Université de Lille

Thèse

pour obtenir le grade de :

Docteur de l'Université de Lille

dans la spécialité

« Bio-informatique »

par

Thomas Baudeau

Studying the properties of viral long reads mapping methods

Etude des propriétés des méthodes de mapping de lectures longues virales

Thèse soutenue le 30 juin 2025 devant le jury composé de :

M.	Cédric Notredame	Senior Group Leader, Notredame Lab , Centre for Genomic Regulation	(Rapporteur)
M.	Dominique Lavenier	Directeur de recherche, IRISA , CNRS	(Rapporteur)
Mme.	Anne Goffard	Professeure - praticien hospitalier, CIIL, Université de Lille	(Présidente du jury)
Mme.	Jasmijn Baaijens	Assistant professor, Bioinformatics Lab, Delft University of Technology	(Examinatrice)
Mme.	Camille Marchet	Chargée de Recherche, CRIStAL, CNRS	(Encadrante)
M.	Mikael Salson	Maître de Conférences, CRIStAL, Université de Lille	(Directeur de Thèse)

Laboratoire CRIStAL
Bâtiment ESPRIT
Avenue Henri Poincaré
59655 Villeneuve d'Ascq
France

Résumé

L'émergence des technologies de séquençage de troisième génération, en particulier les lectures longues (long reads) produites par Oxford Nanopore Technologies et Pacific Biosciences, a profondément modifié les stratégies d'alignement des séquences en bioinformatique. Initialement conçus pour traiter des lectures courtes, les algorithmes de mapping ont dû évoluer afin de prendre en compte les caractéristiques propres aux long reads, notamment un taux d'erreur plus élevé, la fréquence accrue d'insertions et de délétions, ainsi que des volumes de données considérables.

Cette thèse propose une étude approfondie des différentes stratégies algorithmiques mises en œuvre dans les outils de mapping pour long reads. Après une présentation des fondements théoriques du mapping, nous analysons les compromis nécessaires à une gestion efficace de ces lectures complexes, tant du point de vue de la performance que de la précision des alignements.

Dans ce contexte, les virus constituent un cas d'étude particulièrement pertinent. En effet, leurs caractéristiques biologiques spécifiques associées à la présence fréquente de matériel génétique viral mélangé à celui de l'hôte, soulèvent des défis méthodologiques majeurs. Cette thèse s'intéresse aux implications de ces spécificités virales sur la conception, l'évaluation et l'utilisation des outils de mapping adaptés aux lectures longues.

Une attention particulière est portée à la comparaison et à l'évaluation (benchmarking) d'outils de mapping sur des jeux de données viraux, simulés et réels. Nous examinons leur robustesse, leur sensibilité, ainsi que leur impact sur des analyses bioinformatiques en aval, telles que l'appel de variants (variant calling). En complément, nous proposons une nouvelle méthodologie pour améliorer la détection de structures transcriptionnelles virales complexes, notamment les ARN sous-génomiques (sgRNA) dans le cas du SARS-CoV-2, tout en mettant en lumière l'influence de certaines pratiques sur les résultats.

Abstract

The emergence of third-generation sequencing technologies, in particular the long reads produced by Oxford Nanopore Technologies and Pacific Biosciences, has profoundly changed sequence alignment strategies in bioinformatics. Initially designed to deal with short reads, mapping algorithms have had to evolve to take account of the specific characteristics of long reads, including a higher error rate, the increased frequency of insertions and deletions, and huge volumes of data.

This thesis proposes a study of the different algorithmic strategies implemented in mapping tools for long reads. After a presentation of the theoretical foundations of mapping, we analyse the compromises required for efficient management of these complex reads, in terms of both performance and alignment accuracy.

Viruses are a particularly relevant case study. Their specific biological properties, combined with the frequent presence of viral genetic material mixed with host genetic material, present significant methodological challenges. This thesis examines the implications of these viral specificities for the design, evaluation and use of mapping tools adapted to long reads.

Particular attention is paid to the comparison and benchmarking of mapping tools on viral, simulated and real datasets. We evaluate their robustness and sensitivity, as well as their impact on downstream bioinformatics analyses such as variant calling. In addition, we propose a new methodology to improve the detection of complex viral transcription structures, in particular sub-genomic RNAs (sgRNAs) in the case of SARS-CoV-2, while highlighting the influence of certain practices on the results.

Remerciements

Je tiens à exprimer ma profonde gratitude aux membres du jury pour avoir accepté d'évaluer ce travail. Merci à Anne Goffard, pour avoir présidé ce jury avec bienveillance. Je remercie chaleureusement Dominique Lavenier et Cédric Notredame pour avoir accepté le rôle de rapporteur et pour le temps consacré à la lecture de ce manuscrit, ainsi que pour la qualité de leurs retours. Je remercie également Jasmijn Baaijens pour sa participation à ce jury et pour l'intérêt qu'elle a porté à mes travaux.

Je tiens tout particulièrement à remercier Mikaël Salson et Camille Marchet. Mikaël, merci infiniment d'avoir été mon directeur de thèse. J'ai grandement apprécié ces trois années à tes côtés. Merci pour ta bienveillance, tes conseils, et ton accompagnement constant durant cette thèse. Camille, tu as d'abord été mon encadrante de stage, avant de devenir mon encadrante de thèse. Tu m'as accompagné depuis mon arrivée à Lille, et je sais que je n'aurais pas pu faire tout ce chemin sans toi. Tu as toujours pris le temps de m'écouter lorsque je doutais, et tu as su me guider avec justesse dans mes recherches. Pour tout cela : merci, vraiment.

Merci aussi à tous les membres de l'équipe !

Ariski, merci pour ton expertise sur JavaScript. Laurent, merci pour nos discussions toujours passionnantes sur l'horlogerie. Un grand merci à Hélène d'avoir accepté d'être ma directrice de thèse pendant une année ! Antoine, merci pour nos conversations toujours enrichissantes ; même si pas toujours très scientifiques (PS : il ne faut ouvrir qu'une seule boîte). Merci à Bastien d'avoir toujours été disponible pour répondre à mes questions, et d'avoir pris le temps de m'expliquer quand j'en avais besoin. Merci à Jean-Stéphane pour ton aide précieuse dans mes démarches administratives, notamment pour l'enseignement.

Merci également à tous les doctorants et stagiaires de l'équipe. Un grand merci à Lillian pour m'avoir écouté à mon arrivée et pour m'avoir fait comprendre qu'il était normal de ne pas tout saisir au début des discussions. Merci beaucoup Léa, pour m'avoir supporté pendant ces trois années ; bon courage pour la rédaction de la fin de ta thèse ! Merci à Timothée, Pierre, Caleb et Madelaine pour les parties de volley. Merci à Karl, et bon courage pour l'écriture de ta thèse l'année prochaine !

Je ne pourrais pas nommé tout le monde, mais merci à toute l'équipe Bonsai pour m'avoir accueilli pendant ces trois ans et demi. Merci de m'avoir offert un cadre de recherche idéal, d'avoir toujours répondu à mes questions, et de m'avoir soutenu tout au long de cette aventure.

Je voudrais également remercier tous les membres de l'ANR INSSANE.

Je tiens aussi à remercier Kristoffer Sahlin pour m'avoir accueilli en Suède et pour avoir accepté de collaborer avec moi pendant ces cinq mois.

Merci au laboratoire CRIStAL, à l'Université de Lille et à l'école doctorale pour m'avoir permis de réaliser ce doctorat.

Merci à Ludovic, Paul, Liantsoa et Mathieu pour la collocation

Enfin, merci à ma famille et à mes amis pour avoir été à mes côtés toutes ces années. Merci à mes parents pour l'éducation et le soutient dont vous avez preuve durant toute ces années d'étude.

À la mémoire de Jean-Pierre Parolin.

Grand-père, tu m'as vu commencer à écrire ce manuscrit, mais tu n'as pas été là pour le voir terminé. Pour le goût de la rigueur et toutes les choses que tu m'as apprises, je te dédie cette thèse.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Bioinformatics is an interdisciplinary discipline at the interface between biology and computer science. This thesis is specifically dedicated to the field of sequence bioinformatics. We will begin with an introduction to the biological aspects needed to understand the challenges of biological sequence analysis. The aim of this introduction is to give a general and simplified overview, without going into overly technical or exhaustive detail. It is essential to bear in mind that, unlike computer systems which follow well-defined rules, the world of living organisms is marked by considerable diversity and variability. It would therefore be neither relevant nor possible to explore all its subtleties in a single introduction.

1.1 The biology behind the algorithms: The biological sequences

DNA, RNA, and proteins are the fundamental elements that govern life, providing the necessary instructions for cellular function and heredity. These sequences follow well-defined mechanisms: DNA serves as the genetic repository, RNA acts as the intermediary, and proteins execute cellular tasks. This molecular framework, known as the central dogma of molecular biology, is universally conserved across the three major domains of life: eukaryotes, prokaryotes, and archaea. While eukaryotic cells compartmentalize genetic processes within a nucleus, prokaryotes and archaea lack this structure, yet all three domains rely on the same fundamental principles of gene expression. Although specific adaptations and variations exist between species, the core mechanisms of transcription and translation remain highly conserved.

1.1.1 DNA: A molecular foundation for biological information

DNA, or deoxyribonucleic acid, is the fundamental molecule that contains the genetic information necessary for the development, functioning, and reproduction of living organisms. It is often compared to a biological instruction manual that guides each cell in its activities. The structure of DNA is formed by basic units called nucleotides.

Nucleotides are the building blocks of DNA. Each nucleotide consists of three parts: a sugar (deoxyribose), a phosphate group, and a nitrogenous base. There are four types of nitrogenous bases: adenine (A), guanine (G), cytosine (C), and thymine (T). These bases

bond together in a specific way: adenine with thymine, and guanine with cytosine. The assembly of these nucleotides into long chains forms DNA. Inside the cell, DNA adopts a particular structure. Two strands of nucleotides twist around a central axis to form a double helix structure, a configuration discovered in 1953 by James Watson and Francis Crick, based on the work of Rosalind Franklin(124). The two strands of DNA run in opposite directions, with one strand oriented from the 5′ to 3′ end, while the complementary strand runs in the opposite direction. The sugar molecule has five carbon atoms, numbered 1′ through 5′. When we refer to the 5′ end and the 3′ end, we are talking about the orientation of the sugar molecules in the DNA strand. When two nucleotides are linked together one end has a free 5′ phosphate group (the 5′ end), and the other has a free 3′ hydroxyl group (the 3′ end).

Within the nucleotide sequence that makes up DNA, specific parts are called genes, which are particular segments of the molecule containing the instructions for RNA synthesis. Each gene corresponds to a particular sequence of nucleotides, and these sequences determine the production of proteins essential to cell function. For the purposes of this manuscript, we will adopt a simplified definition of what a gene is. It is important to note, that the notion of a gene is much more complex. For example, it is now established that some genes code only for functional RNA and will never be translated into protein(2). Moreover, genes may represent only a part of the total DNA. A variable proportion of the genome may contain non-coding DNA, which does not encode proteins but can have regulatory or structural functions. The process by which the information contained in a gene is expressed consists of two main steps. The first step is transcription. The gene is copied into messenger RNA, a molecule that transports the genetic information to the ribosomes in the cytoplasm. Then, in the second step, translation begins; the messenger RNA is read by the ribosomes, which assemble amino acids in the correct order to produce a protein.

Within a gene coding for a protein, there are several regions, including Open Reading Frames (ORFs). An ORF is a continuous sequence of nucleotides located between two specific nucleotide sequences. It represents the portion of the gene that will actually be translated into a protein. However, not all regions of a gene form part of the ORF and are directly translated into protein. Nevertheless, these non-coding parts play a crucial role in the regulation and proper functioning of gene expression. Non-coding regions include UTRs (Untranslated Regions). There are two types of UTR: the 5'UTR, upstream of the ORF, and the 3'UTR, downstream of the ORF. These regions, although non-coding, are essential for various biological functions. UTRs are involved in the regulation of mRNA translation, RNA stability, and sometimes even in the control of RNA export from the nucleus to the cytoplasm. There are therefore a multitude of non-coding regions in a gene. These non-coding regions are not limited to the UTRs. Genes are therefore not confined to the coding portion (ORF), but also include these essential non-coding regions, which regulate gene expression, stability and translation. These non-coding regions enable fine control of gene expression, ensuring that proteins are produced at the right time, in the right quantities and in the right cells. This regulatory mechanism is fundamental to the proper functioning of cells and the adaptation of the organism to its environment.

To refer to the entirety of an organism's genetic information, we use the term genome, which encompasses all the DNA sequences, including both protein-coding genes and non-coding regions involved in regulation and structural functions. The genome serves as the blueprint of life, guiding cellular processes and ensuring the continuity of species through inheritance.

1.1.2 RNA: Transcription and its role in gene expression

RNA or ribonucleic acid closely resembles DNA, but unlike DNA, it consists of a single strand and contains ribose instead of deoxyribose. While RNA also contains four nucleotides, one of them differs from those found in DNA. Thus, RNA includes adenine (A), guanine (G), and cytosine (C), but in RNA, thymine (T) is replaced by uracil (U). RNA exists in many forms in living organisms, but here we will focus on messenger RNA (mRNA). Beyond mRNAs, many other RNA types exist and play equally crucial roles in the cell, whether in regulation, structural support, or enzymatic activity, highlighting the vast functional diversity of RNA in biology.

Messenger RNA (mRNA) is an essential molecule in gene expression. It is produced during the transcription of DNA. Transcription is the process by which a gene in DNA is copied into RNA. This operation is carried out by an enzyme called RNA polymerase II, which binds to the promoter region of the gene with the help of transcription factors. Once positioned, RNA polymerase uses one of the two strands as a template to synthesize a complementary RNA strand. Once transcipted the RNA is called pre-mRNA.

During this synthesis, RNA polymerase respects the complementarity of the nitrogenous bases. Thus, when synthesizing an RNA strand from a DNA template, adenine from DNA becomes uracil in RNA, thymine becomes adenine, cytosine becomes guanine, and guanine becomes cytosine. The process of transcribing DNA into RNA is a mechanism common to all types of RNA. However, the molecular players involved, notably RNA polymerases, can vary according to the type of RNA synthesized. Once transcription of a gene is complete, the resulting RNA is called pre-messenger RNA (pre-mRNA). In order to become mature mRNA, it must undergo several modifications. This process is known as maturation. In simplified terms, the pre-mRNA undergoes three essential modifications. The first is the addition of a 5′ cap, which is a protective structure made of a modified nucleotide. This cap facilitates the recognition of the mRNA by the ribosome during translation. The second modification is the addition of a poly(A) tail at the 3′ end, a series of adenines added by an enzyme. This tail protects the mRNA from degradation and regulates its stability. Finally, the last step, although particular, plays a crucial role in the complexity of mRNA: splicing.

1.1.2.1 RNA splicing: Mechanisms and regulation

Splicing is an essential process in the maturation of eukaryotic messenger RNA (mRNA). It removes introns (non-coding sequences) and joins exons (coding sequences) to form a functional mRNA. This mechanism, which takes place in the nucleus, is crucial for the production of correct and functional proteins. Splicing is carried out by an enzymatic complex called the spliceosome, a molecular machinery composed of proteins and small nuclear

RNAs. The major spliceosome recognizes specific sequences in the RNA called donor sites and acceptor sites, which are widely conserved across most eukaryotic species. There is also a minor spliceosome, which recognizes different splicing sites and is involved in a rarer type of splicing, specific to certain species. A key phenomenon associated with splicing is alternative splicing, which allows a single gene to produce multiple different proteins by selecting different combinations of exons and introns. This process increases protein diversity and plays a major role in protein diversity (13).

Thus, whether major or minor, splicing is a fundamental mechanism contributing to the complexity of eukaryotic organisms. Through these different maturation steps, the mRNA becomes a functional molecule, ready to be exported to the cytoplasm, where it will be translated into a protein.

1.1.3 Proteins: From gene to function

Translation is the process by which messenger RNA (mRNA) is converted into a protein. This essential mechanism takes place in the cytoplasm of cells. The genetic information contained in the mRNA is read in the form of nucleotide triplets called codons. Each codon encodes a specific amino acid or a particular signal in protein synthesis. For example, the AUG codon codes for methionine, which usually marks the start of translation. There are also three stop codons (UAA, UAG, UGA) that signal the end of translation. Thus, an Open Reading Frame (ORF) begins at the start codon and ends at the stop codon. The ribosome is a molecular complex that binds to the mRNA and moves from 5′ to 3′, reading the codons one by one. It is responsible for assembling amino acids into a polypeptide chain, ensuring proper protein synthesis.

1.2 The borderline of life: Understanding what makes viruses unique

The rule that every organism possesses DNA, transcribes it into RNA, and then translates it into proteins is not universal. Organisms that follow this principle are believed to have originated from a single common ancestor, known as LUCA (Last Universal Common Ancestor)(67). LUCA's descendants are classified into three major domains: eukaryotes, archaea, and bacteria. These groups are considered phylogenetic because they are defined based on their evolutionary history and relationships. In other words, their classification is based on a common origin and the characteristics they share due to this ancestry. However, one group of biological entities escapes this classification: viruses. Unlike organisms descended from LUCA, viruses do not originate from this common ancestor (50). Additionally, some viruses do not possess DNA, or have it in an unusual form. Furthermore, viruses cannot replicate autonomously; they must infect a host and hijack its cellular machinery to ensure their multiplication.

In the rest of this manuscript, when referring to a virus, we will use two methods for classification: the Baltimore classification(8) and that of the International Committee on Taxonomy of Viruses (ICTV)(70). The Baltimore classification categorizes viruses based on

the type of molecule they use to store their genetic material. It was the first classification system adopted to group viruses into different categories. This classification consists of seven groups:

- **Group I :** Double-stranded DNA viruses.
- **Group II:** Single-stranded DNA viruses.
- **Group III:** Double-stranded RNA viruses.
- **Group IV:** Positive-sense single-stranded RNA viruses.
- **Group V:** Negative-sense single-stranded RNA viruses.
- **Group VI:** Single-stranded RNA retroviruses.
- **Group VII:** Double-stranded DNA retroviruses.

The ICTV taxonomic classification has now become the standard in virology. More precise and comprehensive than the Baltimore system, it organizes viruses into categories and subcategories (called taxa), based on their evolutionary relationships, genetic composition, and structural features. In Figure 1.1, we present the current classification framework used to categorize viruses.

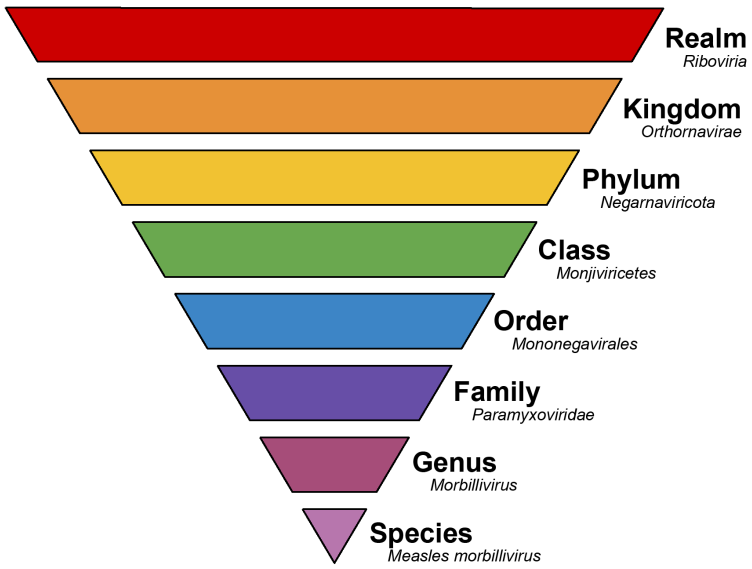

Figure 1.1 – *Virus classification from https://ictv.global/about/taxonomy. The classification of viruses is completely separated from that of cellular organisms. It does, however, use the same concept of hierarchy used in the phylogenetic classification of living organisms. However, the classification of viruses is not based solely on phylogenetic criteria: it also incorporates functional aspects, such as the type of host infected or the structure of the virus.*

1.2.1 The Complexity of viruses: A bioinformatics perspective

From a bioinformatics perspective, certain characteristics specific to viruses can pose unique challenges. Without providing an exhaustive list, this section will highlight some viral particularities and explain why they can complicate bioinformatic analysis.

One of the first challenges is related to the mode of viral replication. When viruses infect a host, their viral genome often becomes mixed with the host's genetic material. Methods exist to extract a pure viral genome without contamination from host DNA or RNA(26), but these techniques are expensive and not always available for all viruses. As a result, they represent only a fraction of the datasets used. Consequently, during bioinformatics processing, an additional step is required to separate the viral genome from the host genome(60). However, this can be a difficult task(46), and in the absence of accurate host reference genomes, it becomes even more complicated to distinguish host sequences from viral or microbial sequences. In all cases, separation requires the use of specialised software tools and careful adjustment of parameters to strike a balance between eliminating contamination and preserving relevant viral data. Beyond this separation, another challenge arises from the small size of viral genomes. Unlike the host genome, which is typically much larger, viral genomes often account for only a tiny fraction of the sequencing data. Amplification techniques can artificially increase the proportion of viral genomes in a sample. However, these methods have consequences: on one hand, they can introduce errors (although their impact remains limited), and on the other hand, they modify the nature of the data by fragmenting the viral genome into smaller segments, making bioinformatic analysis more complex.

Furthermore, certain viral species exhibit biological features that complicate analysis even further. Some viruses mutate extremely rapidly, even within the same host. When multiple viral variants coexist within a single organism, this is referred to as a quasi-species (91, 21, 88). The rapid accumulation of point mutations makes sequence analysis more difficult, particularly in genome assembly and variant identification.

Many other viral characteristics exist. For example, in this manuscript, we will explore the case of subgenomic RNAs, which will be described in greater detail in section 4.1.1.1. The key takeaway is that viruses, due to their unique features, cannot be analyzed in the same way as other living organisms. Their study requires specific bioinformatics strategies. Before detailing these methods, we will first examine how a biological sequence is transformed into a bioinformatic sequence, an essential first step in their analysis.

1.3 Decoding life: An overview of sequencing technologies

Genome sequencing is the process of determining the order of nucleotides in a DNA or RNA molecule. The sequences obtained from the sequencer are called reads. Over time, several sequencing techniques and generations have been developed. These can be categorized into three main types of sequencing techniques.

1.3.1 Sanger sequencing: The first revolution in DNA analysis

Sanger sequencing, developed by Frederick Sanger in 1977 (105), is a method for determining the nucleotide sequence of a DNA fragment. The original methods rely on the use of modified nucleotides called dideoxynucleotides (ddNTPs), which terminate DNA strand elongation when incorporated. Additionally,improvements have been made to the original method (114, 113) . The ddNTPs now carry a specific fluorescence signal for each base.

The process begins with the denaturation of the target DNA to obtain a single-stranded template. A complementary primer binds to this template, initiating the synthesis of a new strand using DNA polymerase. During this synthesis, the polymerase incorporates standard deoxynucleotides (dNTPs) along with dideoxynucleotides (ddNTPs). When a ddNTP is added, the elongation of the DNA strand stops at that position. As a result, a collection of DNA fragments of different lengths is generated. These fragments are then separated based on their size. A laser detects the fluorescence emitted by the ddNTPs, allowing the identification of the corresponding base, which is then used to reconstruct the DNA sequence.

Sanger sequencing can produce reads of 800 to 1000 bases with exceptional accuracy (99.999%)(111). This method remains widely used today for specific applications, particularly in medical and clinical research(22). However, due to its high cost and slow processing speed, it has been largely replaced by second-generation sequencing techniques for large-scale genomic studies.

1.3.1.1 Q-scores, errors and coverage

Before discussing the different sequencing platforms in more detail, it is important to revisit the concept of precision. The precision of a sequencer refers to the percentage of correctly identified bases compared to the actual sequence.

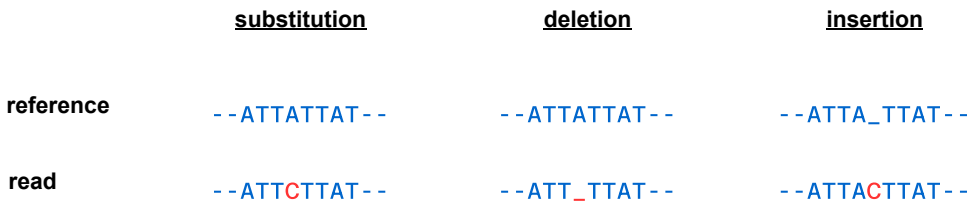

Figure 1.2 – *Diagram of the different types of error found in sequencers.*
The three cases correspond respectively to a substitution, a deletion and an insertion. The character in red corresponds to the error introduced in the reads. In the first example, the second A is replaced by a C in the read, while in the second example, it is deleted. In the third example a C is added after the second A.

The most common sequencing errors (see Figure 1.2) fall into three main categories: substitution, insertion, and deletion.

- A substitution error occurs when one nucleotide is replaced by another.
- An insertion error happens when an extra nucleotide is added to the sequence.
- A deletion error corresponds to the removal of a nucleotide from the original sequence.

These errors impact the quality of sequencing data and vary depending on the technology used. In sequencers, the probability of an incorrect base being called is denoted as e. The sequencing quality score, or Q-score, is then calculated using the following formula:

$$Q = -10 \log_{10}(e) \tag{1.1}$$

A Q-score of 20 corresponds to an error rate of 1/100, meaning a sequencing accuracy of 99%.

The last key concept to explain regarding sequencers is coverage. Coverage refers to the average number of reads per region of the genome. For example, if we talk about a coverage of 10, it means that, on average, each region of the genetic material is sequenced 10 times, meaning each nucleotide is covered 10 times on average. However, sequencing coverage is often not uniform. Some regions will have higher coverage than others. This is particularly noticeable at the beginning and end of the genome, which tend to have lower coverage on average.

1.3.2 Next-generation sequencing: Fast, scalable, and cost-effective sequencing

From the 2000s onward, the field of DNA sequencing underwent a true revolution with the emergence of new high-throughput sequencing techniques. These technological advances significantly accelerated the speed of obtaining genetic sequences while reducing associated costs. These new methods are collectively referred to as "second-generation sequencing", or Next-Generation Sequencing (NGS). Unlike traditional methods such as Sanger sequencing, which require analyzing a single DNA fragment at a time, NGS technologies enable the massive parallel sequencing of millions of fragments in a single experiment.

Among the many platforms developed for second-generation sequencing, Illumina and Ion Torrent dominate the market. Second-generation sequencers produce short reads, typically between 150 and 600 bases in length.

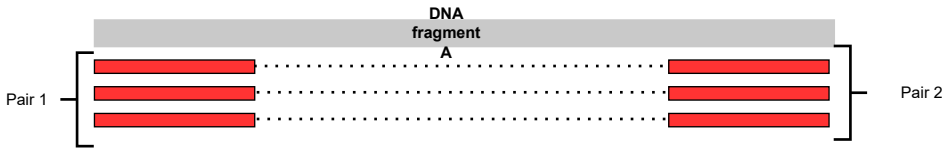

Figure 1.3 – *Schema of paired-end reads. Reads are shown in red. The dotted lines show that each read keeps information about its pair. For example, the reads of a pair can be given in 2 separate files with a common identifier.*

The paired-end sequencing technique makes it possible to compensate for the limited length of reads by sequencing both ends of a DNA fragment, while retaining the information indicating that both reads come from the same fragment (see Figure 1.3). Despite the limitation in read length, NGS sequencers compensate with high sequencing depth, meaning that the same DNA fragment is read multiple times to improve result reliability. The Q-score, which measures the probability of error in base reading, is typically 30, corresponding to 99.9% accuracy with mainly substitution-type errors(119, 108). To summarize, Illumina sequencing technology is characterized by short reads, high depth, and low error rates. From a practical rather than technical perspective, NGS allows for generating reads quickly and at a low cost. For all these reasons, second-generation sequencing has gradually replaced Sanger technology. While Sanger sequencing played a fundamental

role in decoding the human genome, it is now considered slower and more expensive for large-scale sequencing projects(82).

Although Sanger sequencing is still used for applications requiring extreme precision on short DNA fragments, NGS has become the standard for whole-genome sequencing, RNA analysis, and metagenomic studies. However, NGS technologies have certain limitations.

1.3.3 Third-generation sequencing: Towards longer reads and real-time genomics

Second-generation sequencing (NGS) has profoundly transformed DNA analysis by making sequencing faster and more financially accessible compared to the Sanger method. However, it has certain limitations, particularly the short length of reads and the need to amplify DNA before analysis, which can introduce biases and complicate genome assembly. To overcome these constraints, third-generation sequencing (TGS) was developed. Unlike NGS, which generates short fragments, TGS enables the reading of extremely long DNA sequences in a single run. This technological advancement facilitates genome reconstruction and enhances the analysis of complex DNA structures, such as repetitive regions or genomic rearrangements.

Two main platforms currently dominate the TGS market: Oxford Nanopore Technologies(19) (ONT) and Pacific Biosciences(33) (PacBio). These technologies are distinguished by their ability to produce reads exceeding 10,000 bases, although their coverage is generally lower than that of second-generation methods. The error rate varies depending on the platform and model, influencing the quality of the generated data. Initially, ONT and PacBio technologies had a relatively high error rate, ranging between 5% and 20%. Today, they have achieved significantly improved accuracy, with read quality approaching Q20, meaning an error rate of around 1%(59, 126). PacBio has even developed a specific approach called HiFi, which produces reads with Q30 quality, significantly reducing error rates and offering accuracy comparable to NGS. Errors in TGS are predominantly insertions and deletions(23). This type of error is linked to the continuous nature of the signal generated by these platforms, which is based on the direct detection of DNA without amplification. This makes it more difficult to identify the exact lengths of repeated motifs (such as homopolymers), increasing the risk of insertions or deletions.

Each technology has its own advantages and drawbacks. PacBio is often favored for sequencing complex genomes due to the high fidelity of its HiFi(126). On the other hand, ONT stands out for the portability of its sequencers, making it an ideal solution for field applications, particularly during outbreaks where rapid sequencing is necessary(99, 36).

1.3.3.1 Nanopore and basecalling: Translating raw signals into genetic information

Pacific Biosciences, like Illumina and the Sanger method, uses fluorescent nucleotides to detect the DNA sequence. Nanopore platforms do not follow this method. Sequencing using the Oxford Nanopore Technologies platform involves passing a DNA strand through a nanopore, which generates a distinctive electrical signal as the nucleotides pass through

(see figure 1.4. This signal is then processed and converted into bases in a process known as basecalling, which identifies the specific bases (A, T, C, G) in the DNA sequence. To facilitate the movement of the DNA strand through the pore, an adaptor is attached to each end of the sequence, and a motor protein is present at one of the extremities to help control the movement of the strand. Additionally, a tether is positioned on the surface of the flowcell, which guides the DNA towards the nanopore, ensuring that it enters the pore in a controlled manner.

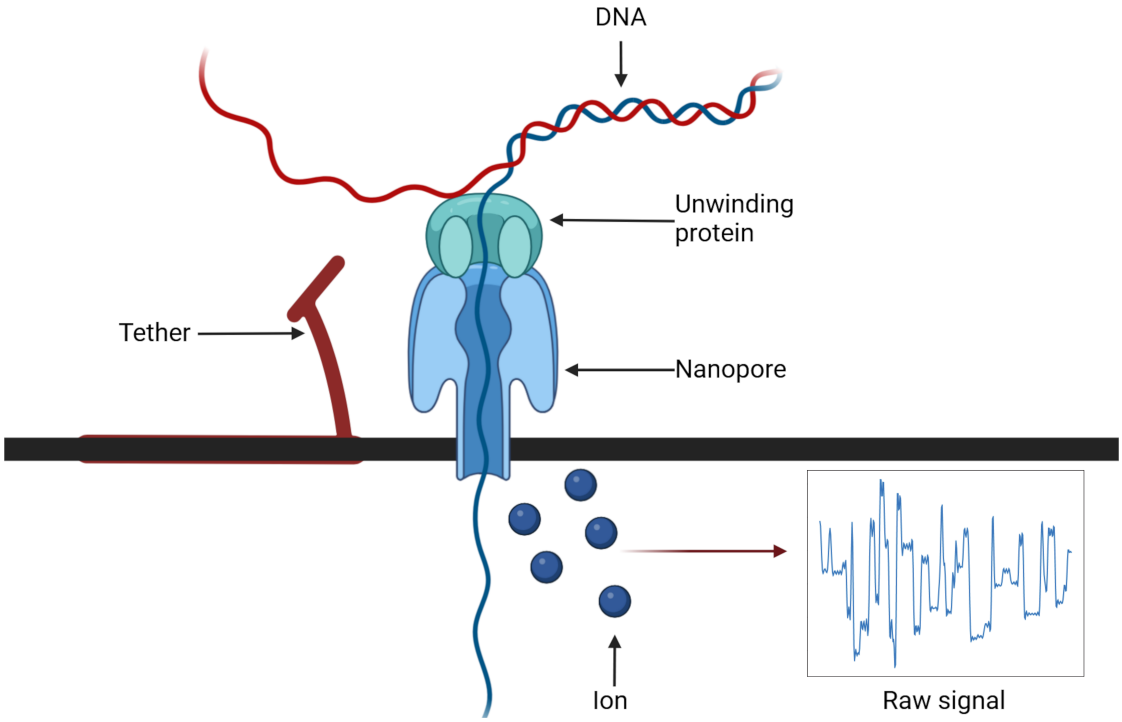

Figure 1.4 – *Schematic representation of pore sequencing. The tether is a specific element which helps the genomic fragment to find the nanopore. When a strand of RNA passes through a nanopore, it disturbs the ionic current generated across the pore. Each nucleotide modifies this current in a characteristic way, enabling the sequence to be transcribed into an interpretable electrical signal.*

The nucleotides pass through the nanopore in groups of five or six, with the basecalling system decoding these groups as a whole. This process of grouping is crucial for ensuring the correct identification of bases as they move through the pore. However, errors can arise due to the inherent ambiguities in the electrical signal, especially when certain groups of nucleotides pass through the pore together. These errors, primarily insertions and deletions, are particularly common in regions with repetitive sequences, such as homopolymers (runs of a single nucleotide, e.g., "AAAAA"). The signal for these repetitive regions can become difficult to interpret, leading to a higher likelihood of errors in basecalling(23). Despite these challenges, ongoing advancements in basecalling algorithms(97) and signal processing are continually improving the accuracy and reliability of ONT sequencing, making it a powerful tool for a wide range of genomic applications. In addition to errors made by basecallers, Nanopore technologies can generate chimeric reads(128). These reads occur when two distinct DNA fragments pass through the same pore successively without interruption. As a result, the read is a mixture of DNA fragments from different origins, which can complicate sequence analysis and assembly.

ONT Basecallers are now tools based on machine learning methods, specifically within a subfield called deep learning. Machine learning encompasses a set of algorithms capable of learning with minimal human intervention. Deep learning is a subcategory of machine learning that includes neural networks, which are often used for basecalling(97). These algorithms function as learning models: once the algorithm is created, it is trained on a specific dataset. Basecalling relies on supervised learning, meaning that the training dataset is labeled. Segments of raw signal are matched with their corresponding nucleotide sequences, allowing the algorithm to learn how to convert a raw signal into a nucleotide sequence. Once training is complete, a basecalling model is obtained, which can then be adjusted if necessary. However, if the training dataset is biased, the model will be biased as well. Biases have already been identified in some basecalling models. For example, regions of the genome rich in guanine (G) and cytosine (C) bases can pose challenges for basecalling, due to the increased stability of GC bonds, affecting the measured electrical signal. This can lead to reduced accuracy in these regions.(24)

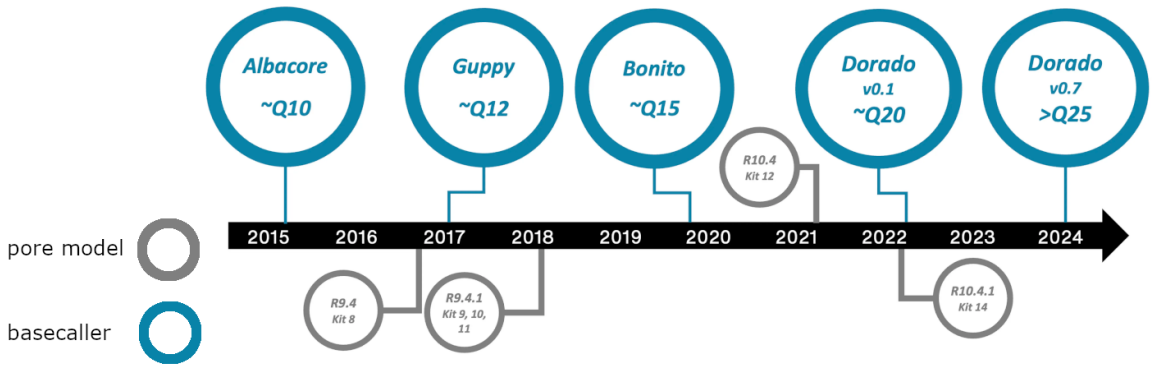

Figure 1.5 – *History of basecallers and pore models from https://nanoporetech.com/blog/transforming-basecalling-in-genomic-sequencing*

For training Nanopore basecallers, it is also necessary to consider the pore model used, as each type of pore has its own characteristics. Thus, a specific basecalling model exists for each pore. Figure 1.5 illustrates the different types of basecaller and pore models created. Today, the neural networks used for basecalling include Transformers (see `https://nanoporetech.com/blog/transforming-basecalling-in-genomic-sequencing`). However, we will not detail their functioning here, as although these architectures influence the quality of the obtained reads, downstream analysis tools generally do not take into account the type of basecaller used.

1.4 Read processing in bioinformatics: From raw reads to meaningful data

Once the signal is retrieved and processed by the basecaller, the read sequences are stored in a FASTQ file.

Each read is represented by a block of four lines:

- The first line contains the read identifier, which starts with @.
- The second line corresponds to the nucleotide sequence.
- The third line is a separator line marked by +.
- The fourth line indicates the quality score of each nucleotide in the sequence.

However, FASTQ files do not mark the end of bioinformatics analysis but rather its beginning. We now have billions of sequences, each with a quality score and an identifier, but without information about their position. Thus,finding the correct reads' position in the reference is akin to solving a puzzle. However, this process is complicated by several factors. First, some sequences may originate from other species due to contamination, leading to a mix of multiple genomes within the same dataset. Additionally, sequencers are not error-free, which can introduce slight variations in fragments that should normally be identical. To overcome these challenges, numerous algorithms and methods have been developed alongside the evolution of sequencing technologies, improving alignment, assembly, and error correction.

1.4.1 Reference free approach

When using reads without relying on a reference sequence, this is often because the species' genome has not yet been sequenced. The bioinformatics approach in this case involves reconstructing the genome directly from the reads. To do this, tools called assemblers(83) are used to perform de novo genome assembly by grouping and connecting sequence fragments based on their overlaps(112). In some cases, an assembly can also be performed even when the reference genome is known. This approach is often favoured when it is believed that using the reference could introduce biases. Indeed, by serving as a guide, the reference may sometimes constrain the placement of reads, thus limiting the ability to identify variations or structures that are not represented in the reference genome(3). In addition to assembly, there are many other approaches that can be qualified as reference free, such as read correction tools(62).

1.4.2 Read alignment and mapping: Processing reads with a reference

When a reference is available, it can be used to determine the original position of each read as well as all the nucleotides it contains. This process, which involves identifying the position of nucleotides by relying on a reference sequence as a model, is called alignment.

