Haplotype assembly from long reads

Roland Faure

COLLEGE MATHS, TELECOMS
DOCTORAL INFORMATIQUE, SIGNAL
BRETAGNE SYSTEMES, ELECTRONIQUE

Université de Rennes

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES

ÉCOLE DOCTORALE N° 601
Mathématiques, Télécommunications, Informatique, Signal, Systèmes, Électronique
Spécialité : *Informatique*

Par
Roland FAURE

Haplotype assembly from long reads

Thèse présentée et soutenue à Rennes, le 27 novembre 2024
Unité de recherche : IRISA, UMR 6074

Rapporteurs avant soutenance :

Christopher QUINCE — Group leader, Earlham Institute
Alexander DITHLEY — Professor, Heinrich Heine Universität

Composition du Jury :

Président :	Rayan CHIKHI	Group leader, Institut Pasteur
Examinateurs :	Paola BONIZZONI	Professeur à l'Università degli Studi di Milano-Bicocca
	Thomas DERRIEN	Chargé de recherche au CNRS
	Simon DELLICOUR	Professeur à l'Université libre de Bruxelles
Dir. de thèse :	Dominique LAVENIER	Directeur de recherche au CNRS
	Jean-François FLOT	Professeur, Université libre de Bruxelles

Acknowledgement

I want to begin my acknowledgments by thanking the jury members for accepting to read my thesis. Now that it is done, you can save some time and skip over all the rest of the acknowledgments, since you still have 150 pages to read. More generally, I found in the community and at conferences many people who took the time to understand and discuss my work, who were pivotal to lead the best research I could.

I spent three wonderful years while working on this Ph.D. This is thanks to all of the people I have been and worked with, so here go my heartfelt thanks.

D'immenses remerciements vont à Jean-François et Dominique pour avoir rendu cette thèse possible. Je ne le réalisais pas complètement à l'époque, mais vous vous êtes tous les deux vraiment adaptés pour construire avec moi un sujet de thèse qui me motivait, et qui m'a motivé pendant trois ans. Pendant les trois années suivantes, je me suis toujours senti encouragé, soutenu et bien conseillé dans mes directions de recherche. Finalement, vous m'avez donné le gout à la recherche et l'envie de continuer ma carrière académique.

Quand je suis arrivé à Rennes en Juin 2021, pendant le troisième confinement, je ne pouvais qu'éspérer trouver un groupe comme Symbiose. Merci évidemment à mes cothésards de notre année à Symbiose, a.k.a l'équipe de tournage de Patatogène, un film qui restera dans les annales du cinéma moderne. Merci aussi aux thésards qui m'ont précédés et suivis, Victor E., Lucas, Nicolas, Matthieu, Garance, Camille, Arnaud, Kevin, Guillaume, Olivier D., Téo, Meven et aussi à Olivier B., Julien, Victor M., Pierre M., Gaétan et Konogan pour ce qu'on a partagé à l'IRISA et en dehors. Et bien sûr Corentin, Yaël et Thomas qui ont partagé mon bureau. Merci aussi à tous les ingénieurs et chercheurs permanents qui ont beaucoup contribué à me donner le sourire en allant au travail tous les jours, Pierre, Karel, Jacques, Emeline, Riccardo, Olivier, Jeanne, Stéphanie, Yann, Catherine, François C., François M., Emmanuelle, Yann, Matéo, Florian, Samuel et finalement Claire et Fabrice, pour avoir partagé aussi des parties de badminton. Finalement, merci Marie d'être la meilleure et de m'avoir *beaucoup* aidé dans les démarches administratives.

After having known all of the Brussels people via Zoom for two years, I finally got to know you in person in September 2023. I discovered a great crew, Hugo, Alice, Claire, So-

phie, Mohammed, Dina, Stephen and I also discovered that Stephen actually was French-speaking from the start. After having conquered a new office, it was very nice settling in with my fellow newcomers Olivier C. (again! that's the fourth one in my acknowledgments), Helena and Felipe and Olivier de T. (damn it! are the Olivier going to stop for one second). All these aperos and times at the Montmartre were a lot of fun. The badminton with Antoine H., Hugo and all my badminton partners were a lot of fun too, in a different style. A special thanks to Claire and Florence for trying to teach me lab work, an ambitious endeavour doomed to fail. It want also to thank all the EBE members with whom I shared meals and drinks.

I had the chance to collaborate with great people during my Ph.D. I will start with Nadège, even though we technically collaborated more before the Ph.D. than during the Ph.D., you taught me a lot on how to ground bioinformatics problems in biological questions and also how to manage a Ph.D. The strainMiner project was completed thanks to the hard work of Tam and Rumen, and the help from Victor and Riccardo. It taught me much on optimisation, a field I was not familiar with. The conference in Rome in winter was nice too. I participated in the Ph.D. project of Alessandro from which I will retain not only the scientific discussion but also the friendship.

Merci aussi à Emmanuelle, Olivier D. et Vonnick pour m'avoir permis de découvrir ce qu'était l'enseignement dans d'excellentes conditions.

Je ne peux oublier ici tous mes amis que je n'ai pas encore cité, à commencer par mes colocataires. D'abord Nicolas, qui m'a fort sympathiquement permis d'aller en conférence en expliquant aux voisins ce qu'était cette huile sur leur balcon. Puis Sixtine, avec qui nous avons notamment exploré Hyrule. Arthur, avec qui j'ai découvert Bruxelles et fabriqué une bien belle brique. Enfin Elvira, avec qui on a partagé de nombreuses soirées, à l'appartement et ailleurs. En plus de ceux cités ici, tous mes amis m'ont permis de vivre cette thèse dans d'excellentes conditions et qui sont bien sûr invités en mai prochain à Montautre.

Pour finir par le plus important, merci à ma famille, à Louise, Ulysse, Arthur, maman et papa, j'ai toujours hâte de venir passer du temps avec vous. À mes grands-mères, qui sont un repère dans ma vie. Merci aussi à Serge pour son coaching intensif en vulgarisation et à Catherine pour veiller à entretenir mon acuité mentale tous les hivers.

Finalement, une mention spéciale pour Alice, qui n'est pas que le nom de mon assembleur. Vive Yelle.

Table of Contents

INTRODUCTION

Large eukaryotic organisms such as plants and animals have been the subject of scientific study for centuries, with their physiology, behavior, and ecosystems well-documented [1]. In contrast, the study of microorganisms has been historically challenging due to their small size and wide diversity. Their very existence was only confirmed in 1676, when Antonie van Leeuwenhoek observed them through his homemade microscope [2].

Progress in understanding the roles of microorganisms was slow, with their involvement in fermentation and disease only established in the 19th century [3, 4]. The development of techniques to culture microorganisms in the laboratory expanded the range of possible experiments and led to significant advances in the field of microbiology [5, 6].

At the turn of the 20th century, scientists discovered that microorganisms are ubiquitous in natural environments and exist in complex ecosystems called microbiomes, which are composed of bacteria, eukaryotes, archaea, and viruses [7, 8]. Despite the growing number and diversity of described microorganisms, the study of microbiomes as ecosystems remained embryonic throughout the 20th century. The scientific methods did not allow even to determine the composition of a microbiome, as many microorganisms are indistinguishable under a microscope and cannot be cultured in the laboratory [9]. A major breakthrough in the study of microbiomes came with the advent of genomic studies in the early 21st century. These techniques enable biologists to analyze the DNA contained in a sample, which can then be used to identify the organisms present and their functions [10].

As genomic techniques improve, we are now slowly beginning to understand microbiomes and realising their importance in ecosystems. The most widely known example is the human gut microbiome, whose composition impacts greatly our health [11]. Microbiomes have also been shown to play key roles in biogeochemical cycles [12, 13], crop growth [14], and some food chains [15].

The objective of my Ph.D. is to enhance the software used in genomic pipelines, with the aim of providing biologists with new and highly precise tools for the study of microbiomes. I hope this, in turn, will contribute to a deeper understanding of biology and ecology.

0.1 Sequencing and assembling a genome

0.1.1 Genomes

Deoxyribonucleic acid (DNA) [16] is a vital molecule found in all living organisms, responsible for encoding the genetic instructions necessary for their growth, development, and function. The DNA molecule is composed of two complementary strands that twist around each other to form a double helix, held together by hydrogen bonds. Each strand is made up of a long chain of four basic nucleotides - adenine (A), cytosine (C), guanine (G), and thymine (T) - connected by covalent bonds. The structure of DNA is such that the sequence of nucleotides on one strand is the reverse complement of the other - A and T are always paired and C and G are always paired. Combined with the fact that the two strands of DNA are oriented in opposite direction, this entails that the sequence of nucleotides of one strand, for example, "ATCG," is the reverse-complementary sequence on the other strand, "CGAT".

Information can be encoded in the sequence of nucleotides of the DNA strands. The set of all the DNA sequences present in each cell represent the genome of an organism. The early days of molecular biology revealed that some regions of the genome encode the design of proteins, the primary workers of the cell [17]. However, the role of the genome is not limited to this. It can also encode non-translated RNA molecules, such as ribosomal RNAs (rRNAs), regulate the transcription of different RNAs in different conditions through promoter and enhancer sequences, regulate translation through microRNAs [18–20]... In other words, the genome contains most information needed by the cell to function, specialize and adapt to different conditions. It is worth noting that the genome may not contain *all* the information concerning the behavior of the cell. Epigenetic modifications, such as DNA methylation and histone modification, can alter gene expression and cellular function without changing the underlying DNA sequence [21]. Other less studied mechanisms such as the conformation of proteins can also theoretically carry information [22]. Nevertheless, knowing a genome is an invaluable source of information to identify an organism and understand its biology.

