the possibility to modify the data contained in them after the initial construction. This division is therefore between what are called static and dynamic data structures.
A static data structure cannot be modified after construction: if a set of elements has to be added or removed from the one it was used to construct it, a new instance of the data structure has to be constructed with the modified set. These data are structure usually allow more compression of the input data, hence less space. They are suitable for applications in which a reference set is used to compare new datasets so there is no need to often modify the reference set.
A dynamic data structure allows a certain number of updating operations such that the input set it represents can be modified. A certain number of operations

can performed, depending to the application the tool is designed for. The most common operation is the insertion of a new set of *k*-mers, that is equivalent to an union operation between the two sets when there is no metadata, or in the case of a counting data structure a change of the count value (if an element is already present the count is increased). Other operations can be deletion of the *k*-mer, or modification of the metadata associated to it.
While most methods are static, dynamic structures that allow efficient insertion and, less frequently, deletion of *k*-mers are being developed in recent years [**marchet2024kmersets**]. Finally, the *k*-mer file format (kff) is a proposed framework that allows the lossless storing and manipulation of *k*-mer sets that combines space savings with interoperability across tools [**kmer-file-format**].

4.3 Our contributions: an outline

The three projects span different topics and can be devised 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 development 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, in contrast to the original one, 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 on top of it with a *Fimpera* scheme to mimic large *k*-mers while store smaller ones to store space.

A **Super*k*-mer** sorting implementation to explore a different data structure

Some of the research and development I did, mostly in the second and third projects just outlined, 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

As presented in the introduction of this manuscript, 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 (and many more are coming), others related to other genomic areas like transcriptomics with GEUVADIS and metagenomics with MetaSub and Tara Ocean. In pangenomics, large dataset present significant challenges for traditional variation graph analyses due to their size and complexity, as presented in the introduction of this chapter. k-mer-based methods can be instead used to study the data with techniques such as k-mer counting and matrix representation. These methods can lead to accuracy in abundance estimation of loci across multiple samples, paving the way for more comprehensive analyses of complex genomic datasets. One example is enabling GWAS studies on all possible variations inside genomes, as current ones focus only on SNPs and small indels. Another example is the possibility of use such matrices as training data for Deep Learning models to learn traits that discern healthy to non-healthy populations for specific diseases and so on.
For these reasons, we propose a novel method to build plain text abundance unitig matrices that can be directly used for downstream applications.

4.4.2 Related work

Cutting-edge tools that compute a cdBGs or ccdBGs form input samples have been developed in recent years. While BCALM and Cuttlefish output a 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` [**kmtricks**], 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 1.2.1, 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

The main idea behind the construction of an abundance unitig matrix is that it is now possible to construct k-mer matrices in a quite efficient way and that is is also possible to build unitigs in a quite efficient way. Therefore by compacting k-mers into unitigs and by estimating unitig abundance by averaging k-mer counts, it is possible to construct a more compact and manageable representation that preserve the high level information needed for for genomic variation diversity studies, loci variation visualization and ingestion by machine learning libraries. If abundance is not needed, presence-absence unitig matrices can still render sequence variation between individuals and be of use for diversity studies. Finally, the abundance matrix is also filtered in one of the main steps to retain only k-mers that reflect the difference between samples, while the presence-absence one does output the entire set of k-mers of the input genomes.
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 an 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

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 and unfiltered unitigs) on 360 ancient oral samples.

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. unitig matrix the ccdBG is built using ggcat. Unitigs are in FASTA format while colors are in a compress 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.

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 using the just presented 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. 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.