In more formal terms, the alignment of two sequences q and t, let $q = (q_1, \dots q_n)$ be the read sequence of size n and $t = (t_1, \dots t_r)$ the sequence of the reference region of size r. Let $\Sigma = \{A, C, G, T\}$ be the alphabet of q and t, and $\Sigma_a = \{A, C, G, T, -\}$ be the alphabet of the alignment. Let $\mathcal{A} : \Sigma_a^* \to \Sigma^*$ be a transform that maps a string (such as "ATTT-C") to its subsequence with all "$-$" characters removed (e.g. "ATTTC"). An alignment is then a pair of strings $(q', t') \in (\Sigma_a^S)^2$ with $S \geq max(n, r)$ such that:

1. q' and t' have the same size: $|q'| = |t'| = S$

2. the initial sequences are retrieved through the transform: $\mathcal{A}(q') = q$ and $\mathcal{A}(t') = t$
3. any pair of characters can be matched at a position i of the strings but two dashes: $(q'[i], t'[i]) \neq (-, -)$, for $0 < i \leq S$

The matching of two characters at a position i is weighted using a score function. The alignment is considered optimal if a defined score function reaches the minimum possible score. Three types of alignment exist, global, local and semi-global.

- The global alignment consists in aligning the totality of a sequence q on a sequence t.
- The local alignment is based on aligning a part of a sequence q on a part of a sequence t.
- The semi-global alignment is an intermediate alignment strategy between global alignment (where all two sequences are aligned, end-to-end) and local alignment (where only the most similar regions of the two sequences are aligned). The principle of semi-global alignment is not to penalise certain ends of the sequences, which makes it possible to take into account cases where we want to align an entire sequence q against only part of a sequence t, or on the contrary to detect overlaps between q and t.

There are several categories of score functions, the fixed cost score functions and the variable cost score functions. The two basic scores for fixed cost score functions are the Hamming distance and the edit distance. The Hamming distance is defined as the number of positions at which the corresponding characters in two strings of equal length are different. It represents the minimum number of substitutions (so, with no "$-$" inclusion) required to transform q into t. This metric only takes into account substitution phenomena.

The edit distance (or Levenshtein distance) is defined as the minimum number of operations; including substitutions, insertions, and deletions; required to transform the string q into the string t. Unlike the Hamming distance, this score function therefore accounts for indel events, that may occur between the two sequences. The indel events correspond to the insertion or deletion. An insertion in the read will be translated in the alignment by the obligation to open a gap in the reference corresponding to the missing nucleotide(s). A deletion event will require a gap in the reads to align it correctly with the reference. A gap will be represented in our alignment by the obligation to add one or more "$-$" characters. The score functions were quickly modified to be able to get closer to the existing situations (in the living being, from an evolutionary point of view, and to integrate a model of sequencing errors). Thus the cost of substitution or indel can now be different considering that an indel phenomenon is more expensive than a substitution because it is a rarer biological event. The cost of the scoring functions can also be adapted according to the sequencer by taking into account the types of errors that are committed or the expected frequency of mutations. The most common models are linear and affine functions. In the

linear model, each gap has a cost that increases proportionally with its length. This means that every additional base in a gap adds the same penalty.

In contrast, the affine gap penalty model, which is widely used in practice, introduces a gap opening cost and a gap extension cost(47). This reflects the biological intuition that starting a gap is more penalizing than extending an existing one. Other types of gap penalty functions also exist. For example, concave gap functions can assign lower marginal costs as gaps get longer, reflecting the idea that a single long gap might be more biologically plausible than multiple shorter gaps. This is particularly useful in contexts like RNA splicing or structural variation detection.

To understand the difficulty of aligning all the reads to a reference, it is important to visualize the size of the data that are processed and also to address notions of complexity. The human genome measures 3.2 billion base pairs, an ONT sequencing retrieves up to 2.1 Gb of data for a 24h run. The brute force algorithm that would consist in enumerating all the existing alignments is impossible to achieve. If gaps are allowed for two sequences of sizes n and m there are :

$$f(n,m) = \sum_{k=0}^{\min(n,m)} 2^k \binom{m}{k} \binom{n}{k} \tag{1.2}$$

So for $f(8,4)$ there are 3649 possible alignments. It is therefore impossible to perform the alignment in a naive way. Even classical algorithms based on dynamic programming; which we will discuss in more detail in the section 2.3.1; have a time complexity of $\mathcal{O}(nm)$, where n and m are the lengths of the sequences. For whole-genome-scale data, this complexity becomes prohibitively expensive. Thus, to speed up this process, heuristics have been implemented. From a computational perspective, a heuristic is a method that allows finding a solution to a problem quickly, without guaranteeing its exactness. Algorithms that use heuristics to align reads to a reference sequence are called mappers.

Chapter 2

From Sanger reads to long reads: The evolution of mapping

As we mentioned in the previous section, to align reads with a reference, we need to go through a step called mapping. Mapping is a succession of several heuristics, the aim of which is to reduce the time spent on alignment as much as possible. The aim of this chapter is not to provide an exhaustive list of each of these heuristics and how they work, but rather to help the reader understand how mapping has evolved to adapt to the data and the computing capabilities.

Before diving into the details of mappers, let's take a step back. BLAST (Basic Local Alignment Search Tool)(4) is one of the most important and well-know tools in sequence analysis. It was developed at a time when only Sanger sequencing technology was available. In the following text, BLAST relies on a database of known sequences. When a user inputs a sequence or a sequence fragment, the software identifies all matching sequences in the database. As we explained in section 1.4.2, aligning a sequence directly against an entire reference is computationally expensive. To make this search more efficient, BLAST does not compare the entire sequence directly with the whole database. Instead, it applies a heuristic approach: it segments the sequence into fixed-length words, a method known as seeding. There are many seeding methods, each with its own characteristics. In this example, let's take the simplest seed possible, called a k-mer, where k represents the word size. For example, Figure 2.1 illustrates 5-mers extracted from a reference sequence $S1$. These k-mers overlap, sharing $k-1$ characters with their neighbours.

At first glance, breaking a sequence into k-mers does not immediately speed up the process; instead, it introduces an additional step. However, unlike the human brain, a computer relies on specific data structures to optimize certain tasks. In the case of mapping, a structure called an index is used. An index has two purposes: to store information and to query the stored information. When it comes to indexes in mapping, two parameters need to be taken into consideration: the size of the index, i.e. the space it occupies, and the time required to execute a query. There are different types of indexes (which we will discuss later, see section 2.1.1 and 2.2.1), each with advantages and disadvantages depending on the application.In BLAST, all k-mers from the references are collected, linked to their original sequence and position, and then stored in an index.

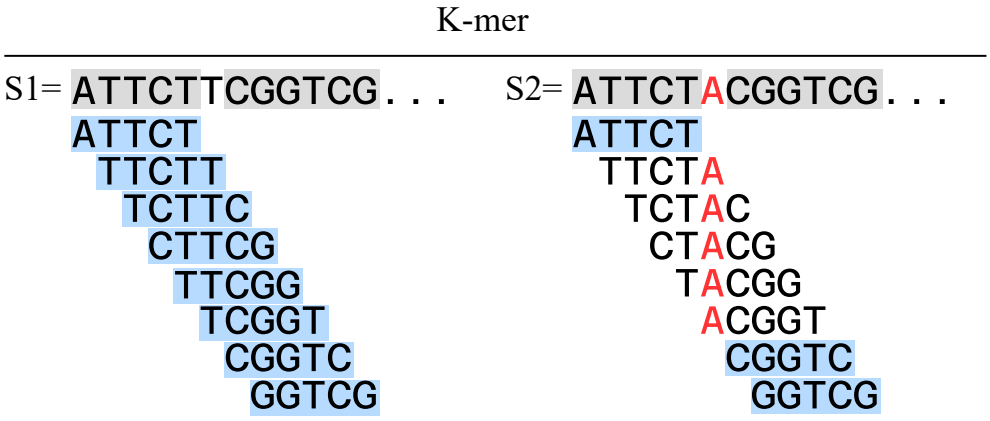

Figure 2.1 – *Schematic representation of k-mers*
The S1 sequence is an error-free sequence, the S2 sequence corresponds to the S1 sequence with one error introduced, a substitution of a T for an A (in red). The k-mers highlighted in blue correspond to k-mers collected without error.

Once the index is built, the user can input a query sequence. This sequence is also fragmented into successive k-mers. The software then searches for matching k-mers in the index, which constitutes what is known as a hit or match. Finally, in the last step, these matches are extended and aligned to produce the final alignment between the user's sequence and those in the database. This approach is known as seed and extend: after identifying seeds, the algorithm attempts to extend them into full alignments by exploring their surrounding regions in both the query and the reference sequences. This final step produces the full alignment between the user's sequence and those in the database.

2.1 Mapping short reads: Seed-and-extend approaches

With the increase in data volume enabled by next-generation sequencers, new tools had to be developed to efficiently process this data. It was at this point that the rise of mappers truly began. Second-generation mappers follow the same logic as BLAST, using a "seed and extend" heuristic, which consists of two key steps: a seed collection phase followed by an alignment phase, corresponding to the extension.

2.1.1 Indexing with short reads

Many types of index have been used throughout the history of mapping. In this section, we will look at the main indexing strategies and explain why they have been chosen for specific types of data.

There are two main families of indexes: full-text indexes and hash tables. We will focus here mainly on full-text indexes, as they are the most commonly used for short-read mapping, although some tools have used hash tables. A full-text index is a data structure that enables efficient search of arbitrary-length substrings within a text. These indexes make it possible to retrieve exact matches of patterns of any size.

Several indexing methods are used in short-read mappers, such as the FM index (39), based on the Burrows-Wheeler Transform (BWT) (16), suffix arrays (81), and suffix trees (125). The table 2.1 shows the key performance metrics of these indexes. Each of these indexing methods has undergone various modifications and new implementations in order to optimise and improve certain aspects of their performance.

Index	Space requirement (bytes)	Querry time (locate)
Suffix Tree	$20n$	$O(m)$
Suffix Array	$4n$	$O(m \log(n))$
FM-Index	$0.5n$	$O(m)$

Table 2.1 – *Full-text index comparisons. Given a text t' of length n and a query q' of size m. We will assume that n fits into 4 bytes of memory. (That is, $n < 2^{32}$)*

To better understand the different forms an index can take, we illustrate the process of creating a suffix array index in Figure 2.2.

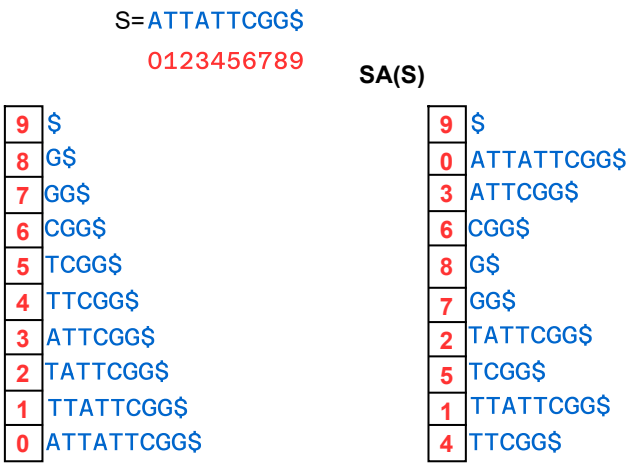

Figure 2.2 – *Method for building a suffix array. To the left of the image is a list of all the suffixes of the word S. Each line is preceded by a number indicating the starting position of the suffix in the original text. On the right, we see the suffix array for S, abbreviated SA(S), which is an array containing these positions, sorted in alphabetical order of the corresponding suffixes.*

To the left of the image is a list of all the positions in S, each corresponding to the start of a suffix. For ease of understanding, the full suffix is displayed at each position, although it is not part of the index as such. This list of positions is then sorted according to the alphabetical order of the suffixes they point to. This sorting constitutes the suffix array. This structure makes it possible to carry out very efficient searches, particularly binary searches.

In Figure 2.3, we illustrate the process of querying a suffix array. For each query, we select the middle index of the array as the pivot value. In our example, this corresponds to index 8, associated with the suffix "G$". We then compare this suffix with our query, letter by letter. Here, there is no match from the first letter. Therefore, we continue searching by

selecting the middle index of the lower half of the array as the new pivot. This process is repeated until the characters match or ended if no match is found.

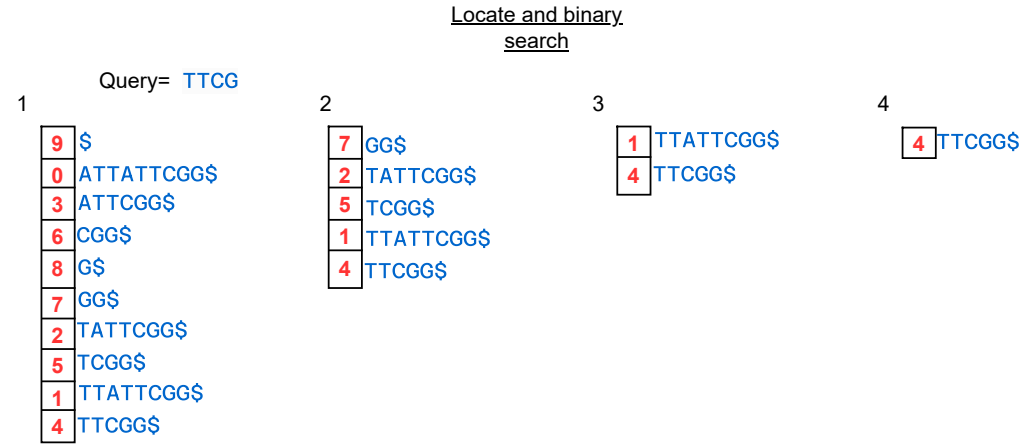

Figure 2.3 – *Method for querying a suffix array. In the figure, we are trying to locate the TTCG pattern in our index. To do this, we use a binary search on the suffix array SA(S). We start by examining the position in the middle of the array, in this case position 8, which corresponds in the text to a suffix beginning with the letter G. We compare this letter with the first letter of the pattern we're looking for (T). Since G is alphabetically smaller than T, the top part of the list can be eliminated: it contains only suffixes beginning with letters smaller than T, so cannot be matched. We repeat the process on the remaining part of the suffix array until we find, if it exists, an entry whose suffix containing the pattern we're looking for.*

2.1.2 Seeding with short reads

Once our index is built, one might think that it would be possible to directly search for our read. However, it is important to keep in mind that the sequenced species may have mutations within its genome, making it slightly different from our reference sequence.

Moreover, with short-read mappers, which typically have an accuracy of $Q30$, we expect an error for approximately every $1,000$ bases sequenced. So there is a high probability that there will be no perfect match between our read and the reference. Therefore, a seeding step is necessary. A large number of seeding strategies are used for short reads. In the following sections, we will describe two of them.

2.1.2.1 Maximal exact matches (MEMs)

The first major family of seeds used in mapping is dynamic seeds. Unlike fixed seeds, dynamic seeds do not rely on a predefined k-mer length. Instead, they are extended character by character during the search, stopping when a mismatch is encountered. While they do not strictly merge the seeding and alignment steps, they initiate the alignment process directly during the seed search. This strategy is particularly effective when the read aligns perfectly or nearly perfectly to the reference, allowing for fast and accurate identification of exact matches.

The most well-known example of this category is the MEM(25) (Maximal Exact Match). To create a MEM, we start with a k-mer and try to find a match for it in the reference. Once a

match has been found, the k-mer is extended on both sides until an error is encountered. In the case of an error-free read, it is technically possible to perform the alignment at the same time as the seeding. Several MEMs are therefore selected. To avoid over-selection, mappers often decide on a minimum size that the MEM must have to be selected.

However, this type of seed is computationally expensive, as it may require multiple extensions to identify the position where the MEM is maximally extended. This approach has been extensively used in one of the most popular short-read tools, BWA-MEM(72), as well as in one of the earliest long-read mappers, BLASR(17).

2.1.2.2 Spaced k-mers

Errors and mutations in the reads have been a problem since the short reads. To overcome this issue, specific type of seeds has been developed. These particular seeds enable the alignment of two sequences that do not share exactly the same characters.

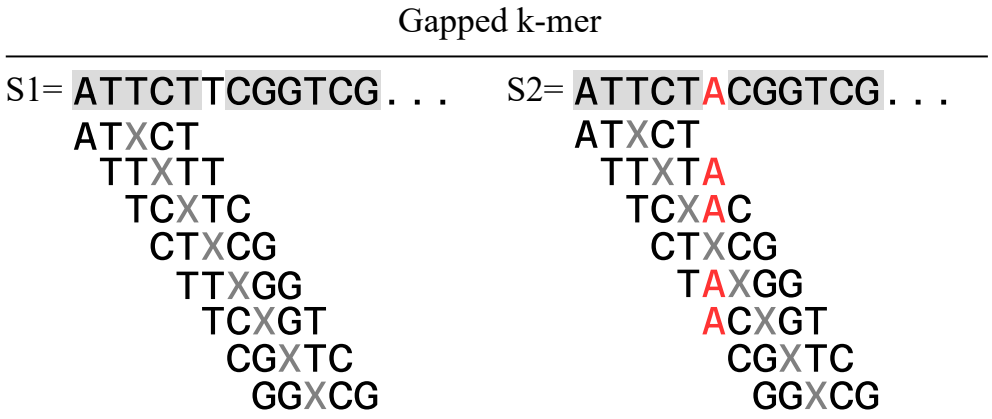

Figure 2.4 – *Schematic representation of spaced k-mers*
The S1 sequence is an error-free sequence, the S2 sequence corresponds to the S1 sequence with one error introduced, a substitution of a T for an A (in red). The gray "X" correspond to the wildcards. The 4th spaced k-mer in S2 highlights the ability of spaced k-mer to override errors compared to the k-mer

Spaced k-mers, work by introducing spaces in the collected seed, called wildcards, that can match any character. These wildcards ensure that, in the case of a substitutions, there is a better chance that a seed can recover the match without error. As shown in figure 2.4, spaced k-mer allows the collecting of seeds that will match the reference despite errors. The higher the error rate, the greater the advantage over k-mer. For spaced k-mers to be efficient, the pattern or patterns containing the wildcards must be designed to effectively handle errors. In fact, it is possible to construct sets of spaced seed patterns that guarantee 100% sensitivity for a given maximum number of substitutions (i.e., Hamming distance)(63). However, in this simple form, they still cannot handle insertion or deletion errors. The search for the optimal wildcard pattern is still ongoing(43).

GraphMap, a third-generation mapping tool, has enhanced spaced k-mers to handle indels. To achieve this, each time a query seed is tested, two additional forms of seeds are also evaluated: one adds an adjacent wildcard to test for insertions, and the other removes

a wildcard to test for deletions. A schematic representation of this process is shown in figure 2.5.

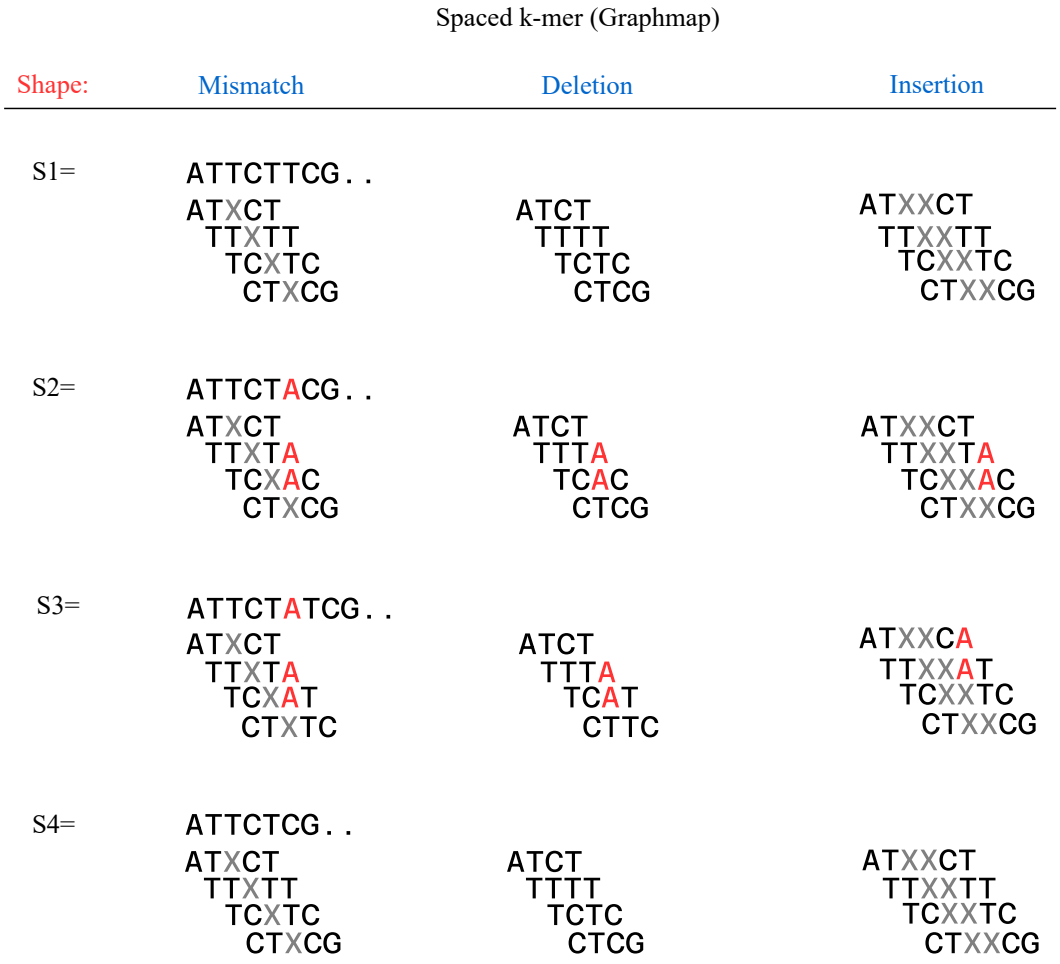

Figure 2.5 – *Schematic representation of Graphmap's spaced k-mers*
The S1 sequence is an error-free sequence, the S2 sequence corresponds to the S1 sequence with one error introduced, a substitution of a T for an A (in red). The S3 sequence corresponds to S1 with the insertion of an A after the 3rd T. The S4 sequence correspond to S1 with the deletion of the 3rd T. Shapes in blue correspond to the three wildcard shapes used by graphmap. Here the mismatch shape will include a wildcard in the middle of the 5-mer. The deletion shape correspond to a seed where we skip the nucleotide in the wildcard position resulting in a 4-mer. The insertion shape correspond to seeds where two wildcards are included resulting in a 6-mer.

2.2 Mapping long reads: Seed-chain-and-extend approaches

The text found in the following sections is largely drawn from Sahlin, K., Baudeau, T., Cazaux, B. et al. A survey of mapping algorithms in the long-reads era. Genome Biol 24, 133 (2023). https://doi.org/10.1186/s13059-023-02972-3. This is the first article written during my Ph.D.

Since the advent of sequencers capable of performing reads of more than 500 bases

but with a high error rate mainly composed of INDELs, it has been necessary to adapt the techniques used by short-read mappers. Before the apparition of mappers specifically designed for long reads, versions of short-read mappers with an option for long reads were developed. Then, mappers incorporating a new form of heuristics appeared. Thus, the previous paradigm of seed-and-extend of the second-generation sequencing tool has been transformed, with the third generation, into a paradigm of seeds, chain and extend. But not only has there been a paradigm shift, there has also been an evolution within these heuristics themselves

2.2.1 Indexing with long reads

One of the distinction between mappers designed for short reads and those intended for long reads lies in the type of index used. While early long-read mappers, such as BLASR (17) and GraphMap (118), still relied on full-text indexes, these were gradually replaced by hash tables, which have become the preferred choice. The hash table is the second indexing structure widely used for mapping. For long-read mapping, it is now the default index for the most recent tools. There are two main families of hash tables: dynamic hash tables and static hash tables. In this manuscript, we will focus only on dynamic hash tables, as to our knowledge, no static hash table has been implemented for mapping purposes.

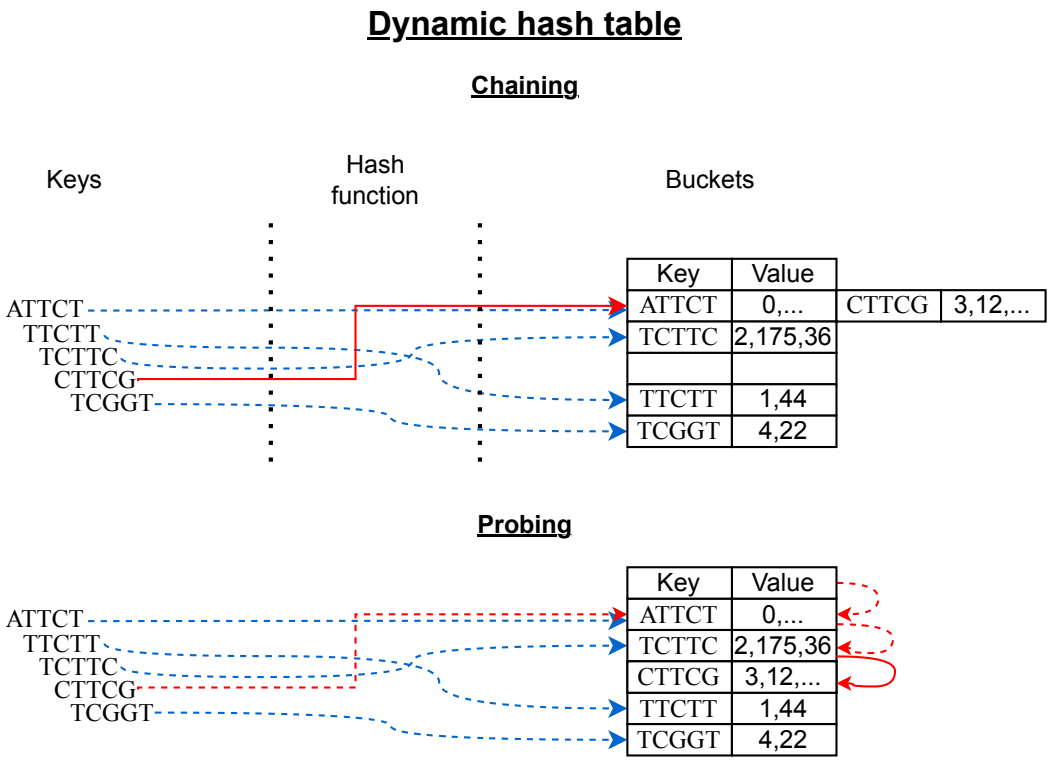

Figure 2.6 – *Dynamic hash table scheme. The dotted blue lines represent the position of each key in the hash table. The red lines show the position of keys that have been involved in a collision. The dotted red line shows the path of a key that has encountered a collision.*

A dynamic hash table works as follows (see figure 2.6): each hash value corresponds to a slot in an array, also called a bucket. Each bucket stores a key-value pair. In our index, the key is a k-mer, and the value is the list of positions where this k-mer occurs. When two different k-mers generate the same hash, it is referred to as a collision. Depending on how collisions are handled, dynamic hash tables can be divided into two subcategories: those using chaining and those using probing.

- With chaining, when a collision occurs, a new key-value pair is added to a linked list stored in the same bucket.
- With probing, when a collision occurs, the table continues to search for the next available slot (according to a defined strategy) to store the new pair.

In theory, if we consider that the key and value fit into 32 bits, the space required for these two indexes, if there are no collisions, is $8n$ bytes, where n is the number of k-mers stored. In terms of query time, in practice we have an average complexity of $O(1)$. Compared to suffix array and FM-index, hash tables require more memory space and don't have the capacity to search for variable-size seeds like MEMs. On the other hand, querying is much faster.

To understand why full-text indexes have lost popularity, we need to consider the nature of the data. Whereas short reads used to have an average of one error every 1,000 nucleotides, long reads now have at least one error every 100 nucleotides, mainly in the form of insertions and deletions. Under these conditions, MEMs (Maximal Exact Matches) become much less useful, as they stop at the first error. And yet, one of the great advantages of full-text indexes was precisely their ability to query seeds of variable length. The only remaining advantage of full-text indexes lies in their memory footprint, which is often smaller than that of conventional hash tables. However, as we'll see in the next section, the emergence of new seeds now makes it possible to significantly reduce the impact of hash table size, thus calling this advantage into question.

2.2.2 Seeding with long reads

Just like with indexes, the first long-read mappers initially used the same types of seeds as those employed for short reads. It was the introduction of minimizers(101) into the mapping process(73) that marked an important turning point in seed selection. To understand the adoption of new types of seed, we need to examine the nature of the data processed by long-read mappers. We are moving from short-read data, characterized mainly by substitution errors, to long-read data, composed of very long sequences with much higher error rates, dominated by insertions and deletions.

In this new context, MEMs, which are effective on high-quality data, become much less useful. Their length is drastically reduced, as they stop at the first sequencing error encountered, which frequently happens with long-read technologies. As a result, the computational cost of searching for MEMs is no longer justified by their informative value. Spaced seeds, on the other hand, are designed to deal with mismatches, but do not give good results with indels. Furthermore, long reads contain much more sequence data, which means

that the number of seeds extracted per read increases considerably. This poses new computational and memory challenges. Among the seed strategies presented so far, only the k-mers remains. Its main advantage lies in its simplicity and efficiency, particularly when combined with hash tables. However, if we were to extract all k-mers from long reads, we would again face the issue of generating a huge number of redundant or low-value k-mers, leading to excessive memory usage and computational inefficiency.

The idea behind minimizers emerged from this challenge: since long reads contain a lot of information (but also a lot of noise), the solution is to sample only a subset of the available k-mers. Sampling allows mappers to retain only parts of the read while drastically reducing the number of seeds to process. This concept is at the core of recent long-read mappers.

2.2.2.1 Minimizers and their evolution

There are many ways to select a sample of k-mers. For example, one could choose to retain only one k-mer out of ten. However, in practice, sampling methods are more sophisticated. The problem with retaining only one k-mer out of ten is that it does not adapt to the content of the sequence. Uniformly sampled k-mers are not guaranteed to be shared between similar regions, especially in the presence of indels, making them unreliable for detecting overlaps or matches. This is why more sophisticated sampling methods are used. Among them, the most successful approach in the field of mapping is the use of minimizers (73, 74, 54), a strategy directly linked to efficient sampling techniques. Minimizers, strictly speaking, refer to the smallest k-mer, known as the minimizer, selected within a sliding window of size w. This is the most widely adopted sampling technique, providing guarantees on the distribution of retained k-mers.

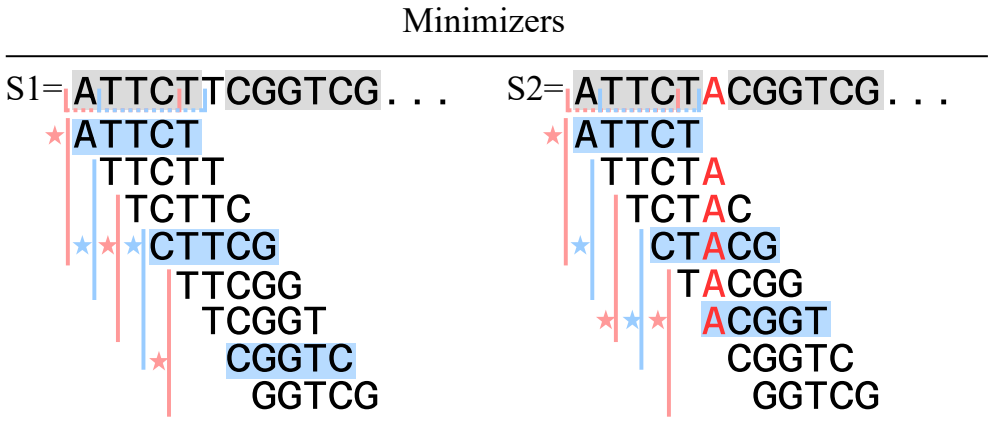

Figure 2.7 – *Schematic representation of minimizers*
The S1 sequence is an error-free sequence, the S2 sequence corresponds to the S1 sequence with one error introduced, a substitution of a T for an A (in red). The vertical lines show which k-mers belong to each window. The stars correspond to the minimizers selected for each window. The red and blue colours are only used to differentiate the different windows or minimizers selected.

More formally, a minimizer is defined by three parameters: m, w, and h. The parameter h is a function that defines the ordering of k-mers, such as lexicographical order. For a

given sequence, within each window of w consecutive k-mers located at positions $[m, m + w - 1]$, the minimizer is the k-mer with the smallest h-value in that window. Minimizers are extracted by identifying the k-mer with the minimum h-value in each sliding window $w \in [0, \mid S \mid -w + 1]$ across the sequence S. Thanks to this approach, two minimizers cannot be more than w apart. When a seed selection method ensures that the chosen seeds do not exceed a certain maximum distance, it is referred to as seeds with a distance guarantee. However, the use of a window also has its drawbacks. Since the minimizer is selected within a given window, it becomes dependent on the local context of that window. As a result, a small modification in the sequence within the window can completely change the selected minimizer.

Figure 2.7 illustrates how minimizers are selected. The parameters of the minimizer are $w = 4$, $k = 5$, and h, which corresponds to alphabetical order. In sequence `S2`, where an error was introduced, we observe that the minimizer `ACGGT` is over-selected, whereas it was not selected before. This highlights the context dependence of minimizers and how this can result in the over-selection of an erroneous k-mer. The last drawback of minimizers is related to repeated regions. Repeated regions are specific areas in the genome where the same sequence motifs appear multiple times. With minimizers in these particular regions, it is almost certain that the same minimizers will be selected at each repetition of the sequence, making the mapping of these regions more complex. To address this limitation of minimizers, C. Jain has proposed(58) an improvement to minimizers called weighted minimizer.

Weighted minimizer

Weighted minimizers are minimizers where a weight is assigned to each minimizer. Every time a same minimizer is selected its weight increases. Once a minimizer reaches a certain weight, it is no longer selected and is replaced by the second k-mer with the lowest hash value. This approach ensures a distance guarantee since we are still selecting a k-mer in the windows while avoiding the selection of minimizers that would be overrepresented.

To further improve seeding in repeated regions, Minimally Confidently Alignable Substrings(57) (MCASs) have recently been created. When attempting to map a read from a region with many repeats, a large number of seeds can be shared between the read and the reference. This makes the process of accurately locating the read in such a repetitive region computationally very expensive. An MCAS identifies, within these repetitive regions, specific seeds that can distinguish between different copies of the region. To achieve this, a unique and minimal substring is selected within each area to represent the MCAS. The MCAS selected will be more or less large so that it can be unique. This substring serves as an anchor, effectively differentiating the various parts of the repetitive region, thereby reducing the complexity of the alignment process.

Sync-mer

To improve minimizers, particularly by reducing their context dependence, a new selection method has been developed(32). This new seed, called syncmer, aims to be less context-dependent compared to minimizers. A syncmer is defined with 4 parameters : k

the size of the k-mer, s the size of the s-mer inside the k-mer, $x1$ the position where the s-mer must occur in the k-mer to be selected, and h is the function that defines the order for selecting the minimum s-mer.

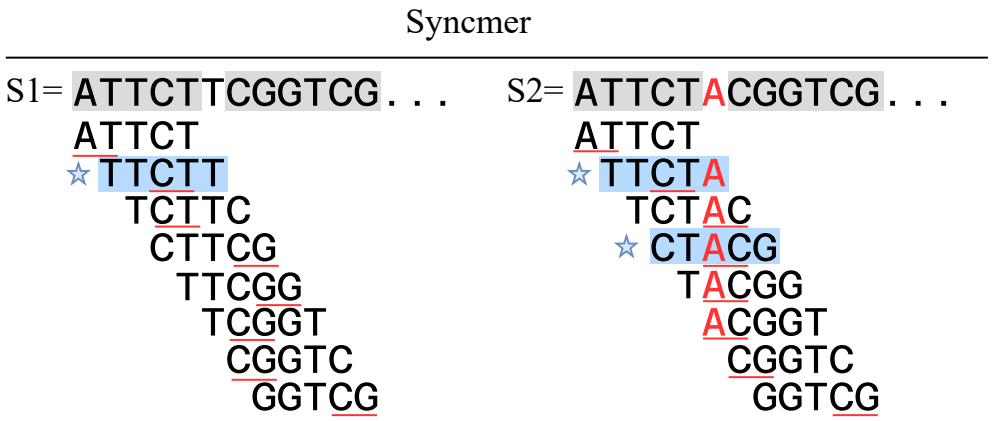

Figure 2.8 – *Schematic representation of syncmers*
The S1 sequence is an error-free sequence, the S2 sequence corresponds to the S1 sequence with one error introduced, a substitution of a T for an A (in red). The horizontal red lines show the s-mer position for each k-mer. The stars correspond to the syncmers selected.

In figure 2.8, we observe that with a syncmer defined by the parameters $k = 5$, $s = 2$, $x1 = 3$, and h (lexicographic order), the addition of a mutation, although it affects the k-mer, has a significantly smaller impact on the syncmer. This method ensures that only 1 out of the 5 k-mers containing the mutation will be selected, whereas in the same scenario with minimizers, 3 seeds containing the error would be selected.

2.2.2.2 Handling error with long reads

New types of seeds have been developed to better handle errors while remaining less computationally expensive than seeds using wildcards. Among these approaches, we can mention those employed by BLEND (41), which enhance the methods introduced with MinHash(34). These seeds are generated by linking multiple k-mers together and grouping them under a single hash function. MinHash-type seeds efficiently handle substitution errors, but they remain highly sensitive to indels. To address this limitation, a new type of indel-resistant seed, called strobemer, was recently proposed(103). Initially designed for short-read mapping, this type of seed also shows promising potential for long reads.

2.2.2.3 Blurring the lines: How seeding strategies adapt across read lengths

Previously, we chose to distinguish between seeds used for short-read mapping and those used for long-read mapping. However, in reality, there is no strictly defined boundary between these categories. Some seeds historically developed for short reads have been adopted by mappers designed for long reads. Additionally, a mapper initially designed for short reads can sometimes perform exceptionally well on long-read data, and vice versa. It is essential to remember that seeding is primarily a heuristic aimed at accelerating

the mapping process, regardless of the type of reads. Thus, while certain implementation trends emerge among tools, the distinction between seeds for short and long reads remains ambiguous and fluid.

Furthermore, the use of seeding is not limited to mapping. Different types of seeds have been utilized for other bioinformatics applications. For example, seeds without distance guarantees, introduced by MinHash (15), are widely used for read overlapping, demonstrating the diversity of approaches and their flexibility depending on analytical needs.

2.2.3 Chaining

Once the seeds are collected from our indexed reference, we also collect seeds in the reads that we'll request from our index. If a seed from the read matches one in the index, it is considered a match. A seed that has a match is called an anchor. Therefore, once the seeding step is complete, we are left with several anchors. The purpose of the chaining step is to link these anchors together in a consistent manner, in order to identify candidate alignment regions. To understand why the chaining step was unnecessary for short reads, it is essential to examine the relationship between read length and seed size. With short reads of approximately 150 bases, a dozen k-mers are usually sufficient to cover the entire sequence. In contrast, for long reads that can exceed 10,000 bases, thousands of k-mers are required.

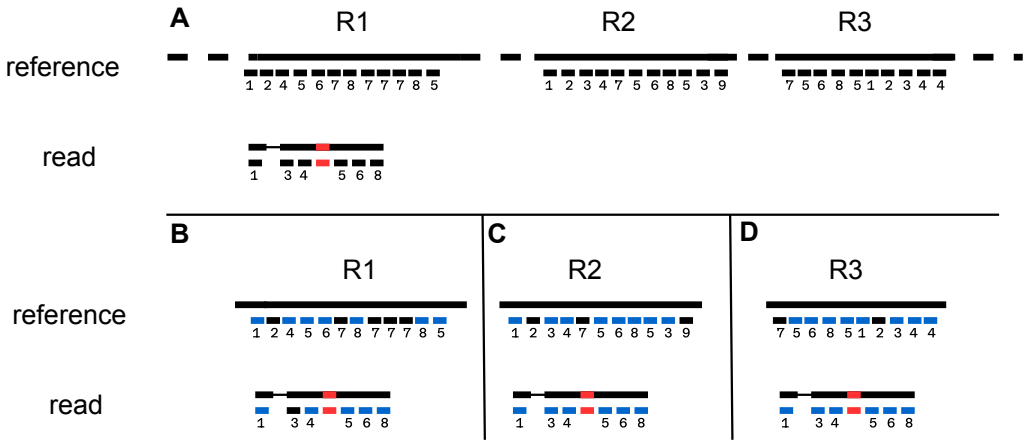

Figure 2.9 – *Schematic representation of anchor distribution*
A : Shows seed distribution in three regions in the reference and for one read. Seeds are numbered from 1 to 8. The thin line corresponds to the deletion of a seed in the read, the red part corresponds to a mutation present in the corresponding seeds creating a new seed that does not exist in the reference. B, C and D show the anchors (in blue) in three region of the reference.