Quite remarkably, all known organisms use DNA to carry the information necessary for the functioning of their cells [1]. The reason for the ubiquity of DNA as a carrier of genetic information is not fully understood, but two properties of DNA make it well-suited for this role. Firstly, it is a highly stable, as evidenced by the successful sequencing of

1. except if RNA viruses, such as coronaviruses, count as organisms, but they do not have cells

mammoth DNA thousands of years after the death of the cells [23]. Secondly, the information in DNA is stored twice, on two complementary strands, which provides a built-in mechanism for error-checking and repair during DNA replication [24]. Though it would not be impossible theoretically that some living organisms carry their genome on another molecule (or molecules), modern biology is exclusively focused on DNA-centered cells.

A central protocol of biology today thus consists in recovering the genomes of organisms. In the context of microbiome analysis, we call the set of genomes a metagenome and this approach is particularly precious, as microbiomes cannot be easily observed and cultured. Observing the genomes contained in a sample can give a precise view of the composition of the microbiome and the behaviour of its different species [25, 26].

The protocol to obtain the (meta)genome of a sample can be broken down in three main steps. The first is to extract the DNA molecules from a sample and discard the other molecules contained in the sample (DNA extraction). The second is to obtain the sequence of the molecules from the purified DNA (DNA sequencing). The final step and main focus of my Ph.D. is to transform informatically the output of the sequencing machine into full genomes (genome assembly).

0.1.2 DNA extraction

DNA extraction is a process used to isolate DNA molecules from cells. This involves breaking open the cells and, if necessary, the nuclei, using chemical, mechanical, or enzymatic methods. For example, detergents can be used to dissolve cell membranes, while liquid nitrogen freezing can be used to physically break open the cells. Proteases, which are enzymes that break down proteins, can also be used to remove proteins and other contaminants from the sample [27].

Once the cells and nuclei have been broken open and the contaminants removed, the DNA molecules can be isolated from the rest of the solution. This is often done by exploiting the chemical properties of DNA, such as its ability to precipitate out of solution when mixed with alcohol. Alternatively, DNA can be isolated by using its negative charge to make it bind to positively charged beads of silica, which can then be physically separated from the rest of the solution [27].

The first DNA extraction was published by Friedrich Miescher in 1871, where the isolated substance was called *nuclein* [28]. While the process of DNA extraction has been known for a long time, it still poses many practical challenges to biologists today. Different

types of cells can react differently to the same protocol, and molecules within the cells can interact with the chemicals used for extraction, leading to unexpected results. In practice, protocols often have to be adapted and optimized for the specific experiment and type of cells being used. I myself failed a fungi DNA extraction twice despite having followed scrupulously two different protocols.

0.1.3 DNA sequencing

The first DNA sequencing method appeared almost a century after the first isolation of DNA, in 1970 [29]. In 1977, Frederick Sanger published the first widely used method for DNA sequencing [30], for which he obtained a second Nobel prize. The method, known as Sanger sequencing, involves duplicating *in vitro* many fragments of the DNA to be sequenced using a DNA polymerase enzyme. Sanger's key innovation was to introduce modified fluorescent nucleotides into the solution, which are incorporated into the newly synthesised DNA sequence by the polymerase. The color of the fluorescence can be used to distinguish the four nucleotides and determine the DNA sequence.

Sanger sequencing was the prominent method of sequencing until the mid-2000s [31]. It was used to obtain the first genome of a virus in 1977 [32]. However, Sanger sequencing is limited in its throughput: all regions of a few hundred basepairs need to be separately amplified and sequenced. Because of this, it took almost twenty years to produce the first non-viral complete genome, *Hemophilus influenza* in 1995 [33]. The Human Genome Project was completed in 2001 [34], after 13 years of effort and approximately $3 billion [35, 36].

During the Human Genome Project, a central question arose regarding the order in which to sequence the DNA. Two competing strategies emerged [37, 38]. On one side, the main effort of the National Institutes of Health (NIH) methodically sequenced precise locations of the genome. On the other side, Craig Venter and the Celera company conducted shotgun sequencing, which involves sequencing random fragments of the genome. Shotgun sequencing results in a significant amount of redundant sequencing (in order to have the whole genome sequenced at least once) and requires extensive bioinformatics effort to reorder the fragments. However, it eliminates the need for laborious lab work to isolate specific regions of the genome. Both projects were eventually completed and published one day apart [39, 40], but shotgun sequencing emerged as the more efficient and popular strategy. With the exponential decrease in sequencing and computing costs, shotgun sequencing has become hegemonic.

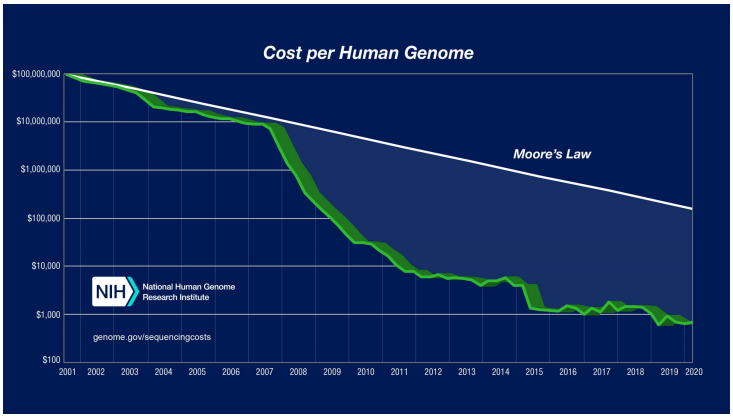


Figure 1 – Evolution of the estimated sequencing cost of a human genome. Source: NIH

Since then, the arrival of second generation sequencers has radically changed the field of genomics. The initially tremendous cost of sequencing has been brought down drastically, outrunning Moore's law by a large margin (see Figure 1). Many technologies emerged simultaneously, such as 454 pyrosequencing [41], Ion Torrent [42], SOLiD [43] and Solexa [44], which have in common the fact that they massively parallelize the sequencing process. Of these, the Solexa technology, which has been bought by Illumina and rebranded, has emerged as the most widely used second-generation sequencing method. This approach involves synthesising the reverse-complement strand of the sequenced DNA by the sequential addition of fluorescently tagged nucleotides. Though this idea bears resemblance with Sanger sequencing, the main breakthrough consisted in filming the fluorescence in real time and sequencing thousands of DNA strands in parallel. These technologies produce a massive amount of "reads" (i.e. fragments of DNA sequence) of relatively short length (typically between 100 and 500bp) and high accuracy (below 1% errors today).

Second generation sequencing divided the cost of sequencing a thousand fold in just five years between 2006 and 2011, democratising DNA sequencing (Fig. 1). However, the short length of the reads produced by these technologies imposed important limitations to biologists and bioinformaticians. Indeed, almost all genomes, and especially the human genome, contain many repeated sequences whose lengths exceed the lengths of the reads [45]. This entails that a single read can represent the sequence of several distinct positions in the genome. This hinders many genomic analyses, among which genome assembly (see section 0.1.4).

In 2011, a new generation of sequencing technology known as long-read or third-generation sequencing was introduced by the company Pacific Biosciences (PacBio). This

new technology, named single-molecule real-time sequencing (SMRT), was still based on fluorescent nucleotides but accelerated tremendously DNA extension. Concurrently, Oxford Nanopore Technologies (ONT) commercialized in 2014 a radically new sequencing method that involves measuring the electric resistance of nanoscopic pores as DNA molecules pass through, each base entailing a different resistance [46]. These technologies were a breakthrough, as they could produce reads two orders of magnitude longer than those produced by Illumina sequencers [47]. However, the initial versions of PacBio SMRT and ONT sequencing had a high error rate, up to 15%, greatly complicating downstream analyses. Improving the error rate has been the main focus of development of the two technologies since then.

Today, the error rate of third-generation sequencing is plummeting. PacBio released a high-fidelity technology (PacBio HiFi), capable of generating reads of 5kb with an error rate of less than 0.5% [48]. Oxford Nanopore has more recently introduced duplex reads, which they claim could provide reads of similar quality [49]. In both cases the drastic improvement in quality relies on sequencing several time the same sequence and generating a high-quality consensus. Both technologies are still in their infancy and pose difficulties regarding their cost and reliability. Nevertheless, we can expect in the following years a shift towards these new accurate long reads.

0.1.4 Genome assembly

(Meta)genome assembly aims at reconstituting the genome(s) of the organism(s) present from the set of randomly sequenced reads. As reads are extract of genomes, reconstituting the genomes can be seen as a one-dimensional puzzle, where the reads are the pieces and several pieces can originate from the same position in the genome. A major difficulty is that a given set of reads may ambiguously be reconstructed into several distinct genomes (see Figure 2), among which only one is the actual sequenced genome. Therefore, genome assemblers generally output their results as "assembly graphs," representing a set of potential genomes. In these graphs, nodes represent contiguous nucleotide sequences (contigs) and the possible genome sequences are paths through the graph.