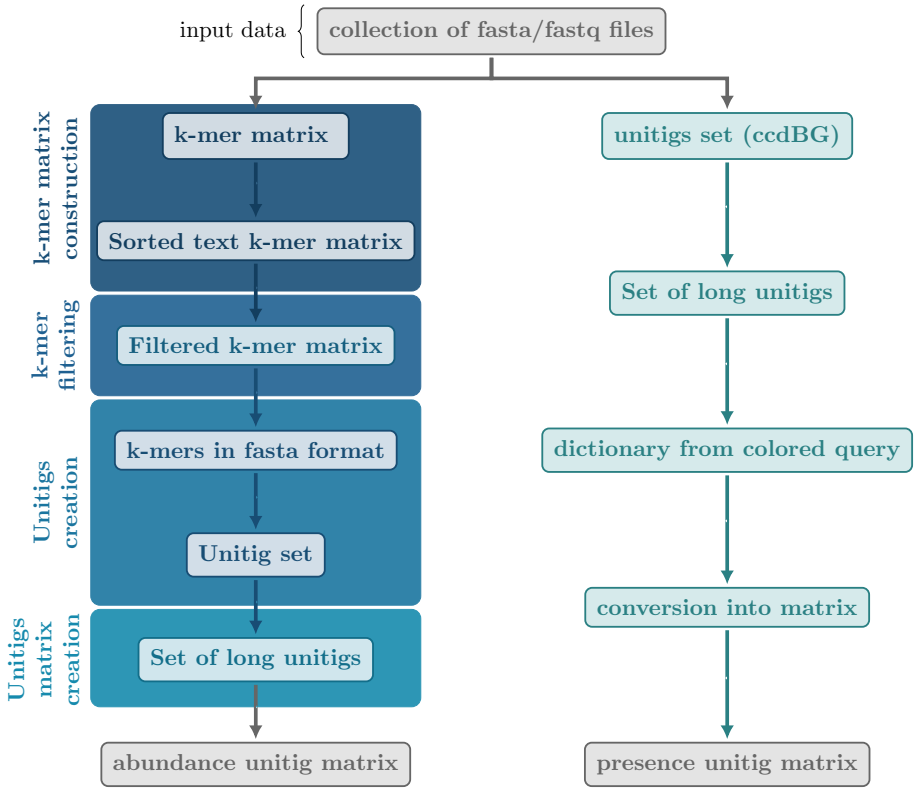


Figure 4.1: Scheme representing the main steps of the `muset` pipeline.

4.4.4 Conclusions and perspectives

4.4.4.1 Unitig matrices are pangenomes

Although not very considered in the pangenome community, matrices are a valid pangenome representation. Every genome can be seen as a binary vector in a matrix that reports the alleles of a pangenome graph. A presence-absence matrix can be inferred by both variation graphs and ccdBGs, in which as rows there are the ids of the nodes and as columns the input samples. Even if with a plain text matrix it is not possible to visualize genome variations like for graphs, they are arguably a friendlier starting point for downstream applications: biostatistics methods and population genetics can be easily generalized to this model.
Before `muset` abundance matrices could be theoretically generated only from paths in variation graphs, while it is now possible to obtain a sufficient approximation (by averaging) for unitigs in each sample.

4.4.5 Perspectives on other applications

`muset` is the first pipeline that uses various tools together to produce the unitig matrix representation. These tools have not been developed with this application in mind, except the `kmat_tool` software that is done to handle the various inputs and outputs format of the tools and for the filtering steps. This approach is important because it is the first to do so but it is not well optimized, as the pipeline uses various steps in which the data is dumped or read as plain text on disk, slowing the process. For this reason a future direction for this project would be to develop a tool to handle some or most of these operation in memory, gaining a lot of computing time.

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

While abundance unitig matrices are useful for specific genomics applications, they represent data in a way that is less efficient for others. Some use-cases require to understand if a sequence is present or absent the dataset(s) of interest in the minimum possible time. Indexing is a way to organize information in data structures that enable fast queries of the data they contain. Sequence query is done by pseudo-alignment (by k-mer izing the sequence and query each of its k-mers). As cdBGs or ccdBG can solve the task but the graph construction is a major bottleneck and require explicitly associating each k-mer with its abundance.
Approximate Membership Query (AMQ) data structures, such as Bloom filters, quotient filters, and cuckoo filters, can be used to represent a set or multi set of elements in a more efficient space. They have become essential tools not only in computational biology but also other domains like databases, storage systems and networks. Their are called approximate because they allow queries to rarely return a false positive value at a rate, δ, meaning that while they will 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 allows the AMQ to save space. More recently, there have been developed version of these that allow to count the elements in a dataset instead of just reporting the absence or presence (cAMQ). We will focus of the ladder as they are more useful in many bioinformatics applications, from pangenomics to transcriptomics and metagenomics. For counting data structures, false positive values should report a count greater and not inferior to the real one.
At the present moment, the main areas of improvement of such data structures are:

1. The latency of *memb()*operations, as it defines the kind of applications they can be used for (e.g. computer networks) and the amount of data that can be queried in a reasonable amount of time (e.g. bioinformatics). This is mostly due to a combination of the following factors:

- how well the data structure is designed in order to have data locality. Reducing cache misses by storing information such that data that might be needed consecutively fits the processor cache to save time.
- how well the implementation is coded, i.e. how engineered are certain operations, e.g. code branches, SIMD operations and multiprocessing. This is a very important aspect, as data structures that are less efficient in theory can in fact outperform in practice as they are easier to optimize.

2. The amount of space the data structure uses to store elements, that poses a hard constraint on the computer architectures that can use it. Although the cost of RAM has been declining, the amount of data is growing faster therefore pushing for more efficient use of each bit to encode information.

3. the operation they allow, especially the possibility to modify the data structure after it has been initialized on some dataset. The most useful operations are:

   - increase the count of elements, or add them if not already inside;
   - decrease the count of elements, or delete them if their count reaches 0;
   - enumerate the elements inside with their respective count;
   - automatically resize the filter when it reaches maximum capacity, to avoid the necessity to use a cardinality estimation tool a priori when initializing the data structure and allow for addition of new elements.

A data structure that tackles relatively efficiently all these problems has been proposed and takes the name of Counting Quotient Filter or CQF. It is based on a Rank-Select quotient filter on top of which a counting scheme has been devised in order to fill the slots of the filter with an efficient encoding of the count of the inserted element that offers less memory usage and lookup speedup compared to a Quotient Filter.

4.5.1 Brief state-of-the-art overview

The AMQ data structure that can be used in these applications are variations of three main ones: the Bloom filter, the Quotient Filter and the cuckoo filter. Although the first two have been described at the beginning of this chapter, here I briefly present them with the implementations built on top of them.