The number of detected anchors for each read is directly proportional to its length. The more anchors available, the greater the number of possible combinations and positions for read alignment. Testing all these possibilities would be computationally expensive. The purpose of chaining is therefore to identify the most probable positions to narrow the search space and optimize the alignment step. Figure 2.9 shows a possible anchor schema,

in which we see the reference and the seeds collected at three positions in the reference (figure 2.9A), labeled R1, R2, and R3, as well as the seeds found in the read. The read contains a deletion, represented by a thin line, and a SNP, represented by the red part, creating a seed that doesn't exist in the reference. Figures 2.9B, C, and D show the selected anchors in blue for the three regions of the reference. The aim of our chaining is to find out which of these 3 regions is the most suitable for alignment.

Another way to represent anchors, offering a clearer visualization of the best chain, is to place them in a read-reference space , as shown in figure 2.10. In this dotplot-like representation, the read and the reference are respectively the ordinate and the abscissa of the space. Here, the three positions from figure 2.9 are retained.

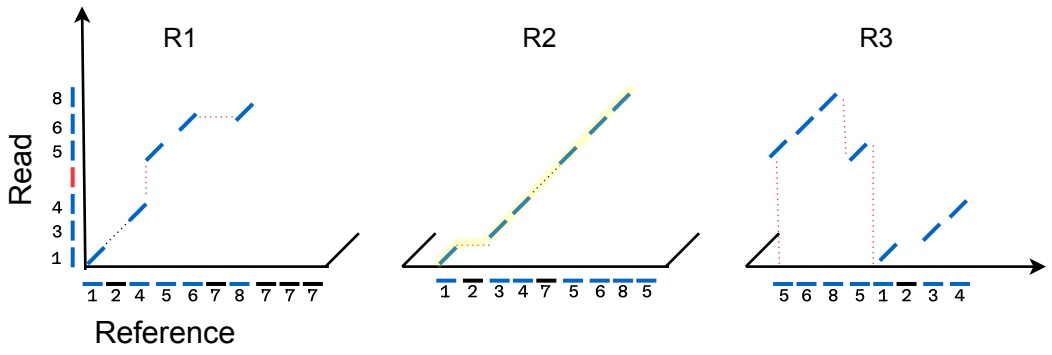

Figure 2.10 – *Schematic representation of the read reference space. R1, R2 and R3 represent 3 zones of the reference. The blue lines represent the anchors in a space where the reads represent the ordinate and the reference the abscissa. Red dotted lines represent deallocations due to insertion or deletion. Anchor chains highlighted in yellow represent the best anchor chain.*

With this visualization, if a read perfectly matches the reference, the anchors align along a single affine line. In the case of a missing anchor, for instance, due to a mutation preventing the collection of a seed, subsequent anchors will continue along the same line. However, in the presence of an insertion or deletion, the anchors will follow another affine line with a different y-intercept. This type of representation enables the establishment of rules to determine which series of anchors should be selected to correctly position the read. These rules include:

1. Anchors must follow an affine line with a positive slope, meaning the anchors have to be collinear.

2. The number of anchors on the line must be maximized.

3. Anchors should ideally avoid large gaps between them. Large spaces between anchors can lead to unaligned regions, meaning that significant parts of the read or reference are left without any matching seeds, which may reduce alignment quality or increase the risk of missing relevant regions. However, if no better alternative exists, aligners may still attempt to align reads with sparse anchors, though this increases the uncertainty and computational cost of the alignment process.

4. The space between their diagonals must be minimized. This means prioritizing chain with anchors that follow the same diagonal, or are close.

5. If inexact matches are allowed in the seeds, each seed corresponds to a pair of intervals: one on the target and one on the query. The goal is to identify a subsequence of seeds that minimizes the total Levenshtein distance across these interval pairs. In essence, this ensures that the aligned regions on the target and the query are as similar as possible, despite potential mismatches or indels.

A final point to bear in mind, not shown in our figure, is that there are a certain number of overlapping anchors. Overlapping anchors is not a problem in itself. However, we must bear in mind that some nucleotides will be taken into account several times. So if we take six adjacent 5-mers the total number of different nucleotides seen is only 10, whereas if they don't overlap it's 30 different nucleotides that are seen. The aim of chaining is therefore to maximize the total covered region of the read (or the reference) by selecting a consistent subset of anchors that respects the previous conditions. There are several possible strategies for the mappers to address this challenge.

2.2.3.1 Clustering in chaining: Eliminating less probable alignment paths

To speed up chaining, some authors (118, 100, 73) first aimed to identify the most relevant groups of anchors for alignment. To achieve this, they introduced a preliminary step called clustering. To identify approximate clusters of anchors, methods typically analyse a discrete (reference, query) position space (like the representation in figure 2.10). In this representation, a perfect match between an anchor and the reference appears as a straight line segment, corresponding to the main diagonal of the (reference, query) alignment matrix. For a group of anchors, the same principle applies; however, due to insertions and deletions, the line segments may deviate slightly from the exact diagonal, forming approximate lines in the (reference, query) space.

To detect these approximate lines the Hough transform(30) has been employed. This method is capable of identifying imperfect straight lines in a 2D space, allowing the detection of multiple diagonal patterns instead of a single best-fit line, as would be produced by linear regression. By doing so, it enables the identification of distinct groups of anchors, each representing a potential alignment.

2.2.3.2 Finding the optimal chain: LIS problem and scoring functions

Previous clustering techniques focus on identifying approximately collinear groups of anchors. However, many anchor clusters are still retained. Therefore, it is necessary to further reduce their number to minimize the alignments to be performed.

A method is needed to distinguish anchor groups that follow the expected rules (see page 27) from those that do not. To achieve this, a subset of anchors can be ordered using the longest increasing subsequence (LIS) method(107). In this approach, each anchor is assigned a position based on its order in the reference sequence. By comparing these reference positions with the order of the anchors in the query sequence, a permutation of

(where n is the total number of anchors) is formed. The LIS algorithm can then be applied to this permutation to identify collinear chains efficiently in $O(n \times \log n)$ time.

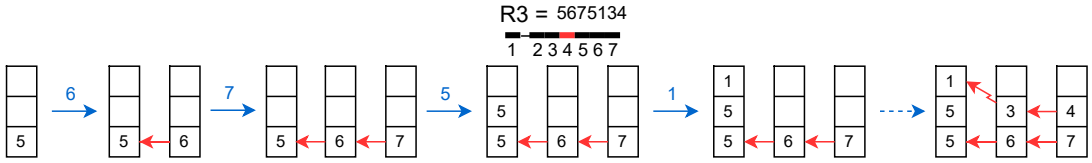

Figure 2.11 – *Longest increasing subsequence explanatory schema.*
In blue the new next number of the sequence. Anchors from R3 are used. To find the LIS among these anchors, we begin by numbering all the seeds in the read, here from 1 to 7, in the order in which they appear. Keeping this numbering, we then apply it to each anchor, giving us a sequence of numbered anchors, here: 5, 6, 7, 5, 1, 3, 4. Next, we build a first stack in which we place the first number (here 5). Then, for each subsequent number, two cases exist. For 6, and 7: if it is greater than the last element of all existing stacks, we create a new stack containing this number. For 5, 1, 3, and 4: if it's smaller or equal, we add it to the stack whose last element is the smallest strictly superior number (we replace this last element). With each addition, we keep a pointer (red arrow) to the previous element in the previous stack, which will then enable us to reconstitute the subsequence. The total number of stacks at the end of the algorithm gives the size of the LIS, and the pointers are used to find the elements that make it up.

The figure 2.11 schematically illustrates how to find the longest increasing subsequence (LIS). To identify the LIS, we will gradually construct several lists. We start by adding the first number of the sequence to our first list. Then, for each new number, we compare it to the elements already present. If the new number is larger than the last element in our list, we create a new list; otherwise, the number is added to the existing list. This process is repeated until all the numbers have been processed. As with the index array, binary search can be used to efficiently insert each element in the correct position, thereby improving the performance of the process. At the end, the number of lists created corresponds to the length of the LIS. To reconstruct the LIS sequence, a pointer is added with each insertion to trace the selected elements.

For fuzzy seeds, where inexact matches are allowed, the problem becomes more complex. Beyond the basic LIS approach, it is necessary to account for mismatches by addressing the LCSk problem, which seeks the longest common subsequence in substrings of at least k-length. This adjustment ensures that the resulting anchor chains are closely aligned to the sequences at the base-pair level. Interestingly, there is a relationship between LIS and LCSk: the LIS of a sequence P is equivalent to the LCSk between P and the sequence (1,2,. . . ,c).(12)

Despite their usefulness, LIS and LCSk methods have inherent limitations. Neither fully addresses key constraints such as allowable distances between anchors or the potential for large gaps in alignments. As a result, while these methods can establish the general structure of anchor chains, they may fall short in capturing fine-grained alignment details.

To overcome these shortcomings, some tools like graphmap rely on graph-based models constructed over anchors to guide chaining and alignment. However, these graph-based strategies often fail to consider anchor distances adequately. Consequently, they have been largely replaced by dynamic programming techniques that integrate gap-scoring functions, enabling more accurate and flexible handling of anchor spacing and alignment gaps.

The era of chaining functions

To better address the limitations of techniques based on the Longest Increasing Subsequence (LIS), particularly in handling gaps between seeds, alternative approaches have been developed. One such approach relies on chaining functions, first implemented in minimap2 (75). The purpose of these functions is to optimize the mapper's performance when dealing with deletions. In our examples, we have illustrated single-nucleotide deletions, but in reality, they can be much longer, involving multiple consecutive nucleotides. In some cases, such as splicing, deletions can span several hundred bases. Therefore, it is crucial to design scoring functions that can adapt to varying gap lengths.

One commonly used type of function for this purpose is the concave function(55). The cost function is specifically designed to address gaps caused by frequent indels, recognizing that these gaps tend to occur in clusters rather than as isolated events. Indels often impact consecutive positions, either due to errors in specific regions of the query or because of local repeats in the reference sequence. To accommodate this, the cost of initiating a gap is treated differently from the cost of extending one, making linear gap models inadequate for this purpose. Instead, concave gap functions, which adhere to the Quadrangle Inequality, are employed. These functions utilize the "wider is worse" principle(45), enhancing computational efficiency. In practical applications, concave gap functions are typically affine, consisting of either a single affine function, a set of affine functions, or a combination of affine and logarithmic functions, as outlined in minimap2 paper. Minimap2's gap functions serve as a prominent example, having been widely adopted in many studies(57, 56, 40), with only a few exceptions such as LRA(100). In addition to the concave gap function LRA proposes a solution to deal with the difficulty of minimap for large gaps. This updated approach includes two components: a cost function for regular gaps and a specialized long-gap patching procedure. It is also interesting to note that, with the exception of LRA, tools using score functions for chaining no longer use the clustering step and go directly to chaining using the score function.

Once the anchor chains with the best score have been identified, they need to be aligned with the reference. It is important to note that multiple different anchor chains for the same read can be retained for alignment. Among these, the primary chain is the one with the best score for chaining. On the other hand, secondary chains correspond to alignments with a lower score than the primary chain, but still sufficiently close to it. These secondary chains can represent plausible alternative solutions for alignment, especially when ambiguities or errors exist in the anchor sequence.

As mentioned earlier, chaining and seeding are heuristics used to accelerate alignment. However, these heuristics do not necessarily guarantee an exact solution. This is why multiple alignments are provided to the user, allowing them to choose the one that seems most likely based on the criteria they deem relevant.

2.3 Extension : Finding the best alignment

Thanks to the combination of seeding and chaining, the number of positions where our read can be aligned has been reduced to one or two probable positions. The extension mainly involves aligning the remaining fragments between the anchors of the chain. To achieve this, several methods have been introduced, all based on similar principles. We will begin this chapter by explaining the basics of alignment before delving into the methods used today in both short and long reads mapping.

2.3.1 Original dynamic programming algorithms for alignment

To reiterate what we introduced in the 1.4.2 section, the aim of our mapping will be to align our reads q with our reference t. There are three main types of sequence alignment: global, local, and semi-global.

In bioinformatics, when it comes to sequence alignment, two pioneering algorithms in this field are the Needleman-Wunsch(90) algorithm to perform global alignment and the Smith-Waterman(115) algorithm to perform local alignment.

2.3.1.1 Needleman-Wunch

The Needleman-Wunsch algorithm is a global pairwise alignment algorithm. This algorithm is based on a dynamic programming system. The first step is to assign a score when two nucleotides match and a different score when they do not. Using these scores, a scoring matrix is constructed. For example, if we choose a scoring system such that

$$M_{a,b} = \begin{cases} 1 & \text{if } a = b \\ -1 & \text{if } a \neq b \end{cases}$$

Next, it will be necessary to define a penalty system for gaps. As previously described, we can choose either a linear function or an affine function. If the penalty is linear, it will be described by: $P_L(d) = Gd$ where d is the length of the gap and G the penality for the gap. If it is affine, it will be described by: $P_A(d) = G + (d-1)E$ where E is the extension parameter.

In the figure 2.12, to align the nucleotide sequences r =`CAGCTAGCG` and q =`CCATACGA` using the Needleman-Wunsch algorithm, we define a scoring matrix F where each term $F[i,j]$ represents the alignment score for the subsequences $r[1..i]$ and $q[1..j]$. The algorithm computes this matrix starting from the top-left corner at $F_{0,0}$, proceeding row by row from left to right and filling in each cell.

The rows of F correspond to positions in r, and the columns correspond to positions in q.

Next we characterize the boundary conditions, that define the scores at the edges of the matrix based on penalties applied to align gaps:

1. $F_{i,0} = i \times G$, for $1 \leq i \leq |x|$
2. $F_{0,j=}j \times G$, for $1 \leq j \leq |y|$

These boundary values represent the cumulative cost of aligning a sequence with an increasing number of gaps with $G = -1$. Figure 2.12A shows the matrix filled with the boundary condition.

After that, the cells in F are filled using a recurrence relation, each term of a cell is defined using some adjacent cells following the rules:

$$F_{i,j} = \max \begin{cases} F_{i-1,j} + G & \text{skip a position of x} \\ F_{i,j-1} & \text{skip a position of y} \\ F_{i-1,j-1} + M_{r[i],q[j]} & \text{match/mismatch} \end{cases} \tag{2.1}$$

A

	_	T	C	T	T	C	G
_	0	-1	-2	-3	-4	-5	-6
T	-1						
C	-2						
T	-3						
A	-4						
C	-5						
G	-6						

B

	_	T	C	T	T	C	G
_	0	-1	-2	-3	-4	-5	-6
T	$F_{i-1,j-1}$	$F_{i-1,j}$	0	-1	-2	-3	-4
C	$F_{i,j-1}$						
T	-3						
A	-4						
C	-5						
G	-6						

C

	_	T	C	T	T	C	G
_	0	-1	-2	-3	-4	-5	-6
T	-1	1	0	-1	-2	-3	-4
C	-2	0	2	1	0	-1	-2
T	-3	-1	1	3	2	1	0
A	-4	-2	0	2	2	1	0
C	-5	-3	-1	1	1	3	2
G	-6	-4	-2	0	0	2	4

D

	_	T	C	T	T	C	G
_	0	← -1	← -2	← -3	← -4	← -5	← -6
T	↑ -1	↖ 1	← 0	← -1	← -2	← -3	← -4
C	↑ -2	↑ 0	↖ 2	← 1	← 0	← -1	← -2
T	↑ -3	↑ -1	↑ 1	↖ 3	← 2	← 1	← 0
A	↑ -4	↑ -2	↑ 0	↑ 2	↖ 2	← 1	← 0
C	↑ -5	↑ -3	↑ -1	↑ 1	↑ 1	↖ 3	← 2
G	↑ -6	↑ -4	↑ -2	↑ 0	↑ 0	↑ 2	↖ 4

Figure 2.12 – *Schematic representation of the Needlman-Wunch matrix*
A shows the initialized matrix. B shows how to compute one cell. C shows the completed matrix and D shows the various possible alignment with the arrow, the red arrow show the path of the best alignment.

With these rules and the initial conditions in place, the rest of the matrix can now be filled. Following these rules, the matrix is filled from top to bottom and from left to right (Figure 2.12B). Once the matrix is filled, (see figure 2.12C) the optimal alignment can be obtained by following the path with the highest score. Figure 2.12D shows the traceback matrix, which indicates the rule used to calculate the score for each cell. This algorithm

computes the optimal global alignment in $O(nm)$ if n and m are the sizes of the aligned sequence

2.3.1.2 Smith-Waterman

The Smith-Waterman algorithm is similar to the Needleman-Wunsch algorithm, but it enables local alignment. One key difference is the introduction of a fourth rule for calculating the matrix score, which prevents the matrix from having negative scores.

$$F_{i,j} = \max \begin{cases} F_{i-1,j} + G & \text{skip a position of x} \\ F_{i,j-1} & \text{skip a position of y} \\ F_{i-1,j-1} + M_{r[i],q[j]} & \text{match/mismatch} \\ 0 & \text{lowest possible score} \end{cases} \tag{2.2}$$

With this algorithm, the backtracking to determine the optimal alignment does not start from the bottom-right cell but rather from the cell with the highest value. As a consequence, the initialization of the first row and column is set to zero, unlike in global alignment. This algorithm compute the optimal local alignment in $O(nm)$ if n and m are the sizes of the aligned sequence

2.3.1.3 Piece-wise alignment

To manage the extension of alignments the quadratic complexity of traditional methods proves impractical for long reads due to excessive computational and memory requirements. As a workaround, long-read mappers often employ piece-wise extension, where alignment is extended incrementally between adjacent anchors or sequence segments. This technique has been widely adopted in the field(74, 100).

One of the main benefits of piece-wise extension is its ability to accommodate multiple chains corresponding to distinct regions of the query sequence. This capability allows read mappers to produce alignments across several regions simultaneously, aiding in the detection of structural variations(110). Given the importance of structural variation detection in downstream applications, piece-wise extension is now considered a standard methodology in long-read mapping.

Nevertheless, piece-wise alignment can become computationally demanding if the gaps between anchors are significantly large, posing challenges that still require optimization.

2.3.2 Wave Front Alignment (WFA)

For years, despite some improvements(6), sequence alignment algorithms have remained close to quadratic. However, a recent method called WFA (Wavefront Alignment Algorithm) has emerged. While older methods achieved optimal alignment in $O(mn)$, WFA accomplishes this in $O(ns)$, where s is the optimal alignment score and n is the sequence length. Instead of performing a standard top-to-bottom traversal of the dynamic programming (DP) matrix, WFA operates diagonally, focusing only on the cells along the diagonals

with the highest scores (or lowest penalties in the WFA framework). This selective computation skips over many irrelevant cells, particularly when sequences are highly similar, where s is small. To my knowledge, this algorithm has not yet been directly implemented in long-read mappers, but some alignment tools are starting to use it(131).

2.4 Theory and practical implementation

Seeding and chaining can be used to quickly find an alignment. To speed up the process even further, or to reduce memory costs. Mapper authors regularly implement new heuristics on top of existing ones.

2.4.1 Seed heuristics

The seeding step is the one with the fewest additional heuristics. One of the most notable heuristics involves the selection of seeds, such as Graphmap or minimap2. In this case, the tools rejects all seeds that appear more than a certain number of times. These seeds, deemed non-specific because they occur too frequently, are excluded to limit the number of potential anchors during the chaining step and reducing the space of the index.

It is important to highlight that this approach differs from the concept of weighted minimizers. With weighted minimizers, a seed is excluded to allow for the selection of another. In contrast, in Graphmap, the seed is simply excluded without substitution.

2.4.2 Chaining heuristics

In the chaining step, most mappers adopt a similar strategy when calculating the optimal anchor chain. If the chain score between two anchors falls below a certain threshold, the chaining process is terminated. Each tool uses its own threshold, which depends on the scoring system implemented.

2.4.3 Alignment heuristics

For the alignment step, this is where numerous heuristics can be found. For the DP matrix, many tools use the banded alignment, which, like WFA, calculates only a portion of the matrix(93, 78). However, unlike WFA, the banded alignment does not guarantee an optimal alignment. In practice, since the optimal alignment is often located near the diagonal of the matrix, the banded alignment achieves optimal alignment in most cases. Another commonly used heuristic for alignment is the Z-drop heuristic, employed by Minimap, which is an adaptation of the X-drop heuristic from BLAST. This heuristic stops the alignment before reaching the edges of the sequence if the alignment score drops below a certain threshold.

Finally, the last heuristic we'll talk about, widely used by most long-read mappers,is soft clipping. Soft clipping consists in not aligning a part of or an extremity of the reads. It avoids the time-consuming task of trying to align regions that don't match, or match only slightly. It has been present since short reads, and makes it easy to process data with

very erroneous read edges, or to handle cases where barcodes are present. It has proved particularly useful with 3rd generation data and especially nanopore data for dealing with chimeric reads (where two DNA fragments are accidentally sequenced as one). Although this method is now standard practice in many tools, it comes with a lack of transparency: while most mappers implement soft clipping, the criteria or thresholds for deciding when and which part of a read should be clipped are rarely made explicit.

2.4.4 Mapping tools and their evolution

If we look at the history of the most well-known mapping tools (figure 2.13), it begins with YAHA(38) and BLASR(17) in 2012. These two tools used seeds similar to MEMs, which were widely employed by short-read mappers. In 2016, minimap(73) and graphmap(118) were released. minimap use the famous minimizers and graphmap used spaced k-mers. At the time of its release, GraphMap demonstrated better performance compared to tools originally designed for short reads but adapted for long reads, such as LAST(44), BWA-MEM(72), and DAligner(89). As for Minimap, it is more challenging to provide clear results because the original paper did not include benchmarks comparing it to the tools cited in the bibliography.

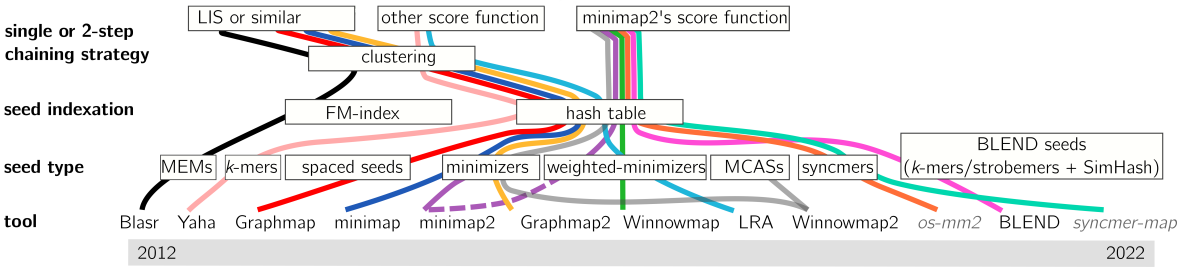

Figure 2.13 – *"Long read mapping tools over time. Tools and techniques are presented from oldest to most recent, from left to right. The figure presents implementation names at the bottom, then goes up to the different steps: seeding, with seed selection strategies and indexation, then chaining and pairwise alignment strategies. The dotted line for minimap2 means its implementation evolved from strategy in plain line to strategy in dotted line. The grey italic names denote for proofs-of-concept rather than tools." From Sahlin, K., Baudeau, T., Cazaux, B. et al. A survey of mapping algorithms in the long-reads era. Genome Biol 24, 133 (2023). https://doi.org/10.1186/s13059-023-02972-3.*

In 2018, minimap2 was released, retaining the same seeds as Minimap, with changes primarily focused on the chaining step. Minimap2 quickly became one of the most widely used tools for long-read mapping.The following year, a preprint for GraphMap2 appeared. However, no official paper has been published to date. When examining the seeds used, it is evident that GraphMap2 initially retained the concept of spaced k-mers before also adopting minimizers. In 2020, a new tool named Winnowmap was released. This tool introduced weighted minimizers(58).

In 2021, a new paper by the author of minimap2 indicated that minimap2 now incorporates seeds similar to weighted minimizers(76). These seeds improve mapping performance in highly repetitive regions and enhance the detection of structural variants in these areas.

The same year saw the emergence of a new tool called LRA, delivering excellent results in repetitive regions, such as centromeres(100). This tool also utilizes weighted minimizers but differs from minimap2 in its other mapping steps.In 2022, Winnowmap2 introduced MCASs (Minimally Confidently Alignable Substrings)(57).

Finally, in 2023, a new tool called BLEND was developed, employing fuzzy seeds for read mapping(41). It is interesting to observe the prominent role minimap2 in the mapping of long reads. It is also worth noting that even for short-read mappers, some tools have been enhanced by incorporating minimizers.

Even though new seeds and chaining strategies continue to emerge, such as Blend, which also uses a type of spaced seed, it is evident from the literature that minimap2 have had a significant impact.

Chapter 3

Evaluation of long read mapping tools for viral genome analysis

The text found in the following sections is largely drawn from Thomas Baudeau, Camille Marchet, Mikaël Salson. Assessing Long-Read Mappers for Viral Genomics. bioRxiv 2024.11.25.625163; doi: https://doi.org/10.1101/2024.11.25.625163. This article has not yet been published, but has been submitted to PCI Genomics.

The long-read technology being relatively recent, there are few comprehensive comparisons between mappers designed for this type of data. Moreover, benchmarks for different mappers, when available, often lack diversity(28, 123) and, to the best of our knowledge, have never been tested on viral datasets. To address this, a benchmark of the various tools from the literature was constructed to evaluate their performance on these specific datasets.

3.1 The relevance of comparing mapping tools

As presented in section 2.4, third-generation mappers incorporate numerous heuristics to speed up the reads alignment. However, when we examine these heuristics, it becomes clear that they are often adapted to address eukaryotic or even prokaryotic organisms. For example, the size of the seeds, or the use of splicing signal for splicing. Viruses, on the other hand, are frequently overlooked in these considerations. Yet, many studies are conducted without accounting for the specific characteristics of viral data. To address this issue, a benchmark of the most commonly used tools in the literature was developed.

3.1.1 Tools selection

To create a selection encompassing the majority of the different algorithms available for long-read mapping, nine tools were chosen based on their specific features.

We included bwa-mem2 (122) and Magic-BLAST (14) in our selection. Originally tailored for short-read data, bwa-mem2 now offers features for long-read mapping, adapting to broader sequencing needs. Magic-BLAST, derived from the BLAST framework, employs fixed-length exact seeds but is designed to accommodate indels commonly associated with

Oxford Nanopore Technologies (ONT) sequencing errors. It also handles large gaps, such as those introduced by splicing.

Among early long-read mapping tools, we considered BLASR (17) and GraphMap (118). As discussed in 2.4.4, BLASR was one of the pioneering mappers developed for long reads, indexing all k-mers to identify primary matches in the seeding phase. It also introduced the critical "chain" step in the seed-chain-extend paradigm. Similarly, GraphMap, though no longer actively maintained like BLASR, remains notable for its strategy using spaced-seeds

For more recent approaches for long-read mapping, we evaluated minimap2 (74), Winnowmap2 (57), BLEND (42), and lra (100). minimap2 has become a versatile and widely adopted tool across various bioinformatics applications, including genome assembly, variant calling, and metagenomic analysis. It is often integrated into pipelines and incorporates numerous innovations (see Chapter 2). Winnowmap2 builds on its predecessor Winnowmap(56), tailoring its performance for long-read mapping. BLEND specializes in mapping high-quality long-read data, while lra introduces a novel chaining algorithm designed to better align structural variants. This capability has enabled lra to achieve the best results in certain datasets from a recently published benchmark study(79). It is important to mention that some tools, such as Graphmap2, were removed from the benchmark due to the impossibility of obtaining results that could be processed.

3.1.1.1 Tools parametrisation

An important detail that was not addressed in the previous section is that each step of the seed-chain-extend algorithm is accompanied by several user-adjustable parameters. In this benchmark, we focused on the parameters related to seed selection. Examining the default settings of the tools (Table 3.1), we observe that they tend to revolve around a seed size of 15. To understand this parameter choice by the tools, it is essential to consider the number of occurrences of k-mers in a genome.

In figure 3.1, we observe the distribution of k-mer occurrences in the human genome for different values of k. With a k-mer size of 20, the majority of k-mers are unique, although a significant proportion still appear several times. When the size is set to 15, this trend continues, but with a lower proportion of unique k-mers. However, when k is reduced to 10, the trend reverses: very few k-mers remain unique and most appear several times. For even smaller values, such as 5 or 7, almost all the k-mers appear more than 255 times.

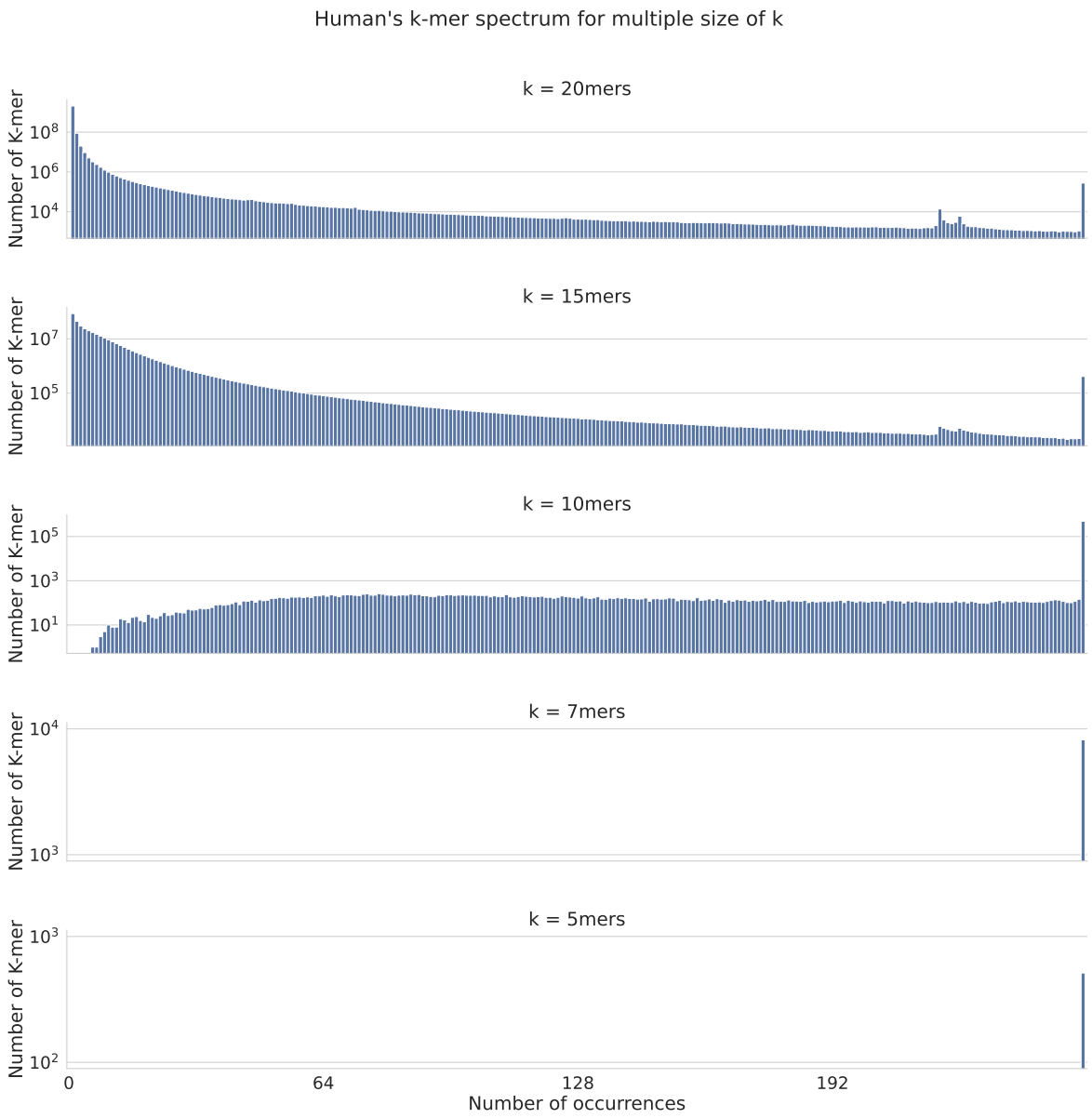

Figure 3.1 – *Human's k-mer spectrum for multiple size of k*
A k-mer spectrum represents the distribution of how many times each unique k-mer appears in the genome. The max number of occurrences is set at 255.

This shows that the larger the size of the k-mers, the higher the probability that the seeds collected are unique. From the explanations given in the section 1.4.2 on seed collection, one might wonder why we don't just collect unique seeds, which would simplify the chaining step. However, it is essential to understand that increasing the size of k also increases sensitivity to sequencing errors. Consider a scenario in which sequencing errors occur every 20 bases; if we use a seed size of 20, most seeds will contain at least one error, making it unlikely that exact matches will be found. On the contrary, shorter seeds are more tolerant of errors.

The objective is thus to use a seed size that is large enough to reduce computation time by limiting the number of matches, but small enough to ensure that a sufficient number of reliable anchor points can be found, even in the presence of sequencing errors.

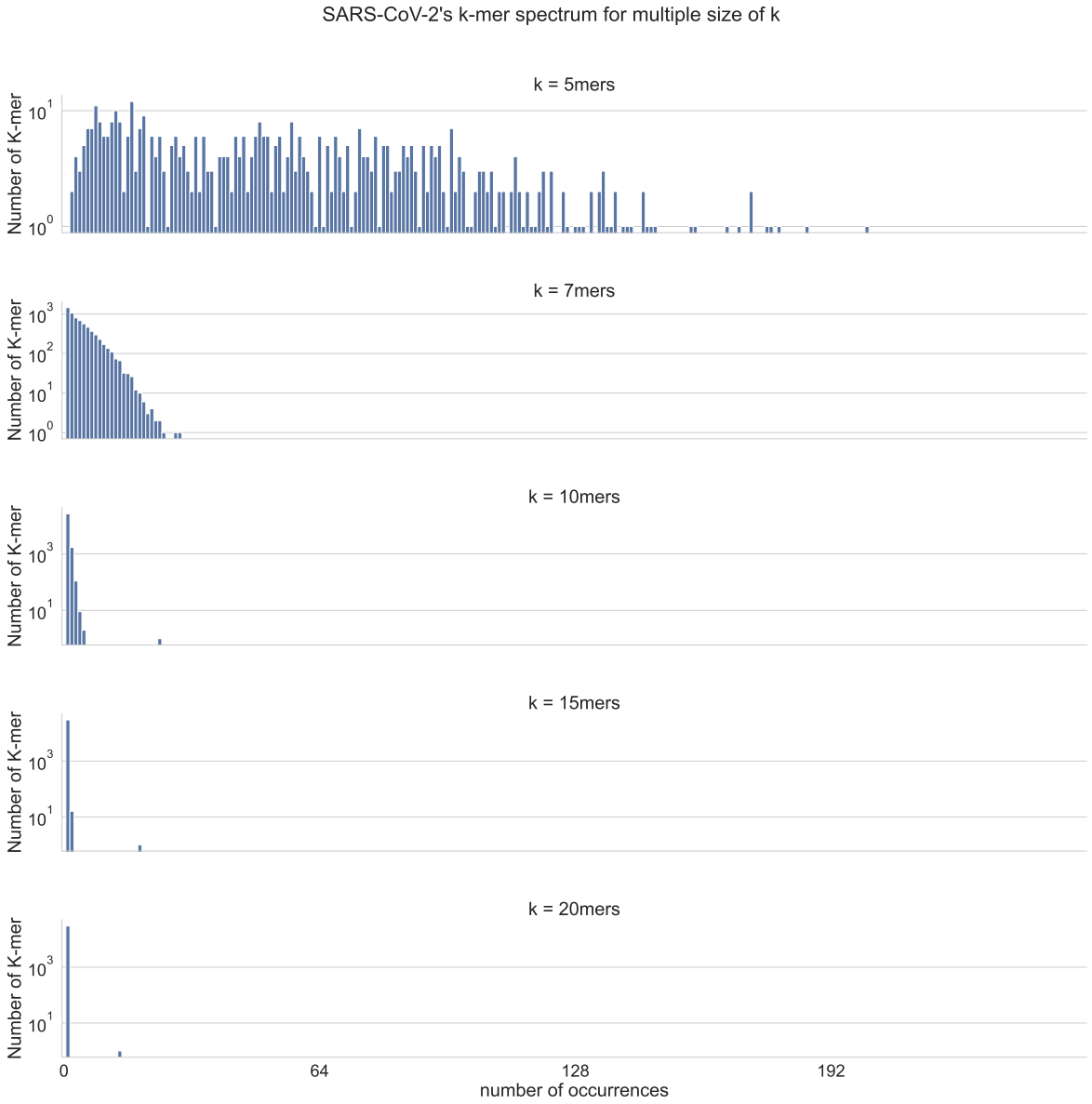

Figure 3.2 – *SARS-CoV-2's k-mer spectrum for multiple size of k*
A k-mer spectrum represents the distribution of how many times each unique k-mer appears in the genome. The max number of occurrences is set at 255

To get an idea of the most suitable size for viruses, we can compare their k-mer spectrum with that of humans. Thus, with viral genomes such as SARS-CoV-2 or HIV-1 (figure 3.2 and 3.3), we find that with k sizes of 20, 15, or 10, almost all k-mers are unique in the genome. With a k size of 7, the distribution of k-mers in terms of occurrence becomes slightly broader. With k-mers of size 5, the number of occurrences increases, with some k-mers observed up to 60 times for SARS-CoV-2 and over 99 times for HIV-1.