Throughout this thesis, we will encounter various methods that ignore the edges between the contigs and treat the contigs as disconnected entities. This practice has historical roots; although assemblers naturally produce graphs (see below), many have traditionally outputted only the contigs for simplicity, discarding the connecting edges. Consequently, there is sometimes ambiguity regarding whether assemblies should be considered as graphs

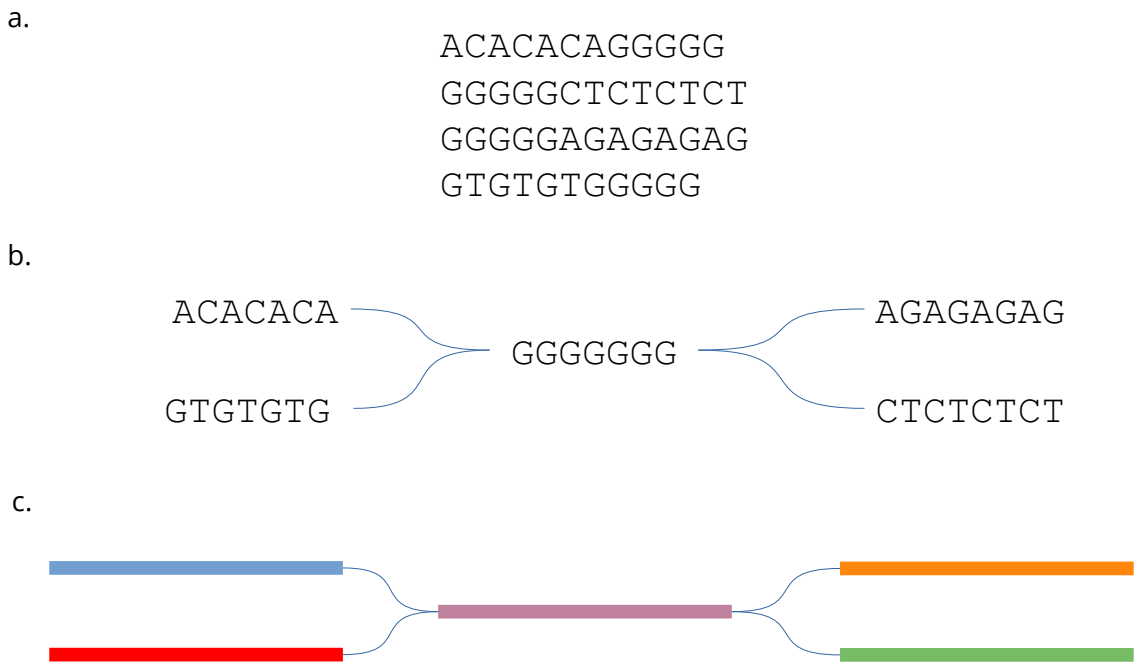


Figure 2 – Assembly graphs. a) Sequencing reads. b) An assembly graph which represent several potential genomes. Here, the genome could be {`ACACACAGGGGGGGAGAGAGAG`, `GTGTGTGGGGGGGGCTCTCTCT`} or {`ACACACAGGGGGGGCTCTCTCT`, `GTGTGTGGGGGGGGAGAGAGAG`} or the combination of both. It is impossible to disambiguate from the reads. c) Because sequences are generally very long in assembly graphs, a convention is to replace them by rectangles of color.

or merely as sets of contigs. Following the trend in the community, this thesis will consider assemblies as graphs.

Perfect assemblies represent each chromosome of the sample as a single sequence. Good assemblies are *complete*, i.e. represent all the genome(s) of the sample, *accurate*, i.e. do not represent non-existing contigs, and *contiguous*, i.e. with the longest possible contigs and the simplest possible graph (discussed further in Chapter 1).

Described assemblers can be categorized in three groups of algorithms: greedy, De Bruijn Graphs (DBG) and Overlap-Layout-Consensus (OLC).

Greedy assemblers Greedy assemblers were the first type of genome assembly algorithm to be developed and were used to assemble the first bacterial genomes [50]. These assemblers are based on the assumption that repeated sequences are unlikely to occur in

a genome. Therefore, if two reads contain a long common sequence, or overlap, they must originate from the same position in the genome and can be merged to form a longer "super-read." Greedy assemblers iteratively extend these super-reads with the longest available overlaps until no overlaps are left.

It turns out that the assumption behind greedy assemblers - that repeated sequences are unlikely to occur in a genome - seemed true for the first viral and bacterial genomes analysed, but is completely inaccurate for many genomes, including genomes of eukaryotes. As a consequence, two reads sharing a large overlap may not come from the same location at all, and greedy assemblers generally fail to produce accurate genomes.

De Bruijn Graph assemblers De Bruijn graphs were described for the first time by Camille Flye Sainte-Marie in 1894 [51], but bear the name of Nicolaas Govert de Bruijn, who described them in 1946 [52]. Nodes of De Bruijn graphs represent all existing sequences of length k over a given alphabet (A,C,G,T for us). In the context of genome assembly, we will call these sequences of length k *k-mers*. In DBGs, there is a directed edge between k-mer a and k-mer b if and only if the $k-1$ suffix of k-mer a is equal to the $k-1$ prefix of k-mer b. DBG assemblers do not build the full De Bruijn graph but only a subgraph containing only the k-mers present in the reads. Additionally, k-mers that are present less than c times in the reads are discarded to keep only k-mers that are confidently present in the genome. For convenience, the obtained subgraph will also be called "De Bruijn Graph", even though it is not, mathematically. Under the assumption that all k-mers of the genome are present at least c times in the reads, this graph contains the genome. The graph then gets simplified to improve the contiguity of the assembly. Typically, this involves trimming short dead ends and collapsing "bubbles" of the graph, which have been empirically identified as typical artefacts of sequencing errors. See Figure 3 for an example.

In the past two decades, highly efficient DBG assemblers have been developed, leading to their widespread use and popularity in the field of genomics. Examples of such assemblers that are commonly used today include SPAdes [53], MEGAHIT [54], minia [55], and mDBG [56]. However, the main limitation of DBG assemblers is their reliance on the assumption that all k-mers of the genome are present at least c times in the reads. This assumption can be difficult to meet when working with highly erroneous reads, as most of the sequenced k-mers may contain errors. As a result, DBG assemblers fell slightly

Reads	k-mers
GCATAG	GCAT, CATA, ATAG
CAGCATGGACGGAT	CAGC, AGCA, GCAT, CATG, ATGG, TGGA, GGAC, GACG, ACGG, CGGA, GGAT
CAGCATGGACGGATTA	CAGC, AGCA, GCAT, CATG, ATGG, TGGA, GGAC, GACG, ACGG, CGGA, GGAT, GATT, ATTA
CAGCATGG	CAGC, AGCA, GCAT, CATG, ATGG
CGGATTA	CGGA, GGAT, GATT, ATTA
CAGCTTGGACGG	CAGC, AGCT, GCTT, CTTG, TTGG, TGGA, GGAC, GACG, ACGG
GCATGGACGGATTA	GCAT, CATG, ATGG, TGGA, GGAC, GACG, ACGG, CGGA, GGAT, GATT, ATTA
CAGCATAG	CAGC, AGCA, GCAT, CATA, ATAG

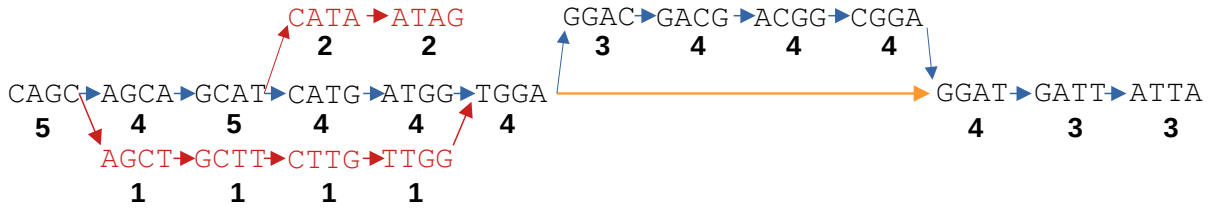


Figure 3 – De Bruijn Graph assembly with k=4. Depicted is the De Bruijn subgraph containing strictly the k-mers seen in the reads. The number associated with each node represent the number of times it is seen in the reads. Assemblers typically discard all the k-mers seen below a threshold c and short dead ends in the graph as sequencing errors (in red here). Most assemblers also choose to collapse bubbles, which would mean here discarding the orange arrow.

out of favor with the rise of long, error-prone reads. However, as the error rate of long reads is decreasing, DBG assemblers may become once again the prime choice for genome assembly.

Overlap-Layout-Consensus assemblers The Overlap-Layout-Consensus (OLC) approach for genome assembly was first introduced by Rodger Staden in 1979 [57]. As the name suggest, this method follows a three-step strategy. In the overlap step, the algorithm identifies overlaps between all the reads that meet a certain quality threshold and constructs a directed graph. The nodes in the graph represent the reads, and the edges represent the overlaps between them. The layout step removes any redundant edges in the graph that can be inferred from other edges. The graph can also be simplified by trimming dead ends and collapsing bubbles. Finally, the consensus step transforms the overlap graph into an assembly graph by merging the reads into contigs (longer stretches

of DNA sequence) by selecting the most consensual base at each position among all the reads. See Figure 4 for an example.

OLC assemblers are generally less computationally efficient than DBG assemblers because they require the expensive computation of inexact overlaps between reads. However, OLC assemblers are well-suited for handling high error rates and low sequencing depth. The first human genome was assembled using the OLC assembler Celera [59], and OLC assemblers have become increasingly popular with the advent of long-read sequencing technologies. Some of the most widely used OLC assemblers today include Raven [60], miniasm [61], Canu [62], and hifiasm [63].

0.1.5 Finishing an assembly

The ideal genome assembly contains exactly one contig per chromosome, but this is rarely achieved in practice. The main obstacle is not the assembly method itself, but the information contained in the sequencing reads. If a genome contains a repeated region that is longer than the length of the reads, it becomes impossible for the assembler to determine which sequence before the repeated region corresponds to which sequence after the repeated region (see Figure 2).

The introduction of long reads has mitigated this problem. While obtaining perfectly contiguous assemblies was possible only for viruses a few years ago [32], single-contig circular bacterial genomes are now routinely assembled [64]. Yet reads have not improved enough in length to provide contiguous eukaryote or metagenomic assemblies, for two reasons. The first is that eukaryote genomes can contain very long repeats on a single chromosome, especially in telomeric and centrometric regions [65]. The second is the presence of several *haplotypes* in these (meta)genomes. The term haplotype is a contraction of "haploid genotype" [66] and designates a single set of non-homologous chromosomes in a sample. For example, a single bacterial genome contains one haplotype, a human genome two (the maternal and the paternal) and a mix of ten bacterial strains ten. Different haplotypes can share long identical regions, making it impossible to obtain a perfectly contiguous final assembly.