Definition 4.5.1.

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.

About all these implementations, it is important to remember that they provide a fast lookup table in which information can be stored into fixed-size slots.
Finally, from now on I will discuss only about implementation details of Quotient Filter, as it is the basic data structure that we considered for out implementation. When a new element has to be inserted in a quotient filter, it is first hashed and then the resulting value is separated into a quotient of length q and a remainder of length r.

4.5.1.1 Quotient Filter structure: Rank and Select

The Ransk-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 store the remainder of each item. When inserting an item, the filter tries to place the remainder in the slot based on the quotient. If the home slot is occupied, a linear probing technique is used to locate the next available slot. To help tracking where the runs, i.e. the collections of subsequent slots containing the remainders of a specific quotient, are, the the RSQF uses two metadata bitvectors:

occupieds Tracks which slots are currently occupied by data.

runends Indicates the end of each set of consecutive entries (or a "run") in the quotient filter.

The combination of these two bit vectors allows the RSQF to efficiently find and manage inserted items by using rank and select operations. Specifically, the `RANK` function counts the number of occupied slots up to a certain point, and `SELECT` identifies where in the filter a particular run ends. This allows efficient lookup, insertion, and enumeration of the data in the filter.
Moreover, to store the data in a cache-friendly way, the filter is divided 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 for every 64th slot. Therefore computing these offsets involves scanning only small sections of the metadata (64 bits or fewer) per operation, making the filter significantly faster.
Each block fragments both the slots vector and the metadata vectors. It contains therefore one offset value, 64 occupieds, 64 runends, and 64 slots for remainders data. By storing these elements together, the system should minimize memory access and enhances cache efficiency.

4.5.1.2 Counting

Counting data structures, depending on the application, can store exact counts or order of magnitudes. This because in some applications the count number is expected relatively low and precision is important (like human genomics) while in others skewed abundance is more probable and an estimate of the count is good enough (for example in metagenomics).
Finally, counts can be stored with three different strategies. In any case the reminders associated to a certain quotient are stored in monotonic order inside the data structure.

1. a count can be encoded as the number of times a reminder is inserted into consecutive slots. This is the most basic implementation and the less space-efficient one as the data structure occupation would be related to the total number of counts in it. It also makes poor use of the bits in the slots that could be used for count or not.

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

   - if $1 <= N <= 2$, than the reminder R is inserted N times;
   - else $K = C - 2$ can be encoded into a sequence of slots, whole boundaries are flagged by two reminders R (hence the - 2). The encoding uses the slots in between to store the actual value of the count K. If K is greater than R, a 0 is placed just after the first reminder to signal that the next slots are used for encoding, else not as the monotonic insertion of the reminders would implicitly flag the count. If K is greater than the max value that can be encoded in the r bits of a slot (i.e. $2^r - 1$), it is encoded as a sum of the slots containing the count.

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

3. another way of storing c 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.

4.5.2 Developing a new library for a Quotient Filter

The implementation of the proposed CQF is efficient but represent an object that is not modifiable and does not allow for experimentation of different models based on the RSQF or CQF. For this reason, we decided to re implement such data structure in a modular way so that the basic RSQF data structure could be used as building block for multiple applications. In this way it would be possible to develop models:

- exact (when no space trade-off is chosen) or approximate;
- with exact or approximate counts;
- with different count encoding;

The developed data structure is therefore layered as this:

low level it 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 whatever metadata as it only imposes bits to a certain region of memory corresponding to a slot;
- clearing of a slot to zeros for a) removals of elements from the filter and b) shifting operations;
- reading the value at a specific slot;
- shift of slots a certain amount y of slots from a position x of z positions. This allow to keep elements when a new one has to be inserted in an already occupied position. By shifting right of z slots the y already set slots, the position x to $x + z - 1$ are therefore free to be used.
- metadata operation to set to 1 or to 0 the occupied and runends bits in their bitvector to keep track of which slots are in use.
- operation to adjourn the offset vector to keep track of the runs.

medium level it comprises operation to insert elements without explicitly caring about handling of metadata and shifts in the slots. 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;

high level it comprises operations done on top of the RSQF.

- The addition and removal are extension of the RSQF to allow for multiple operations together (i.e. adding or removing multiple slots to store the encoded counter).

- a function to encode and decode counters, as described in point 2 of section 4.5.1.2.
- the query uses an intelligent linear probe that recognizes when a counter is starting and seeks the start and end of the counter of the queried reminder.

application specific application specific functions like

- hashing of *k*-mers
- initialization of the data structure
- enumeration of the elements inside the data structure;
- dynamic resize of the data structure (it comprises the enumeration of the elements, doubling the size of the filter by moving a bit from the remainder to the quotient and re-inserting all the elements);

This has been developed as a library or as standalone software to use.

4.5.2.1 Allowing multiple types of counts: CQF and BQF

While I focused mostly on re-implementing a CQF from the RSQF implementation, the basic implementation 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 that is called the Backpack Quotient Filter (BQF), an implementation that uses the count encoding presented in point 3 of paragraph 4.5.1.2.
This proves the flexibility of the proposed implementation.