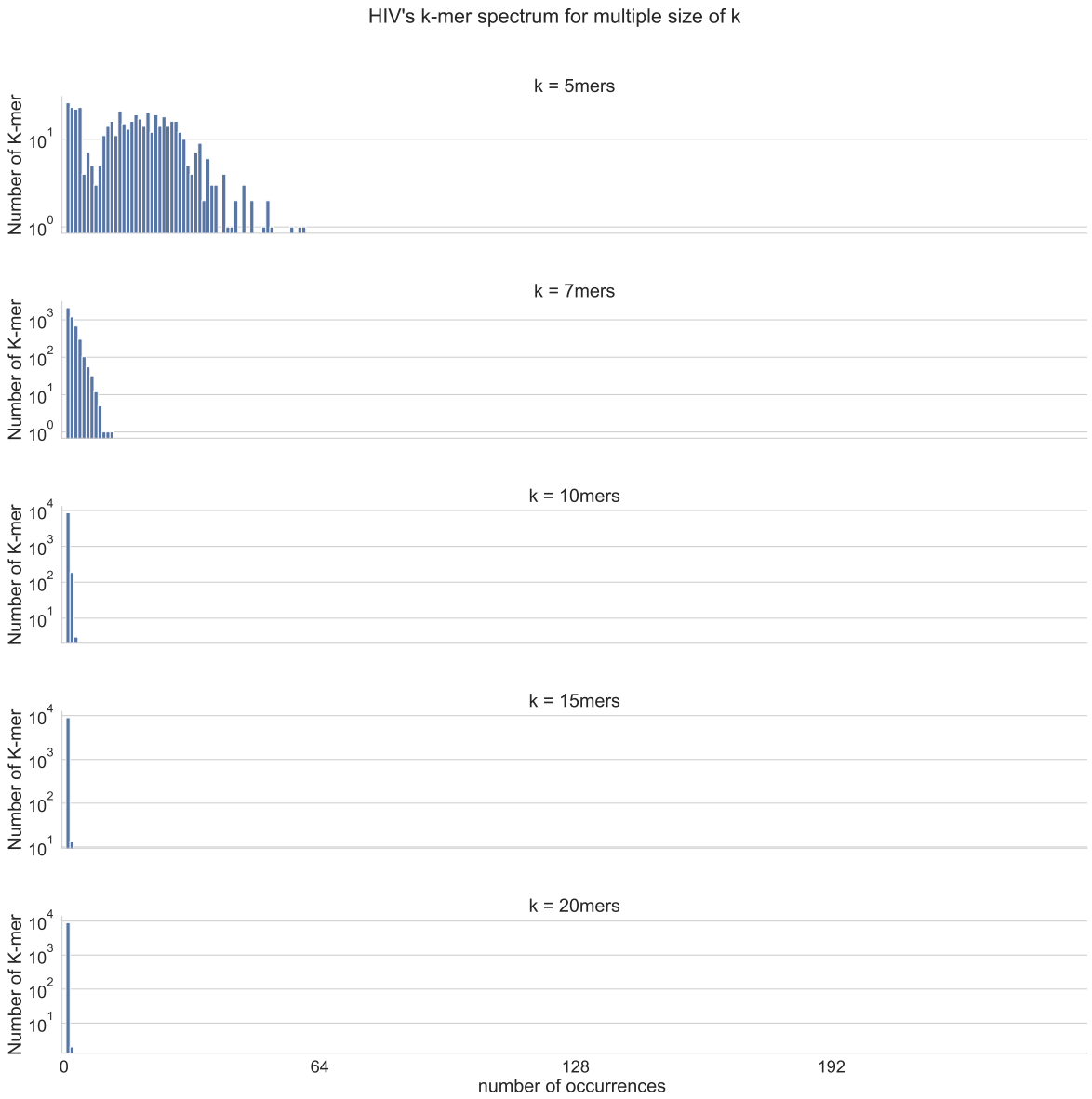

Figure 3.3 – *HIV's k-mer spectrum for multiple size of k*
A k-mer spectrum represents the distribution of how many times each unique k-mer appears in the genome. The max number of occurrences is set at 255

As we have seen, viruses have a much narrower spectrum of k-mers than that observed for the human genome, due to their much smaller genomic size. The hypothesis formulated at the start of this study was therefore that it would be more appropriate to use smaller seeds to map viral data. This intuition was confirmed by a recent study by Ayad et al. (7), which introduces the concept of seedability, i.e. the ability of a set of seed parameters to optimise alignment performance.

In this work, the authors evaluate different seed parameters in the context of the human genome, using the minimap2 tool. Their results show that, even for a large genome such as the human genome, the use of shorter seeds makes it possible to obtain more alignments, and of better quality. These observations reinforce the idea that, in the case of viruses, the use of small seeds is not only appropriate, but probably optimal. Especially since, given

the small size of viral genomes, the computational impact of using smaller seeds is very limited.

tool name	Version	Parameter details	Default parameter	Parameter One	Parameter Two
minimap2	2.28	-k the size of the minimizers -w the size of the window	k = 15 w = 10	k= 12 w = 4	k = 7 w = 5.
minimap2 (old)	2.1.1	-k the size of the minimizers -w the size of the window	k = 15 w = 10	k= 12 w = 4	k = 7 w = 5.
syncmer-minimap2	2.1.1	-k the size of the k-mer, -w the size of the window, --downsample the down-sampling factor, --s-mer the size of the sync-mer, --pos the sync-mer parameters	k = 15 ; w = 10 downsample = 1 s-mer = 5 pos1 = 3 pos2 = 9	k= 12 w = 4	k = 7 w = 5.
Magic-BLAST	1.5.0	-word size , the minimum number of consecutive bases that must match exactly. the minimum value for this parameter is 12.	-word size = 18	-word size = 16	-word size =22
GraphMap	0.5.2	The parameter k corresponds to the size of k-mers inside the graph, while A corresponds to the minimal size of a match to be chosen as an anchor.	k = 6 A = 12	k = 7 A = 8	k = 5 A = 7
BLASR	5.3.5	–minMatch, which corresponds to the seed size, and -nCandidates, which affects the number of alignments tested.	minMatch =12 nCandidates =10	minMatch =12 nCandidates =10	minMatch =12 nCandidates =10
bwa-mem2	0.5.2	-k minimum seed length; -W discard a chain if seeded bases shorter than W; -w band width for banded alignment	k = 14 w = 1 W = 1	k = 7 W = 10 w = 1	k = 5 w = 1 W = 1
Winnowmap2	2.03	-k , which corresponds to the size of the k-mer, and -w, which corresponds to the size of the window, are used.	k = 15 w = 50	k = 15 w = 1	k = 12 w = 4
BLEND	1.0.0	-k, representing the size of the k-mer, -w, representing the window size, –fixed-bits number of bits used for the hash values of seeds, and –neighbors representing the number of kmer to generate a seed.	k = 7 w = 10 fixed-bits = 32 neighbors = 11;	k = 7 w = 5 fixed-bits = 24 neighbors = 8	k = 12 w = 5 fixed-bits = 30 neighbors = 10
lra	1.3.2	-W The minimizer window size,-K The word size.	-W =10 -k = 17	-W = 4 -k = 12	-W = 5 -k = 7

Table 3.1 – *Table of used parameter settings for each tool*
Parameters one and two are two sets of set parameters that lead to a selection of smaller seeds. Parameter one is also referred to as 'long' in some figures, as the seeds, even if smaller than the default parameter, are slightly larger than parameter one, which is referred to as 'short'. The idea behind these no's is that long would be a higher parameter for longer reads and short would be more suitable for shorter reads.

3.2 Data generation

In aiming to test the different mappers, we used two types of dataset. The first is a simulated dataset, which provides access to the underlying reality of the dataset, i.e. we know

where the sequencing errors are, what type they are, and we can also add mutations in the sequenced species. The second dataset is made up of real data, to confirm that the results obtained with the simulated dataset are also found with the real dataset.

3.2.1 Generating ground truth datasets

To generate data simulating ONT reads, several tools were available. These include pbsim2(94) and pbsim3(96). When the benchmark was created, only pbsim2 was available. When pbsim3 was released, it was decided to retain pbsim2, as it had already been used in several studies and was widely accepted by the scientific community. Another simulator, NanoSim(129), had also been considered. This tool generates artificial reads based on a real data set. However, NanoSim did not allow for easy control of the error rate of reads, which limited its flexibility for the creation of different datasets. Nine parameters were chosen and varied according to the different data sets :

- **Length** : The first parameter is the length of the reads. When working with viral data, many protocols involve an amplification step. The amplicons being much smaller than the usual long reads (i.e. without the use of amplicon).
- **Virus**: The second parameter is the species used for the reference genome. This makes it possible to check that the results obtained are not specific to a particular species of virus. Two viruses of different orders were chosen.
- **Error Rate**: The third parameter is the average error rate of our data set. This parameter will allow us to see how the mapper behaves at different error rates.
- **Model**: The fourth parameter is the pore model (see 1.3.3.1) used by pbsim2. This is to see if it has any influence on mappers or other possible data processing tools.
- **Contamination**: The fifth parameter is the percentage of human reads that will be added to the sample. The percentage is calculated in relation to the number of reads of the reference species. This is to check the specificity of the various mappers.
- **Coverage**: The sixth parameter is the coverage for each data set. The purpose of this parameter is not so much to test the mapper as to see how it will influence the variant calling. We'll talk about metrics in more detail in the section entitled 'Metric for mapping comparison'.
- **Nb of variants**: The 7th parameter is the number of variants present in the sample. In viruses, it is possible to have several variants of the same species in the same sample. The aim is to see if and how this can influence the mapper.
- **Proportion of variants**: The eighth parameter is the percentage of reads from each variant created with parameter 7, which will influence the number of reads of the variant in relation to the reference species.
- **Variations**: The ninth is the number of mutations introduced for each variant. Here, the choice has been made not to mimic the possible evolutionary pressures of the genome. In other words, the variation can be either a substitution or an indel with the same percentage of chance on the genome in a completely random way. However, two mutations cannot cancel each other out or overlap. As the aim was to test the mappers, it was felt that it was not necessary to have mutations that copied the possible pressure on the mutations.

We defined eight data sets, each varying according to some of the conditions mentioned above, in order to assess their impact on the mapping stage. All the parameters and names of these sets are shown in Table 3.2. The choice of error rate, read size and contamination level was made to reflect realistic values that could be observed in real data. We summarize below the characteristics we wanted to asses in each dataset:

S1 and **S3** test low and high coverage respectively and three percentage of error rates.

S2 different viruses and three error rates.

S4 high host contamination of the sample.

S5 high divergence between the sequenced virus and the reference one.

S6 and **S7** different sequencing pores.

S8 two similar viruses in the sample.

Name	Length	Species	Error Rate	Model	Contamination (%)	Coverage	nb Variant	prop Variant	%v Variant
S1	500,7500	covid	0.95 0.99 0.90	R103	10	10	1	99	2
S2	500,7500	covid,vih	0.95,0.99,0.90	R103	10	100	1	99	2
S3	500,7500	covid	0.95 0.99 0.90	R103	10	1000	1	99	2
S4	500,7500	covid	0.95	R103	10000	100	1	99	2
S5	500,7500	covid	0.95	R103	10	100	1	99	7
S6	500,7500	covid	0.95	R94	10	100	1	99	2
S7	500,7500	covid	0.95	R94/R103	10	100	1	99	2
S8	500,7500	covid	0.95	R94	10	100	1	50	2

Table 3.2 – *Table of the different conditions for each experiment.*

With all the possible conditions, a total of 34 data sets are generated.

3.2.2 Real datasets

In addition to the generated data sets, four real data sets are used. These four sets, taken from a previous study (52), are available at the following address: github.com/iqbal-lab-org/covid-truth-datasets or on the ENA project under identifiers PRJEB51850 and PRJEB50520. In this study, SARS-CoV-2 variants were detected using high-quality sequencing performed with Illumina and ONT technologies, and each variant was manually validated. We selected four datasets from this study, prioritizing those for which ARTIC sequencing analysis was available. These four sets will be named R1a, R1b, R2a and R2b, and all were sequenced on a Nanopore platform.

- R1a contains 55,615 reads sequenced according to the midnight-v2 amplicon scheme.
- R1b contains 557,718 reads sequenced according to artic-v4.1.
  - Both datasets are from the same strain (PRJEB51850), with four substitutions and one deletion in comparison with the standard reference genome MN908947.3.
- R2a contains 45,484 reads sequenced according to the midnight-v2 scheme.

- R2b contains 157,189 reads sequenced according to artic-v4.1.
  - Both datasets are from the same strain (PRJEB50520), characterized by 32 substitutions and three indels in comparison with the standard reference genome MN908947.3.

All data sets contain barcodes and no cleaning steps were applied. This was done so as not to add an additional tool that could impact the results, and because no cleaning had been carried out in the study from which the data came.

3.2.3 File format for the data generation part

To handle the generation process, we have created a python pipeline available on github (https://github.com/ThomasBaudeau/BenchMapL). This pipeline is implemented using Snakemake and follows a multi-step process.

Figure 3.4 shows how the pipeline works schematically. Note that it is possible to obtain several sets of data from a single input file. As can be seen from the schema, there are many formats available when it comes to bioinformatics. The different formats will be detailed below.

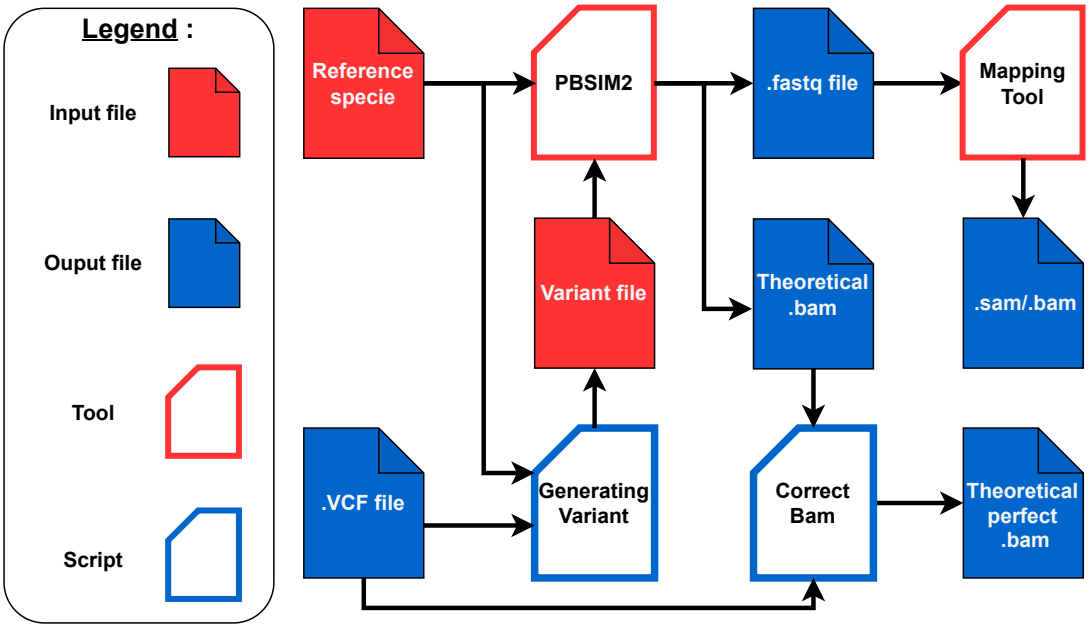

Figure 3.4 – *Schema of the simulated data generation part of the analysis pipeline*
The reference species file corresponds to the FASTA file for the species to simulate data on. The variants file is a tabulated file containing the mutations to introduce. The theoretical BAM file corresponds to the theoretical alignment of the reads generated by pbsim2; there is one theoretical BAM file per variant generated, and the alignment is carried out using the variant as a reference. The theoretical perfect BAM file corresponds to the theoretical BAM generated by pbsim2, but modified so that the alignment is carried out in relation to the reference species, and not the variant. The theoretical perfect BAM is abbreviated to perfect in the result section.

3.2.3.1 The fasta format

The fasta format is a simple file format used for storing biological sequences such as DNA, RNA or proteins. The format is structured in two parts: a header line consisting of a chevron > followed by the sequence's unique identifier. A description of the sequence can be added to the header. The second line consists of the sequence. It can be on one line or on several lines. When several sequences are present in the same file, we speak of multi-fasta. This is the format used for references in all long-read mappers.

3.2.3.2 Sam/bam

SAM(77) (Sequence Alignment/Map) is a format used to store sequence alignments against a reference. It is now widely used by short and long reads mappers. SAM is in text format and also has a binary equivalent called BAM. The SAM format is now maintained and updated by the community via the samtools tool. The latest specifications and best practices relating to this format are available at https://samtools.github.io/hts-specs/

The SAM format consists of two parts: the first is the header, which contains several fields identified by tags, providing general information about the alignment and the sequences analysed. The possibles tags are :

- **@HD** : General file information (version, alignment order).
- **@SQ** : List of reference sequences (name, length).
- **@RG** : Information on read groups (samples, libraries).
- **@PG** : Details of the program used for alignment.

The second part contains the alignment results for each read. It is structured into 11 mandatory columns, to which optional fields can be added to provide additional information.

3.2.3.3 Variant Calling Format

The Variant Call Format (VCF) is used to store genetic variations, such as single nucleotide polymorphisms (SNPs), insertions, deletions (indels) and structural variations. A VCF file consists of two main parts, the first one is the header which defines metadata and the second one consists of the diffents row for each variant detected. The second part begin with the name of the different fields.

3.3 Metric for mapping comparison

In order to compare our mapper, we have used several metrics. Some of these metrics are specifically applied to simulated files, while others can be used for both simulated and non-simulated datasets.

3.3.1 Standard metrics for mapping

To evaluate the performance of our mappers, we first analysed the alignment obtained using several metrics. The first and most basic metric is the percentage of reads mapped. In our simulated dataset, this percentage corresponds to the number of reads aligned relative to the total number of reads from the reference species (mutated or not). For real datasets, it's the ratio between the number of mapped reads and the total number of reads. However, in the latter case, it is impossible to guarantee that reads aligned with the reference species actually belong to it.

In our simulated dataset, a second metric is available: the number of reads from the contaminating species that have been mapped. In our case, the contaminating species is human RNA from sample ID SRR16071311, available from the NCBI SRA database.

A third metric, used for real datasets, concerns the number of reads whose alignment contains a soft-clipped portion (see section 2.4.3). Reads that are soft-clipped are divided into five non-overlapping groups to assess the impact of soft-clipping. The groups are : no soft-clipping in the reads (0), less than 5 soft-clipped bases (from 1 to 5), less than 10 (from 6 to 10) soft-clipped bases, less than 20 soft-clipped bases (from 11 to 20) and more than 20 soft-clipped bases (21 and more).

Finally, for the simulated dataset, a fourth metric was used to assess the accuracy of the reads' alignment. Figure 3.5 provides a schematic illustration of how this metric works. Thanks to the perfect theoretical file, we have access to the expected alignment positions of the reads. We then evaluate each read by comparing its actual alignment position with its theoretical position. This metric is then divided into several categories, similar to those used in the metric that analyses soft clipping in the reads.

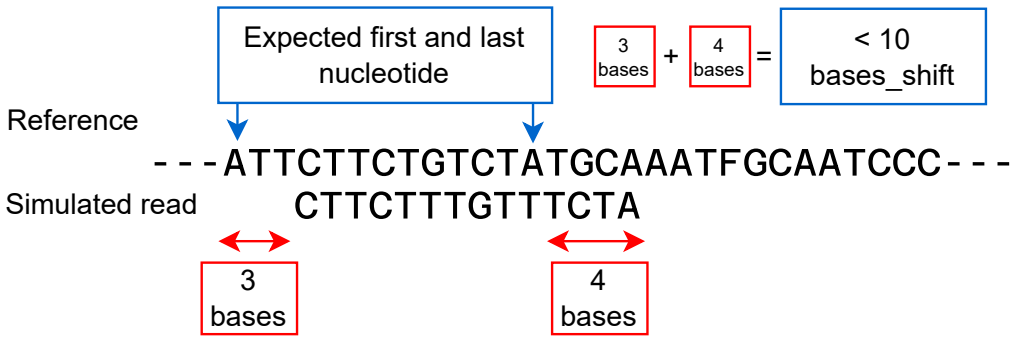

Figure 3.5 – *Explanatory schema of the base-shifts metrics*
The expected position is given by the theoretical perfect bam file. The position of the two ends of the reads are taken into account to calculate the offset.

3.3.2 Impact of mapping tools on downstream analysis

To assess the influence of mapping tools and parameters on subsequent analyses, we focused on an application widely used in genomics: variant calling. For this study, we selected two tools: bcftools (71) and medaka (github.com/nanoporetech/medaka). Medaka was chosen for several reasons. Firstly, it is an integral part of the widely used ARTIC

pipeline, which is designed to analyse viral ONT sequencing data (1). Medaka relies on neural network-based models to improve variant calling accuracy. For our study, we used the two pre-trained models available in medaka's GitHub repository, R9.4 and R10.3, which correspond to different versions of the ONT pore. These models enable variant calling to be optimized for datasets generated using ONT sequencing technology. In recent years, an increasing number of variant callers have incorporated machine learning approaches, in particular pre-trained or trainable neural networks. While these models offer significant advantages, such as improved accuracy and adaptability to sequencing noise, they also have limitations. One of the main challenges is the difficulty of acquiring high-quality viral sequencing data for model training. This constraint makes it impractical to develop and train customized models for each new dataset. Consequently, the availability of pre-trained models is a valuable solution, as their performance on specific species has already been validated. However, despite these advantages, the lack of transparency in the processes by which these models are trained raises concerns about potential biases and sensitivity issues. A model trained on a limited set of viral genomes may not generalize well to all datasets, leading to false positives or the omission of important mutations.

To alleviate these concerns, we incorporated bcftools as a comparative method. Unlike medaka, bcftools does not rely on neural network but it is based on statistical methods , making it a valuable monitoring tool for identifying potential biases introduced by deep learning-based variant calling tools. By comparing the results obtained by the two tools, we sought to assess whether certain mapping parameters disproportionately influence neural network-based variant calling. If discrepancies between bcftools and medaka appear under specific conditions, this could indicate biases in neural network training, potentially linked to the mappers used in the upstream analysis.

Thus, by selecting both a traditional statistical variant caller (bcftools) and a neural network-based variant caller (medaka), our analysis provides a more comprehensive perspective on the potential impact of mapping strategies on variant detection results. This approach allows us to assess whether certain mapping methods introduce biases that propagate through the analysis pipeline, ultimately influencing biological interpretations and downstream applications.

To evaluate the results, we'll look at the number of false positives (a base detected incorrectly as a variant), false negatives (a variant not detected) , true positives (a variant found correctly) and the F1-score ($\frac{TP}{TP+\frac{1}{2}(FN+FP)}$).

We use vcfdist(31) to standardize the comparison between the VCF files produced by bcftools and medaka, and the ground truth VCF. This tool is specifically designed for accurate benchmarking of variant calling, and has been successfully employed in several recent studies(35, 48).

3.3.3 Non selected metrics

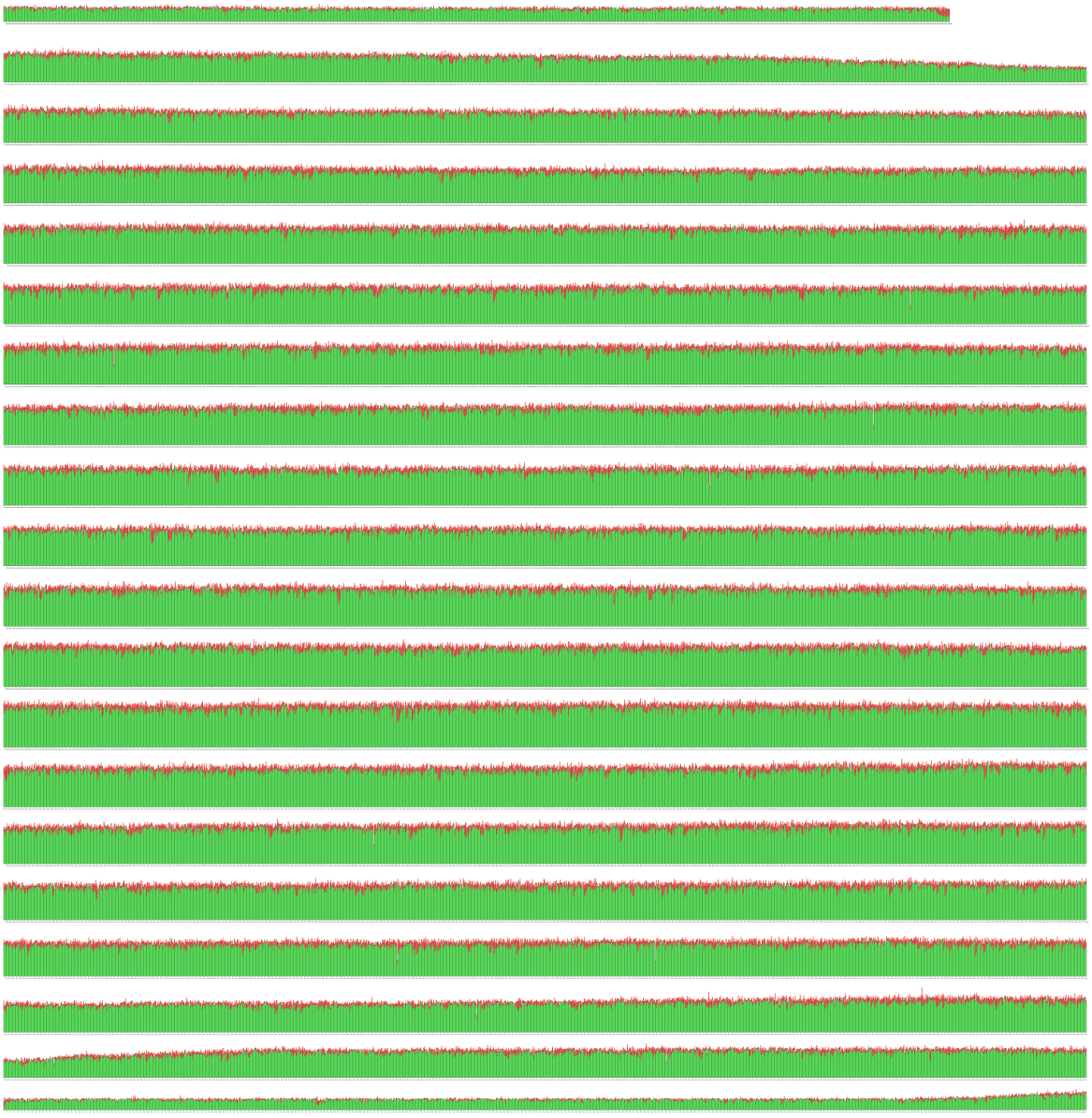

Figure 3.6 – *Schema of the alignment score metric*
Alignment score from a minimap2 BAM file. The species is SARS-CoV-2 and the file contains no variants but has a 20% sequencing error rate. One pixel is drawn for each nucleotide in the theoretical perfect BAM alignment. The pixel in the bottom-left corner corresponds to the first nucleotide of the reference. If multiple reads cover the same nucleotide, an additional pixel row is added. For visualization purposes, the reference has been split into multiple sections. A green pixel indicates that the alignment in the observed BAM matches the theoretical perfect BAM at that position; a red pixel indicates a mismatch.

In addition to the metrics mentioned above, tests have been carried out to compare the alignment of reads at global level. Using the reference .bam file, we try to create a new metric called the alignment score. To calculate the alignment score, the theoretical perfect bam that gives the expected alignment was used to compare with the alignment obtained.

As a first step, a graphical visualization of the alignment was used to identify any difficult-to-align areas, i.e. regions where alignment errors are more frequent. Figure 3.6 shows an

example of such a visualization used to detect problematic regions for mappers. In this representation, each pixel corresponds to a base in the reference, and its color indicates whether the alignment of a read at that position matches the expected (green) or not (red). Multiple stacked rows indicate multiple reads covering the same base.

As shown in the figure, there are no clearly defined areas with consistently high alignment errors. In other words, apart from a few isolated positions, no regions display a strong, continuous accumulation of red pixels that would indicate particularly challenging zones for alignment.

In a second step, a metric called CIGAR correspondence was tested to measure the correspondence between two alignments. Just like the alignment score metric, we compare the two bam. For each position in the reference we have a score, with $+1$ when a nucleotide of a read is badly positioned, -1 when a nucleotide is soft clipped, 0 when it matches.

However, as designing a satisfactory scoring system for both the alignment score and CIGAR correspondence metrics would have been well beyond the scope of this thesis, it was decided to abandon these two approaches.

To understand why these two metrics were abandoned, let's consider the different cases that can be found when comparing two alignments. Four possible cases were defined:

- Case A: The read has been mapped and the nucleotide is aligned in the same way as in the theoretical alignment.
- Case B: The read has not been mapped, which means that all the nucleotides in the read are incorrectly aligned.
- Case C: The read has been mapped, but part of the nucleotides alignment differ from the theoretical perfect bam.
- Case D: The read is partially aligned, a part of it is soft-clipped, meaning that a portion of the read was not aligned.

As we can see, Case A is the ideal scenario that we want to observe most frequently in a mapper. However, Cases B, C, and D highlight different aspects of a mapper's performance. Case B reflects a mapper's ability to correctly select reads for alignment, while Case C evaluates its accuracy in globally aligning a read. Case D, on the other hand, concerns the mapper's ability to properly align the read's edges.

It could have been possible to retain only cases C and D for our two metrics. However, this would have introduced an artificial bias by favouring mappers that completely discard difficult reads. Mappers that attempt to align as many reads as possible would have been penalized, leading to a distorted evaluation of overall performance.

3.4 Results

The Results section is largely drawn from the corresponding part of Thomas Baudeau, Camille Marchet, Mikaël Salson. Assessing Long-Read Mappers for Viral Genomics. bioRxiv

2024.11.25.625163; doi: https://doi.org/10.1101/2024.11.25.625163. Only a few sentences have been modified to incorporate figures that were previously included in the supplementary materials.

3.4.1 General mapping results

Raw mapping rates: First, we analyze the mapping rate for each experiment. Figure 3.7A shows that high mapping rates were observed across almost all tools tested.

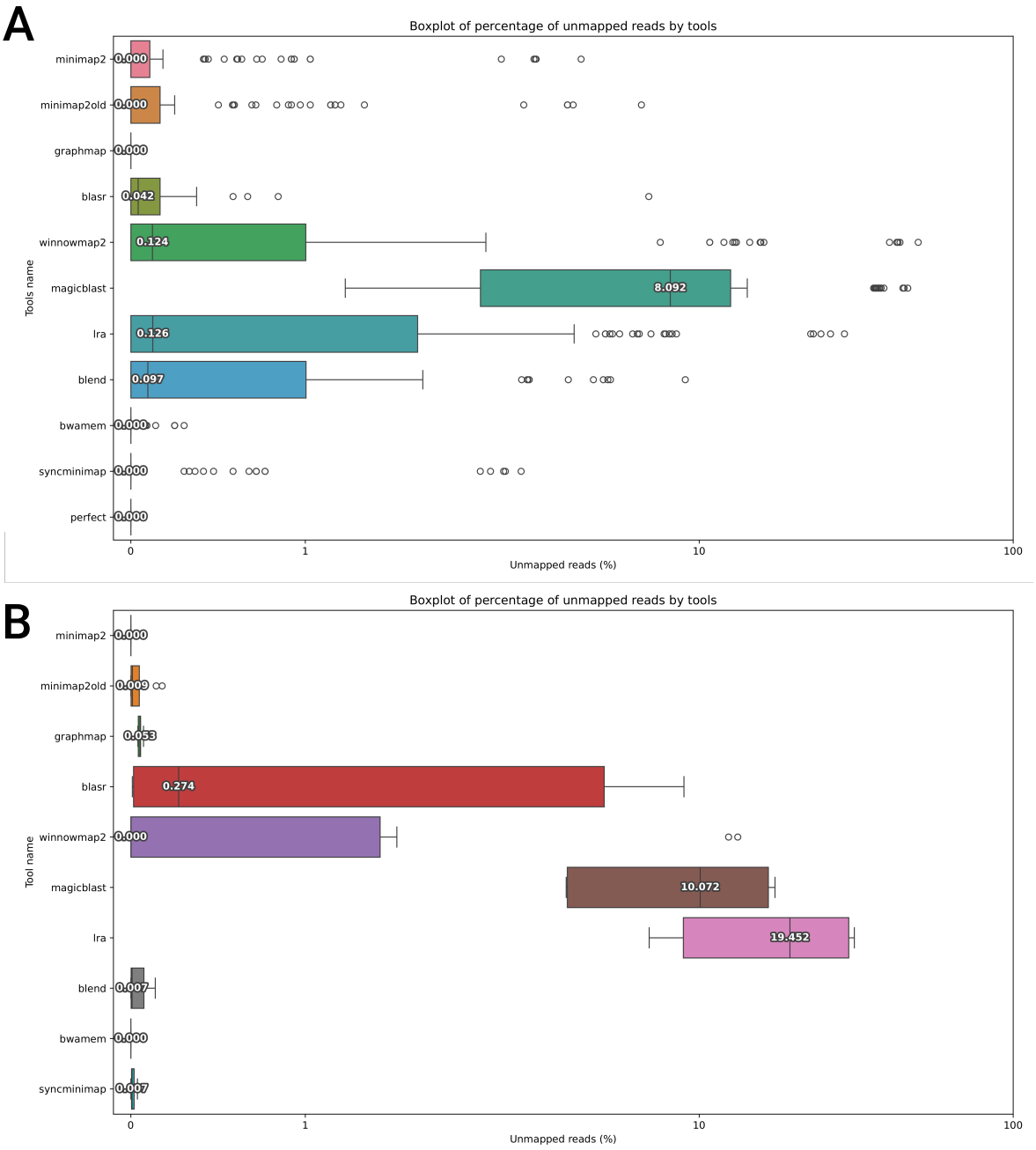

Figure 3.7 – *Boxplot of the percentage of unmapped reads for each tool for the 34 samples generated. The x-axis is in symlog scale. For minimap2, GraphMap,bwa-mem2, syncmer-minimap2, and perfect, the median is 0. For BLASR, it is 0.084, 0.140 for Winnowmap2, 0.195 for BLEND, 0.210 for lra, and 9.575 for Magic-BLAST.*

When combining the 8 experiments, 3 groups of tools can be distinguished (fig 3.7A).

The first group comprises the tools with the best results, including bwa-mem2, GraphMap, minimap2, syncmer-minimap2, BLASR and perfect ,to recall ,perfect corresponds to a bam given by pbsim2 with the theoretical alignment. The second group corresponds to tools with a lower number of mapped reads in some datasets, made up of lra, BLEND and Winnowmap2; and finally a third group with Magic-BLAST, which on the whole has poorer results than the other tools.

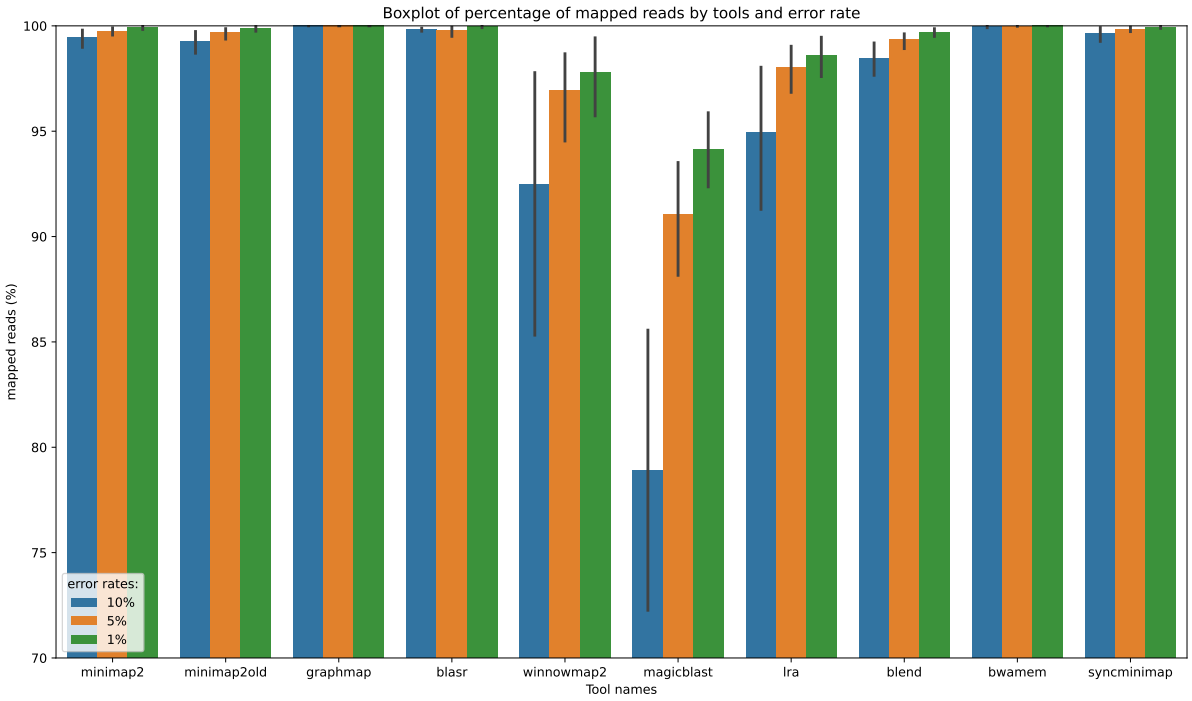

Figure 3.8 – *Barplot of the percentage of mapped reads by tools and reads error rates*

A more detailed analysis of each dataset shows that under conditions replicating ARTIC characteristics (such as read length = 500 and error rate = 10 or 5 %), bwa-mem2 and GraphMap mapped a higher number of reads compared to other approaches under their default settings (Figures 3.8 and 3.9). Furthermore, we also see the impact of error rate and length of the reads on the mapping results. lra, BLEND, Winnowmap2, and Magic-BLAST map less reads when they are shorter, but their results are similar to the other mappers with longer reads. We observe the same effect on error rates, for those tools, with a higher error rate leading to fewer reads mapped.

The results for the average percentage of unmapped reads with real data show similar outcomes for the vast majority of the tools (Figure 3.7B). However, lra shows a significant decrease in performance, with a median of 19.4% unmapped reads. Magic-BLAST and Winnowmap2 also see an increase in their number of unmapped reads. BLEND, on the other hand, shows improved results with only 0.007% unmapped reads compared to 0.097% with simulated data.

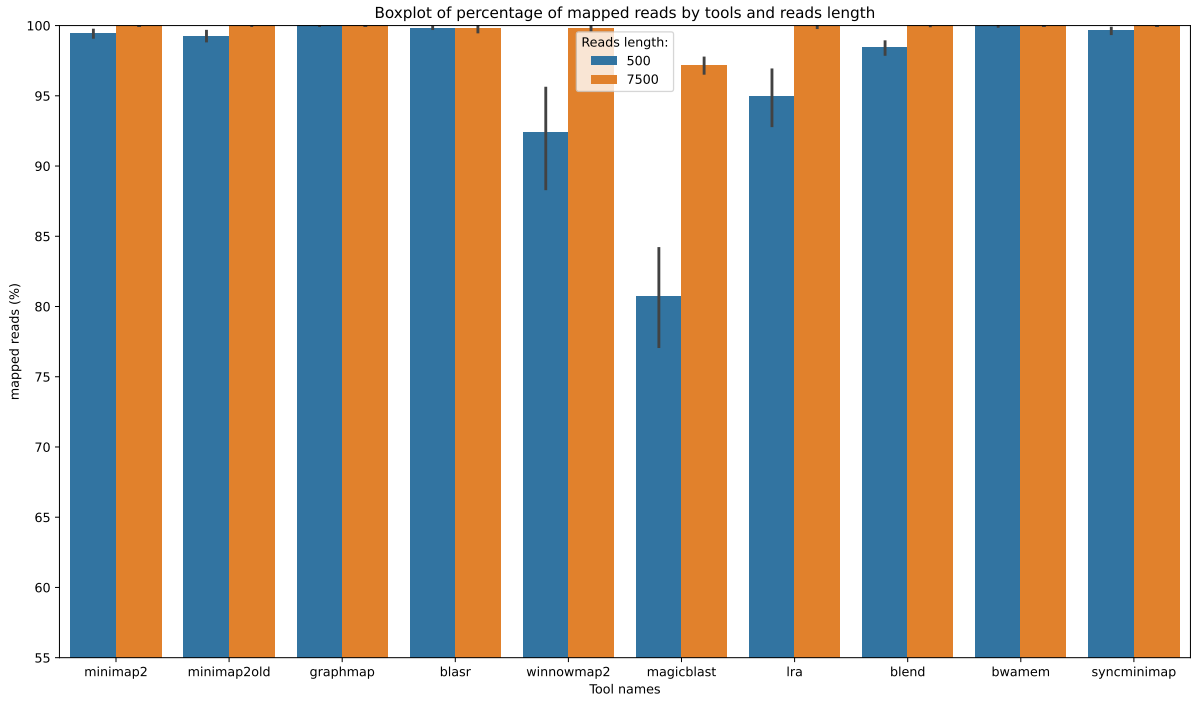

Figure 3.9 – *Barplot of the percentage of mapped reads by tools and reads length*

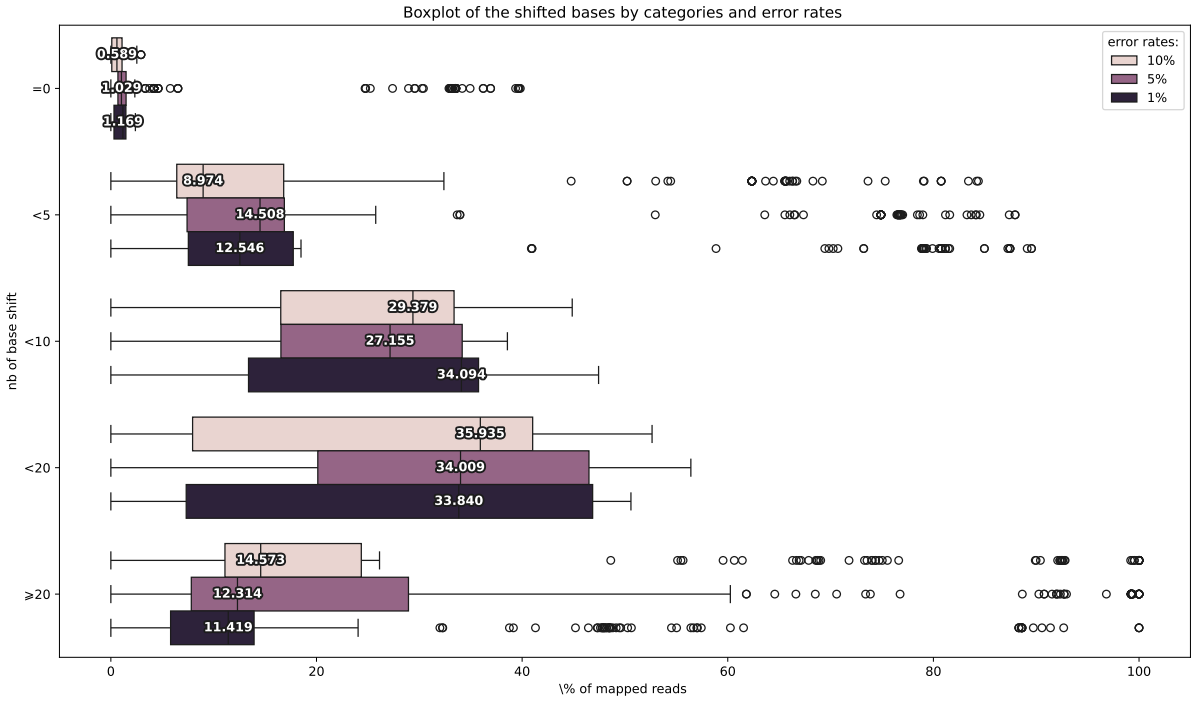

Figure 3.10 – *Boxplot of the percentage of shifted bases per categories and error rates*

Table 3.3 shows that on average less than 3% of reads have their sequence endings and

beginnings correctly positioned. We note that two tools, Magic-BLAST and lra, deviate significantly from this average, with 1.34 and 0.07, respectively. We also see that on average more than 10% of the mapped reads have their mapping coordinates shifted by more than 20 nucleotides compared to the theoretical expected mapping.