If the length of sequencing reads is still insufficient to resolve repetitive regions in a (meta)genome, other technologies have been developed to provide additional information about the long-range structure of the genome. These technologies, such as mate pairs, genetic maps, optical mapping, linked reads and proximity ligation cannot be used to assemble a genome on their own, but can be used in conjunction with sequencing reads to

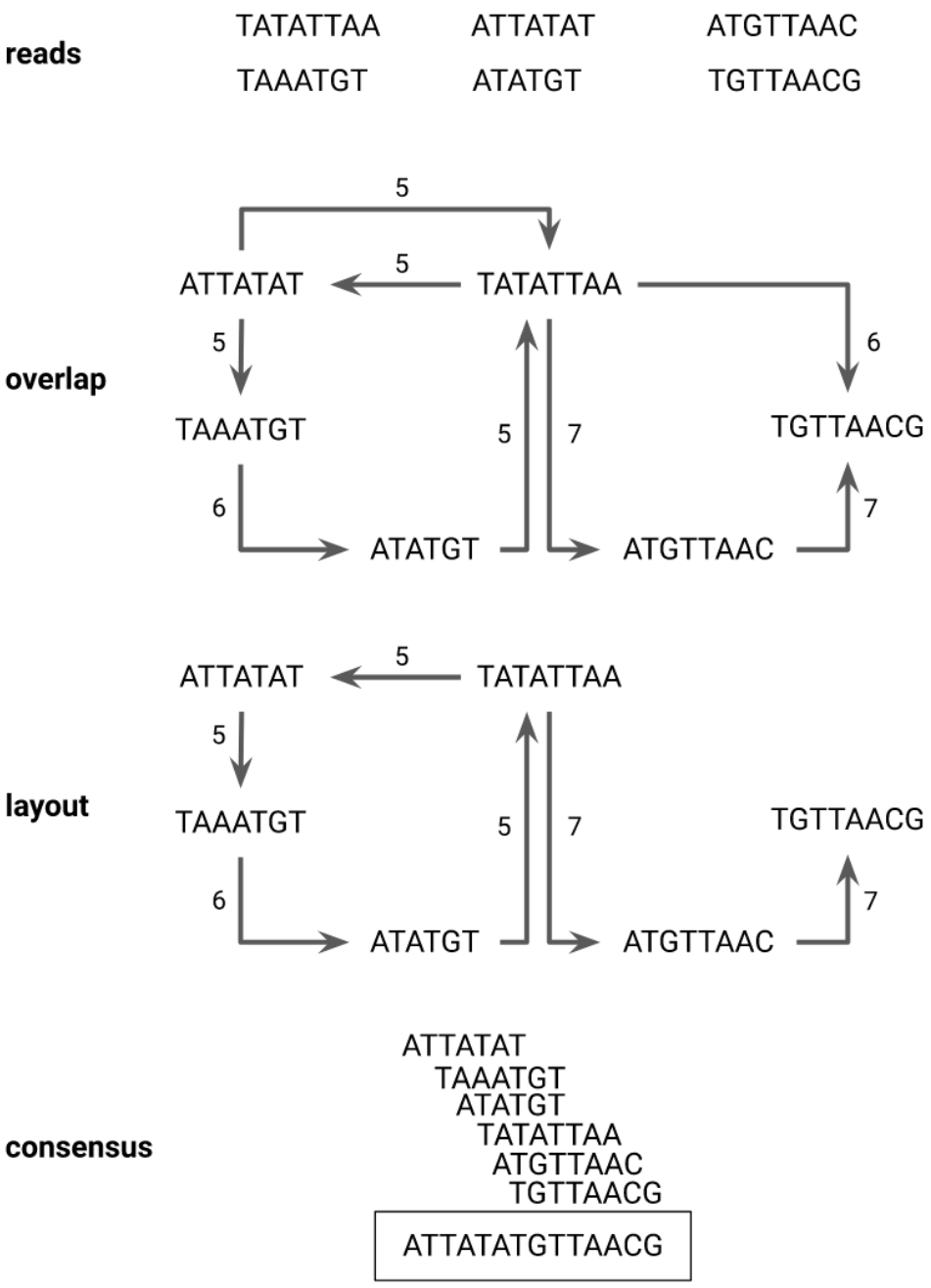


Figure 4 – Overlap-Layout-Consensus assembly as illustrated by Nadège Guiglielmoni [58]. The overlap graph is built with all overlaps of at least 5 bases with a tolerance of 1 mismatch. The layout step removes overlaps that can be inferred from other overlaps. The consensus step then optionnaly simplifies the graph and generates the contigs. The edge between reads "TATATTAA" and "ATTATAT" was here discarded during the consensus step by the assembler to simplify the graph.

improve the contiguity of the assembly [67]. Interestingly, long reads themselves sometimes fill this purpose, when their quality is not judged sufficient to assemble a genome - for example when ultra-long nanopore reads are used only to improve a high-fidelity reads assembly (e.g. [68]). The most widely used technology, proximity ligation, inventories pairs of sequences which touch physically in the nucleus [69]. Leveraging the fact that sequences that are far on the linear strand are less likely to physically touch than sequences close on the linear strand, this data can be exploited to reconstruct entire chromosomes [70].

These long range data are sometimes directly integrated in assembly software [63, 71] but more often are used in specific programs named *scaffolders*. Scaffolders order and orient contigs to produce new assemblies composed of long "scaffolds", which are series of contigs stitched together, with unknown sequences between them. We will focus exclusively in this thesis on improving the contiguity of assemblies with proximity ligation data and long reads.

0.2 Challenges of metagenome assembly and outline of the thesis

0.2.1 Metagenome assembly

Historically, researchers have first focused on retrieving the genome of single organisms. A reason for this focus is that, at the time massive sequencing developed, the central role of microbiomes in ecosystems was not as well established as today.

As a consequence, genome assemblers have not historically been designed to assemble a collection of genomes but single organisms only. When they were applied on metagenomic samples, it became evident that they would not work out of the box. Indeed, single-organisms assemblers can safely assume that all regions of the genome are approximately sequenced in the same amount. Because of this, they can confidently reject rare sequences as sequencing errors and confidently decide if a region is repeated or unique. This assumption crumbles when strains are present in different amounts in a metagenome. A rare sequence might then represent either a rare strain or a frequent sequencing error. A shallowly covered contig, constructed with few reads, might exist in ten different strains, while an abundantly covered contig, constructed with many reads, might be unique in the metagenome.

This led to the design of specialized metagenome assembly software, often building

on previously existing single-organisms software - metaSPAdes [72] based on SPAdes, metaMDBG [73] based on MDBG, hifiasm-meta [74] based on hifiasm, metaFlye [75] based on Flye... In practice, adapting an assembler to perform metagenome assembly generally means being more careful in the overlap graph or De Bruijn graph simplification. While these assemblers are quite successful when dealing with a diverse collection of genomes, they all struggle on one difficulty: the presence of several highly similar strains (or haplotypes) in a sample.

Metagenomic samples can contain multiple strains of the same species, with highly similar genomes but distinct functions. It is thus essential to differentiate between conspecific strains in order to understand the behavior of a microbiome, including its pathogenicity. A well-known example is the pathogenic and non-pathogenic strains of the bacteria *Escherichia coli* that can coexist in the human gut. In fact, we know that the very first isolation of *E. coli* by Theodor Escherich in 1886 was from a multi-strain gut sample: although he correctly identified *E. coli* as the cause of many childhood diarrhea cases, he actually isolated and accused a non-pathogenic strain of his sample [76, 77]. Though this is amusing in retrospective, an assembly of the same sample sequenced with ONT today would make the same mistake! Assemblers are unable to distinguish close strains, and would produce an assembly of the most abundant strain only.

In Chapter 1, I will define what constitutes a high-quality assembly and present an approach to enhance the quality of assembly graphs by correcting structural errors introduced by assemblers, implemented in a software named GenomeTailor. While this approach can be applied to any assembly, it is particularly valuable for assembling highly similar strains, which often contain numerous misassemblies.

0.2.2 Assembling haplotypes with erroneous reads

Assembly tools designed for error-prone long reads, whether they are based on Overlap-Layout-Consensus (OLC) (e.g., [60, 61]) or approximate De Bruijn Graphs (e.g., [78, 79]), begin by identifying overlaps between all reads. When dealing with erroneous reads, these tools must allow for some discrepancies in the overlaps; otherwise, no overlaps would be detected. As a result, reads from highly similar haplotypes are often considered overlapping because the biological differences between them are obscured by sequencing errors. This leads to the collapse and merging of haplotypes into a single consensus sequence, typically representing the most abundant haplotype.

Various solutions have been developed that cater to diploid or even polyploid cases,

e.g. [80–83]. These methods rely on the assumption that there is a known number of haplotypes, often two, with similar abundances in the data. However, this assumption does not translate well to metagenomics. In a metagenome, the number of haplotypes for a given sequence is not known beforehand and varies across the different species of the sample. Additionally, the abundance of these haplotypes is not balanced. Some haplotypes may be very rare, making them difficult to detect and assemble accurately.

Many metagenomic assemblers choose to simplify the problem by assuming that there are no highly similar strains to assemble in a sample - "metaSPAdes focuses on reconstructing a consensus backbone of a strain mixture" [72] ; metaMDBG "simplifies graph regions [...] caused by strain variability"[73]. Other assemblers do not mention the difficulty (such as MEGAHIT [54]). Lastly, some assemblers try assemble strains separately (such as metaFlye [75]), but no assembler show a convincing example of highly similar strain separation.