4.5.2.2 Handling toricity

One major property of the filter that is never mentioned in the original implementation of the CQF is the handling of specific cases. Remember than runs of reminders (or counts or combination of both) associated to the same quotient are stored contiguously in a monotonic order. Moreover this has to be done in increasing slot id order. For example, when in an empty filter two elements with the same quotient are added, the slots used are the one associated with the quotient and the one *on the right*, or better the one associated to the quotient+1. And so on.
The problem arises on the *rightmost* or final part of the filter, where if new elements are added, it is probable that at some point a slot should be pushed on the next element of the final slot. To overcome this issue, the filter has a toroid structure, i.e. the next slot from the last slot is the first slot of the filter. This property avoids any problem associated with filling the filter in the final slots as it imposes a equal property to any slots of the filter.
Implementing the filter with such a characteristic is nontrivial as the toroid property imposes a different handling of all the comparisons inside the functions and the shifting operations. [FIGURE HERE]

4.5.2.3 Using the Fimpera scheme to reduce space

The size of a RSQF data structure is given by $2^q * (r + m)$ bits with m 2.125 metadata bits (and some other relatively negligible overhead). It gives that if there could be a way to store almost the same input information by reducing r, this would provide great space savings and allow the use of the data structure on larger datasets. To this end, bot the BQF and the CQF implementations have been developed with the Fimpera [**fimpera**] scheme on top. Fimpera splits each k-mer into smaller *s-mers* and stores them into the filter (each of the the k-mer count). The k-mer abundance can be therefore retrieved through its constituent *s-mers* as if a k-mer is present, so are 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 [**fimpera**].In this case, Fimpera is used to reduce the dimension of the filter without increasing the false-positive rate, as storing smaller hashes from *s-mers* gives smaller r.
To estimate the correct abundance of a k-mer, the smallest The major drawback of this scheme is the loss of the k-mer enumeration feature, as only the *s-mers* can be retrieved. The dynamic resizing of the data structure is instead preserved as only *s-mers* are needed for that.
Finally, it is possible that false poisitive k-mers are introduced: k-mers that are non present in the dataset made by *s-mers* that are instead found in other k-mers are going to be reported as present. As this is a joint probability, it is the result of the multiplication of each independent probability so when s is large enough it is very low. This means that the s parameter has to be chosen as a trade-off between space efficiency and false positive rates. For the BQF, this has been estimated as under $10^{-5}\%$ with $s > 20$ when $k = 32$. This is a reasonable false positive rate.

4.5.3 Conclusions and perspectives

This implementation could allow for other metadata to be inserted, that would render it no more a cAMQ but could for example contain color information for specific pangenomics applications. This is nontrivial as color storing is memory expensive and a new encoding would be needed. A possible lossless direction would be to use Shannon coding for the colors. If the filter is built in a way that needs multiple resizes, at each resize one could evaluate the distribution of colors and encode the most probable ones with few bits while the less recurrent ones could be encoded with less compression.
Finally, this implementation has not found a way as a standalone publication as the actual CQF implementation (without Fimpera) was slower than the, certainly better optimized, original one. Nevertheless there is value in prototyping for the community data structures that are more open to customization depending on the specific application.
The BQF implementation has been presented at RECOMBseq [**recombseq**] conference and will be soon published in the associated journal [**bqf**].

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

As seen until now in this chapter, there is no single recipe to represent k-mers in an efficient way. It depends on the specific application that is addressed. For example when comparing two different sets of datasets, one possibility is, for each set, to enumerate all the k-mers and then store them in a sorted manner in a list. The difference between the two sets will be therefore be estimated with a set metric (like the Jaccard index) that does take into account the difference between the k-mers in the two lists. This is straightforward when the two lists are sorted. Sorted lists of k-mers are also a quite fast representation for k-mer queries without indexing. Through binary search it is possible to speed the query time to $log(N)$ with N being the cardinality of the set.
The advantages of sorted k-mer lists are the absence of the indexation step, the predisposition for set comparison operations and the relatively fast query of the k-mer into the list. The drawback is that this representation is very expensive in terms of space. To partially overcome this issue, one can look at another side of k-mer research that is the space compression of k-mer by encoding them within a string set, i.e. all the *-tigs*. The rationale is to build a set of strings in which all enumerated k-mers and nothing else can be spelled. In recent years various models have been proposed, among which unitigs, eulertigs and simplitings. Although they provide efficient storing of k-mers, sometimes together with relative biological meaning (like unitigs), they are not easy to directly query [**marchet2024kmersets**].
Here we propose another data structure that is in between the two sides. Its aim is to:

1. encode k-mers into longer strings in order to save space;
2. maintain the sorted list structure for relative fast query of non-indexed elements.