In figure 3.10 and 3.11 we cannot see any noticeable differences for the effect of the error rate on the read offset, regardless of the category, but the length seems to have an effect with a larger number of reads perfectly aligned with a small read size.

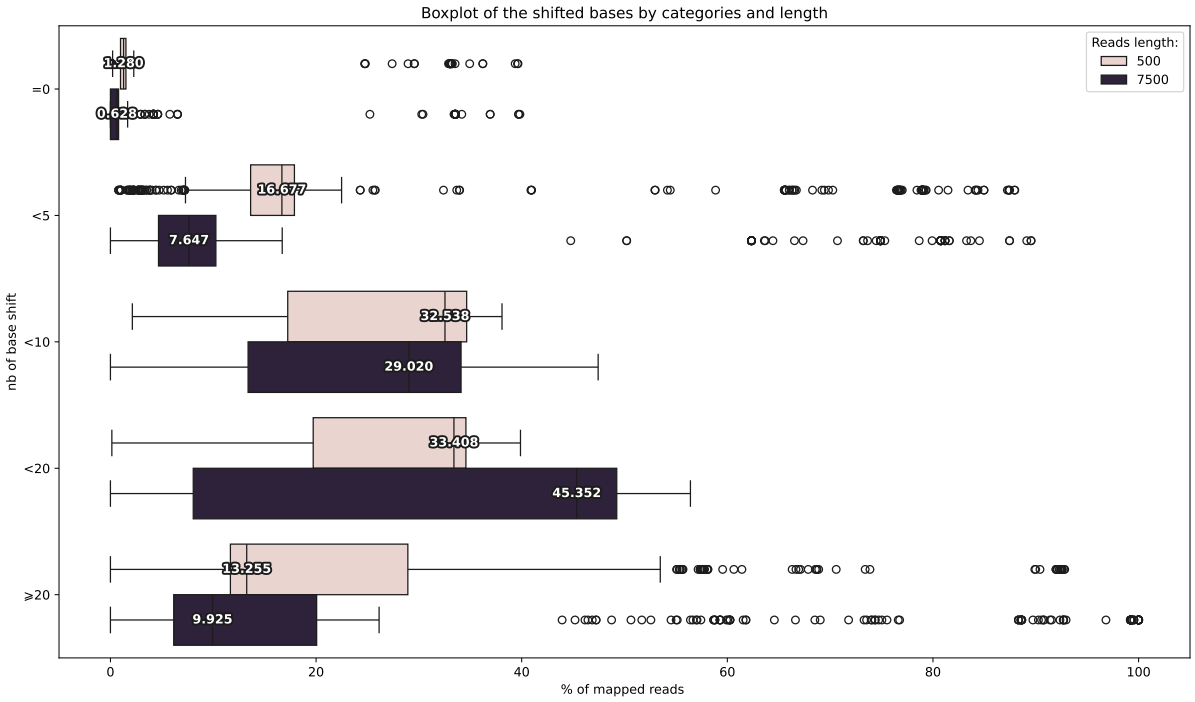

Figure 3.11 – *Boxplot of the percentage of shifted bases per categories and length*

Table 3.4 shows the soft clipping results for each read, indicates that on average, 50% of the reads are not affected by soft clipping. However, certain tools like lra and Magic-BLAST have only 0.93% and 27.53% of their reads, respectively, unaffected by this phenomenon. For GraphMap, 83.49% of the reads show no soft clipping.

Figure 3.12 shows the trends observed for soft clipping on real datasets, it is important to note that barcode has been present and has not been trimmed on the real data. We observe that, apart from GraphMap and bwa-mem2, the other tools use soft clipping for more than 90% of the reads, with significant soft clipping (greater than 20 bases). It is also noteworthy that bwa-mem2 has 6% of its reads without soft clipping, whereas for the other tools, it's less than 1%.

	Shift in mapping coordinates				
	0	1–5	6–10	11–20	>20
GraphMap	2.46	18.82	34.71	32.80	11.20
BLEND	2.07	17.35	33.15	34.44	12.98
BLASR	1.77	15.36	31.25	36.87	14.74
bwa-mem2	1.92	15.00	28.14	27.97	26.97
syncmer-minimap2	2.07	17.41	33.22	34.47	12.83
minimap2	2.06	17.31	33.08	34.46	13.09
minimap2 (old)	2.06	17.31	33.09	34.47	13.07
Magic-BLAST	1.34	6.50	13.01	13.74	65.40
Winnowmap2	2.16	17.35	33.31	34.41	12.77
lra	0.07	3.08	9.70	30.84	56.30

Table 3.3 – *Proportion for each tool of the mapped reads according to the shift in mapping coordinates, compared to the theoretical ones, over the 34 experiments. A shift of 0 corresponds to a read that starts and ends at the exact expected positions, a shift of x positions means that the start and end have in total a shift of x positions compared to the expected theoretical alignment. When measuring the ability of a mapper to correctly map a read, the proportion of reads should be as high as possible for a shift of 0 positions (in green), and the values for the other categories (especially for the last ones) should be as low as possible (the highest values are in red for each categories).*

	Nb bases soft-clipped				
	0	1–5	6–10	11–20	>20
GraphMap	83.49	15.62	0.51	0.15	0.02
BLEND	58.40	30.53	6.43	2.62	1.10
BLASR	46.93	28.28	12.79	7.71	2.02
bwa-mem2	64.27	22.69	2.62	1.82	8.18
syncmer-minimap2	58.55	30.67	6.36	2.54	0.97
minimap2	58.12	30.51	6.50	2.73	1.20
minimap2 (old)	58.18	30.51	6.49	2.71	1.18
Magic-BLAST	27.53	7.97	1.64	2.21	60.39
Winnowmap2	59.59	30.00	6.08	2.43	1.02
lra	0.93	10.31	12.69	29.19	44.77

Table 3.4 – *The percentage of average soft clipped bases per read for each tools. Each category corresponds to the number of nucleotides soft-clipped in the whole read at either end. The higher the percentage in the column for 0 nucleotide soft-clipped (in green), the better the ability of the mapper to correctly align the read. The values in the other columns should be as low as possible (the highest value is in red for each category).*

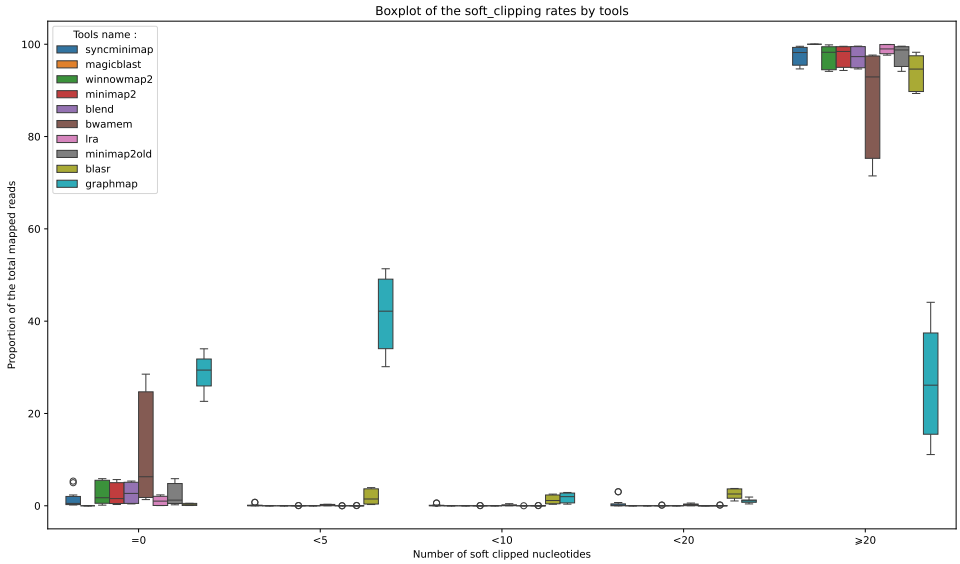

Figure 3.12 – *Boxplot of the percentage of soft clipping for each tool for the 34 samples generated. The x-axis show the proportion of soft clipping of the total mapped reads.*

3.4.2 Robustness of parametrization:

Varying parameters can have more or less impact on the mapper. With mappers such as GraphMap and bwa-mem2 BLASR or Magic-BLAST, it is difficult to see any impact of the settings on the mapping. For minimap2, syncmer-minimap2 and BLEND we can see a slight improvement (less than 2%) in mapper efficiency. Finally, for tools such as Winnowmap2, lra, we can see that with tuned parameters the number of unmapped reads goes from 10% to less than 1% with Winnowmap2 and goes from less than 10% to 1% or even less than one percent depending on the parameters used for lra

Number of indexed seeds. The number of mapped reads is positively influenced by smaller values for minimizer sizes and windows. Some tools like GraphMap index all k-mers as a regular part of their algorithm. Other, like minimap2, index a subsample of k-mers using a combination of windows and minimizers. By setting the window size and minimizer size to low values, one can be close to indexing all positions on the reference. The figures 3.7 and 3.13 show that for tools which natively have shorter and more numerous seeds, and for tools parametrized to harvest on almost all positions, mapping rates are better than for regular parametrization. This comes at the price of an increased running time, We can see particularly for minimap2, Winnowmap2, lra, and syncmer-minimap2 that the parameter settings, which affect the number of indexed seeds, have an extremely positive impact on the number of mapped reads. For Winnowmap2, for example, we see that the boxplot median of the average number of unmapped reads falls from just under 1 to 0.

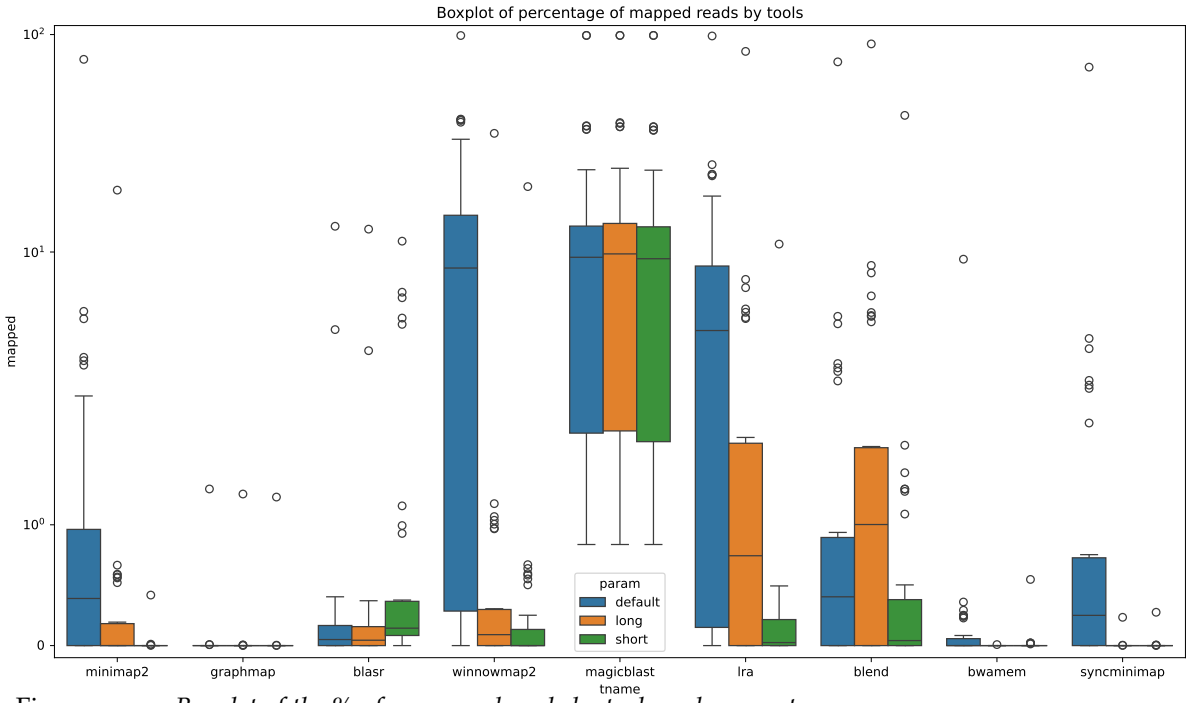

Figure 3.13 – *Boxplot of the % of unmapped reads by tools and parameters*

3.4.3 Impact on variant calling

Each dataset includes a variant in greater or lesser proportion. In order to see if the aligned reads could correctly find the variants, we look at the different F1-scores associated with 2 different variant callers, medaka and bcftools and the F1-scores are normalized with vcfdist.

It is observed in figure 3.15A that with medaka, despite the perfect BAM file not reaching a perfect F1-score (0.995), the vast majority of tools have an average F1-score of 1. Only GraphMap, BLASR, and Magic-BLAST have respective F1-scores of 0.914, 0.939, and 0.976. The second variant caller, bcftools, shows an F1-score around 0.5 for all tools.

Figure 3.15B, which shows the F1-score with real data, presents completely different results. In this configuration, bcftools gives better results: 0.909 for minimap2 in both versions, 0.857 for GraphMap, 0.869 for BLASR, 0.894 for Winnowmap2, 0.871 for Magic-BLAST, 0.959 for lra, 0.902 for BLEND, 0.869 for bwa-mem2, and 0.894 for syncmer-minimap2. Variant calling with medaka is around 0.65 for most tools, except for GraphMap with 0.425, Magic-BLAST with 0.554, and bwa-mem2 with 0.554.

In figures 3.14, which calculate the F1-score without vcfdist, different results appear with both simulated and real data. With the simulated data (figure 3.14A), the two variant callers have much closer F1-scores, with an average of 0.75 for bcftools and an average of 0.83 for medaka. In this configuration, Magic-BLAST with bcftools achieves the highest score, with an F1-score of 0.877.

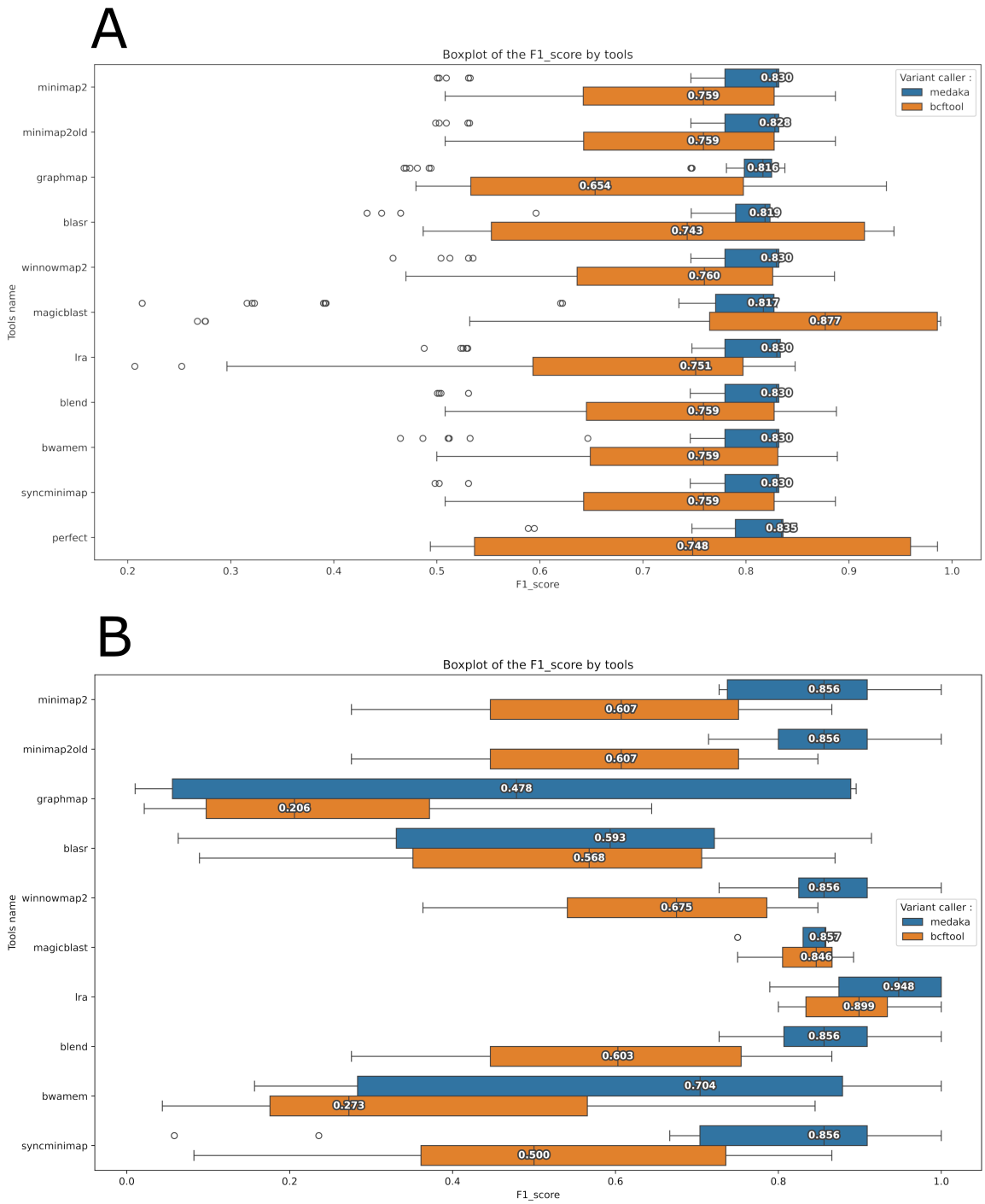

Figure 3.14 – ***A**:Boxplot of the F1-score by tools without using vcfdist across the 34 simulated datasets,**B**: Boxplot of the F1-score whithout using vcfdist for each tool across the 4 real datasets.*

For the real data (figure 3.14B), the F1-scores diverge much more between tools. With medaka, the higher F1-score is achieved by lra (0.948) and the lowest by GraphMap (0.478). For bcftools, most tools achieve an F1-score below 0.61, except for Winnowmap2 (0.675), Magic-BLAST (0.846), and lra (0.899).

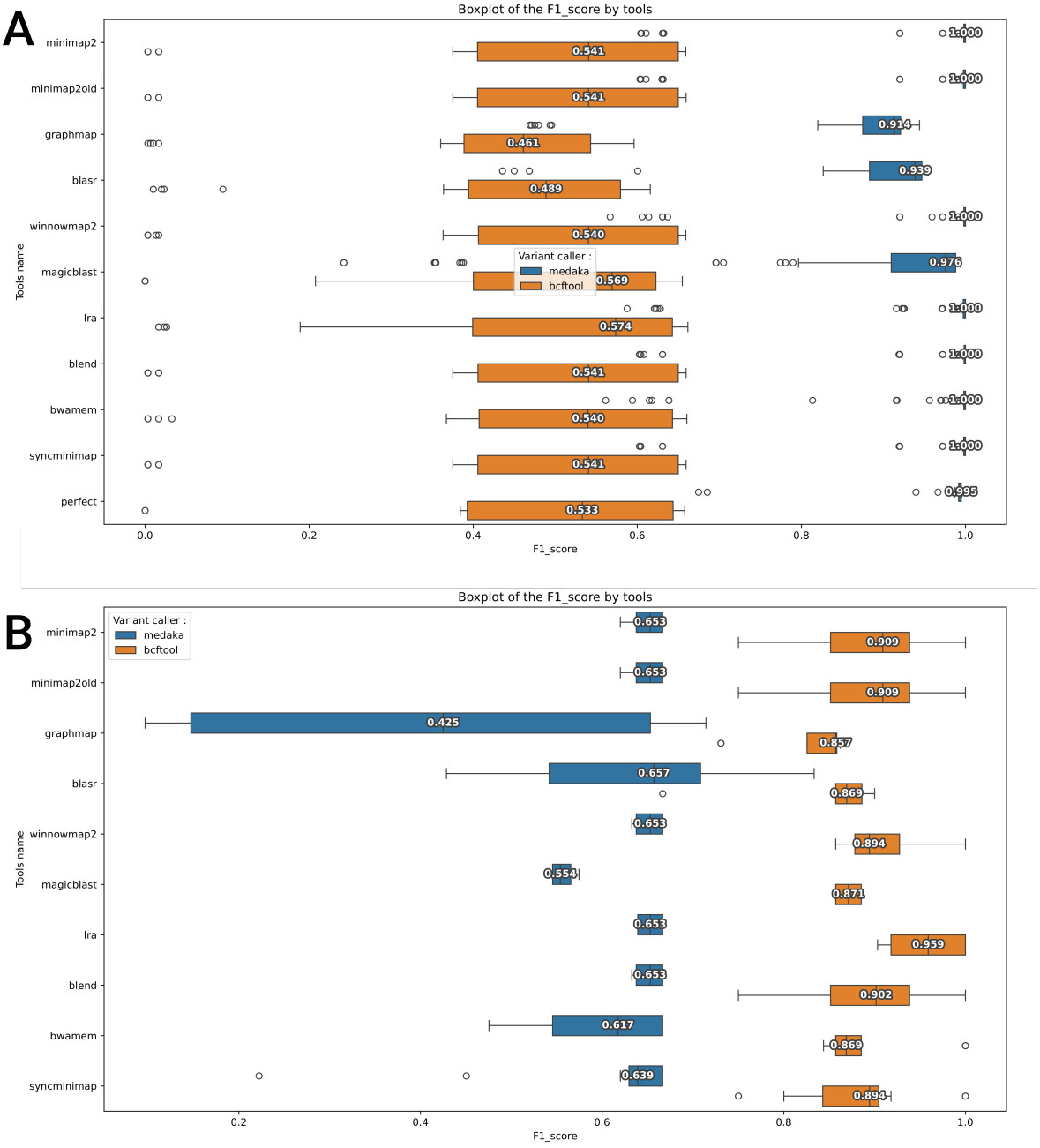

Figure 3.15 – *A: Boxplot of the F1-score using vcfdist for each tool across the 34 simulated datasets. In blue, the F1-score obtained with medaka, and in orange, the F1-score obtained with bcftools. : Boxplot of the F1-score using vcfdist for each tool across the 4 real datasets. In blue, the F1-score obtained with medaka, and in orange, the F1-score obtained with bcftools*

Figure 3.16 shows the average F1-score of all tools, indicating that there is no difference between experiments D1, D6, and D7. Therefore, it can be reasonably assumed that, for the simulated data, the choice of pore in the data generation process did not impact the results. Regarding the impact of coverage, as expected, mapping itself was not affected by it (experiments D1, D2, and D3).

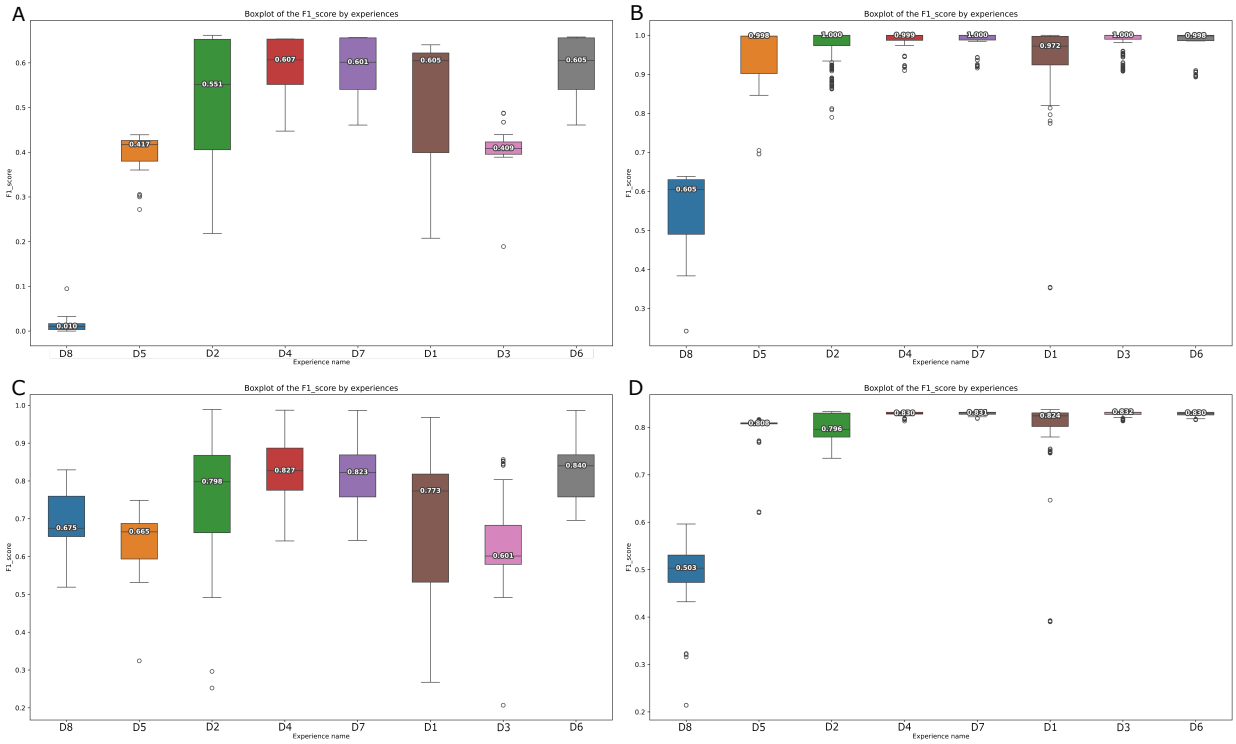

Figure 3.16 – *Boxplot of the F1-scores by experiences, A) bcftools F1-scores with vcfdist; B) medaka F1-scores with vcfdist; C) bcftools F1-scores without vcfdist; D) medaka F1-scores without vcfdist;*

However, we do observe an impact on the F1-score, which will be discussed in the variant calling section. Experiment D2 suggests that, in our case, species did not influence the performance of the mappers.

3.4.4 General results regarding the different datasets.

Although further results across more viral species would be needed to confirm this, we can still assume that viruses are less likely to show disparities in mapping results. Indeed, their small genome size leaves little room for short tandem repeat phenomena, which can be a challenge for mapping. The contamination (experiment D4) does not seem to affect the tools, which all proved to be extremely robust on the tested species, although tools like GraphMap occasionally mapped 1 or 2 human reads.

Regarding read length and error rate, both factors impact the number of mapped reads. Shorter reads make mapping more difficult, likely due to lower chaining scores. This hypothesis is supported by the fact that selecting more seeds allows for a similar number of mapped reads. The error rate, as expected, affects both the number of mapped reads and the F1-score. In experiment D5, which tested mapping on a more distant species, we also observed an impact on the number of mapped reads. However, this effect is similar to what can be seen with a high error rate. Finally, the number of species present does not impact the mapping performance of the tools.

3.5 Discussion and conclusion

The Discussion section is largely drawn from the corresponding part of Thomas Baudeau, Camille Marchet, Mikaël Salson. Assessing Long-Read Mappers for Viral Genomics. bioRxiv 2024.11.25.625163; doi: https://doi.org/10.1101/2024.11.25.625163.

3.5.1 Soft clipping.

According to the author of minimap2, clipping is the only way to report the chimeric or poor-quality reads that can be produced by long-read sequencers such as Nanopore and PacBio. In many tools, including minimap2, this implementation is not optional and it is not possible for the user to remove it. However, the implementation is not clearly indicated. minimap2's clipping strategy can be impacted by several heuristics such as the z drop or the end bonus (see section 2.4.3), but in most cases, it is during the extension part and during alignment that the clipping position will be determined by taking the positions that maximize the alignment score. This aspect of soft clipping, although it can be used to process chimeric reads or to give the mapper the possibility of managing data sets where the primers have not been removed, also represents a strong constraint on the ability to correctly retrieve information on the positioning of reads(10). Furthermore, no soft clipping doesn't seem to be the best strategy either. GraphMap, which greatly minimizes the use of soft clipping, forces the alignment of barcodes attached to the reads, in the real data.

3.5.2 Tool parameters.

With the benchmark, we try to understand how the choice of a specific seed size affects the results using 3 types of parameters with a variation for the length of the reads. The various results show that parametrization has an impact on the number of reads mapped by certain tools, especially those that samples seeds. In this way, we can see that parametrizations that increase the number of seeds have a positive impact on the number of mapped reads.

However, this comes at a price and will slow down calculation time. Figure 3.13 shows that for most tools, the default parameters are a solid choice with very good results. Despite this, for some specific samples, we can see that the use of a shorter seed improves the results. Increasing the number of seeds seems to positively affect the results independently of the types of data. The results show that decreasing the size of the seeds collected increases the results however it is important to note that decreasing the size of the windows will also increase the number of seeds collected so it is more likely that the most important aspect is the number of seeds.

3.5.3 Variant calling

Regarding variant calling, we observe a significant difference between the results obtained using vcf_dist, which applies corrections and filtering on the variants, and those obtained without using vcf_dist. For medaka, without vcf_dist, regardless of the origin of the BAM files, the F1-score consistently hovers around 0.8. Although the theoretically perfect BAM achieves the highest F1-score on average, the difference is only about 0.05 points compared

to the average of other tools. Moreover, figure 3.16 shows no notable differences between the various possible parameters. However, when vcf_dist is used, we see that, apart from GraphMap, BLASR, and Magic-BLAST, all tools achieve an F1-score of 1, while the theoretically perfect BAM only scores 0.995.

In the case of bcftools, there is greater variability between mappers, and on average, magic_blast achieves the best results despite a lower mapping percentage compared to other tools, particularly for reads with a length of 500 bases. With bcftools and vcf_dist, we observe less disparity, but also a significantly lower score, with a maximum of 0.574 obtained by lra.

To explain why we see such a difference in values with and without the use of vcfdist, we need to understand that this tool includes a filtering stage. In this filtering stage, reads judged to be of poor quality are removed. In the simulated dataset bcftools, making mainly false positives, removing the reads of bad quality allows it to increase its F1-score value. This effect is also observed on real data. Concerning medaka, it seems that vcfdist has a positive effect with the simulated dataset, but that the deletion of the data in the real dataset no longer allows it to find the variants since we see an increase in the false negatives.

It is interesting to note that the F1-score for variant identification is widely used to compare mappers. While this measure provides a good indication of a mapper's ability to process biological data, the results highlight some practical nuances. It appears that variant callers are highly specific to datasets with significant variability. Moreover, mappers with a lower average number of mapped reads like Magic-BLAST can sometimes achieve better variant calling results. Although we cannot fully explain these results at this stage, one hypothesis is that Magic-BLAST, by being very strict on the reads it maps, might only map reads with few errors, thus leading to better variant calling results compared to other mappers.

Practices in bioinformatics. Long-read mapping tools are often designed to address specific problems at particular points in time. The first tools, such as BLASR and GraphMap, used seed types better suited to short-read data. This was not an issue when they were first developed, as long reads were relatively short. However, with the advancement of long reads, their increased length, and improved quality, new methods were introduced, most notably minimap2, which uses minimizers for mapping.

One common aspect of all these tools is that they are often tested on eukaryotic and prokaryotic data. However, they all incorporate certain heuristics that, if correctly adapted to the dataset, can improve both results and implementation. Unfortunately, a general benchmark does not exist to comprehensively test and compare these tools. Although all tested tools are open source and typically provide a changelog with new releases, such a benchmark could help pinpoint improvements or degradations in their versions across different datasets.

Furthermore, as noted in this work, different tools employ various strategies for each step (seed, chain, extend), but their implementations are not modular enough to test independently and with different combinations the seeding, chaining and extension part.

Finally, some post-hoc analyses become extremely specialized in using one tool in particular, such as medaka. It is very difficult to use a mapper other than minimap2 in its workflow, which has two main consequences. First, according to the way the neural network behind medaka was trained, it is likely that medaka learns from minimap2 alignments and may experience a degradation in performance with other mappers. Second, if a user wishes to use another mapper for a valid reason, it would require significant engineering effort to replace it in the workflow.

3.5.4 Benchmark limit

In this section, we have attempted to compare the various existing tools for aligning long reads, by carrying out an analysis of the alignment on the one hand and, on the other, an analysis of the subsequent studies that can be carried out using the results of the alignment, in particular via the comparison of variants.

In this study, we have omitted a detail considered crucial by the tools: the computation time and memory required for their execution. The reason for this choice is that we wanted to focus on the impact of the various mapping stages (seeding, chaining and extension) on alignment quality. In addition, calculation time and memory space are significantly reduced with viral genomes, which rarely exceed $100,000$ bases. To achieve this, we varied certain parameters without taking into account the time and memory required by each tool.

Furthermore, the mappers presented are complex programs, and their design, independently of more sophisticated algorithmic solutions such as the choice of seed type, has a direct impact on their quality. For example, minimap2, one of the best-known tools, benefits from continuous improvements by the community, progressively optimizing its code and making it more efficient. This uneven optimization of tools is a second argument explaining why computation speed and memory footprint were not taken into account as metrics in this study.

Another point to consider is that our analysis concerns an assessment of tools at a given point in time. However, as in biology, bioinformatics tools are constantly evolving and adapting. As a result, this benchmark only reflects the performance of the tools in the specific versions used for this study.

Finally, our comparisons present results as averages. However, a detailed analysis by dataset would have revealed edge effects specific to certain datasets. Although our study includes 36 datasets, this represents only a tiny fraction of the existing biodiversity, with its many particularities. Some of these specificities may pose unique challenges for certain tools and algorithms.

3.5.5 Conclusion

In our experiments, we assessed whether and to which extent long-read mappers could accurately map long viral sequences. The different results obtained with the different data sets show that the tools manage to map the vast majority of reads and also manage to find the variants. However, we have seen that the particularity of viral data prevents the tools to

be at their full potential with the default settings. What's more, the great variability present in viruses can be a real challenge if the species sequenced are too far from the reference species. We have shown that recent or well-maintained tools, such as minimap2 or lra, are generally preferable for aligning viral reads. When precision is needed at the edges of the alignment - as in the search for subgenomic RNAs (10) (as discussed in the next chapter) or splicing events (92).

We have also demonstrated that an interaction exists between mappers and downstream tools. Thus, when a mapper appears to align a greater quantity of reads effectively, the details of the alignment can either favour or hinder variant calling tools, depending on how they have been trained.

Chapter 4

Customizing bioinformatics tools to study atypical biological mechanisms

The text found in the following sections is largely drawn from Thomas Baudeau and Kristoffer Sahlin, Improved sub-genomic RNA prediction with the ARTIC protocol, Nucleic Acids Research, Volume 52, Issue 17, 23 September 2024, Page e82, https://doi.org/10.1093/nar/gkae687.

In the previous chapter, we explored mappers and their capabilities. However, the specificities of viral data are not limited to the small size of their genomes compared to eukaryotic and prokaryotic species. Indeed, due to their cellular parasitism, viruses not only adapt to the biological mechanisms of their hosts but also possess mechanisms of their own. These mechanisms, due to their uniqueness, are generally not considered in the development of bioinformatics tools.

4.1 The challenge of SARS-CoV-2 and sgRNA

In this section, using the example of SARS-CoV-2 subgenomic RNAs, we will examine how certain specificities can pose challenges to mapping algorithms.

4.1.1 SARS-CoV-2

SARS-CoV-2 is a strain belonging to the species *Betacoronavirus pandemicum*, within the *Nidovirales* order and the *Coronaviridae* family. According to the Baltimore classification, it belongs to Group IV. The SARS-CoV-2 genome consists of a single-stranded, positive-sense RNA with a 5′ cap and a 3′ poly(A) tail. A positive-sense RNA means that the viral RNA has the same orientation as messenger RNAs. Once introduced into the host, it can be directly translated by ribosomes. The SARS-CoV-2 genome is approximately 30,000 nucleotides long. Eleven genes have been identified so far (Figure 4.1).

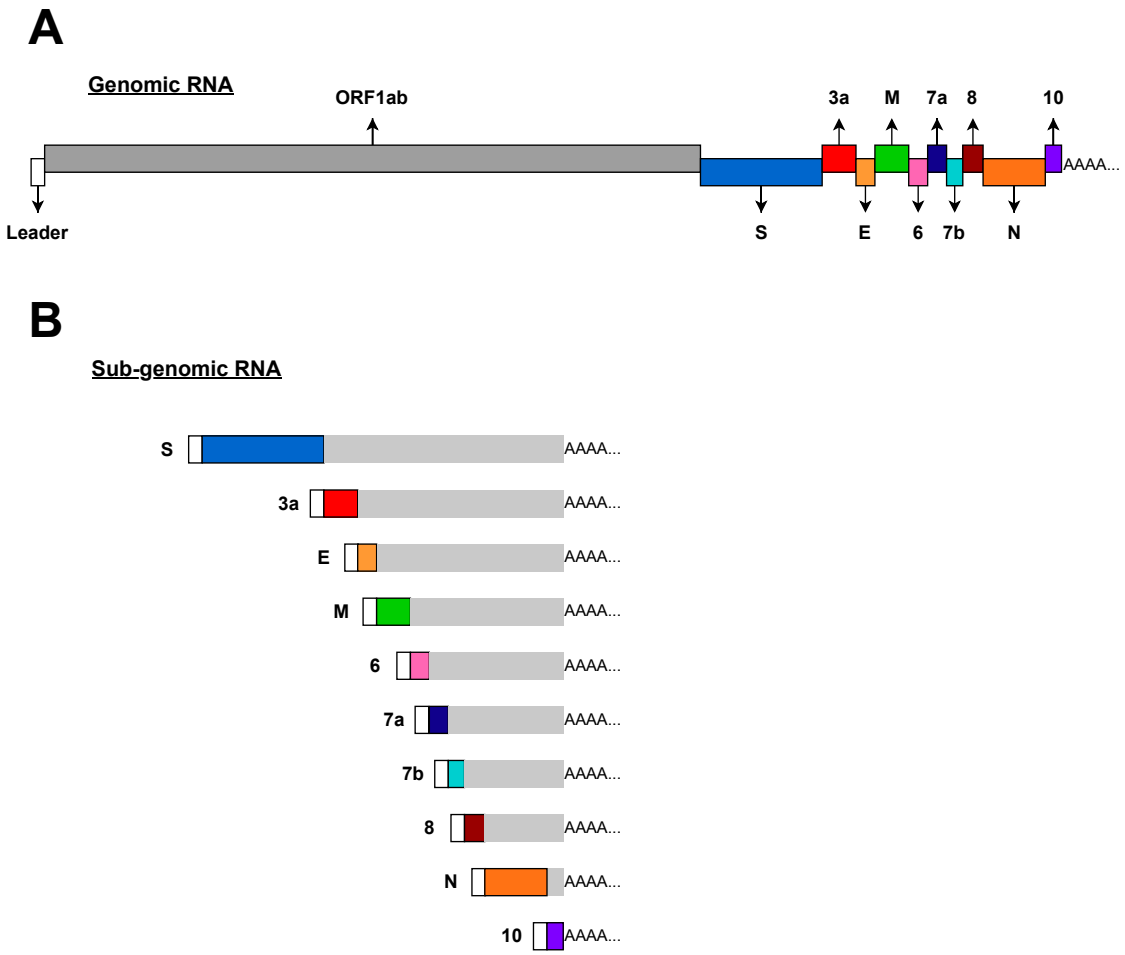

Figure 4.1 – *A: Representation of the SARS-CoV-2 genomic RNA*
B: Representation of the SARS-CoV-2 subgenomic RNAs

Among them, the S, E, M, and N genes encode the four structural proteins of the virus:

- The S gene encodes the Spike (S) protein,
- The E gene encodes the Envelope (E) protein,
- The M gene encodes the Membrane (M) protein,
- The N gene encodes the Nucleocapsid (N) protein.