In response to these challenges, several software tools have been developed to separate strains from a draft assembly and a set of reads. Notable examples include Strainberry [84], iGDA [85], stRainy [86], and for viral haplotypes Strainline [87] and HaploDMF [88]. However, as we will discuss in Chapter 2, these tools have limitations in handling the complexity of metagenomes. To overcome these limitations, I propose a two-step strategy for the precise assembly of complex metagenomes. In Chapter 2, I will introduce a novel statistical approach to detect single-nucleotide differences between strains (Single Nucleotide Polymorphisms - SNPs), which has been successfully implemented as a software named HairSplitter and tested on mixtures of up to ten strains and unbalanced datasets.

0.2.3 Assembling haplotypes with highly precise reads

High-fidelity (HiFi) read assemblers can significantly outperform their erroneous reads counterparts in strain assembly due to the much lower error rate of the reads, making it easier to distinguish noise from signal. However, we will see in Chapter 3 that even hifiasm [63], the most widely used HiFi assemblers, still faces difficulties when assembling highly similar strains and could benefit from significant acceleration. Inspired by MDBG [56], an extremely fast assembly method, I will propose in Chapter 3 a new sketching method, implemented in the proof-of-concept assembler Alice, that excels at distinguishing strains while being significantly faster than state-of-the-art assemblers.

0.2.4 Finishing a multi-haplotype assembly

Numerous scaffolders have been developed over the years, including Lachesis [89], 3D-dna [90], and YaHS [91], although many are limited to scaffolding haploid genomes. More recently, tools such as ALLHiC [92], Greenhill [93], and hapHiC [94] have been introduced to handle diploid and polyploid assemblies. The work presented in Chapter 4 began with the observation that these scaffolders process assemblies as unconnected sets of contigs and do not use assembly graphs during the scaffolding process. In the context of multi-haplotype assemblies, where highly similar contigs are present and the assembly is often fragmented, we hypothesized that the topology of the assembly graph could provide valuable information to improve scaffolding. I will introduce the concept of *untangling* an assembly graph and implement it in a software tool named GraphUnzip. I will demonstrate that combining untangling with classical scaffolding methods improves the quality of the final scaffolds.

In metagenomics, proximity ligation has been employed to bin contigs from different species [95, 96]. However, to the best of my knowledge, there is currently no metagenomic scaffolder capable of dealing with highly similar strains. This problem is complex, and as we will see, GraphUnzip does not yet effectively extend to metagenomes. Nevertheless, I believe that future metagenomic scaffolders designed to handle highly similar strains will untangle assembly graphs.

Chapter 1

Evaluating and improving assembly graphs

Abstract: This chapter explores the characteristics of a high-quality assembly graph and defines the concepts of *correctness*, *completeness*, and *contiguity* to evaluate assembly graphs. The motivation for this chapter stems from the observation that metagenomic assembly graphs are often incomplete and incorrect. We introduce a new method to improve assembly graphs using sequencing reads alone, implemented in a software named GenomeTailor. GenomeTailor focuses on correcting structural (large-scale) errors, leaving base-level errors to be addressed by other tools such as HairSplitter (Chapter 2). Tested on both simulated and real data, GenomeTailor significantly improves the quality of metagenomic assemblies. Though it has been developed for metagenomics, it can be used on any assembly and successfully improved a yeast assembly.

1.1 Evaluating assembly graphs

Assemblers produce assembly graphs, where each path through the graph represents a potential sequence of the sequenced genome(s). Among these paths, some correspond to the actual sequences of the sequenced genome(s). Assemblies are typically evaluated based on their *correctness*, *completeness*, and *contiguity*, using concepts originally developed for assemblies represented as sets of contigs. We will briefly discuss these three notions in the context of assembly graphs.

An assembly is deemed *correct* if all contigs in the graph are present in the genome(s) [97–101]. Conversely, an incorrect contig is one that does not exist in the genome(s). An assembly is generally considered *complete* if the genome(s) can be fully covered by the contigs [102] or if all k-mers of the genome(s) are present in the assembly [103, 104]. However, we believe these definitions of correctness and completeness are insufficient to assess the completeness of an assembly graph – so-called "correct and complete" assemblies may still fail to accurately represent the actual genome(s). For example, see Figure 1.1. To address this, we propose that an assembly graph be considered *complete* if all chromosomes

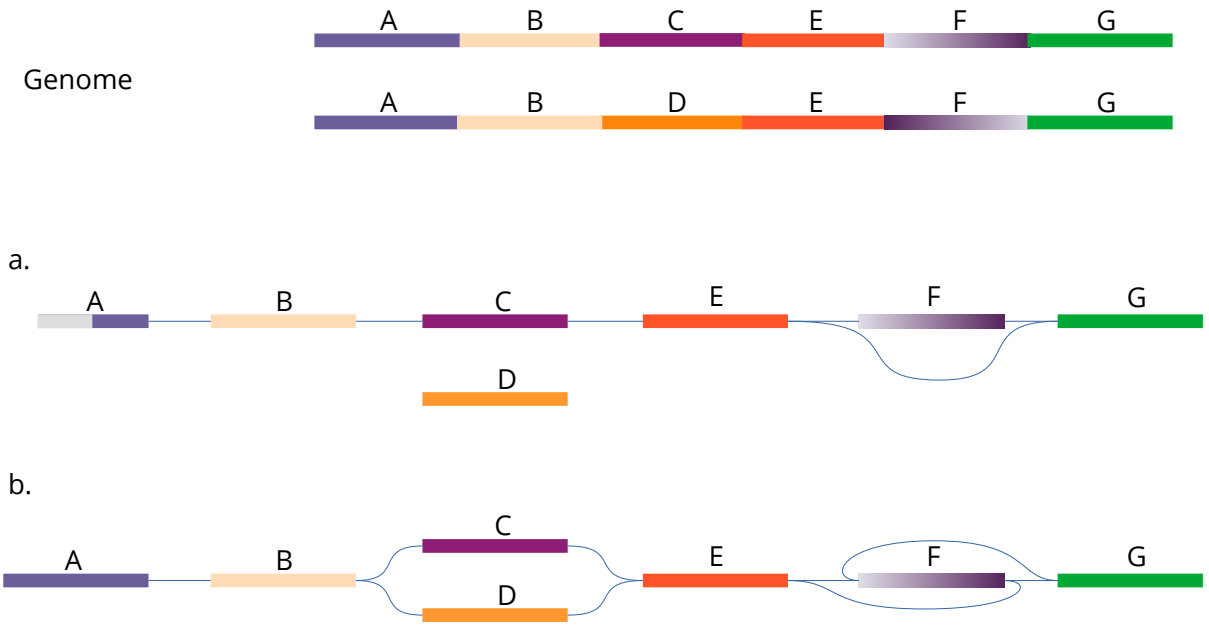


Figure 1.1 – Two different assembly graphs aiming to represent the same genome. The genome consists of two haplotypes, with two heterozygous regions (C and D) and one inverted region, F. Assembly *a* contains several issues. First, contig A is incorrect, as the sequence (gray and blue) is not present in the original genome. Although the other contigs are correct, the graph can be considered incomplete: no contiguous path through the graph can include contig D or correctly represent the inversion of F. On the contrary, *b* is a complete assembly graph.

are represented by the assembly graph (i.e. are a single path in the graph).

The final feature evaluated to assess the quality of an assembly graph is its contiguity. Traditionally, contiguity is measured using the distribution of contig lengths in the assembly, with metrics such as the N50. The N50 is the maximum size of contigs such that more than half of the assembly is contained in contigs of this size or larger. Having fewer, longer contigs is generally preferred over having many short contigs, as it provides a more precise representation of the sequenced genomes. However, in the context of assembly graphs, this definition of contiguity is unsatisfactory. As illustrated in Figure 1.2b and c, two assembly graphs can have very different N50 values yet be completely equivalent.

To address this, we propose a broader definition of contiguity within the context of assembly graphs: a contiguous assembly graph represents a small diversity of potential genomes. Measuring this diversity with a simple metric is challenging, particularly when

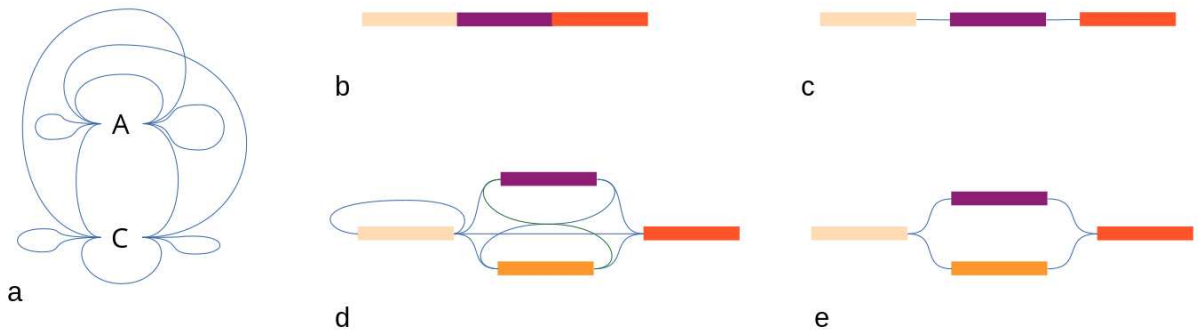


Figure 1.2 – Assembly graphs which illustrate the concept of contiguity we propose. Rectangles of color represent long sequences. Assembly a is a correct and complete assembly of *any* genomic dataset, but is impractical because it represents a huge number of potential genomes. Despite having a different distribution of length of contigs, assemblies b and c are obviously equivalent, thus should in our opinion be considered as having the same contiguity. Reciprocally, even though their contigs are of the same length, we propose that assembly e be considered more contiguous than assembly d because it represents a smaller diversity of potential genomes.

the completeness of the graph is uncertain. Consequently, despite acknowledging that N50 is not an ideal metric for assembly graphs, we will still use N50 as a measure of contiguity due to its simplicity.