To do so, we build a super$k$-mer sorted list.
Super$k$-mers are sequences of adjacent k-mers that share the same minimizer.
A known uses of super$k$-mers is to used them in hash tables: grouping super$k$-mers by their minimizer enables rapid membership queries. To perform a query, one finds the set of super$k$-mer linked to the minimizer of the query k-mer and checks if the query k-mer appears as a sub-string within those super$k$-mer. Implementations like `BLight` [**blight**] and the newer `ssHash` [**sshash**], which also has an extension that supports k-mer counts, use Minimal Perfect Hash Functions (MPHF) to facilitate mapping of minimizers to their corresponding super$k$-mers. The problem of using MPHF is that they generate a static dictionary that cannot be updated [**smsketch**]. For this reason we generate a dynamic data structure that uses sorted lists of super$k$-mers to store k-mer sets and, optionally, their count directly from a sequence set. Finally, another advantage of storing super$k$-mers is that, since consecutive k-mers are stored together, querying sequences is quite fast, as all the ones inside the same super$k$-mer will be queried very fast.

4.6.1 Super*k*-mer sorted lists: Input ad output

The super$k$-mer sorting algorithm we propose is encapsulated inside a larger super$k$-mer data structure development I contributed to, whose details are going to be omitted for the sake of space, that already processes a set of sequences and to output an enumeration of super$k$-mers. Therefore the input of the algorithm is going to be a list of pre-computed super$k$-mers. The output is instead a list of sorted super$k$-mers to be used for fast lookup. While most of the operation are going to be displayed in text format, they space in which the algorithm works is binary. This is done by multiple reasons, i.e. for the parsimony of space and for fast machine operations and comparisons.

4.6.2 Super*k*-mer list model

In order to understand the steps of the sorting algorithm, it is helpful to imagine the super$k$-mer list not only as a succession of objects representing the strings but also as a matrix. The matrix is defined by 2 parameters: the number of super$k$-mers N and the maximum length of a super$k$-mer M. M is defined as $M = 2k - m$ with k as the length of k-mer and m as the length of the minimizer. In this model the rows are the distinct strings and the columns identify a precise position in them. As not all su per$k$-mers are going to be maximal,it is possible that in a certain position described by a column some super$k$-mer won't contain any character. This information is not explicitly encoded in the matrix but instead in the object that represent the super$k$-mer. Figure XXX shows the equivalences between the two models. Most of the steps of the algorithm can be considered as operations on the columns of such matrix. Another important part of this model is that at each columns position a super$k$-mer can have the first nucleotide of a valid k-mer or not. This depends if at that position it has a nucleotide and the next $k - 1$ columns contain also valid nucleotides. If not, that column position is not valid for a k-mer.

4.6.3 Sorting *k*-mers in the same position

The fist step in the sorting algorithm requires to produce a sorted list of super$k$-mer ids for each column of the matrix. The sorting is done by comparing the valid k-mers at the column position using as ordering function the value of the hash of the k-mer. If the hash is smaller, it comes first.
To do so, a first scan over the input super$k$-mer is done to select the super$k$-mer ids of the ones that have a valid k-mer at the column position. Then, the ordering is done overt the selected list by comparing the k-mer hashes and finally the list of sorted super$k$-mer ids is returned. This process is done for every column of the matrix.

4.6.4 Returning overlaps between *k*-mers

The next step is, for each pairs of consecutive columns, to detect the overlap between the $k - 1$ suffix of a k-mer of the first column with the $k - 1$ prefix

of the subsequent. This process therefore done independently on each possible pair of consecutive columns. The $k-1$ prefixes of the k-mers of the second column are inserted in a vector. Then for each $k-1$ suffix computed from the first column, the matching value is searched in the prefix vector. If found, the pair of super$k$-mer ids of the first and second column is inserted in the list of candidate overlaps. Overlap is possible between a single element of one column and multiple elements of another column. In this case, all the possible overlaps are reported. At the end of the scan, each pair of column will have the list of candidate overlaps between the two.

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

In the previous step all the possible k-mer overlap pairs are computed. As the ordering of the k-mers in the columns has to be respected to produce a sorted list, it naturally follows that it is possible that some overlaps won't be compatible between each other because one of the two situations occur:

a the same k-mer overlaps with more than one k-mer of the the other column, but can be used just once;

b pairs that (if visualized as edges between positions in the column) cross each other cannot be used together because they would not respect the relative order of the columns.

Figure XXX shows and example of non valid pairs and of a maximal chain while a more formal definition is provided below. To choose the maximal set of "non-crossing" pairs between the ones calculated at the previous step, we use co-linear chaining.
Co-linear chaining is an algorithmic technique that is known to be used in alignment algorithms. In the context of pairing elements in a sequence, it can be summarized as finding pairs of connected elements such that no "crossing" connections occur when visualized geometrically.
When aligning a read to a reference sequence, it takes as input pairs of maximal exact matches (MEMs). It then computes a chain of pairs such that the order of the selected pairs is concordant with the order in which they appear in the two strings while maximizing the amount of bases covered by the chain in the read. Lately it has been also used on alignment of sequences to graph in pangenomics applications.