4.1.1.1 Subgenomic RNA

SARS-CoV-2 has a mechanism unique to viruses of the order *Nidovirales*: the subgenomic RNAs (sgRNA). An sgRNA is a shortened portion of an mRNA. While the synthesis of sgRNA is already a distinctive feature of *Nidovirales*, viruses in the *Coronaviridae* and *Arteriviridae* (84) are the only ones to have a subsequence called the leader within their sgRNA. In SARS-CoV-2, the leader has a length of approximately 70 nucleotides.

In SARS-CoV-2, the genomic RNA only produces the protein from the 1a and 1b genes. The rest of the proteins, such as the previously mentioned structural proteins, are synthesized from subgenomic RNAs (sgRNA).

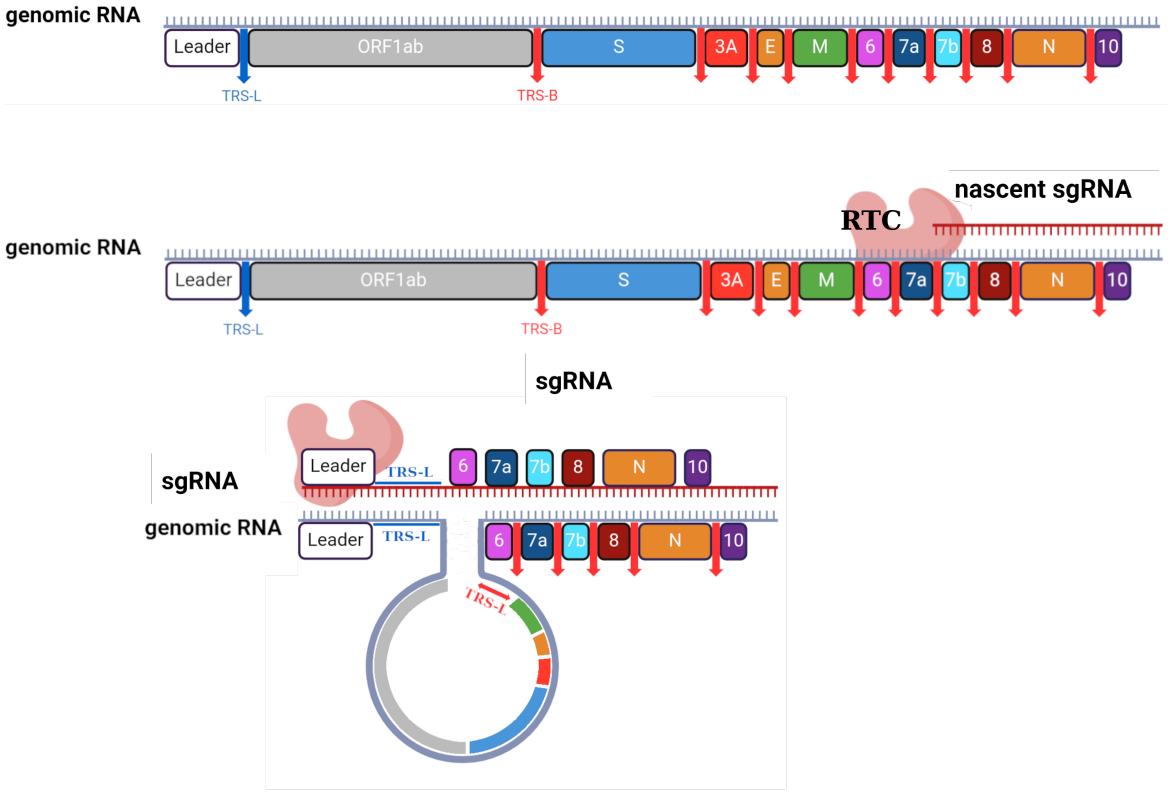

Figure 4.2 – *Schematic representation of the SARS-CoV-2' sgRNA production*

In Figure 4.2, a schematic representation of sgRNA synthesis is shown, as described in (116).

Within the viral genome, there are Transcription Regulatory Sequences (TRS). One of these, called TRS-L, is found just after the leader sequence. Additionally, except for the 1a gene, a TRS is located before each gene, known as TRS-B.

When the viral replicase-transcriptase begins replicating the genome, it adopts a hairpin structure at the TRS-B site. This causes replication to halt and the replicase to detach. Due to the complementarity between the TRS-B and TRS-L sequences, the transcriptase can resume transcription at the TRS-L site.

Thus, each sgRNA formed through this mechanism contains the leader sequence. The sgRNAs generated in this way are called canonical sgRNAs. There are also non-canonical sgRNAs, known as TRS-independent or TRS-B-independent, meaning that no TRS is involved, or only the TRS-L participates.

4.1.2 Generating dataset with sgRNA

Since sgRNA production is specific to certain viruses, and establishing ground truth for genomic data is already a complex task, even without additional biological mechanisms. To obtain a gold standard, we design a pipeline to integrate sgRNA into genomic data.

To determine the best way of generating simulated data that most closely reflects the real data, it is essential to look at the methods used to obtain viral data in practice. Various strategies are employed to sequence viral genomes, each with its own advantages and limitations. One approach involves culturing the virus in a controlled laboratory environment, which helps enrich viral material but may introduce artificial mutations and does not work for viruses that are difficult to culture. Another method relies on metagenomic sequencing, which enables the detection of viral genomes without prior knowledge but often results in low viral read depth, making genome reconstruction challenging. A more targeted alternative is PCR amplification, which selectively amplifies viral genetic material, offering a more focused and reliable analysis compared to broader sequencing approaches (11). A well-established RT-PCR-based method is the ARTIC protocol (99, 121), which has been widely applied for tracking viral outbreaks (99, 61, 49, 37, 5).

For sequencing ARTIC amplicons, both Next-Generation Sequencing (NGS), such as Illumina, and Third-Generation Sequencing (TGS), including Oxford Nanopore Technologies (ONT), are commonly employed. While Illumina remains the preferred choice for variant identification due to its high accuracy and sequencing depth, ONT is continuously improving in terms of read quality and throughput. Furthermore, ONT's portability and ability to generate long reads enable full-length sequencing of amplicons, making it particularly well-suited for viral surveillance. Its faster turnaround time also facilitates the sequencing of a significantly larger number of samples, up to five times more than Illumina, providing a major advantage for real-time epidemic monitoring (120).

4.1.2.1 Sequencing with the ARTIC protocol

The ARTIC protocol is a multiplex PCR protocol developed by the ARTIC consortium. A PCR, or Polymerase Chain Reaction, is a laboratory technique used to create copies of a DNA fragment. To summarize, a PCR operates in three repeating steps. In the first step, the DNA is denatured so that double-stranded DNA becomes single-stranded. Then, it binds to two short complementary DNA sequences called primers. Finally, a polymerase extends the primers to create a new DNA strand called amplicon. A multiplex PCR follows the same principle, but the primers have specific characteristics. With the ARTIC PCR, there are two pools of non-overlapping primers. The advantages of this method compared to a classic PCR are that it ensures more homogeneous coverage, as each region is covered by a primer. Additionally, this method guarantees that the amplicons have a fixed size and prevents the

formation of amplicons resulting from the hybridization of different primer pairs. In the case of RNA viruses, such as SARS-CoV-2, we refer to RT-PCR, which stands for Reverse Transcription Polymerase Chain Reaction. A reverse transcriptase is a specialized DNA polymerase that has the ability to synthesize a complementary DNA (cDNA) strand from an RNA template.

4.1.2.2 Simulating the sequencing of sgRNAs with the ARTIC protocol

In order to generate ARTIC like datasets, we developed a pipeline named `sgENERATE`. `sgENERATE`, developed in `Python` using Snakemake (85), first generates datasets resembling ARTIC sequencing data by extracting amplicon sequences from a reference genome, using ARTIC-defined amplicon positions. Long-read sequencing data is then simulated with pbsim2 (95), with parameters configured to match amplicon sizes (`--length-mean` 2000, `--length-min` 2000, `--length-max` 2000) and Nanopore sequencing error rates and proportion (`--difference-ratio` 23:31:46) corresponding to the proportions of substitutions, deletions, and insertions (*i.e* among the errors introduced, 23% are substitutions, it is the default setting). The randomness for `--hmm_model` analysis is controlled using `--seed 1234`, while the `R103` pbsim2 model replicates Nanopore sequencing characteristics. Users can adjust `--accuracy-mean` to modify error rates and `-depth` to control sample size. To reconstruct subgenomic RNAs (sgRNAs), we use a reference table listing sgRNA junction sites from a prior study (18). This table, provided in `.TSV` format, can be modified by users to accommodate different sgRNA structures.

To simulate chimeric amplicons (see figure 4.4), we determine primer pairs that could generate non-canonical products from the amplicon and sgRNA position, mimicking how an amplicon might arise from an sgRNA. In the sgRNA leader, the primers are conserved, while the primers within the deleted sequence are absent, which can lead to the formation of new amplicons called chimeric amplicon, as illustrated in Figure 3.15A. We assume that a chimeric amplicon results from a primer pair and consider amplicon length and polymerase activity in its formation. These chimeric amplicons are generated with pbsim2 using the same parameters as before, but with a maximum length set to 1.5 times the mean amplicon size. A final script merges all amplicons to create the dataset, ensuring that chimeric amplicons account for no more than 1% of total reads to reflect realistic sequencing conditions. The output is a FASTQ file mimicking ARTIC protocol sequencing with primer scheme v3 (`https://artic.network/resources/ncov/ncov-amplicon-v3.pdf`).

Our pipeline incorporates certain simplifications compared to the ARTIC sequencing workflow. Specifically, RT-PCR amplification errors are not simulated, and reads lack sequence tags, which are normally used to distinguish primers in different amplicon pools. However, such tags can be removed with tools like TagCleaner (109). Despite these simplifications, our approach accurately models key sequencing steps and highlights important biases. The generated dataset represents an idealized case, free from noisy amplicons and unexpected biological variants such as non-canonical sgRNAs. Following dataset generation,

the pipeline runs benchmarked tools (in this study, `periscope` and `periscope_multi`) using the simulated data.

The benchmarking process produces six key output files: one FASTQ file containing raw reads, two BAM files (one with raw alignments and one with classified reads (*i.e.* a tag is added to each read and tells if it's a sgRNA or a gRNA and from which amplicon it comes), and three CSV files (sgRNA counts, sgRNA counts normalized using Periscope's method, and an amplicon-sgRNA mapping file (a file showing the count of the sgRNA from an amplicon perspective and not a reads perspective)). `sgENERATE` then analyses these files, providing graphical representations of sgRNA proportions and total read counts. It also identifies shared and unique sgRNAs between different tools, using Matplotlib (53) and `matplotlib-venn` for visualization. Designed to be modular, `sgENERATE` can integrate additional tools that follow the same input-output structure: taking a FASTQ file as input and producing both a CSV file (formatted like `periscope` and `periscope_multi` outputs) and a BAM alignment file with tagged annotations. This modularity allows our pipeline to serve as a benchmarking framework for future bioinformatics tools designed for this sequencing protocol.

While previous studies (69, 86) have evaluated tools such as Periscope (98), LeTRS (29), sgDI-tector (27), and CORONATATOR (80), none have used simulated datasets that provide a ground truth for assessing tool accuracy in detail. We will discuss these tools in more detail in the section 4.2.

4.1.2.3 The selected datasets

To test how the tools perform, several datasets were selected:

- **SIM datasets**: A simulated dataset generated with `sgENERATE`, consisted of 419,095 reads of median length 400 and average error rate 7%, where the fraction of sgRNA read was 1/100
- **SIM multi-error** :A dataset with several samples. The reads in each sample have various error rates of 4%, 7%, 10%, 15%, and 20%. In all five datasets, a fixed quantity of 100 reads were simulated from each sgRNA. This corresponds to a rough abundance rate of 0.02% for each sgRNA type compared to the total number of reads generated by pbsim2 for the sample (between 415,567 and 500,135 reads for each dataset due to coverage variability in pbsim2)

To verify that the results obtained on simulated datasets are consistent with those on real datasets, three additional datasets were sourced from real sequencing data available on the European Nucleotide Archive (ENA; https://www.ebi.ac.uk/ena/browser/search). This data is publicly available as part of the periscope study (98) and two samples were generated using Oxford Nanopore Technology, more specifically a gridION system, when the third sample used Illumina sequencing. The ARTIC protocol was used to amplify the viral samples, using the primer scheme V3.

- **BIO-SMALL datasets**: Extracted from sample ID ERR5159329 which consists of 681,228 ONT reads sequenced with the ARTIC protocol.
- **BIO-LARGE datasets**:Extracted from project number PRJEB40972, it consists of 1,145 samples of ONT reads with a total of 305,338,021 reads.
- **BIO-SMALL-illumina datasets**: Extracted from sample ID ERR5165937, it consists of 681,228 reads and sequenced with paired end Illumina protocol.

Read visualization was performed using IGV (102). All figures are created using `Python 3.9.13` and `Matplotlib (53) 3.5`.

4.2 Available tools to detect sgRNA for long reads datasets

4.2.1 Periscope

`Periscope` (98) is a Python pipeline developed to find sub-genomic RNA in both ONT and Illumina reads using the ARTIC (99) protocol. The ARTIC protocol is an amplicon-based sequencing protocol designed for viruses. However, we focus on the `periscope` pipeline for analyzing ONT reads, due to its lightweight portable sequencing and, thus, potential for surveillance and bio-monitoring capabilities. For the ONT reads, `periscope` uses minimap2 (74) with default parameters (`-ax map-ont -k 15 -w 10`). Minimap2 maps the reads from a sample to a single reference representation of the viral sequence. After this, the resulting `.bam` file is processed in several steps. The first step retrieves the starting position of each read and, based on the starting position, assigns the reads to the corresponding amplicon using information on the amplicon positions, amplicon lengths, and primer positions provided by the ARTIC protocol (`https://artic.network/ncov-2019/ncov2019-bioinformatics-sop.html`). After the read has been assigned to an amplicon and the read start position on the amplicon is determined, the tool deduces if a read begins in an ORF using the known ORF positions given by a file with each amplicon's starting and ending position. If the read is mapped with the start position inside the ORF, it is assigned to the ORF.

The second step is to find the leader sequence in the mapped reads. To perform this task, `periscope` uses a local alignment approach using dynamic programming with biopython (20) between the whole read and the leader sequence given by the user. The scoring parameters for the local alignment are $+2$ for a match, -2 for a mismatch, -10 for the gap opening penalty, and -0.1 penalty to extend the gap. If the read is assigned at an ORF, it is classified to the corresponding sgRNA with a label based on the leader's alignment score. If the score exceeds 50, it's categorized as HQ (High Quality). If the score falls within the range of 50 to a specified threshold, it's classified as LQ (Low Quality). If the score is below the specified threshold, it receives the label LLQ (Low Low Quality). The reads not assigned to an ORF are labeled as non-canonical sgRNA if their leader score is HQ or LQ. All the amplicons that are not labeled as non-canonical sgRNA or sgRNA are labeled as genomic RNA (gRNA). Once the two-step-based read classification is complete, statistics such as the proportion of each sgRNA are computed.

4.2.1.1 Other available tools

When designing this study, we mainly focused our comparison on `periscope`, the tool that we considered to be the most advanced for the detection of sgRNAs in the context of SARS-CoV-2. However, in order to place our approach in a broader context, we also compared our results with those of other tools described in the literature, in particular two of them which we present below.

LeTRS :

Like `periscope`, LeTRS(29) follows a two-step approach but differs in its choice of mapping tools. For short reads, it uses Hisat2 v2.1.0 (64), while for long reads, whether amplicon-based or not, it relies on minimap2 in a mode optimized for spliced read mapping. The idea behind this tool is to compare subgenomic RNA with conventional mRNA, and thus rely on the detection of splicing events to identify subgenomic RNAs. However, it is important to note that tools such as minimap2 were originally designed to detect specific motifs associated with splicing in eukaryotes. Consequently, the use of adaptive modes for splicing can introduce significant biases into the results, due to the mismatch between the signals expected by these tools and the particularities of sgRNAs.
In the second step, the tool first identifies and collects reads with a splice junction. It then applies several successive filters to determine whether these reads correspond to sgRNA. If the Leader-TRS junction is known, the read is assigned to one of the canonical sgRNAs. Otherwise, it is classified as non-canonical.

sgDI-tector :

sgDi(27) also follows a two-step approach; however, instead of using a standard mapper, it relies on a specialized tool called DI-tector. DI-tector is designed to identify Defective Viral Genomes (DVGs) within viral sequencing data, specifically detecting deletion-type DVGs.
By leveraging DI-tector, sgDI-tector retains all reads exhibiting a deletion event. It then analyzes the region surrounding the deletion site, associates the read with known Open Reading Frames (ORFs), and links it to the corresponding sgRNA. If the ORF is unknown, the read is classified as non-canonical sgRNA. Finally, using the detected canonical sgRNAs, the leader sequence is inferred and provided to the user.

4.2.2 Persicope_multi

We here present `periscope_multi`, a modified version of `periscope`. `periscope_multi` is implemented to preserve the same user interface, identical functionality, and output format to `periscope`. A schematic representation of the strategies used by `periscope` and `periscope_multi` can be found in figure 4.3. At a high level, `periscope_multi` uses multiple reference sequences representing the known variation of the sgRNAs of the genome under study. This multi-reference better represents the genomic variation that occurs and increases the length of the possible alignment between the read and references,

which allows for more accurate and time-efficient analysis. Specifically, we made three design changes to the `periscope` pipeline.

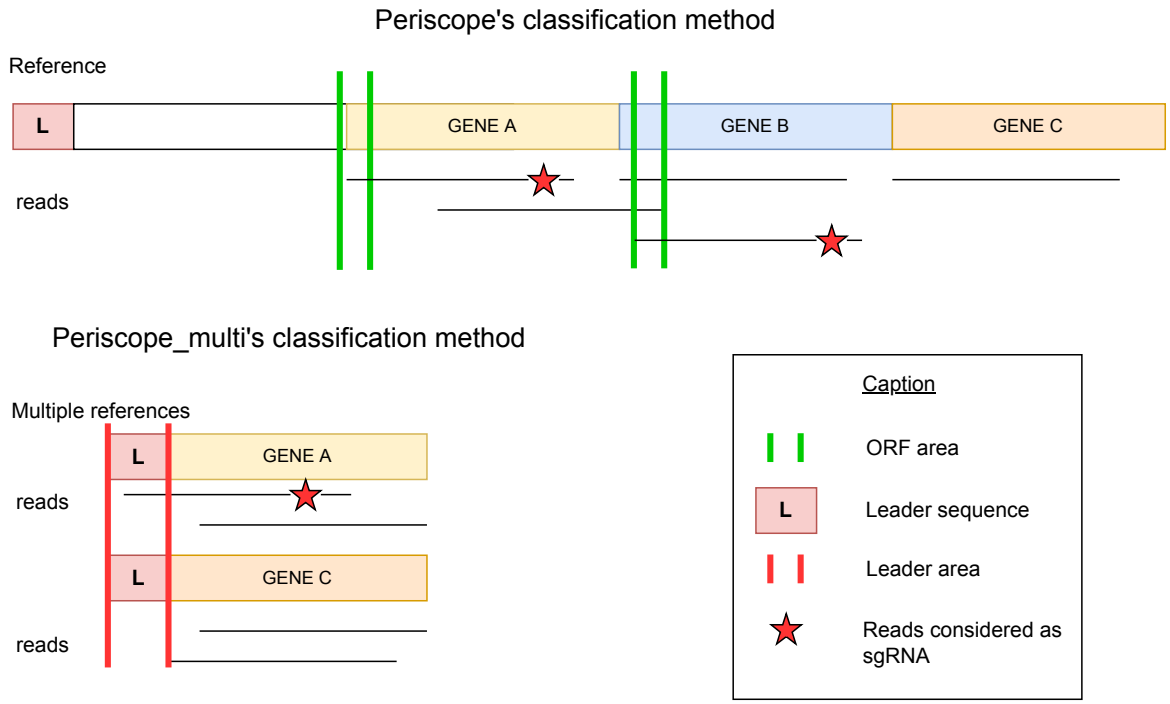

Figure 4.3 – ***Schematic representation of the sgRNA classification methods between `periscope` and `periscope_multi`.*** *The green area represents the beginning of an ORF, it's the part taken into consideration by `periscope`. In `periscope` a read is considered as sgRNA when it starts in this area. The leader part is normally soft clipped by minimap2. The read considered as sgRNA is labelled with a red star. In `periscope_multi` a read is considered as sgRNA when a read is aligned in the leader area.*

First, `periscope_multi` creates a reference file with multiple sequences corresponding to the beginning of all the known sgRNA concatenated with the leader sequence, using a gff file from the National Center for Biotechnology Information available at : https://www.ncbi.nlm.nih.gov/datasets/taxonomy/2901879/. We don't use genomic RNA as a reference to minimize the redundant parts of our mapping. Each sequence consists of the leader part, a given gene, and the start of the following downstream gene (see the sgRNA ORF2 in Fig 4.4A). Finally, the leader sequence is added to the front of each sequence. Thus, the resulting length of our reference sequences will be the length of the gene and the leader (*e.g.*, ORF2 in Fig.4.4B) plus the length of an amplicon to ensure that amplicons overlapping two genes can also be aligned. We also modified the S gene's position to fit the position from previous work (18).

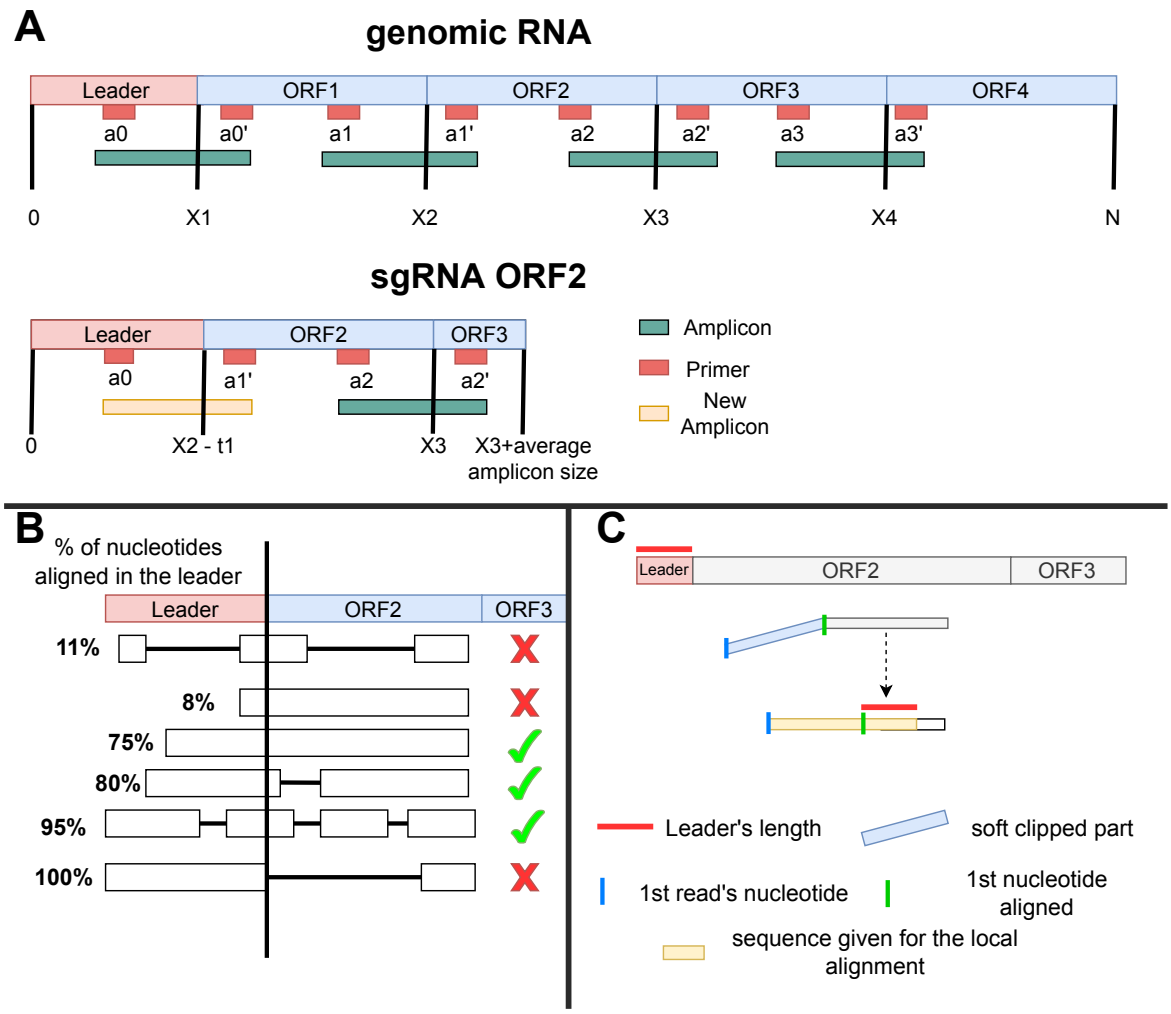

Figure 4.4 – *Amplicon selection schema*
Panel A shows the construction of each sgRNA for the new reference in `periscope_multi`*.The red segments represent the position of each pair of primers and the green bar are the resulting amplicons. The yellow segment shows the amplicon resulting from the new primer pair of* a_0 *to* a'_1 *due to the new location of each primer in the sgRNA. Panel B shows the local alignment in* `periscope_multi` *for six example reads. Panel C shows examples of the alignment needed for* `periscope_multi` *to consider a read as sgRNA. Only the soft-clipped part of the read plus the leader length is used. The corresponding yellow segment to the left shows the sequence given for each of those reads.*

Secondly, `periscope_multi`, similar to `periscope`, uses minimap2 to perform read mapping to our multi-reference file. `Periscope_multi` then extracts the reference position of each mapped read. If the read aligns with more than a threshold X ($X = 0.3$) of the nucleotides in the leader sequence(approximately 8 nucleotides), `periscope_multi` labels the read as sgRNA (Fig. 4.4B). Thus, unlike `periscope`, all the reads mapped to the beginning of the reference are labelled as HQ, avoiding the usage of the HQ, LQ and LLQ categories. In addition, less reads needs to be locally aligned to identify the leader sequence, as the leader is already aligned with the reference, speeding up the computational processing.

Thirdly, in order to further reduce the computational cost of `periscope_multi`, only the soft-clipped part of the read; that have not been aligned with the leader; plus the leader

length is used from the read to locally align against the leader sequence to detect non canonical sgRNA. This simplification is based on the following reasoning. If the mapping is correct, the leader part has to be at the beginning of the alignment. Otherwise, that means the alignment is spurious, which means we discard it for further analysis. The local alignment score is then used to determine if the read is a non-canonical sgRNA, similarly to `periscope`. If the score is higher than a given threshold similar to `periscope`, the read is labelled as non-canonical sgRNA. Finally, `periscope_multi` converts back the positions of the read with respect to the original genome provided both to `periscope` and `periscope_multi`.

4.2.3 Finding truncated proteins in BIO-LARGE

To construct a database of non-canonical sgRNAs and the effect on the coding protein, we performed the following analysis. We first re-aligned all reads classified as non-canonical sgRNA in four different modes; (1) minimap2 in splice mode aligned to the SARS-CoV-2 genome, (2) minimap2 in the same configuration than `periscope_multi`, (3) minimap2 default parameters but without the leader part and with only the first 200 nucleotides of the read aligned to the multiple reference, and (4) minimap2 default parameters but without the leader part and the soft-clipped at 5' end aligned to the multiple reference. We used these four settings because the mode (1) with splicing can detect chimeric read or spurious alignment. (2) is our reference alignment, (3) is to ensure that the starting position of the alignment is correct. Indeed, if we consider a sequence A containing two sub-sequences Ab and Ac from two distinct regions of the genome, it is highly likely that the mapper will choose the position of the longer sub-sequence as the starting position for the alignment. This analysis therefore allows us to look at the starting position of the 1st sub-sequence. (4) allows us to discard possible negative interaction with primer of the read. We then designed a python script to compare for each read the four alignments produced under the different settings.To identify consensus alignment positions, we use a Python script to collect the start position of each alignment on the reference genome. A position is considered a consensus if it is supported by at least three alignments. We then extract the corresponding reference sequence at these consensus positions. To identify Open Reading Frames (ORFs), we run ORFfinder(127) on the reference sequence; this avoids artifacts due to sequencing errors that could disrupt ORF detection. For each ORF identified, we compare it to the canonical ORF associated with the corresponding sgRNA. Finally, we generate a CSV file listing the reads and indicating, for each gene, whether the encoded protein is complete or truncated based on the ORF position. The genes used as references are extracted from NCBI under the gene IDs: 43740578; 43740568; 43740569; 43740570; 43740571; 43740572; 43730574; 43740575; 43740576; 43740577.

4.3 Results

The Results section is entirely taken from the corresponding part of Thomas Baudeau, Kristoffer Sahlin, Improved sub-genomic RNA prediction with the ARTIC protocol, Nucleic Acids Research, Volume 52, Issue 17, 23 September 2024, Page e82, https://doi.org/10.1093/nar/gkae687. Only a few sentences have been modified to incorporate figures that were previously included in the supplementary materials.

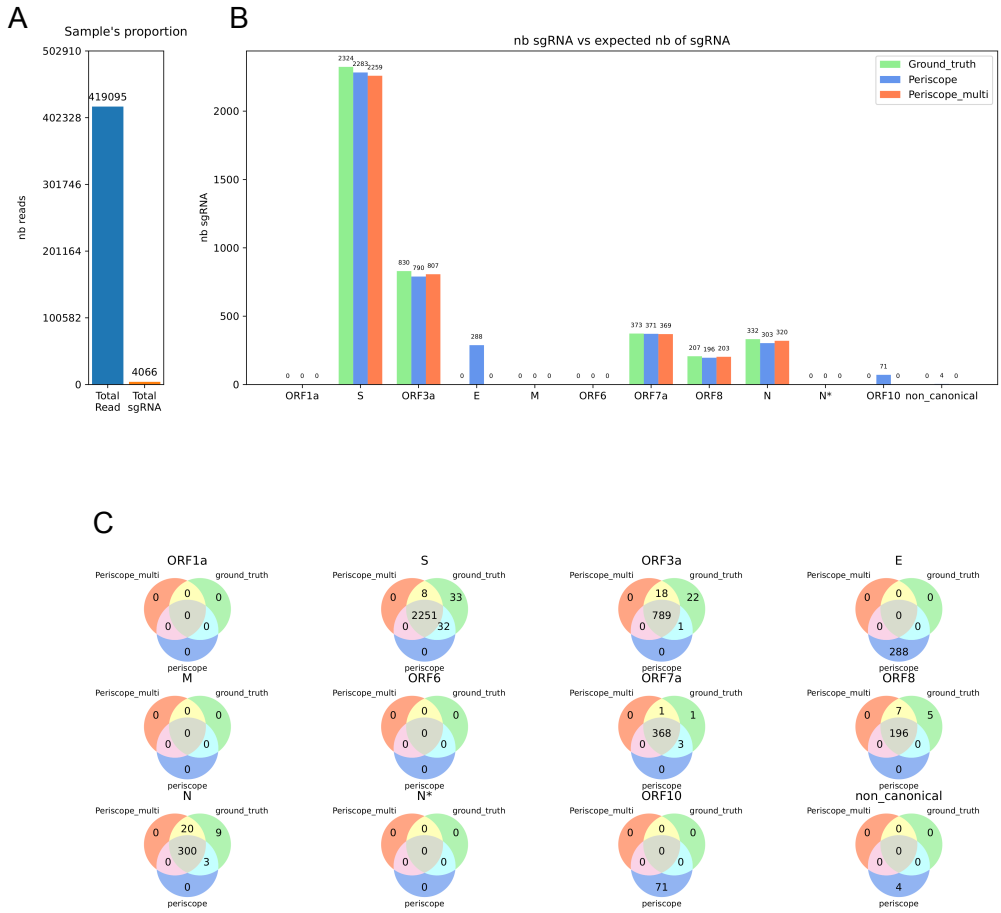

Figure 4.5 – ***Result of `periscope` and `periscope_multi` on the SIM dataset from `sgENERATE` with an error rate of 7%.*** *A) Total number of reads and total number of sgRNAs introduced in the sample. B) The number of sgRNAs found for each gene by the tools, the blue bar corresponds to `periscope`, the orange to `periscope_multi` and the green one to the real number of sgRNA in the sample. C) Venn diagrams showing the proportion of shared reads between `periscope`, `periscope_multi` and the ground truth.*

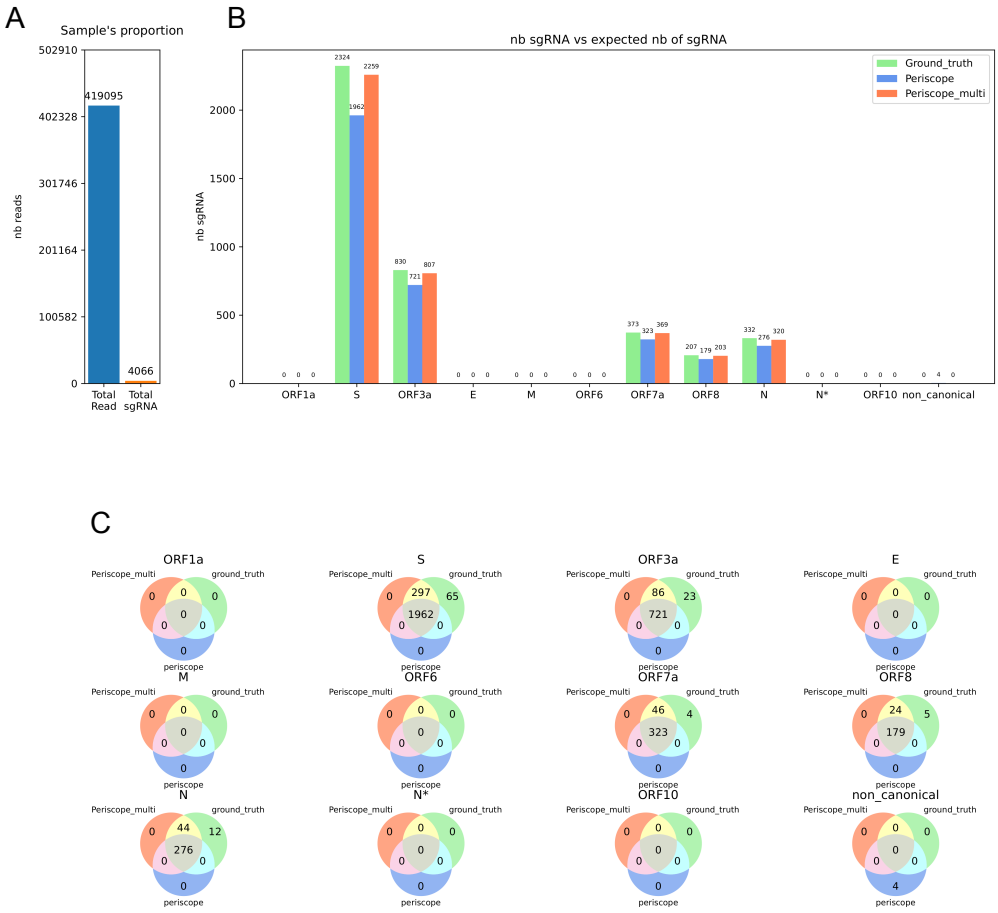

Figure 4.6 – ***Result of `periscope` and `periscope_multi` on a simulated dataset from `sgENERATE` with an error rate of 7% without the LLQ-labeled sgRNA for `periscope`.*** *A) Total number of reads and total number of sgRNAs introduced in the sample. B) Number of sgRNAs found for each gene by the tools, the blue bar correspond to `periscope`, the orange to `periscope_multi` and the green one to the real number of sgRNA in the sample. C) Venn diagrams showing the proportion of shared reads between `periscope`, `periscope_multi` and the ground truth.*

4.3.1 Periscope-multi accurately predicts sgRNAs

We first analyzed periscope and periscope_multi, using all prediction classifications classes (HQ, LQ, LLQ). The SIM datasets show that `periscope` falsely detects sgRNA classified reads from E and ORF10 which are not in the sample (Fig. 4.5B-C). Unlike `periscope`, `periscope_multi` does not produce any false classifications. However, in this configuration, `periscope` also finds slightly more TP than `periscope_multi` (Fig. 4.5C).

We also investigated whether excluding the LLQ-tagged sgRNA reads improves `periscope`'s FP predictions in Figure 4.6. With this configuration, `periscope` no longer generates FP, but there is a sharp increase in the number of FN in all genes. `Periscope` underestimates the sgRNA from other genes. For example, it misses 364 sgRNA for S gene (15.65% of ground truth sgRNA from the S gene), 111 for ORF3a (13.3%), 52 for ORF7a (13.8%), 30 for ORF8 (14.35%) and 58 for N gene (17.3). In comparison, `periscope_multi` does not generate false positives and still allows us to find the majority of the sample's sgRNAs with missing only 67 sgRNA for S gene (2.88% of ground truth sgRNA from the S

gene), 25 for ORF3a (3%), 6 for ORF7a (1.6%), 6 for ORF8 (2.8%) and 14 for gene N (4.19). Figure 4.6 shows that in this configuration, `periscope_multi` outperforms `periscope` with a large number of correctly detected sgRNAs (Fig. 4.6C).

In summary, `periscope_multi` has a precision and recall of 1.00 and 0.97, respectively, on SIM, while `periscope` has precision and recall of 0.92 and 0.97, respectively (with LLQ-labelled sgRNA) and 1.00 and 0.85 (without LLQ-labelled sgRNA). Thus, `periscope_multi` achieves beneficial sensitivity precision tradeoff.