1.2 Improving metagenome assemblies

Metagenomic assemblers typically make two main types of errors when highly similar strains are present in a sample. We have addressed both of these problems separately. The first problem involves structural errors. When haplotypes contain structural variations, assemblers often retain only one allele, either completely discarding the alternative haplotype or detaching it from the assembly graph. In these cases, the structural variation is lost in the assembly graph. This problem can also occur in multiploid genomes or even in repeated regions of haploid genomes [105]. In this chapter, we will propose a solution to this problem. The second problem is that assemblers generally collapse highly similar sequences into a single sequence, discarding SNPs and small indels. This second problem will be addressed in Chapter 2.

Numerous methods have been developed to identify structural errors, or misassemblies,

in genome assemblies using long reads. Notable tools for long reads include Inspector [98], CRAQ [99], GAEP [100], and klumpy [101]. These tools inspect the alignment of reads on the final assembly to detect patterns characteristic of assembly errors. However, they primarily focus on improving the correctness of individual contigs and not the completeness of the assembly graph. Inspector offers error correction, but it does so through local reassembly without considering the other contigs in the graph. Additionally, Inspector, CRAQ, and GAEP are designed to work exclusively with haploid or diploid genomes.

In this chapter, we introduce GenomeTailor, a software designed to work natively with assembly graphs to enhance both the correctness and completeness of these graphs. The concept behind GenomeTailor is to take an initial assembly and perform a series of cuts, stitches, and gap-filling operations to produce corrected and completed assemblies, "tailored" to the sequencing reads. GenomeTailor focuses on correcting large structural errors, while leaving base-level polishing to specialized polishing tools.

1.3 GenomeTailor

The GenomeTailor procedure is illustrated Figure 1.3.

Blunting the graph Later in the process, GenomeTailor aligns reads to the graph using minigraph [106]. However, minigraph does not handle overlaps in assembly graphs. As an initial step, the assembly graph is therefore "blunted." We attempted to use specialised tools such as GetBlunted [107] and gimbricate/seqwish [108], but both were challenging to install and resulted in numerous extremely short contigs. Consequently, GenomeTailor introduces a native python script to remove overlaps in a GFA graph. This module iteratively and greedily deletes and detaches the ends of contigs to eliminate overlaps, ensuring that the graph remains valid at all times (i.e., a graph where overlaps are shorter than the contigs). This module is slightly less robust than specialized software and may fail to completely blunt the graph in rare cases. In practice, it successfully removed all overlaps in the miniasm assemblies we tested. To deal with all cases, the module cuts links with remaining overlaps.

Assembling missing sequences GenomeTailor aligns all reads to the blunted assembly graph using minigraph [106] with default parameters, retaining only primary alignments. Since all reads are assumed to be fragments of the sequenced genome, they are

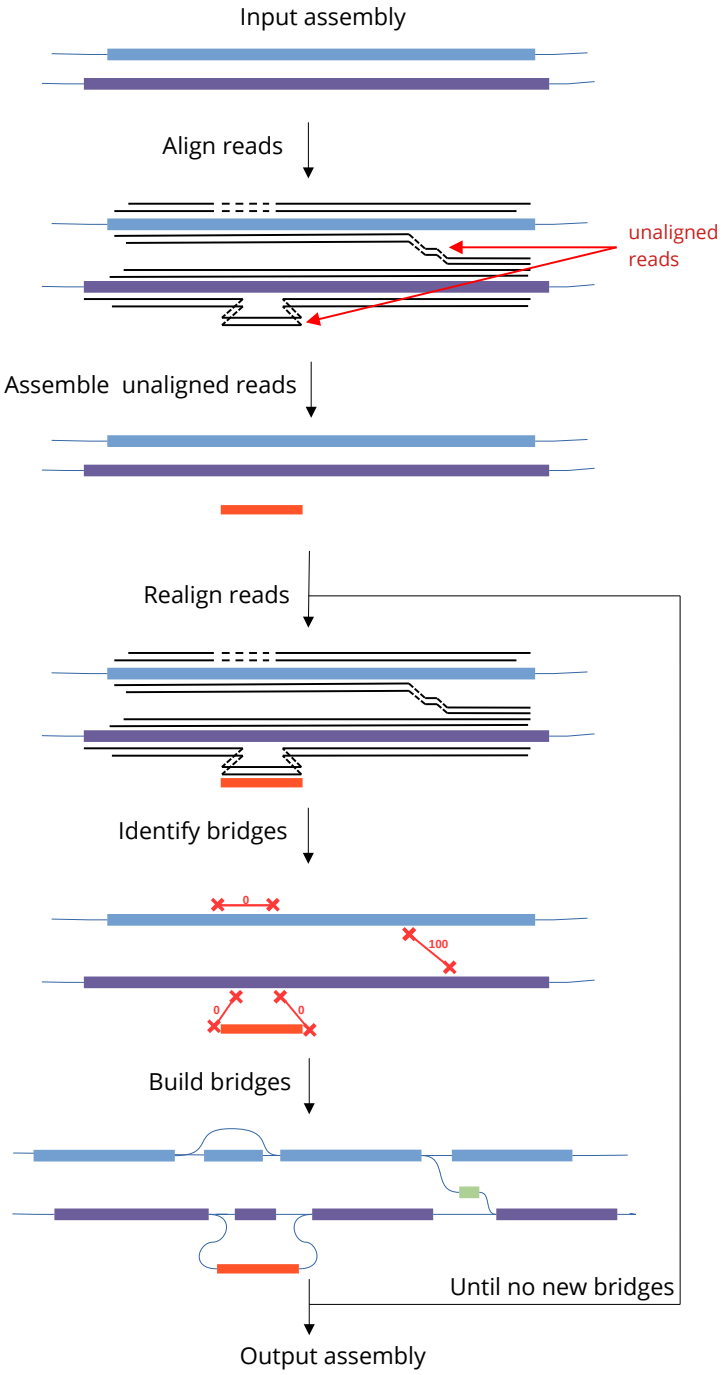


Figure 1.3 – The GenomeTailor procedure. Black lines represent reads. Red crosses represent the end of the bridges and the associated number indicates the length of the bridge. In this examples, bridges are considered valid if they are supported by at least two reads.

expected to align entirely on the graph. GenomeTailor identifies all unaligned regions of reads longer than one kilobase and reassembles them using Raven [60], aiming to recover any non-assembled regions of the genomes. The new contigs are added to the original assembly and reads that were not completely aligned in the first iteration are re-aligned to the new assembly graph. As a side effect, the alignment of the reads on the assembly allows GenomeTailor to recompute the coverage of the contigs.

Identifying misassemblies GenomeTailor operates on the principle that, in a correctly assembled genome, all reads should align from end to end on the assembly graph, except for chimeric reads and contamination. Therefore, locations where reads abruptly stop aligning often pinpoint misassemblies.

Next, GenomeTailor identifies misaligned reads, which are reads that do not align from end to end (with a 1 kbp tolerance for the ends) in a single run on the graph. It then analyzes each misaligned read to categorize the misassemblies into three types. *False simplex* reads are a typical artefact of Nanopore sequencing, where a read and its reverse-complement are concatenated: these misaligned reads do not reflect a misassemblies and are not considered further. *Bridges* are errors where a read stops aligning, jumps to another location, and starts aligning again. They are characterized by two loci (the ends of the bridge) and an unaligned sequence in between (the bridge itself). *Piers* are errors where a read stops aligning, and the remaining portion of the read does not align anywhere else on the assembly graph. They are characterized by a loci and an unaligned sequence.

After identifying all bridges and piers, GenomeTailor aggregates them into solid bridges and solid piers based on user-defined parameters. Two piers are considered equivalent and aggregated in a solid bridge if their loci are within the aggregative distance (1kbp by default). Two bridges are considered equivalent and aggregated in a solid bridge if both their loci are within the aggregative distance and their unaligned sequences are of coherent length, allowing for a leeway of the aggregative length.

Only solid bridges and piers supported by a minimum number of reads (default: 5) are retained. Those with fewer supporting reads are considered potential artifacts due to chimeric reads or alignment errors and are discarded.

Building bridges For each solid bridge, the following steps are performed:

— Polishing: one of the bridge sequences is polished with the others using Racon [109].

— Local Realignment: the polished sequence is locally realigned near the bridge ends

to precisely determine the ends, aiming to create the shortest possible bridge.
— Insertion in the graph: if the bridge length is 0, a link is added to the graph; if the bridge is longer, a new contig is created based on the polished sequence and inserted into the graph.

For solid piers, a similar procedure was followed, but in real-data tests, it often resulted in dead ends in the assembly that did not appear to correspond to reliable genome sequences. We attribute this to a combination of difficult-to-sequence regions, noisy reads and suboptimal aligner parametrisation and would merit further investigation.

Iterative correction of the graph GenomeTailor iteratively aligns reads, build bridges and realigns reads until no bridges are found. In practice, reads that were already aligned end-to-end do not need to be realigned between two iterations.

1.4 Results

1.4.1 Datasets

We evaluated GenomeTailor using both real and simulated datasets. Instead of artificially introducing errors into correct assemblies, which would not accurately represent the unknown error patterns of different assemblers, we opted to start with sequencing reads from three datasets provided in [110]. The first dataset consists of Nanopore reads simulated from a mixture of 10 strains of *Escherichia coli*. The other two datasets are sequencing of the Zymobiomics Gut Microbiome Standard mock community generated using two different generations of Nanopore flowcells, R9.4.1 (Q9 kit) and R10.4.1 (Q20+ kit). The mock community contains 5 strains of *Escherichia coli*. We assembled these datasets using Flye [78] and miniasm [61]. We chose to test two different assemblers, as different assemblers tend to produce different types of assembly errors. Miniasm could not assemble the Q9 sequencing because it filled the memory of the server on which it ran. We then ran GenomeTailor on these assemblies and used our knowledge of the true genomes to assess its performance.