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

- two ordered lists of super$k$-mer ids of two contiguous columns computed as in section 4.6.3.
  - $A = \{a_1, a_2, ..., a_n\}, |A| = N$, representing the left column,
  - $B = \{b_1, b_2, ..., b_m\}, |B| = M$, representing the right column;

- a set of tuples $V = \{(v_1, w_1), (v_2, w_2), ..., (v_k, w_k)\}$ such as $v_i \in A$ and $w_i \in B$ and (v_i, w_i) represents an overlap between k-mers of the two contiguous columns, as computed in section 4.6.4;

The goal is to find the maximal list of tuples U, with $set(U) \subseteq V$ such that

- if $v_i \prec v_j$ in $U \wedge v_i = a_i, v_j = a_j \implies a_i \prec a_j$ in A, and the same for B;
- $\forall v, w \in U | v_i = a_x \vee w_i = b_y \implies \nexists v_j ! = v_i | v_j = a_x \wedge \nexists w_j ! = w_i | w_j = b_y$.

The first condition implies that the ordering of the column lists A, B is respected in U, while the second implies that each super$k$-mer id from A can occur only once in the list of tuples U and the same for B.

This problem is solved using dynamic programming [**genome_scale**]. First the tuples in V are sorted by their order in B. This guarantees that no "crossing" happens in the w ids because $w_i = b_i : b_i \prec b_j$ will always be processed before $w_j = b_j$. To break ties on equal v values, the ordering of $v_i = a_i$ in A is used. Then the dynamic programming is done over the A ids to find the largest set of pairs where the A ids form a non-decreasing sequence. The score is therefore stored for each element v_i in the list of pairs and a table C of length M is filled, in which index j gives the maximum possible score using tuples from V such that $(v, w) has w \in \{b_1, ..., b_j\}$. For any tuple a recurrence is obtained depending if it violates or not the conditions above. To calculate the recurrence,

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

if (v_j, w_j) does not overlap with $V_{j'} \subseteq \{\}(v_1, w_1), ..., (v_{j-1}, w_{j-1})\}$ i.e. the non-overlapping subset selected in the previous step. To optimize the search of the best solutions j' between the ones already computed, a binary search tree that contains the best score for each value $w \in A$ is queried and updated, with $O(\log n)$ complexity. The search tree contains the w values sorted by A ordering. The lookup is $O(\log n)$ From this follows that the co-linear chaining costs $O(n \log n)$.

4.6.6 Reconciliation and final output

Once the lists of non "crossing" overlaps for each pairs have been computed using the co-linear chaining algorithm, the next step is to reconcile their information to output the sorted list. To do so, consider the matrix representation of the super$k$-mers, in which for each column the k-mers are sorted from top to bottom. Using the lists of non-crossing overlaps, each k-mer in the matrix is added in a map as key with value an id associated to which super$k$-mer it is going to be inserted. Take the case in which one of the overlap lists returns the pair (x,y) and the subsequent returns the pair (y,z). This means that k-mers x,y and z have to be compacted into the same super$k$-mer in the sorted list and are therefore going to be inserted in the map with the same value, e.g. 1. Other k-mers not to be compacted in the same super$k$-mer will have different values.

To output the list in an ordered way, each column of the matrix is pointed by a pointer that tracks the row in which there is the next k-mer to be inserted. A k-mer or a super$k$-mer is inserted in the final sorted list by iterating over the pointers. If a k-mer or super$k$-mer has all its elements flagged by a pointer, it is inserted in a list. Ties are broken by directly comparing the two super$k$-mers and inserting first the one with smallest hash. After an element is inserted, the pointers on top of its k-mers are moved to the row below. This process ir repeated until all pointer reach the last element of their column.

4.6.7 Searching the list

As described at the beginning of this section, the sorted super$k$-mer list produced by the algorithm can be used for relatively fast search without indexing the k-mers. To this end, a binary search is a fast strategy to query k-mers inside the list. Binary search is an efficient algorithm used to find the position of a target value within a sorted array. It repeatedly divides the search interval in half, comparing the middle element to the target. If the middle element matches the target, the search ends. If the target is smaller, it searches the left half, and if it's larger, it searches the right half. This process continues until the element is found or the search interval is empty.
When querying a k-mer in the super$k$-mer list, the search algorithm works 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 the used super$k$-mer representation is the search algorithm jumps an element (the super$k$-mer) and then looks if at a specific position, that we will call offset, there is a k-mer. By knowing the minimizer position of queried k-mer, the query can be done directly to a specific offset instead of doing a linear probe on the entire super$k$-mer.
The drawback is that, since not all super$k$-mers will be maximal, some queries on a specific super$k$-mer won't be possible, as they won't have a k-mer in the searched position. An offset will be valid if the super$k$-mer contains a k-mer in that position and invalid if it does not. To address this issue, when a lookup is done on a invalid offset of a super$k$-mer, the lookup will move as a linear probe on the previous or subsequent elements of the list until finds the first super$k$-mer having that offset valid. [Could be useful to have a figure here].

4.6.8 Conclusion and future work

This project culminated with a prototype data structure to represent a k-mer set as a sorted list of super$k$-mers. The advantages of using such a representation are:

- it can enumerate the k-mers in the set, differently from some hashing data structures (see bloom filters);
- it can compress k-mers into super$k$-mers thus reducing the space used to store them compared to a ordered list of k-mers;
- it allows fast direct queries, compared to other *tigs* representations;
- it allows metadata to be added by a separate data structure;

This work has not been published yet but we count to close it and submit it in the coming months.