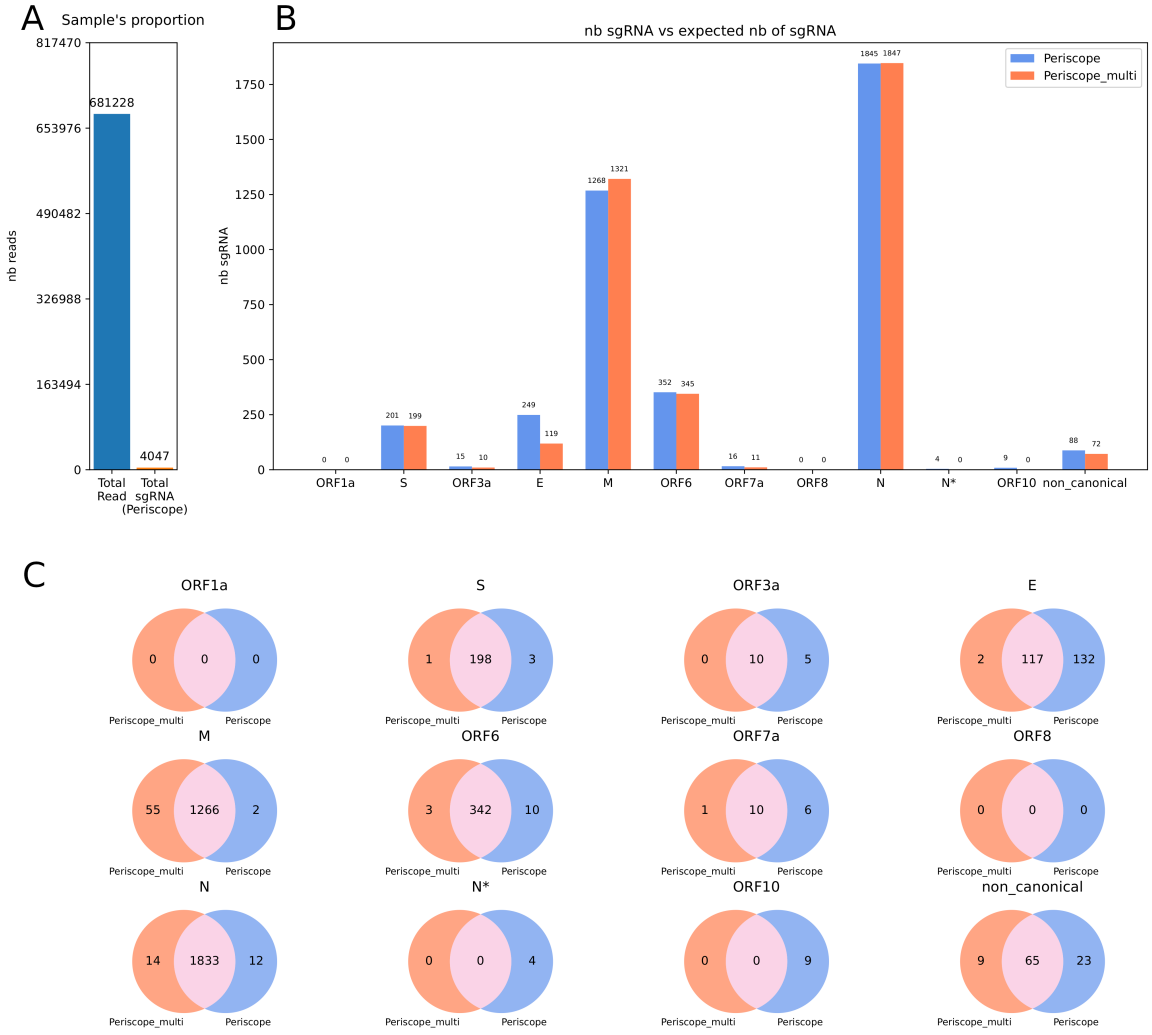

Figure 4.7 – ***Result of `periscope` and `periscope_multi` on the BIO-SMALL dataset.***. *A) The total number of read and the total number of sgRNA found by `periscope`. B) The number of sgRNAs found for each gene by the tools, the blue bar correspond to `periscope` and the orange to `periscope_multi`. C) Venn diagrams showing the proportion of shared reads between `periscope` and `periscope_multi`.*

As simulated data rarely models all the artifacts introduced in sequencing protocols, we investigated whether the overestimation (with LLQ-labelled sgRNA) was also present in biological data. When analysing the BIO-SMALL dataset (Fig. 4.7), we observed similar trends as in the SIM dataset. The number of sgRNAs belonging to E is twice as high in `periscope` as in `periscope_multi`. There were also 9 sgRNAs belonging to ORF10 found only by `periscope`. For the other sgRNAs, the majority are shared between the

two tools. When excluding the LLQ-labeled sgRNAs in `periscope` (Figure 4.8), we see the same effect as with our simulated dataset. That is, a decrease in sgRNAs found by `periscope` and a harmonization of the results found by the two tools concerning E and ORF10.

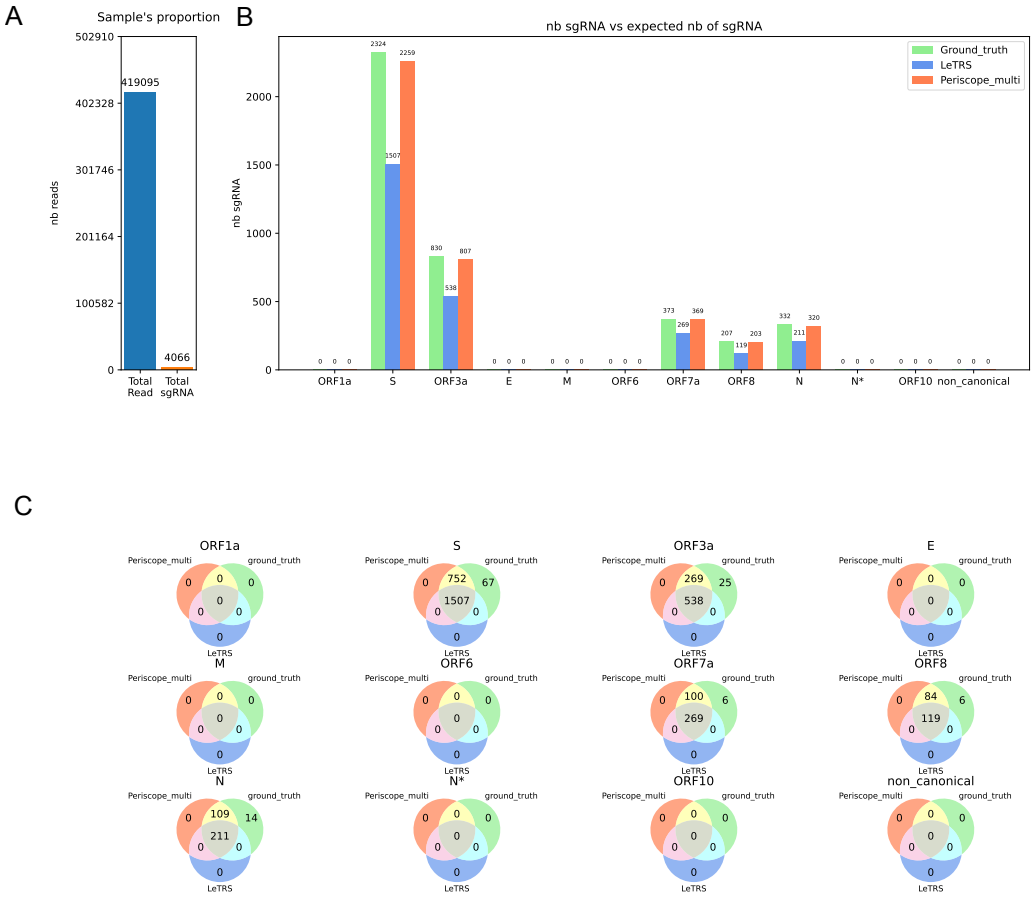

Figure 4.8 – ***Result of LeTRS and `periscope_multi` on the SIM dataset A)*** *Total number of reads and total number of sgRNAs introduced in the sample.* ***B)*** *Number of sgRNA found for each gene by the tools, the blue bar corresponds to LeTRS and the orange to `periscope_multi`.* ***C)*** *Venn diagrams showing the proportion of shared reads between `periscope` and `periscope_multi`*

4.3.2 Additional tools and mapping configurations

In Tables 4.1 and 4.2, we have also included results with other tools (sgDI-tector, LeTRS), read mappers (graphmap (117), BWA-MEM(72)), and configurations of minimap2. Graphmap does not support multiple references and thus is not compatible with `periscope_multi` but we have implemented it in a modified version of `periscope`. For BWA-MEM with `periscope_multi`, many reads are not correctly placed on the genome. This is likely because BWA-MEMs alignment profile is adapted for Illumina short-read alignments to mammalian-sized genomes. Graphmap's very poor results with `periscope` can be explained by the way in which the alignment of graphmap has been designed. Graphmap always tries to maximise the length of the alignment and will therefore make very little use of soft clipping, preventing periscope from correctly identifying the sgRNAs.

With more sensitive mapping parameters, `periscope_multi` finds slightly more reads (< 1% of total) categorized as sgRNA than when using default mapping parameters . All the other tools or configurations give inferior results to `periscope_multi`.

Tool \ sgRNA	S	ORF3a	E	M	ORF6	ORF7a	ORF8	N	ORF10	nc-sgRNA
Ground_truth	2334	830	0	0	0	373	207	332	0	0
periscope_multi (tuned)	2273	827	0	0	0	369	207	326	0	0
periscope_multi	2259	807	0	0	0	369	203	320	0	0
periscope (without LLQ)	1962	721	0	0	0	323	179	276	0	4
LeTRS	1507	538	0	0	0	269	119	211	0	0
Periscope	2283	790	288	0	0	371	196	303	71	4
periscope_multi (BWA-MEM)	2056	699	0	0	0	336	181	295	0	184
periscope (graphmap)	324	699	0	0	0	313	102	21	0	2053
sgDI-tector	0	0	0	0	11	138	155	3	0	8

Table 4.1 – *Results for the SIM dataset. Each column represents the number of sgRNAs found by the tools. The rows are sorted in descending order of accuracy with the most favourable results appearing in the first row.*

Tool \ sgRNA	S	ORF3a	E	M	ORF6	ORF7a	ORF8	N	ORF10	nc-sgRNA
periscope_multi (tuned)	199	10	125	1341	346	11	0	1857	0	67
periscope_multi	199	10	119	1321	345	11	0	1847	0	72
periscope (without LLQ)	193	10	122	1244	336	10	0	1798	0	88
LeTRS	187	10	111	1159	319	11	0	1619	0	/
periscope_multi (BWA-MEM)	189	9	71	1304	277	11	0	1802	0	428
Periscope	201	15	249	1268	352	16	0	1845	9	88
periscope (graphmap)	21	3	10	107	3	1	0	10	0	11743
sgDI-tector	0	1	0	17	0	0	0	11	0	0

Table 4.2 – *Results for the BIO-SMALL dataset. Each column represents the number of sgRNAs found by the tools.*

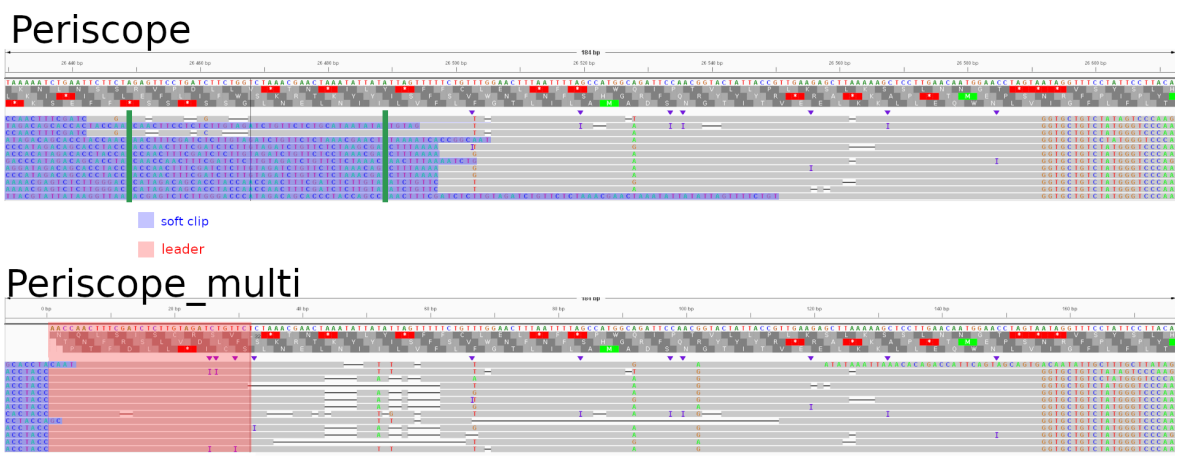

Figure 4.9 – ***Picture of the different alignments obtained by `periscope` and `periscope_multi`*** *Figure shows alignment of reads classified as non-canonical sgRNA by `periscope` and found as canonical by `periscope_multi`. The same 12 reads are present in the two pictures but there position differ. The region between the green bar represents ORF area which determines whether a read is subgenomic in `periscope`, and blue region shows soft-clipped part of the reads. A read alignment has to start in this region between green bars to be considered as a sgRNA, which is not the case for the reads aligned with `periscope` (upper panel). The red part shows the leader position in `periscope_multi`, which is soft-clipped in `periscope`'s alignments.*

4.3.3 Soft clipping leads to an overestimation of non-canonical sgRNA

The absence of non-canonical sgRNA in the SIM dataset allows us to verify the presence of FP among the non-canonical sgRNAs. Figure 4.6 shows that `periscope` misclassified three reads as coming from non-canonical sgRNAs even with LLQ-tagged sgRNA reads excluded. As for the BIO-SMALL dataset, Figure 4.7 shows that `periscope` found 88 non-canonical sgRNAs while `periscope_multi` found 72 with the majority shared by both tools (Figure 4.7C). However, 23 of the sgRNAs are only found by `periscope` while only 9 are only found by `Periscope_multi`. These 23 discordant predictions were split across several locations on the genome. We manually analyzed the region with the most discordant classified reads, consisting of 12 reads (for the M gene). Figure 4.9, shows results for 12 of the 23 non-canonical sgRNAs only found by `periscope` belonging in `periscope_multi` to S gene's sgRNA. In this figure, alignment for `periscope` and `periscope_multi` using IGV (102) are shown.

ER	Tool name	S	ORF3a	E	M	ORF6	ORF7a	ORF8	N	ORF10	nc_sgRNA	F1_score
-	ground thruth	100	100	100	100	100	100	100	100	100	0	1
4	pericope	100	98	242	100	98	98	97	96	124	0	0.90
4	periscope without LLQ	92	93	97	98	95	97	92	91	93	0	0.97
4	periscope multi	98	98	93	100	98	97	96	95	97	2	**0.98**
7	pericope	100	98	271	100	99	98	97	97	141	0	0.88
7	periscope without LLQ	88	85	93	92	95	91	91	93	94	0	0.95
7	periscope multi	98	98	93	100	98	97	96	95	97	2	**0.98**
10	pericope	96	90	615	93	97	98	88	85	199	4	0.70
10	periscope without LLQ	72	78	80	70	72	67	71	67	75	4	0.83
10	periscope multi	93	92	82	91	89	97	89	87	88	5	**0.94**
15	pericope	90	75	1342	87	84	93	80	58	469	5	0.43
15	periscope without LLQ	40	55	42	43	44	45	46	31	52	5	0.61
15	periscope multi	88	72	33	81	69	83	77	57	73	12	**0.81**
20	pericope	80	57	1534	73	71	64	43	51	753	4	0.29
20	periscope without LLQ	22	19	16	31	31	17	20	18	27	4	0.36
20	periscope multi	61	43	14	63	51	55	42	40	52	6	**0.63**

Table 4.3 – *Results of Periscope (with and without LLQ labelled sgRNA) and Periscope_multi for custom datasets with error rates (ER) of 7%, 10%, 15% and 20%. For each error rate, the data set consists of 100 reads for each sgRNA, i.e. a total of 900 sgRNAs. Boldfaced values indicate the highest F1_score.*

The reads are aligned between the positions 26,430 and 26,612 in the SARS-CoV-2 Wuhan reference genome. In the `periscope_multi` alignment, where the reference sequence corresponding to each sgRNA with the leader is included, we can see that the read is systematically mapped in the leader part (red area). However the aligned sequence following the leader includes several smaller or larger gaps. In figure 4.9 when we compare the alignment positions given by `periscope_multi` with those given by `periscope`, we can see

that except for one or two reads which have large alignment gaps after the leader, all the reads begin their alignment inside the ORF area and will be labeled as known sgRNA by `periscope_multi`. To study the detection performance for all sgRNAs simultaneously, additional datasets were simulated where all the canonical sgRNAs are in the sample at equal abundance in low proportions with different error rates (see table 4.3). It can be observed that, even though the error rate impacts `periscope_multi`, it remains the most effective tool, consistently achieving the highest F1 score.

4.3.4 Reanalyzing the BIO-LARGE dataset reveals over detection of non-canonical sgRNA

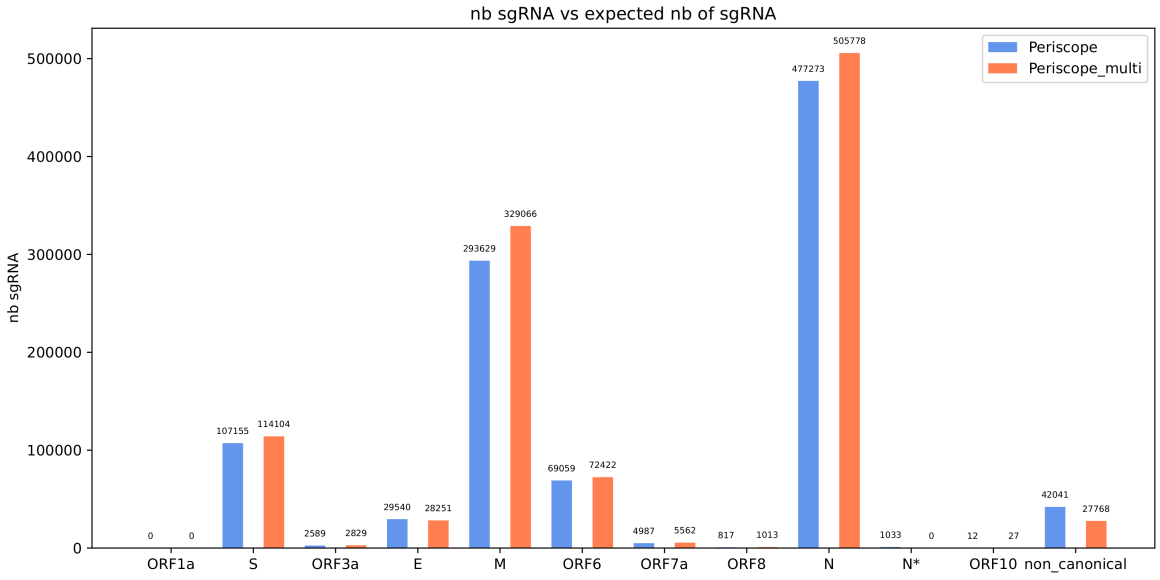

Figure 4.10 – ***Result of `periscope` and `periscope_multi` with the BIO-large dataset without the LLQ-labelled sgRNA*** *The bar represent the number of sgRNA found in the BIO-LARGE dataset, the blue bar correspond to `periscope` result and the orange to `periscope_multi` result.*

To confirm the results shown using the SIM-dataset and the BIO-SMALL dataset, we evaluated `periscope_multi` and `periscope` (using LLQ-labelled sgRNA) on the BIO-LARGE dataset (Fig. 4.12). We observe that, similarly to previous experiments, `periscope` finds substantially more sgRNAs for the E and ORF10 genes than `periscope_multi`. Figure 4.10 shows that, as in the simulated dataset, if the LLQ-labeled are discarded, the same proportion of sgRNAs are found in both tools, but decreases the overall number of sgRNAs found by `periscope` for the other genes. For the M, N, ORF6 and S genes `periscope_multi` found slightly more sgRNAs than `periscope`. For all the other genes, `periscope` finds more sgRNAs than `periscope_multi`. In addition, `periscope` found 42041 non-canonical sgRNAs, a 51% higher estimation compared to `periscope_multi`'s 27768 non-canonical sgRNA classifications.

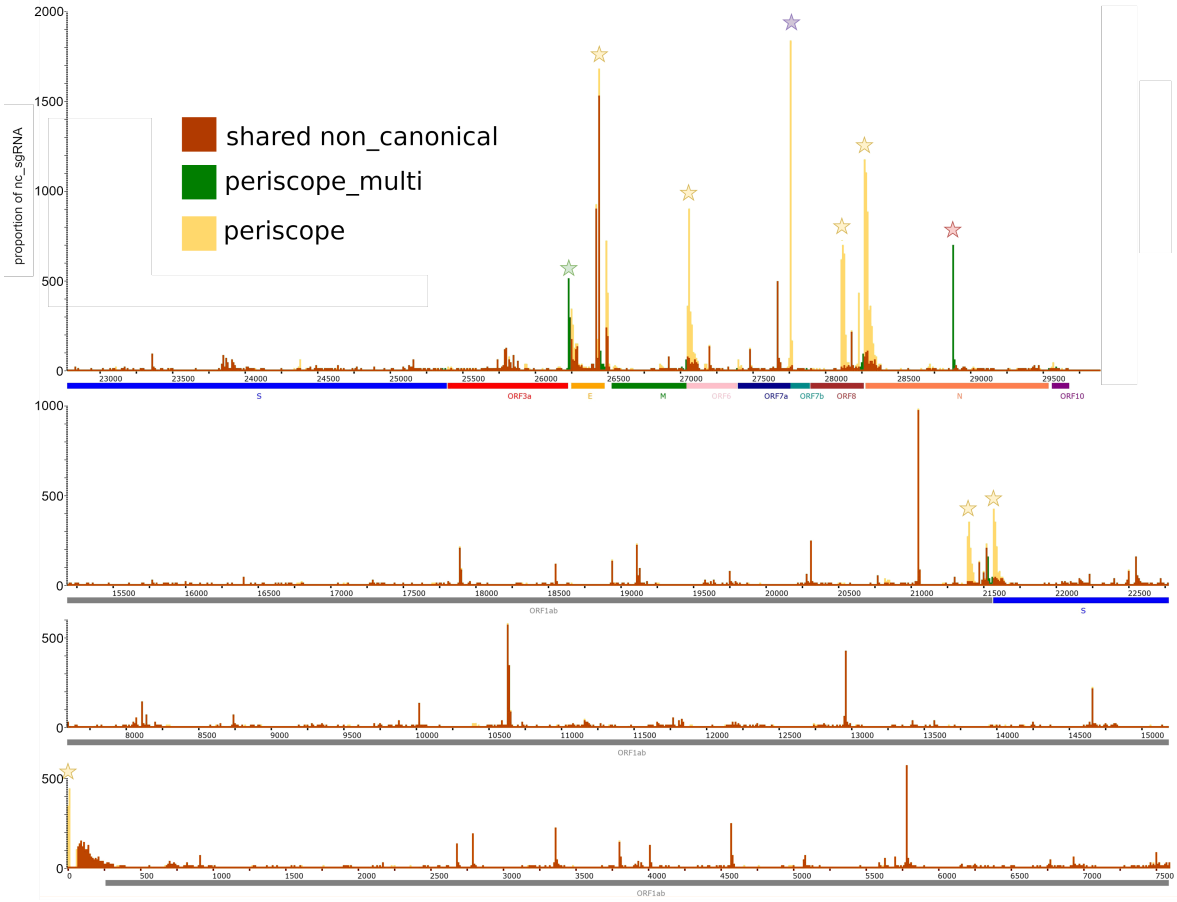

Figure 4.11 – *Positions and proportion of the non-canonical sgRNAs among the SARS-CoV-2 genome from the results of the BIO-LARGE dataset. In the figure, all the non-canonical RNAs are bucketed according to their positions (10 nucleotide windows). The height of the spike illustrates the number of non-canonical sgRNAs in the window. The height corresponds to the total number of sgRNAs in a window divided by 10. The red bar shows that in this window, the two tools share the same number of sgRNAs. The green bar shows non-canonical sgRNAs found only by* `periscope_multi`*, and the yellow represents those found only by* `periscope`*. The star represents an area with a high divergence between the two tools. Yellow stars highlight positions where* `periscope` *reports more nc-sgRNAs than* `periscope_multi`*. Green stars show regions where* `periscope_multi` *reports more nc-sgRNAs than* `periscope`*. The red stars show divergence with* `periscope_multi` *corresponding to the sgRNA labeled as N* in* `periscope` *but not labeled as canonical in* `periscope_multi`*, and the blue star shows divergence with* `periscope` *corresponding to ORF7b that are considered as canonical sgRNA in* `periscope_multi` *and not in* `periscope`*. Below the figure are the genes annotated according to their position.*

We studied the the locations of the non-canonical sgRNA predictions more closely. Figure 4.11 shows a representation of all the sgRNA positions on the SARS-CoV-2 genome. There are many high abundance non-canonical prediction sites where `periscope` and `periscope_multi` agree (red peaks). However, as can be seen in yellow peaks in Figure 4.11, `periscope` predicts several high abundance peaks of non-canonical sgRNAs that `periscope_multi` does not predict. Some notable discrepancies are marked with stars. For example, for the E gene, `periscope_multi` finds a high amount of sgRNAs at one position while `periscope`'s predictions at the site are more spread out.

Two peaks (indicated by red and blue stars) likely arise from the distinct gene classification

systems used in `periscope_multi` and `periscope`. In the `periscope` system, there is an additional canonical sgRNA called N*, which corresponds to the same position as the non-canonical sgRNA identified by `periscope_multi` under the red star. Similarly, in `periscope_multi`, there is an extra canonical sgRNA named ORF7b, sharing the same position as the non-canonical sgRNA marked by `periscope` with the blue star. Overall, we see that `periscope_multi` non-canonical sgRNA predictions are much fewer and, e.g., for the E gene, more precise.

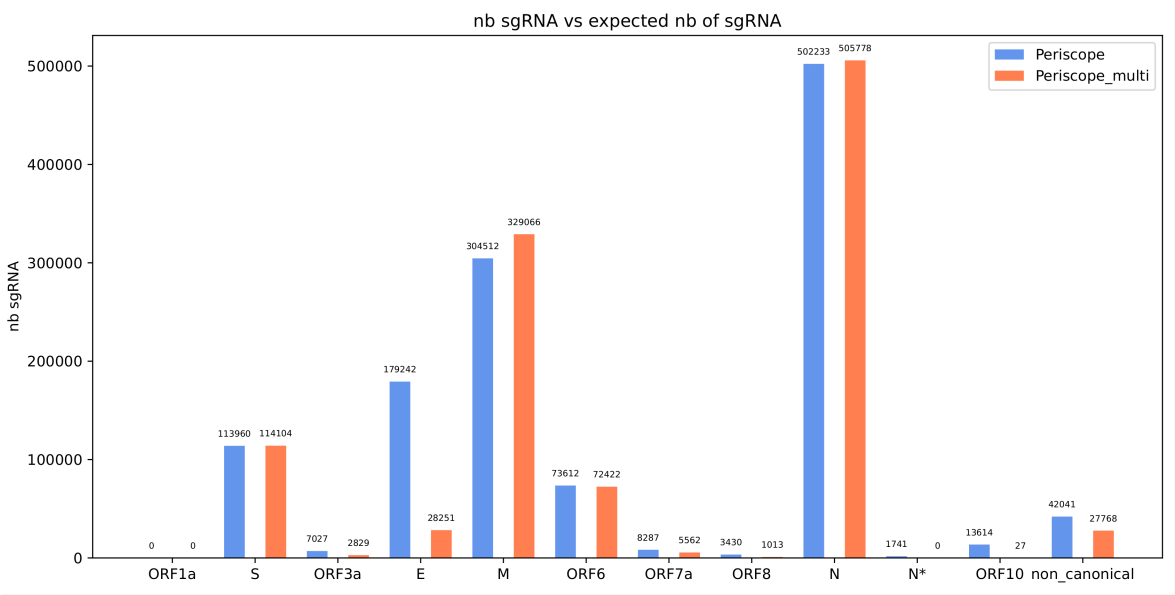

Figure 4.12 – ***Result of `periscope` and `periscope_multi` with the BIO-LARGE dataset*** *the bar represent the sgRNA founded in the BIO-LARGE dataset, the blue bar corresponds to* `periscope` *results and the orange to* `periscope_multi`

4.3.5 Validating non-canonical sgRNAs in BIO-LARGE

An interesting aspect of non-canonical sgRNAs is if they are coding for the same proteins as canonical sgRNAs or whether the proteins get truncated. We wanted to assess this for all `periscope_multi`'s non-canonical sgRNA predictions from the BIO-LARGE dataset. We took all the 27,768 reads classified as non-canonical sgRNA by `periscope_multi` and inferred the protein they coded by using a consensus approach strategies (see section 4.2.3 to find the position in the genome where the non-canonical sgRNA most likely starts. For 16,037 reads (57.8% of total) all four alignments agree, with 9,466 of them (59%) normal proteins and 6,571 of them (41%) truncated. In the case where three alignments agree, 4,632 reads (16.6% of total) are found with 2688 (58%) of them normal proteins and 1944 (42%) of them truncated. The 7,099 (25.6%) remaining reads are labelled as uncertain in the protein columns.

4.3.6 Predictions on BIO-SMALL-ILLUMINA

We also tested `periscope_multi` on the BIO-SMALL-ILLUMINA dataset (Table 4.4. sgDI-Tector found between 10% and 50% more sgRNA than periscope and periscope_multi. LeTRS found slightly more sgRNA than periscope_multi (less than 10, except for

ORF6). Periscope found the fewest sgRNA. To confirm the additional sgRNAs found by periscope_multi, we use IGV to compare the alignments (figure 4.13). The differences are due to certain reads being mapped by BWA-MEM to the leader region, i.e. the first 60 nucleotides of the reference genome. This is due to how soft clipping works. It is the smallest part that cannot or is too hard to align that is soft clipped. However, for some reads, the part containing the leader is larger than the gene segment. So it is the segment containing the gene that is soft clipped. We also observed occasional short reads (less than 1%) were classified as sgRNA by periscope_multi. We believe they are misaligned because they contained several errors in the part aligning to the leader. Finally, there are fifteen reads only found by periscope and not by periscope_multi. These are due to read only containing small parts of the leader region and, thus, are discarded by periscope_multi due to the threshold of a minimum of 30% of the nucleotides of the leaders needed for a read to be considered as non-canonical.

There is no graphical comparison between periscope, periscope_multi and the other tools because LeTRS and sgDI-tector didn't provide from which read each sgRNA come from. Moreover, we lack ground truth for the BIO-SMALL-ILLUMINA dataset, and thus, we cannot be sure about the result. In (69), the authors claim that sg-DI(27) are better for detecting sgRNAs with short reads, which is in line with our results.

Tool \ sgRNA	S	ORF3a	E	M	ORF6	ORF7a	ORF8	N	ORF10	nc-sgRNA
sgDI-tector	63	364	41	161	104	190	65	510	0	50
LeTRS	43	256	26	115	78	159	43	408	1	/
periscope_multi (BWA-MEM)	43	246	20	115	42	147	56	407	0	51
Periscope	34	239	7	109	13	140	36	340	9	38

Table 4.4 – *Results of all the tools for BIO-SMALL-ILLUMINA dataset. Each row and column represents the number of sgRNAs found by one of the tool.*

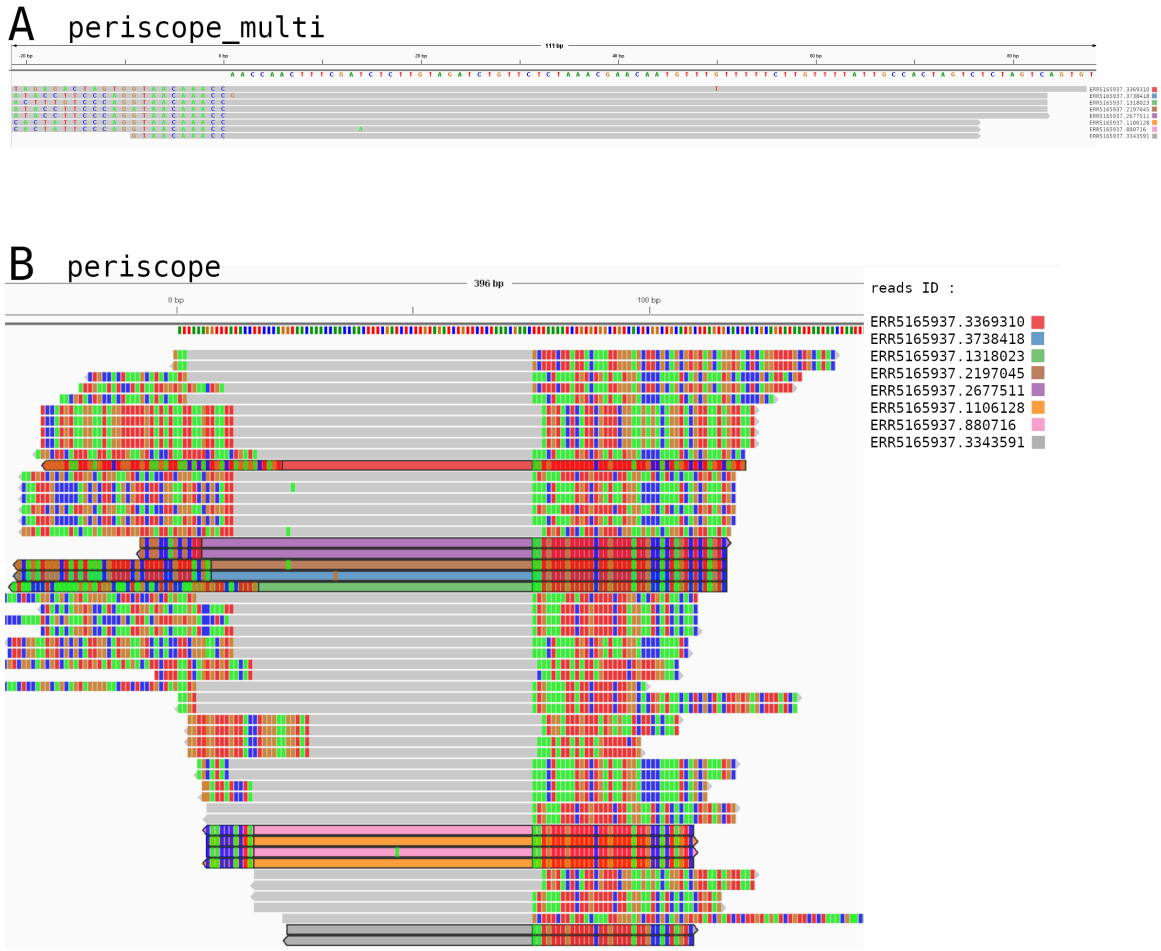

Figure 4.13 – ***Different alignments obtained by `periscope` and `periscope_multi` for the BIO-ILLUMINA dataset.*** *Figure shows alignment of eight reads found as canonical by `periscope_multi` (panel A) and classified as non canonical sgRNA by `periscope`. The same 8 reads are coloured in the periscope alignments (panel B), and the corresponding read ID is displayed to the right.*

4.3.7 Runtime and memory usage

We monitored the computational time of `periscope` and `periscope_multi` on all the 1,145 samples in BIO-LARGE ordered by increasing resources by `periscope_multi`.

In figure 4.14, we observe a large fluctuation in runtime per instance for `periscope`. In contrast, `periscope_multi` is more stable across samples and `periscope_multi` is more than three times faster than `periscope` with a total CPU time of 3.4h compared to 11.0h for `periscope`. When it comes to memory consumption (figure 4.15, the two tools seem to achieve similar results.

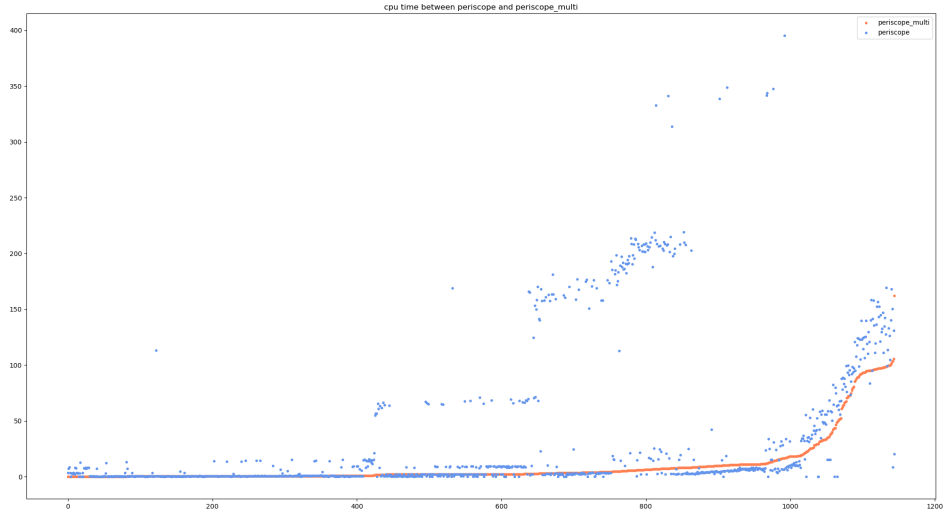

Figure 4.14 – ***CPU time of `periscope` and `periscope_multi` for each sample of the BIO-large dataset*** *The blue point corresponds to `periscope` CPU time and the orange to `periscope_multi`. Each point represents the time for one of the sample of the Bio-large dataset. The samples are sorted in ascending order of time spent with `periscope_multi`*

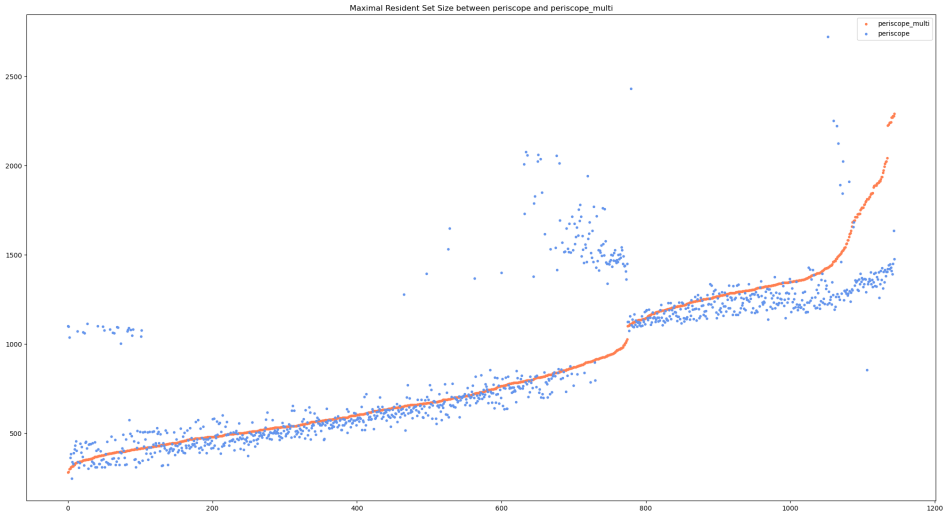

Figure 4.15 – ***Maximal Resident Set Size used by `periscope` and `periscope_multi` for each sample of the BIO-large dataset*** *The blue points correspond to `periscope` and the orange to `periscope_multi`. Each point represents the Maximum RSS for one of the sample of the Bio-large dataset. The samples are sorted in ascending order of the size on `periscope_multi`*

4.4 Discussion

The Discussion section is entirely taken from the corresponding part of Thomas Baudeau, Kristoffer Sahlin, Improved sub-genomic RNA prediction with the ARTIC protocol, Nucleic Acids Research, Volume 52, Issue 17, 23 September 2024, Page e82, https://doi.org/10.1093/nar/gkae687.

We implemented an evaluation pipeline `sgENERATE` that simulates ARTIC-protocol data and used it to compare `periscope` and `periscope_multi` for detecting sgRNAs. We compared the results between the SIM-dataset, BIO-SMALL, and BIO-LARGE datasets. While the release of `periscope` is a useful tool for the study of sgRNAs, we conclude that `periscope` overestimates some of the sgRNA and falsely calls some of the non-canonical sgRNA. We were able to correct much of this bias in our modified version `periscope_multi`. The experiments also shown us that our pipeline `sgENERATE` have been able to simulate data that highlights problematic cases in real datasets, and can provide dataset with ground truth allowing to compare further tools for sgRNA detection.