1.4.2 Metrics

A well-constructed assembly graph should be correct (i.e., no contigs represent non-existing sequences), complete (i.e., chromosomes are ideally represented as a single path

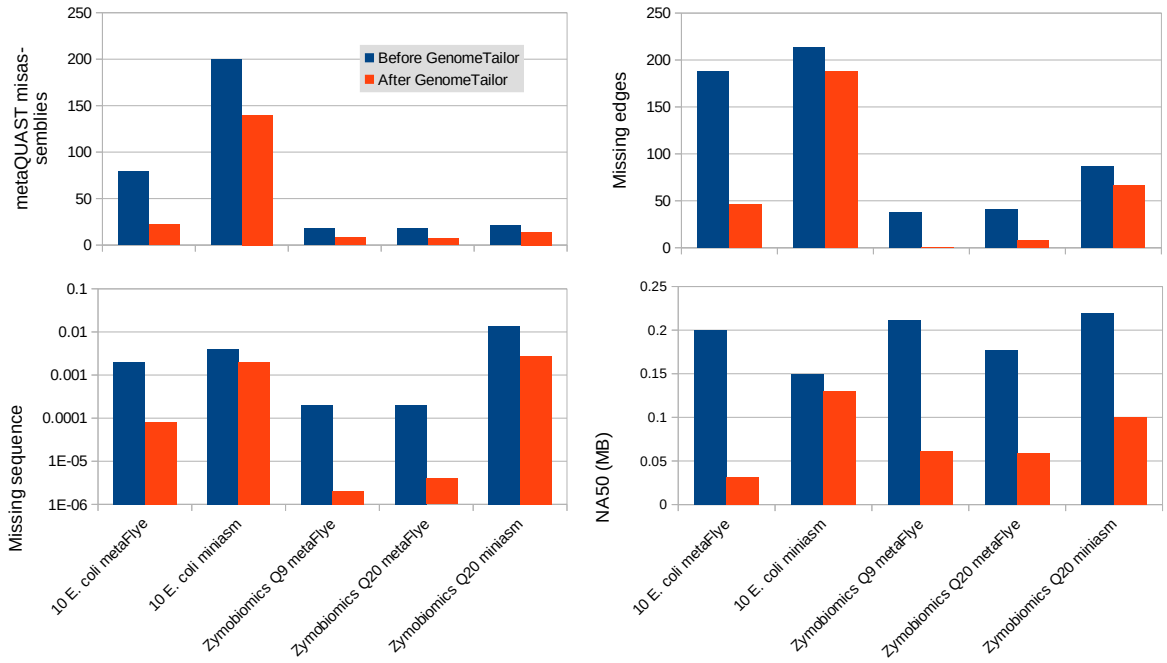


Figure 1.4 – Metrics of the different assemblies before and after GenomeTailor. Note that the scale measuring the proportion of missing sequences is logarithmic.

through the graph), and contiguous (i.e., the assembly graph represents as few potential genomes as possible). To assess correctness, we use the misassemblies metric from metaQUAST [97]. For completeness, we align the solution genomes to the assembly graph using minigraph. From this alignment, we define and measure two metrics: *missing sequence*, which is the proportion of the reference sequence that does not align on the graph (ideally near 0), and *missing edges*, which is the number of non-contiguous alignments of the reference on the graph minus one (ideally near 0). As explained above, contiguity is evaluated using the classical NA50 metric, which indicates the largest size of contigs such that more than half the assembly is contained in non-misassembled contigs of this size or larger.

1.4.3 Benchmark results

The metrics before and after applying GenomeTailor on the five assemblies are presented Figure 1.4.

GenomeTailor decreases significantly the number of misassemblies and the edge com-

pleteness in the Flye assemblies, and reconstructs almost all missing sequences. This comes at the price of many contigs being cut in several chunks, hence a significantly lower NA50. Compared to the Flye assemblies, GenomeTailor edited less the miniasm assemblies, with less impact on correctness, completeness and contiguity.

The sometimes numerous remaining metaQUAST misassemblies (140 in the simulated 10 *Escherichia* experiment) are mostly translocations between strains. These misassemblies occur when the sequence of a contig is composed of the sequences of two different strains. Most often, the misassembly results from the collapse of several strains together, where different strains dominate the consensus of the final sequences at different locations. As long as the reads from all strains align on the contig, GenomeTailor considers that there is no structural error and does not attempt to reconstruct the strains. These misassemblies can then be solved using HairSplitter (see Chapter 2).

Although the number of missing edges decreased in all cases, a careful examination of the results reveals a flaw in the current GenomeTailor algorithm. When certain regions of the genome are duplicated in the assembly, all reads can map from end to end on the assembly, yet the assembly may still be incomplete. This issue is illustrated in Figure 1.5. Addressing this problem is non-trivial, as it involves merging contigs that the assembler initially chose not to merge. We leave this challenge for future development of GenomeTailor.

1.4.4 Completing a *Debaryomyces hansenii* assembly

GenomeTailor was developed for multiple haplotype assembly but can be used to improve the completeness of any assembly. It was applied to an assembly of *Debaryomyces hansenii*, a yeast sampled near Chernobyl and sequenced using Nanopore technology during an EMBO training course. We wish to credit Eugene Tukalenko, Anton Lavrinienko, Janne Koskimäki and the Academy of Finland for the yet-unpublished data. Although the initial Flye assembly appeared highly contiguous and seemingly complete (>99% BUSCO completeness), a closer examination revealed a significant issue. A search for DNA encoding ribosomal RNA using barrnap [111] showed that the assembly contained almost no ribosomal DNA. Biologically, this indicates that the assembled genome was not functional. This is a common problem in eukaryotic assemblies, as ribosomal DNA consists of highly repetitive regions that often pose challenges to assemblers [112].

Consequently, we ran GenomeTailor on the assembly to attempt to recover the missing ribosomal DNA. The initial reassembly step using Raven did not recover the missing

Figure 1.5 – An incomplete graph composed of two contigs is produced. The underlined sequences highlight a sequence of the genome that is present on both contigs. Although a link is missing between the two contigs, GenomeTailor is incapable of completing it, as all reads align from end to end on the assembly.

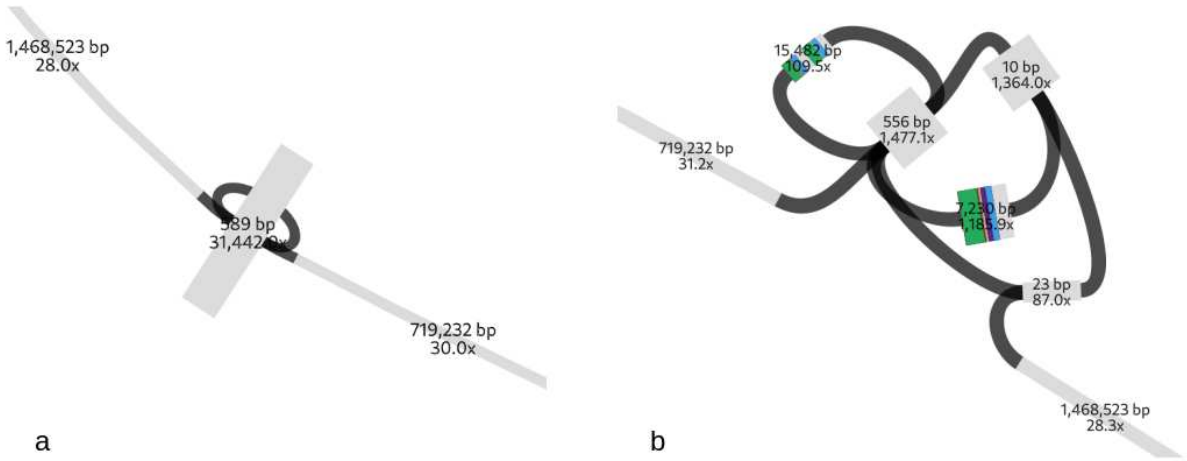


Figure 1.6 – Screenshot of the same region of the *Debaryomyces hansenii* assembly *a* before and *b* after GenomeTailor. The length and coverage of the nodes are indicated. Colourised regions correspond to ribosomal genes, as identified by barrnap. Visualised with Bandage [113].

DNA, as Raven, like Flye, struggles to assemble ribosomal DNA de novo. However, the iterative bridge-building procedure integrated several contigs into the assembly, within which barrnap detected complete 18S, 28S, and 5.8S ribosomal genes. Figure 1.6 visually illustrates the integrated contigs. GenomeTailor not only incorporated the ribosomal genes but also revealed the presence of three slightly different repeats of these genes. The only remaining challenge is to determine the order of the repeats, but the length of the reads did not allow for the complete resolution of the region.

1.5 Discussion

GenomeTailor is a software tool designed to complete assembly graphs. While separate tools already exist for detecting misassemblies, filling precise gaps, and scaffolding contigs, GenomeTailor performs all three tasks simultaneously and outputs a completed assembly graph. It enhances the correctness and completeness of assembly graphs and has successfully improved the assembly of a multi-haplotype sample and ribosomal DNA. When integrated into HairSplitter, it delivers metagenomes of unparalleled completeness and precision (see Chapter 2). However, the current version of GenomeTailor employs a

basic strategy that does not detect all missing edges, as observed when applying it to miniasm assemblies. This could be significantly improved by drawing inspiration from more advanced misassembly detection algorithms such as klumpy or GAEP.