4.6.8.1 Possible improvement

As just described, the super$k$-mer sorted list can be queried using a variation of binary search in which, when the next super$k$-mer to be searched has not a valid offset, it has to linear probe the elements in the list for one that does. This slows the query operation, especially in cases where the super$k$-mers with a valid offset are sparse.
A future improvement of this algorithm would be to mitigate this issue, by adding information on non valid offsets to direct the binary search on the right direction without doing a linear probing. This strategy can be implemented by fill the bits of the super$k$-mer left blank when a k-mer is not stored in an informative way.
Given two super$k$-mers S_1 and S_2 in a super$k$-mer sorted list and an offset t. If there are n super$k$-mers between S_1 and S_2 that do not have a valid k-mer at

that position t while S_1 and S_2 do. The strategy is to fill the bits not used by k-mers in the super$k$-mer with "fake" nucleotides that do not serve as genetic information but that help the search by giving information on where to find the queried k-mer. This can be done by filling the empty bits with an average between the value found in S_1 and S_2 at the same position, starting to fill the bits from the most significant one, for all n super$k$-mers. Another approach would also take into account the cardinality n of the elements to fill and, instead of filling all the super$k$-mers with the same averaged value, it would fill the n elements with progressive values from S_1 to S_2.

4.7 Does it fit to have a general conclusion of the whole chapter here?

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,Pierre Peterlongo. In: *iScience*. Vol. TOFILL, no. TOFILL (2024), pp. TOFILL. DOI: TOFILL.

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*. Vol. TOFILL, no. TOFILL (2024), pp.TOFILL. DOI: TOFILL.

Analysis of metagenomic data

TOFILL "". In: *Nature Reviews Methods Primers*. Vol. TOFILL, no. TOFILL (2024), pp.TOFILL. DOI: TOFILL.

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.

SV pangenome hackaton

TOFILL "". In: *TOFILL*. Vol. TOFILL, no. TOFILL (2024), pp.TOFILL. DOI: TOFILL.

Chapter 6

Perspectives and future work

6.1 On human pangenomics: graphs and beyond

The result of the analysis I conducted in the first phase of my PhD, presented in chapter 3 serves as basis to understand what are the features, the limitations and the usefulness of the software that is currently used or developed to build pangenome graphs. These are based upon the latest developments in terms of computer science algorithms to provide the best computational performance now possible and represents a huge leap compared to the currently standard software used for genomic analysis. Here I will present a few considerations and perspectives that stem from this as well as from 2 more years of thoughts and discussions with peers of my doctoral program, my supervisors and other colleagues and experts in the field.

As we are possibly at the beginning of a change of paradigm between linear reference sequences and genomics analysis to pangenome references and pangenomics analysis, there are a few things that need to be addressed as soon as possible.

Reproducibility and stability of computation has to be the main focus of the next years for pangenome reference software developing.

In the case of leading general-purpose pangenome reference building tools, like `pggb` and `Minigraph-Cactus`, that produce variation graphs, it is of upmost importance that the graph generated from a set of sequences is exactly the same when the same data is fed as input. This means that the heuristics used to generate the variation graph are independent of the ordering of the input sequence and do not contain any stochastic process that might alter the structure of the graph. If a tool can produce two variation graph that can spell the same input genomes but that do not have the same internal order, downstream analysis, like read mapping, loci visualization and other application become biased toward the graph, making it less desirable to genomic analysts that rely on the stability of a linear reference. Current liftover and graph-mapping solutions, in my view, can only be a temporary solution if pangenomics is to be adopted and fully accepted in the genomics and genetics field.

There should be guarantees or estimates on the overall biological correctness of pangenome graphs.

While `Minigraph-Cactus` omits centromeric variation, `pggb` does at the cost of producing more complex graphs. The trade-off is not trivial as gaining on "variation resolution" leads, also, to graphs that are more difficult to interpret, especially as the number of input genomes increases. Moreover, De Bruijn Graphs are difficult to untangle and understand already at small case. A very useful and interesting future development would be to design a method to evaluate

thoroughly the (computational and biological) quality of the pangenomic data structure produced. This tool would be a necessary Quality Control (QC) step in all custom-pangenome based analysis. For human pangenomics, this would be useful for application where a different reference compared to the HPRC precomputed one is needed, for other cases, like bacteria, virus or fungi, where only specific strains and not the entire species is to be considered.

De Bruijn Graph methods need a common color file format or interface to push the development of application-specific tools. Mathematically clear, computationally efficient and output-stable de Bruijn Graph methods, like the ones that use colored-compacted dBGs, won't succeed in being real alternatives of alignment based software to perform pangenomic analysis if there won't be a consensus between the main developers on at least a minimum common interface that let users write tools to exploit the information they contain. Standardized file formats [**kff**] and interfaces for (colored) queries would help other researchers commit into developing tools for k-mer based approaches, independently of the latest tool in the scene. At the current moment, the landscape is quite diverse and new tool are constantly being developed, discouraging, in my view, the needed investment of resources to develop tools for dbg-based downstream analysis. Writing software for genomics application of k-mer based pangenome representations is crucial to make this representation useful to the end users. As the representation of references is better suited to variation graphs, applications of k-mer based tools could provide added value on genomic studies of specific (sub-)populations.