4.4.1 Periscope bias and minimap2 limitation

As it could be seen in Figure 4.5, `periscope` overestimates the number of the E and ORF10 sgRNA, which likely occurs also in biological data (Fig. 4.7 and 4.12). The analysis in the `periscope` paper didn't include the LLQ-labeled sgRNA. However, if LLQ-labeled sgRNA is excluded, `periscope` underestimates the number of sgRNAs (Fig. 4.6, S2, S3). For example, in figure 4.6 almost 15% of the total number of sgRNAs are missing for ORF7a.

This can be explained by the way `periscope` determines if a read is subgenomic. The reads starting within the ORF area (between green bars in Fig. 4.9) are considered subgenomic for the corresponding gene regardless of the leader's presence. The leader is only used to establish a score for category classification (i.e., HQ, LQ, or LLQ), so `periscope` can't distinguish a read with a noisy leader from a read without the leader. In the `periscope_multi` approach, the leader should be mapped to the reference for a read to be considered subgenomic. This allows the tool to distinguish a noisy leader sequence and a random sequence of the reference genome. `Periscope_multi` will, therefore, be able to estimate the number of sgRNAs better.

Figure 4.5 shows that `periscope` identifies false positive non-canonical sgRNAs, and according to figure 4.7 and 4.9, a big part of the non-canonical sgRNAs called by `periscope` is due to minimap2's excessive soft clipping of the read, making the alignment start further downstream on the reference than what is considered correct. It means that the strategies of `periscope` will highly depend on the quality of the sequencing data. Consequently, the divergence of the non-canonical sgRNAs seen in figures 4.12 and 4.11 are likely due to a misclassification by `periscope`. That will lead to an overestimation of the number of non-canonical sgRNAs. However, we can see that adding the leader in the reference, as in `periscope_multi`, improves the estimation of the non-canonical sgRNA. This addition

increases the length of the potential region to align, which is critical for the seed-and-chaining function of minimap2 (104), which can use the seeds from the leader to avoid excessive soft-clipping. `Periscope_multi` strategies allow to obtain a less noisy visualization of the non-canonical sgRNAs, but figure 4.11 shows that even if a big part of the non-canonical sgRNA is discarded, some of them stay present. This can be observed for the E and M genes.

In the case of ORF7b, there is uncertainty. A study (106) claims that ORF7b does not produce its own sgRNA but is produced from ORF7 sgRNA mRNA. However, in the results from the BIO-LARGE dataset a significant number of sgRNAs for ORF7b are found. Regarding the ORF10 sgRNA, `periscope` identifies only 12 sgRNAs if the LLQ-labelled are excluded. Although `periscope_multi` detects slightly more ORF10 sgRNAs (27), we share the conclusions of (98) that no clear assertion can be made regarding the presence of subgenomic RNA for the ORF10 gene with this dataset.

4.4.2 Other species with sgRNA

In our study, we focus on SARS-CoV-2, but other species with sgRNAs exist such as MERS-COV, SARS-COV-1, and other coronaviridae, e.g., OC43 (66). In addition, a long list of positive-strand viruses exist, and share the mechanism of sgRNA synthesis (66). Due to the lack of existing ARTIC long-read datasets, we could not compare `periscope` and `periscope_multi` on other species. Moreover, both `periscope` and `periscope_multi` require the positions of the genes and the leader sequence. If this information is unknown for a species, new tools for discovering sgRNA will be needed.

4.4.3 Conclusion

Although viral organisms are well studied, most bioinformatics methods that study viruses are based on computational tools built for eukaryotes (51, 65). This study presents a bioinformatic evaluation pipeline, `sgENERATE`, and a tool for accurate sgRNA detection, `periscope_multi`. `sgENERATE` simulates data from ARTIC protocol and helps assess the accuracy of sgRNA prediction tools. Using `sgENERATE` we showed that the state-of-the-art method `periscope` is biased by over-estimating the number of some sgRNAs, and by falsely finding non-canonical RNAs. Guided by `sgENERATE` we improve `periscope`'s informatics pipeline to produce more accurate canonical and non-canonical sgRNA predictions while having over three times faster runtime.

Chapter 5

Conclusion and perspective

5.1 Conclusion

This thesis began with an exploration of how mapping algorithms have evolved over time, with a particular focus on long-read mappers. This initial investigation was supported by the knowledge and synthesis developed during the preparation of our first paper (104), a comprehensive review of long-read mapping strategies published in *Genome Biology*.

In the second part, we analysed the behaviour of long-read mappers when applied to viral data. This study, inspired by the results presented in my third paper (9) (submitted to PCI Genomics), highlights the challenges specific to viral genomics. To support this work, we developed an open-source Snakemake pipeline that automates read simulation, mapping using multiple mappers, and result analysis. The pipeline was designed to promote reproducibility and extensibility, making it easy to integrate and benchmark new mappers. It is freely available at the following address: https://github.com/ThomasBaudeau/BenchMapL.

Finally, we focused on the specific case of sub-genomic RNAs (sgRNAs), drawing from the research published in my second article (10), published in *Nucleic Acids Research*. In this study, we proposed two dedicated tools:

- `sgENERATE`, which simulates datasets containing both canonical and non-canonical sgRNAs; (available at https://github.com/ThomasBaudeau/sgENERATE)
- `periscope_multi`, a tool designed to detect and classify sgRNAs in sequencing data. (available at https://github.com/ThomasBaudeau/periscope_multifasta).

5.1.1 Evolution and adaptation of long read mappers

As illustrated throughout this manuscript, there is a huge diversity of mapping techniques. However, among the different seeds and chaining techniques in use, we have confined ourselves to the best-known and most widely used mappers. This raises the question of how and whether mappers will continue to evolve. To answer this, it is essential to consider a critical factor: the data with which these tools are used. Indeed, the rapid evolution of mapping algorithms goes hand in hand with the fast-paced advancements in long-read

sequencing technologies. As mentioned in the introduction, long-read technologies initially had an error rate of approximately 20%, which has since improved to 5%-1%, with current claims of error rates as low as 1% or even lower. This improvement in sequencing accuracy has made it possible to use fuzzy seeds, which were previously only viable for short-read technologies. Thus, mappers and their heuristic strategies evolve alongside sequencing technologies.

Beyond algorithmic efficiency, another crucial aspect is the computational resources required to run these tools. Bioinformatics programs are designed to run on computers with varying levels of physical memory, RAM, and processing power. While some tools may achieve outstanding performance, they might require computational resources that are only available in a handful of well-funded laboratories. This raises another critical question: Can a program that only runs on expensive, high-end hardware truly be considered the best solution? Ultimately, the choice of the best mapper and heuristics depends not only on the dataset but also on the computational resources available to the user. A highly accurate mapper might require extensive memory and processing power, making it impractical for users with limited hardware. Conversely, a faster, lightweight mapper may sacrifice some accuracy but be more accessible. Thus, the 'best' solution is context-dependent, balancing accuracy, efficiency, and feasibility based on the user's constraints.

5.1.2 Handling viral long reads

We have observed that viral data, due to the reduced read size caused by amplification and a high error or mutation rate, can slightly impact the performance of mappers. However, this impact is generally limited and does not appear to significantly alter results. Furthermore, by adjusting tool configurations, it is possible to optimize their performance in most cases.

Nevertheless, certain intrinsic biological characteristics of viruses can pose challenges. As discussed in section 4, some heuristics, such as soft clipping, are not designed to handle specific viral features. However, this functionality remains essential in eukaryotic and prokaryotic organisms, as it helps avoid wasting time and resources on chimeric reads. Finally, a solid understanding of how these tools function is crucial for addressing these particularities and adapting analyses accordingly.

5.1.3 The unseen work behind bioinformatics tools

Gaining a deep understanding of how bioinformatics tools function can sometimes be challenging. When a scientific paper is published, the development of a tool is not necessarily complete. Some tools continue to evolve over time, such as minimap2, which has undergone improvements in its seeding strategy. While some of these changes are documented in follow-up publications, this is not always the case. Although most code-sharing platforms provide a record of modifications made to a tool, they are often not user-friendly for beginners.

Nevertheless, these updates remain beneficial for users, ensuring that the tools they rely on

remain efficient and up to date. This presents a real challenge for the scientific community: improving transparency and providing better insights into the evolution of bioinformatics tools over time. Currently, no viable solution exists. Since scientific publications are primarily intended to present methodological or conceptual advancements, they are not well suited for tracking software updates and modifications.

Beyond the issue of traceability, software maintenance and bug fixes pose another major challenge. Most bioinformatics tools are developed by researchers, often as proofs of concept to demonstrate the efficiency of a new algorithm rather than as long-term software solutions. Unlike commercial software, these tools are typically created within research projects with limited funding, making their long-term maintenance uncertain. Yet, a part of these tools end up being widely adopted by the community.

While some laboratories have the resources to hire engineers to stabilize and maintain their tools, this remains the exception rather than the rule. In most cases, researchers must take on this maintenance work themselves alongside their primary research activities. However, despite its critical importance to the scientific community, such work is rarely recognized academically and does not significantly contribute to career advancement.

5.2 Perspectives

5.2.1 Benchmarking the strategies behind the tool

During the design of the benchmark, the question of the evaluation's objective quickly arose. Due to the lack of comparisons between third-generation mappers at that time, it seemed relevant to evaluate these tools as a whole in order to provide an overview to the community. Since then, a benchmark assessing these tools on eukaryotic and prokaryotic data has been published (79).

However, analysing the performance of different mappers on amplified viral data has raised several questions. We observed a slight but noticeable effect of read length on the mapping rates of the tools. Therefore, we do not yet have a definitive explanation for the observed decrease in the number of mapped reads. Our current hypothesis is that shorter reads may not achieve a sufficient alignment score to be retained during the chaining step and this also helps to explain why reducing the size of the seeds would make it possible to recover a higher mapping rate.

To go further, it would be necessary to distinguish and analyse the different strategies employed by mapping tools, such as seed selection, chaining algorithms and alignment heuristics, in order to compare them more accurately. This is because each mapper combines these components in a unique way and includes tool-specific optimisations, thresholds or pre-processing steps, making it difficult to isolate the effect of a single algorithmic choice.

To overcome this limitation, the basic strategies should be reimplemented in a common, modular framework where each component can be tested independently. Such a setup would allow controlled experiments, where only one parameter is changed at a time, which

would allow rigorous evaluation of the impact of different strategies on performance in various contexts (e.g. high error rate, chimeric reads, repetitive regions, short versus long reads). This approach would require significant development effort, particularly to faithfully reproduce the behaviour of each strategy without the confounding factors introduced by real implementations. However, it would provide more robust and reproducible information about the mapping process and could be a valuable resource for benchmarking and guiding the design of future mapping tools.

5.2.2 Potential enhancement of Persicope_multi 2

Regarding `periscope_multi`, improvements could still be made to the tool. For the moment, it can only manage non-canonical TRS-B dependent sgRNAs, i.e. the tool cannot detect sgRNAs that do not have a leader. To be able to detect them, it would be necessary to be able to detect reads with long deletions. This is technically feasible using an alignment strategy similar to that already used for splicing without taking splice sites into account. In addition, it would be interesting to ensure that the tool could detect sgRNAs without the need for prior knowledge, as proposed by sgDI-tector. For example, the first step would be to review the way in which soft clipping is designed to make it more effective on this type of data. This could be achieved by changing the soft clipping trigger threshold, so that the tool is less inclined to clip and instead attempts to produce a complete alignment. However, in tools such as minimap2, this threshold is not easily configurable and would require direct modification of the aligner source code. This limitation highlights the importance of designing either custom alignment strategies or adapted alignment tools when faced with certain biological particularities. As for detection without knowledge of the leader, this part seems more complicated to implement. In fact, the possibility of relying directly on sgRNA references gives `periscope_multi` a high degree of precision. To maintain the use of references, an intermediate step would have to be added, which would inevitably take up both memory and computing time. One possible solution, using the ARTIC protocol, would be to perform clustering step. With sufficiently fine clustering, it would be possible to separate expected amplicons from chimeric amplicons. The sgRNAs would then be expected to form larger clusters of chimeric amplicon than the nc-sgRNAs. The consensus sequence of each cluster could then be used as a reference to identify canonical sgRNAs. A multiple alignment of the consensus sequences of each cluster would then allow the extraction of the shared sequence among the consensus, probably corresponding to the leader.

5.2.3 Mapping with raw signal

With the continuous improvement in the quality of long reads, strategies once reserved for short reads are gradually becoming applicable to long reads like the appearance of fuzzy seeds with blend. If this trend continues, designing long-read mappers without a chaining step seems increasingly feasible. Such a change would reduce computation time and avoid biases introduced by these heuristics, thereby simplifying the alignment process while maintaining sufficient accuracy for biological analyses.

At the same time, Nanopore technology is opening new horizons for sequencing and anal-

ysis. Currently, the basecalling step, which converts raw signals into nucleotide sequences, represents a bottleneck in the data analysis pipeline. This conversion is not only time-consuming and resource-intensive but can also introduce errors that complicate result interpretation. To overcome this limitation, several approaches have been developed to directly map reads from raw signals without basecalling. Tools such as Uncalled (68) and Sigmap (130) have demonstrated the feasibility of this approach, offering significant time savings. However, these solutions are still limited to small genomes, and their application to more complex genomes remains a major challenge. Optimizing these tools could pave the way for a new generation of mappers, leveraging raw Nanopore sequencing signals for greatly reducing the need for basecalling. We could, for example, imagine applications where only reads of interest would be basecalled for future analysis.

This direction appears particularly promising in the context of adaptive sampling or targeted sequencing, where reads are filtered in real time as they exit the nanopore (87). Unlike conventional approaches that sequence all DNA present in a sample, adaptive sampling analyzes raw signals on-the-fly and interrupts sequencing of unwanted molecules. This is especially relevant for viral genome analysis, where the target DNA or RNA is often present in low abundance compared to host material. By selecting only relevant molecules early in the process, this strategy can improve both speed and specificity, while eliminating the need for prior amplification.

Although these approaches may seem disconnected from the classical concepts of seeds, chaining and alignment that are at the heart of this thesis, they are in fact closely related. Mapping from a raw signal always requires the identification of anchor points between segments of the signal and reference regions. Adapting and extending the techniques explored in this thesis to the signal domain; such as designing robust seed-like models directly on signal features; could be an interesting direction for future research.

Bibliography

[1] Artic Network, March 2023. [Online; accessed 19. Mar. 2024]. 48

[2] Management Group Liefer Laura A. 51 Wetterstrand Kris A. 51 Good Peter J. 51 Feingold Elise A. 51 Guyer Mark S. 51 Collins Francis S. 52, Baylor College of Medicine Human Genome Sequencing Center*, Washington University Genome Sequencing Center*, et al. Identification and analysis of functional elements in 1% of the human genome by the ENCODE pilot project. *nature*, 447(7146):799–816, 2007. 2

[3] Michael Alonge, Sebastian Soyk, Srividya Ramakrishnan, et al. RaGOO: fast and accurate reference-guided scaffolding of draft genomes. *Genome biology*, 20:1–17, 2019. 12

[4] Stephen F Altschul, Warren Gish, Webb Miller, et al. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990. 15

[5] Armando Arias, Simon J Watson, Danny Asogun, et al. Rapid outbreak sequencing of Ebola virus in Sierra Leone identifies transmission chains linked to sporadic cases. *Virus evolution*, 2(1):vew016, 2016. 68

[6] Vladimir L Arlazarov, EA Dinic, MA Kronrod, et al. On economical construction of the transitive closure of a directed graph. In *Dokl. Akad. Nauk SSSR*, volume 194, pages 1209–1210, 1970. 33

[7] Lorraine AK Ayad, Rayan Chikhi, and Solon P Pissis. Seedability: optimizing alignment parameters for sensitive sequence comparison. *Bioinformatics Advances*, 3(1):vbad108, 2023. 41

[8] David Baltimore. Expression of animal virus genomes. *Bacteriological reviews*, 35(3):235–241, 1971. 4

[9] Thomas Baudeau, Camille Marchet, and Mikael Salson. Assessing Long-Read Mappers for Viral Genomics. *bioRxiv*, pages 2024–11, 2024. 91

[10] Thomas Baudeau and Kristoffer Sahlin. Improved sub-genomic RNA prediction with the ARTIC protocol. *Nucleic Acids Research*, 52(17):e82–e82, 2024. 61, 64, 91

[11] Niko Beerenwinkel and Osvaldo Zagordi. Ultra-deep sequencing for the analysis of viral populations. *Current opinion in virology*, 1(5):413–418, 2011. 68

[12] Gary Benson, Avivit Levy, and B Riva Shalom. Longest common subsequence in k length substrings. In *Similarity Search and Applications: 6th International Conference, SISAP 2013, A Coruña, Spain, October 2-4, 2013, Proceedings 6*, pages 257–265. Springer, 2013. 29

[13] Benjamin J Blencowe. Alternative splicing: new insights from global analyses. *Cell*, 126(1):37–47, 2006. 4

[14] Grzegorz M. Boratyn, Jean Thierry-Mieg, Danielle Thierry-Mieg, et al. Magic-BLAST, an accurate RNA-seq aligner for long and short reads. *BMC Bioinformatics*, 20(1):405, December 2019. 37

[15] Andrei Z Broder. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*, pages 21–29. IEEE, 1997. 26

[16] Michael Burrows, David J Wheeler, M Burrows, et al. A block-sorting lossless data compression algorithm. 1994. *Systems Research Center, Palo Alto*, 1994. 17

[17] Mark J. Chaisson and Glenn Tesler. Mapping single molecule sequencing reads using basic local alignment with successive refinement (BLASR): application and theory. *BMC Bioinformatics*, 13(1):238, September 2012. 19, 21, 35, 38

[18] Zigui Chen, Rita Way Yin Ng, Grace Lui, et al. Profiling of SARS-CoV-2 subgenomic RNAs in clinical specimens. *Microbiology Spectrum*, 10(2):e00182–22, 2022. 69, 73

[19] James Clarke, Hai-Chen Wu, Lakmal Jayasinghe, et al. Continuous base identification for single-molecule nanopore DNA sequencing. *Nature nanotechnology*, 4(4):265–270, 2009. 9

[20] Peter JA Cock, Tiago Antao, Jeffrey T Chang, et al. Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25(11):1422, 2009. 71

[21] Martí Cortey, Gastón Arocena, Emanuela Pileri, et al. Bottlenecks in the transmission of porcine reproductive and respiratory syndrome virus (PRRSV1) to naïve pigs and the quasi-species variation of the virus during infection in vaccinated pigs. *Veterinary research*, 49(1):107, 2018. 6

[22] Rosina De Cario, Ada Kura, Samuele Suraci, et al. Sanger validation of high-throughput sequencing in genetic diagnosis: still the best practice? *Frontiers in genetics*, 11:592588, 2020. 7

[23] David Deamer, Mark Akeson, and Daniel Branton. Three decades of nanopore sequencing. *Nature biotechnology*, 34(5):518–524, 2016. 9, 10

[24] Clara Delahaye and Jacques Nicolas. Sequencing DNA with nanopores: Troubles and biases. *PloS one*, 16(10):e0257521, 2021. 11

[25] Arthur L Delcher, Simon Kasif, Robert D Fleischmann, et al. Alignment of whole genomes. *Nucleic acids research*, 27(11):2369–2376, 1999. 18

[26] Jill A Dembowski and Neal A Deluca. Purification of viral DNA for the identification of associated viral and cellular proteins. *Journal of Visualized Experiments: Jove*, (126):56374, 2017. 6

[27] Andrea Di Gioacchino, Rachel Legendre, Yannis Rahou, et al. sgDI-tector: defective interfering viral genome bioinformatics for detection of coronavirus subgenomic RNAs. *RNA*, 28(3):277–289, 2022. 70, 72, 85

[28] Juliane C Dohm, Philipp Peters, Nancy Stralis-Pavese, et al. Benchmarking of long-read correction methods. *NAR Genomics and Bioinformatics*, 2(2):lqaa037, 2020. 37

[29] Xiaofeng Dong, Rebekah Penrice-Randal, Hannah Goldswain, et al. Analysis of SARS-CoV-2 known and novel subgenomic mRNAs in cell culture, animal model, and clinical samples using LeTRS, a bioinformatic tool to identify unique sequence identifiers. *Gigascience*, 11:giac045, 2022. 70, 72

[30] Richard O Duda and Peter E Hart. Use of the Hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972. 28

[31] Tim Dunn and Satish Narayanasamy. vcfdist: accurately benchmarking phased small variant calls in human genomes. *Nature Communications*, 14(1):8149, 2023. 48

[32] Robert Edgar. Syncmers are more sensitive than minimizers for selecting conserved k-mers in biological sequences. *PeerJ*, 9:e10805, 2021. 24

[33] John Eid, Adrian Fehr, Jeremy Gray, et al. Real-time DNA sequencing from single polymerase molecules. *Science*, 323(5910):133–138, 2009. 9

[34] Barış Ekim, Kristoffer Sahlin, Paul Medvedev, et al. mapquik: Efficient low-divergence mapping of long reads in minimizer space. In *Research in Computational Molecular Biology*, 2023. 25

[35] Adam C English, Egor Dolzhenko, Helyaneh Ziaei Jam, et al. Analysis and benchmarking of small and large genomic variants across tandem repeats. *Nature Biotechnology*, pages 1–12, 2024. 48

[36] N. R. Faria, M. U. G. Kraemer, S. C. Hill, et al. Genomic and epidemiological monitoring of yellow fever virus transmission potential. *Science*, 361(6405):894–899, August 2018. 9

[37] Nuno Rodrigues Faria, Moritz UG Kraemer, Sarah C Hill, et al. Genomic and epidemiological monitoring of yellow fever virus transmission potential. *Science*, 361(6405):894–899, 2018. 68

[38] Gregory G. Faust and Ira M. Hall. YAHA: fast and flexible long-read alignment with optimal breakpoint detection. *Bioinformatics*, 28(19):2417–2424, 07 2012. 35

[39] Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Proceedings 41st annual symposium on foundations of computer science*, pages 390–398. IEEE, 2000. 17

[40] Can Firtina, Jisung Park, Mohammed Alser, et al. BLEND: A Fast, Memory-Efficient, and Accurate Mechanism to Find Fuzzy Seed Matches. *arXiv:2112.08687 [q-bio]*, April 2022. arXiv: 2112.08687. 30

[41] Can Firtina, Jisung Park, Mohammed Alser, et al. BLEND: a fast, memory-efficient and accurate mechanism to find fuzzy seed matches in genome analysis. *NAR Genomics and Bioinformatics*, 5(1), 01 2023. 25, 36

[42] Can Firtina, Jisung Park, Mohammed Alser, et al. BLEND: a fast, memory-efficient and accurate mechanism to find fuzzy seed matches in genome analysis. *NAR Genomics and Bioinformatics*, 5(1):lqad004, 2023. 38

[43] Martin C Frith, Laurent Noé, and Gregory Kucherov. Minimally overlapping words for sequence similarity search. *Bioinformatics*, 36(22-23):5344–5350, 2020. 19

[44] Martin C Frith, Raymond Wan, and Paul Horton. Incorporating sequence quality data into alignment improves DNA read mapping. *Nucleic acids research*, 38(7):e100–e100, 2010. 35

[45] Zvi Galil and Kunsoo Park. A linear-time algorithm for concave one-dimensional dynamic programming. *Information Processing Letters*, 1989. 30

[46] Yunyun Gao, Hao Luo, Hujie Lyu, et al. Benchmarking short-read metagenomics tools for removing host contamination. *GigaScience*, 14:giaf004, 2025. 6

[47] Gonzalo Giribet and Ward C Wheeler. On gaps. *Molecular phylogenetics and evolution*, 13(1):132–143, 1999. 14

[48] Michael B Hall, Ryan R Wick, Louise M Judd, et al. Benchmarking reveals superiority of deep learning variant callers on bacterial nanopore sequence data. *eLife*, 13:RP98300, 2024. 48

[49] Sarah C Hill, Renato de Souza, Julien Thézé, et al. Genomic surveillance of yellow fever virus epizootic in São Paulo, Brazil, 2016–2018. *PLoS pathogens*, 16(8):e1008699, 2020. 68

[50] Edward C Holmes. What does virus evolution tell us about virus origins? *Journal of virology*, 85(11):5247–5251, 2011. 4

[51] Ari Hong, Dongwan Kim, V Narry Kim, et al. Analyzing viral epitranscriptomes using nanopore direct RNA sequencing. *Journal of Microbiology*, 60(9):867–876, 2022. 89

[52] Martin Hunt, Angie S Hinrichs, Daniel Anderson, et al. Addressing pandemic-wide systematic errors in the SARS-CoV-2 phylogeny. *bioRxiv*, 2024. 44

[53] John D Hunter. Matplotlib: A 2D graphics environment. *Computing in science & engineering*, 9(03):90–95, 2007. 70, 71

[54] Chirag Jain, Alexander Dilthey, Sergey Koren, et al. A fast approximate algorithm for mapping long reads to large reference databases. In *International Conference on Research in Computational Molecular Biology*, pages 66–81. Springer, 2017. 23

[55] Chirag Jain, Daniel Gibney, and Sharma V Thankachan. Co-linear chaining with overlaps and gap costs. In *International Conference on Research in Computational Molecular Biology*, pages 246–262. Springer, 2022. 30

[56] Chirag Jain, Sergey Koren, Alexander Dilthey, et al. A fast adaptive algorithm for computing whole-genome homology maps. *Bioinformatics*, 34(17):i748–i756, 2018. 30, 38

[57] Chirag Jain, Arang Rhie, Nancy F. Hansen, et al. Long-read mapping to repetitive reference sequences using Winnowmap2. *Nature Methods*, April 2022. 24, 30, 36, 38

[58] Chirag Jain, Arang Rhie, Haowen Zhang, et al. Weighted minimizer sampling improves long read mapping. *Bioinformatics*, 36(Supplement_1):i111–i118, 2020. 24, 35

[59] Miten Jain, Sergey Koren, Karen H Miga, et al. Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nature biotechnology*, 36(4):338–345, 2018. 9

[60] Natasha Jansz and Geoffrey J Faulkner. Viral genome sequencing methods: Benefits and pitfalls of current approaches. *Biochemical Society Transactions*, 52(3):1431–1447, 2024. 6

[61] LE Kafetzopoulou, ST Pullan, P Lemey, et al. Metagenomic sequencing at the epicenter of the Nigeria 2018 Lassa fever outbreak. *Science*, 363(6422):74–77, 2019. 68

[62] Wei-Chun Kao, Andrew H Chan, and Yun S Song. ECHO: a reference-free short-read error correction algorithm. *Genome research*, 21(7):1181–1192, 2011. 12

[63] Uri Keich, Ming Li, Bin Ma, et al. On spaced seeds for similarity search. *Discrete applied mathematics*, 138(3):253–263, 2004. 19

[64] Daehwan Kim, Joseph M Paggi, Chanhee Park, et al. Graph-based genome alignment and genotyping with HISAT2 and HISAT-genotype. *Nature biotechnology*, 37(8):907–915, 2019. 72

[65] Kijin Kim, Kyungmin Park, Seonghyeon Lee, et al. VirPipe: an easy-to-use and customizable pipeline for detecting viral genomes from Nanopore sequencing. *Bioinformatics*, 39(5):btad293, 2023. 89

[66] Gennadiy Koev and W Allen Miller. A positive-strand RNA virus with three very different subgenomic RNA promoters. *Journal of virology*, 74(13):5988–5996, 2000. 89

[67] Eugene V Koonin. Comparative genomics, minimal gene-sets and the last universal common ancestor. *Nature Reviews Microbiology*, 1(2):127–136, 2003. 4

[68] Sam Kovaka, Yunfan Fan, Bohan Ni, et al. Targeted nanopore sequencing by real-time mapping of raw electrical signal with UNCALLED. *Nature biotechnology*, 39(4):431–441, 2021. 95

[69] Denise Lavezzari, Antonio Mori, Elena Pomari, et al. Comparative analysis of bioinformatics tools to characterize SARS-CoV-2 subgenomic RNAs. *Life Science Alliance*, 6(12), 2023. 70, 85

[70] Elliot J Lefkowitz, Donald M Dempsey, Robert Curtis Hendrickson, et al. Virus taxonomy: the database of the International Committee on Taxonomy of Viruses (ICTV). *Nucleic acids research*, 46(D1):D708–D717, 2018. 4

[71] Heng Li. A statistical framework for SNP calling, mutation discovery, association mapping and population genetical parameter estimation from sequencing data. *Bioinformatics*, 27(21):2987–2993, 2011. 47

[72] Heng Li. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv preprint arXiv:1303.3997*, 2013. 19, 35, 79

[73] Heng Li. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, 32(14):2103–2110, 2016. 22, 23, 28, 35

[74] Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, 2018. 23, 33, 38, 71

[75] Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, September 2018. 30

[76] Heng Li. New strategies to improve minimap2 alignment accuracy. *Bioinformatics*, 37(23):4572–4574, 2021. 35

[77] Heng Li, Bob Handsaker, Alec Wysoker, et al. The sequence alignment/map format and SAMtools. *bioinformatics*, 25(16):2078–2079, 2009. 46

[78] Daniel Liu and Martin Steinegger. Block aligner: fast and flexible pairwise sequence alignment with SIMD-accelerated adaptive blocks. *bioRxiv*, 2021. 34

[79] Jonathan LoTempio, Emmanuele Delot, and Eric Vilain. Benchmarking long-read genome sequence alignment tools for human genomics applications. *PeerJ*, 11:e16515, 2023. 38, 93

[80] Lin Lyu, Ru Feng, Mingnan Zhang, et al. Subgenomic RNA profiling suggests novel mechanism in coronavirus gene regulation and host adaption. *Life science alliance*, 5(8), 2022. 70

[81] Udi Manber and Gene Myers. Suffix arrays: a new method for on-line string searches. *siam Journal on Computing*, 22(5):935–948, 1993. 17

[82] Elaine R Mardis. Next-generation DNA sequencing methods. *Annu. Rev. Genomics Hum. Genet.*, 9(1):387–402, 2008. 9

[83] Jason R Miller, Sergey Koren, and Granger Sutton. Assembly algorithms for next-generation sequencing data. *Genomics*, 95(6):315–327, 2010. 12

[84] W Allen Miller and Gennadiy Koev. Synthesis of subgenomic RNAs by positive-strand RNA viruses. *Virology*, 273(1):1–8, 2000. 67

[85] Felix Mölder, Kim Philipp Jablonski, Brice Letcher, et al. Sustainable data analysis with Snakemake. *F1000Research*, 10, 2021. 69

[86] Antonio Mori, Denise Lavezzari, Elena Pomari, et al. sgRNAs: A SARS-CoV-2 emerging issue. *Aspects of Molecular Medicine*, page 100008, 2023. 70

[87] Trevor J Morin, Tyler Shropshire, Xu Liu, et al. Nanopore-based target sequence detection. *PloS one*, 11(5):e0154426, 2016. 95

[88] Chihiro Morishima, Stephen J Polyak, Ranjit Ray, et al. Hepatitis C virus–specific immune responses and quasi-species variability at baseline are associated with non-response to antiviral therapy during advanced hepatitis C. *The Journal of infectious diseases*, 193(7):931–940, 2006. 6

[89] Gene Myers. Efficient local alignment discovery amongst noisy long reads. In *International Workshop on Algorithms in Bioinformatics*, pages 52–67. Springer, 2014. 35

[90] Saul B Needleman and Christian D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970. 31

[91] Peter Neumann, Orlando Yañez, Ingemar Fries, et al. Varroa invasion and virus adaptation. *J. Invert. Pathol*, 103:96–119, 2012. 6

[92] Nam Nguyen Quang, Sophie Goudey, Emmanuel Ségéral, et al. Dynamic nanopore long-read sequencing analysis of HIV-1 splicing events during the early steps of infection. *Retrovirology*, 17:1–24, 2020. 64

[93] Daiki Okada, Fumihiko Ino, and Kenichi Hagihara. Accelerating the Smith-Waterman algorithm with interpair pruning and band optimization for the all-pairs comparison of base sequences. *BMC Bioinformatics*, 16(1):321, 2015. 34

[94] Yukiteru Ono, Kiyoshi Asai, and Michiaki Hamada. PBSIM2: a simulator for long-read sequencers with a novel generative model of quality scores. *Bioinformatics*, 37(5):589–595, September 2020. 43

[95] Yukiteru Ono, Kiyoshi Asai, and Michiaki Hamada. PBSIM2: a simulator for long-read sequencers with a novel generative model of quality scores. *Bioinformatics*, 37(5):589–595, 2021. 69

[96] Yukiteru Ono, Michiaki Hamada, and Kiyoshi Asai. PBSIM3: a simulator for all types of PacBio and ONT long reads. *NAR genomics and bioinformatics*, 4(4):lqac092, 2022. 43

[97] Marc Pagès-Gallego and Jeroen de Ridder. Comprehensive benchmark and architectural analysis of deep learning models for nanopore sequencing basecalling. *Genome Biology*, 24(1):71, 2023. 10, 11

[98] Matthew D Parker, Benjamin B Lindsey, Shay Leary, et al. Subgenomic RNA identification in SARS-CoV-2 genomic sequencing data. *Genome research*, 31(4):645–658, 2021. 70, 71, 89

[99] Joshua Quick, Nathan D Grubaugh, Steven T Pullan, et al. Multiplex PCR method for MinION and Illumina sequencing of Zika and other virus genomes directly from clinical samples. *Nature protocols*, 12(6):1261–1276, 2017. 9, 68, 71

[100] Jingwen Ren and Mark JP Chaisson. lra: A long read aligner for sequences and contigs. *PLOS Computational Biology*, 17(6):e1009078, 2021. 28, 30, 33, 36, 38

[101] Michael Roberts, Wayne Hayes, Brian R Hunt, et al. Reducing storage requirements for biological sequence comparison. *Bioinformatics*, 20(18):3363–3369, 2004. 22

[102] James T Robinson, Helga Thorvaldsdóttir, Wendy Winckler, et al. Integrative genomics viewer. *Nature biotechnology*, 29(1):24–26, 2011. 71, 81

[103] Kristoffer Sahlin. Strobealign: flexible seed size enables ultra-fast and accurate read alignment. *Genome Biology*, 23(1):260, 2022. 25

[104] Kristoffer Sahlin, Thomas Baudeau, Bastien Cazaux, et al. A survey of mapping algorithms in the long-reads era. *Genome Biology*, 24(1):133, 2023. 89, 91

[105] Frederick Sanger, Steven Nicklen, and Alan R Coulson. DNA sequencing with chain-terminating inhibitors. *Proceedings of the national academy of sciences*, 74(12):5463–5467, 1977. 6

[106] Scott R Schaecher, Jason M Mackenzie, and Andrew Pekosz. The ORF7b protein of severe acute respiratory syndrome coronavirus (SARS-CoV) is expressed in virus-infected cells and incorporated into SARS-CoV particles. *Journal of virology*, 81(2):718–731, 2007. 89

[107] Craige Schensted. Longest increasing and decreasing subsequences. *Canadian Journal of mathematics*, 13:179–191, 1961. 28

[108] Melanie Schirmer, Rosalinda D'Amore, Umer Z Ijaz, et al. Illumina error profiles: resolving fine-scale variation in metagenomic sequencing data. *BMC bioinformatics*, 17:1–15, 2016. 8

[109] Robert Schmieder, Yan Wei Lim, Forest Rohwer, et al. TagCleaner: Identification and removal of tag sequences from genomic and metagenomic datasets. *BMC bioinformatics*, 11:1–14, 2010. 69

[110] Fritz J Sedlazeck, Philipp Rescheneder, Moritz Smolka, et al. Accurate detection of complex structural variations using single-molecule sequencing. *Nature methods*, 15(6):461–468, 2018. 33

[111] Joseph Ojonugwa Shaibu, Chika K Onwuamah, Ayorinde Babatunde James, et al. Full length genomic sanger sequencing and phylogenetic analysis of Severe Acute Respiratory Syndrome Coronavirus 2 (SARS-CoV-2) in Nigeria. *PloS one*, 16(1):e0243271, 2021. 7

[112] Jared T Simpson, Kim Wong, Shaun D Jackman, et al. ABySS: a parallel assembler for short read sequence data. *Genome research*, 19(6):1117–1123, 2009. 12

[113] Lloyd M Smith, Steven Fung, Michael W Hunkapiller, et al. The synthesis of oligonucleotides containing an aliphatic amino group at the 5 terminus: synthesis of fluorescent DNA primers for use in DNA sequence analysis. *Nucleic acids research*, 13(7):2399–2412, 1985. 6

[114] Lloyd M Smith, Jane Z Sanders, Robert J Kaiser, et al. Fluorescence detection in automated DNA sequence analysis. *Nature*, 321(6071):674–679, 1986. 6

[115] Temple F Smith and Michael S Waterman. Comparison of biosequences. *Advances in Applied Mathematics*, 2(4):482–489, 1981. 31

[116] Isabel Sola, Fernando Almazán, Sonia Zúñiga, et al. Continuous and discontinuous RNA synthesis in coronaviruses. *Annual review of virology*, 2(1):265–288, 2015. 67

[117] Ivan Sović, Mile Šikić, Andreas Wilm, et al. Fast and sensitive mapping of nanopore sequencing reads with GraphMap. *Nature communications*, 7(1):11307, 2016. 79

[118] Ivan Sović, Mile Šikić, Andreas Wilm, et al. Fast and sensitive mapping of nanopore sequencing reads with GraphMap. *Nature Communications*, 7(1):11307, September 2016. 21, 28, 35, 38

[119] Nicholas Stoler and Anton Nekrutenko. Sequencing error profiles of Illumina sequencing instruments. *NAR genomics and bioinformatics*, 3(1):lqab019, 2021. 8

[120] Derek Tshiabuila, Jennifer Giandhari, Sureshnee Pillay, et al. Comparison of SARS-CoV-2 sequencing using the ONT GridION and the Illumina MiSeq. *BMC genomics*, 23(1):319, 2022. 68

[121] John R Tyson, Phillip James, David Stoddart, et al. Improvements to the ARTIC multiplex PCR method for SARS-CoV-2 genome sequencing using nanopore. *BioRxiv*, 2020. 68

[122] Md Vasimuddin, Sanchit Misra, Heng Li, et al. Efficient architecture-aware acceleration of BWA-MEM for multicore systems. In *2019 IEEE international parallel and distributed processing symposium (IPDPS)*, pages 314–324. IEEE, 2019. 37

[123] Justin Wagner, Nathan D Olson, Lindsay Harris, et al. Benchmarking challenging small variants with linked and long reads. *Cell genomics*, 2(5), 2022. 37

[124] James D Watson and Francis HC Crick. The structure of DNA. In *Cold Spring Harbor symposia on quantitative biology*, volume 18, pages 123–131. Cold Spring Harbor Laboratory Press, 1953. 2

[125] Peter Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory (swat 1973)*, pages 1–11. IEEE, 1973. 17

[126] Aaron M Wenger, Paul Peluso, William J Rowell, et al. Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome. *Nature biotechnology*, 37(10):1155–1162, 2019. 9

[127] David L Wheeler, Tanya Barrett, Dennis A Benson, et al. Database resources of the national center for biotechnology information. *Nucleic acids research*, 36(suppl_1):D13–D21, 2007. 75

[128] Ruby White, Christophe Pellefigues, Franca Ronchese, et al. Investigation of chimeric reads using the MinION. *F1000Research*, 6:631, 2017. 10

[129] Chen Yang, Justin Chu, René L Warren, et al. NanoSim: nanopore sequence read simulator based on statistical characterization. *GigaScience*, 6(4):gix010, 2017. 43

[130] Haowen Zhang, Haoran Li, Chirag Jain, et al. Real-time mapping of nanopore raw signals. *Bioinformatics*, 37(Supplement_1):i477–i483, 2021. 95

[131] Haowen Zhang, Shiqi Wu, Srinivas Aluru, et al. Fast sequence to graph alignment using the graph wavefront algorithm. *arXiv preprint arXiv:2206.13574*, 2022. 34