Graphs are not the only *k*-mer based pangenome representation. For this specific use-case, other data structure based on k-mers can be used to extract valuable insights. As already described, unitig matrices are a new powerful example of k-mer based data structures that can represent the genetic content of a population and its diversity. k-mer matrices [**kmtricks**] can be seen as a pangenome where rows are the k-mers present in the whole populations and columns are the specific individuals. Then the value in the matrix could be an absence/presence binary value, defining a *de facto* equivalent representation of a colored dBG. Unitig matrices then contain the same information of a ccdBG.
In my opinion there is great value in being able to demonstrate equivalence between such representation and to develop tools to change of representation such that, depending on the specific application, each user can decide which is of more interest and not be limited by specific tools building specifically formatted data structures. This is another interesting path forward in the field.

6.2 Exploring *k*-mer data structures for pangenomics

Chapter 7

Conclusions

This manuscript is the result of 40 months of doctoral journey. The common theme is the use of k-mer based algorithms and tools to tackle complex genomic problems that consider multiple different samples at the same time. This effort was always driven by the desire to produce software or analyses that would help answer biological questions.

It hasn't been a straightforward path as many projects I embarked in did not get to a stage of meaningful contribution to the scientific community. Nevertheless such efforts taught me important lessons about working alone or with other colleagues and strengthened my understanding of other genomics and bioinformatics fields. I reckon that the scope of this doctoral thesis might seem quite broad, starting with an analysis of computational pangenomics methods for human genomes representations and ending with data structures to represent k-mers in a cache efficient way. In my view this is the result of constant curiosity about the whole sequence bioinformatics problems tackled by the team where I was and the reflection of a comprehensive and multifaceted approach to the challenges of the field.

We proposed an analysis on the construction and representation of pangenome graph from high quality human haploid assemblies was well received by the community as it shed light on the characteristics of both their internal representation and the methods to generate them. It stresses the importance of selecting the kind of graph that best fits the particular application, specifically in the way it represents variations in the DNA sequence of the individuals. Variation graphs are better to perform specific downstream analysis and are more intuitive to understand, manipulate, visualize and analyze while de Bruijn Graphs are more efficient to generate, scale better and give guarantees on the preservation of the input sequences. Finally, this work stresses the difficulties of proposing a one-fits-all solution and points out areas of research that would make k-mer based approaches more attractive to the genomic community. I believe pangenomics is the key of solving many issues and deficiencies present in current genomics approaches: it is a novel area that is evolving now and will need much more effort to produce viable solutions to all the genomics tasks biased by the use of a single reference sequence.

My work on the Backpack Quotient Filter has been focused on coding the underlying data structure, the Rank Select Quotient filter, in a way that could be easily used and manipulated for different high level interfaces that would implement different filters. The final method proposed on top of this data structure a particular encoding of the counting information together with the integration of the Fimpera scheme for k-mer storing and retrieving. My work lied on the demonstration of the efficacy of the implementation I wrote by

recreating on top of it the Counting Quotient Filter as originally conceived together with the Fimpera scheme. This demonstrated that implementation differences affect the magnitude of the data such data structures can analyze, even if the information provided as output is the same. This work served as confirmation that improvements in the representation ok k-mer sets can drastically change the way in which large data collections can be interrogated and explored, even if very complex. In order to improve in the future such analysis power, new tools will require enhanced methods and refined coding techniques to exploit the maximum out of computational power to analyze exponentially increasing amounts of data.
Finally, I gave my contribution to other projects, like `muset`, that is a pipeline for the construction of uniting matrices (both abundance and presence/absence). This is the first effort to produce such data structure and is an advancement from k-mer matrices, that contained almost the same information while being more complex to analyze. This data structure can serve different applications, from pangenomics to metagenomics data as well as cancer transcriptomics, as it provides a very clear representation of the genetic content of multiple samples and their difference that is easily accessible by data analysis and machine learning tools.
The tools, pipelines and data structures I analyzed or coded are still not the definitive answer to any of the problems that pangenomic and metagenomics still face but are a clear demonstration of what are good steps in the right direction. In the future there is plenty of work to do. In the pangenomic field, methods to demonstrate the biological correctness of pangenome graphs, like it is now done with assemblies, as well to scale pangenomic representation to handle large collections of eukaryotes.

Acronyms

DNA Deoxyribonucleic acid.

dsDNA double stranded DNA.

HOR Higher Order Repeat.

NGS Next Generation Sequencing.

NIH National Institute of Health.

ONT Oxford Nanopore Technologies.

PCR Polymerase Chain Reaction.

QC Quality Control.

RNA Ribonucleic acid.

ssDNA single stranded DNA.

TGS Third Generation Sequencing.