Another potential application of GenomeTailor is to update a pangenome graph using raw reads alone. Currently, the standard pipeline for updating a pangenome graph with a new dataset involves assembling the dataset and then adding the assembly to the pangenome graph [114]. However, the assembly process introduces biases, typically the collapse of haplotypes. GenomeTailor could offer a method to update a pangenome graph directly with the reads, bypassing the assembly step entirely. This remains to be tested and GenomeTailor adapted to this specific task.

Chapter 2

Recovering haplotypes using noisy long reads

Abstract: This chapter presents complete pipelines to transform a collapsed assembly into a haplotype-aware assembly using long, erroneous reads. The first pipeline, HairSplitter, first corrects structural errors with GenomeTailor. It then recovers small variants using a novel statistical method. We demonstrate that HairSplitter significantly outperforms state-of-the-art methods in recovering collapsed strains, particularly in complex metagenomes. To further enhance this pipeline, we formalize the problem as an optimization problem and propose an Integer Linear Programming (ILP) solution, implemented in a module called strainMiner. This formalisation paves the way for future improvements from other fields of computer science and maths.

The most commonly used assemblers for assembling highly erroneous reads start by calculating pairwise overlaps between reads (e.g. [61, 78, 79]). To account for sequencing errors, pairwise overlaps must be computed tolerating sequencing errors. However, two reads from very similar haplotypes overlap very well: when the overlaps are computed, the differences between the haplotypes are largely drowned out by sequencing errors. The assemblers, whether based on the OLC or the DBG paradigm, then assemble the haplotypes into a single contig based on this overlap information. The final assembled contig is a consensus of the different haplotypes, effectively discarding small SNPs and indels that characterise each haplotype.

The only effective way to prevent assemblers from collapsing haplotypes is to remove overlaps between reads from different haplotypes. hifiasm [63] and HiCanu [115] implement this strategy by first correcting reads in a haplotype-aware manner and then allowing only perfect overlaps, but are limited to high fidelity reads. [116] proposes a solution to correct highly erroneous reads in a haplotype-aware manner, but the presented mutliploid assemblies are far from complete.

This thesis implements a different strategy, which is to correct a (partially) collapsed assembly post-hoc. Three papers detailing this work have been reviewed and accepted:

— The first paper has been accepted at the proceeding-conference JOBIM2023 [117]. It presents for the first time a comprehensive pipeline designed to generate a phased assembly from a draft assembly and raw reads, named HairSplitter. This paper is the most detailed regarding the proposed algorithm (for example, it is the only one that details the rationale behind cutting input contigs in windows). However, the results are based only on simulated datasets.

— The main paper is soon to be recommended in *PCI Mathematical and Computational Biology* [110]: HairSplitter has been improved compared to the first paper, adding to the pipeline GenomeTailor to deal with the structural errors of the input assembly (the first versions of GenomeTailor were developed specifically for HairSplitter). The statistics behind HairSplitter are clearly explained. The main contribution of this article is to present a state-of-the art software which significantly improves the completeness of multi-strain metagenomic assemblies, with tests on both simulated and real data.

— Another paper was acepted for the BIOSTEC 2024 conference in Rome [118]. This paper formalises the core algorithm of HairSplitter and proposes an alternative Integer Linear Programming (ILP) solution. This ILP approach is implemented in a module named strainMiner, which initially diminished considerably the memory footprint of the pipeline. In response, the latest versions of HairSplitter were optimised for RAM consumptions and the comparisons provided in the article are already out of date. The conference suggested we produce a journal version of the paper, which is the one presented here. In my opinion, the main contribution of this article is to provide a formalisation of the haplotyping problem as a matrix tiling problem, paving the way for future improvements coming from the optimisation or graph theory fields.

HairSplitter: separating strains in metagenome assemblies with long reads

Roland Faure[1,2], Jean-François Flot[2] and Dominique Lavenier[1]
[1] Univ. Rennes, INRIA RBA, CNRS UMR 6074, Rennes, France
[2] Service Evolution Biologique et Ecologie, Université libre de Bruxelles (ULB), 1050 Brussels, Belgium

Corresponding author: roland.faure@irisa.fr

Abstract *Long read assemblers struggle to distinguish closely related strains of the same species and collapse them into a single sequence. This is very limiting when analysing a metagenome, as different strains can have important functional differences. We present the first version of a new software called HairSplitter, which recovers the strains from a strain-oblivious assembly and long reads. The originality of the method lies in a custom variant calling step that allows HairSplitter to work with erroneous reads and to separate an unknown number of haplotypes. On simulated datasets, we show that HairSplitter significantly outperforms the state of the art when dealing with metagenomes containing many strains of the same species.*

Keywords Metagenomics, Haplotyping, Genome assembly, Strain separation

1 Introduction

A powerful tool for understanding complex microbial communities is de novo metagenome assembly. Current methods can reconstruct the genomes of sufficiently abundant species, but struggle to differentiate strains within a species, even if they are abundant. While strains of the same species are very similar at the genomic level, the small differences can lead to very significant phenotypic and functional changes. The most famous example of such intra-specific diversity is probably *Escherichia coli* [1], some strains of which can be highly pathogenic while sharing an average nucleotide identity of more than 98.5% with commensal strains [2].

Assemblers are designed to correct for sequencing errors by ignoring bases that occur at low frequencies. As a side effect, they generally discard haplotype differences and collapse the different haplotypes into a single sequence. An additional difficulty in the metagenomic context is that the number of haplotypes is a priori unknown and that haplotypes generally have different frequencies in a sample.

Specific software has been developed to overcome these difficulties. For example, two such software based on short reads are STRONG [3] and strainXpress [4]. However, more and more samples are sequenced using only long reads, as their cost has dropped recently and they allow for more contiguous assemblies.

Using error-prone long reads, assemblers such as metaFlye [5] or Canu [6] attempt to assemble strains separately. However, the authors of [7] showed that these assemblers still struggle to recover multiple strains and proposed a new pipeline, called *Strainberry*, which takes an assembly and the long reads as input and recovers the collapsed strains. Strainberry improves significantly the assembly of samples containing 2 or 3 strains of the same species, but is limited when the number of species increases.

We present *HairSplitter*, a new pipeline for recovering the strains lost when assembling exclusively from (error-prone) long reads. HairSplitter does not make any assumption on the number of strains that should be found in the metagenome. It contains an original procedure that combines a custom variant calling method with a new phasing algorithm. We have extensively tested HairSplitter on simulated Nanopore data, replicating the protocol proposed in [7], and show that HairSplitter significantly improves the completeness of assemblies of metagenomes composed of many strains compared to the state of the art. HairSplitter still needs to be tested on real datasets.

2 Description of the pipeline

The HairSplitter pipeline is composed of four main steps: 1) alignment of the reads on the contigs, 2) separation of the reads in their haplotype of origin, 3) generation of the new contigs and 4) strain-aware contig scaffolding. Steps 2 is done by an original software, while step 1 is performed by minimap2 [8], step 3 by Racon [9] and step 4 by GraphUnzip [10]. The pipeline is illustrated Figure 2

Step 1: Aligning the reads on the contigs

HairSplitter starts by generating base-to-base alignments of the sequencing reads on the assembly with minimap2. For each contig, a multiple sequence alignment is generated using the alignment of the reads to the reference. This is the simplest way to build a multiple sequence alignment, but it creates alignment artifacts, especially at positions where the contig contains errors.

Step 2: Splitting a group of reads in one or more haplotypes

The originality of HairSplitter lies in the second and most crucial step, where reads that align on a contig are separated by haplotype of origin.

This operation is performed locally on window of the contig of size w. The phasing is performed locally to avoid clustering reads that do not overlap. Indeed, clustering together reads that do not overlap could lead to the HairSplitter algorithm clustering very different reads together, as shown in Figure 1. w should thus be chosen to be significantly shorter than the reads.

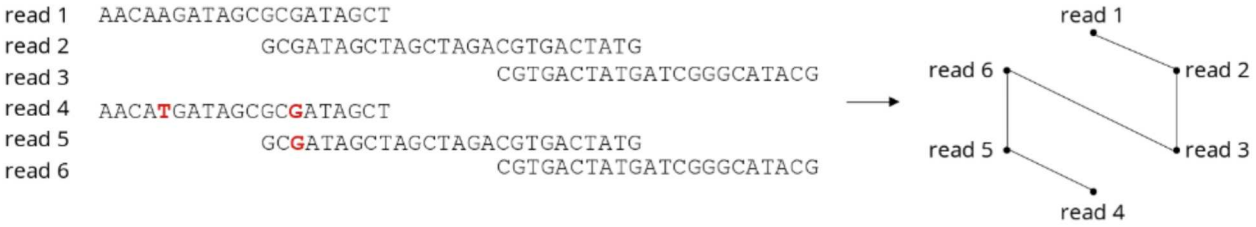


Fig. 1. A read graph is built as follows: each read is a vertex ; reads are connected to the reads they overlaps with 100% identity. Even though read1 and read4 are very different, they are transitively linked through read3 and read6.

A difficulty of the process comes from the error rate of the reads, which can be much higher than the divergence between the haplotypes. Another difficulty is the possibly high number of haplotypes which can be unevenly covered by the sequencing. As this step represents the core of HairSplitter, it is described in detail in section 3.

Step 3: Generating new contigs

The reads on a given window are separated into n groups, the contig sequence in the window is polished n times by Racon using the different groups, yielding n different versions of the window, which we call *subcontigs*. The subcontigs are laid out as an assembly graph, based on the original assembly graph.

Step 4: Strain-aware scaffolding

Due to local homozygosity, some subcontigs will contain multiple haplotypes. These subcontigs limit the contiguity of the graph and must be duplicated to be present once for each haplotype. To do this, the paths of the sequencing reads on the subcontig graph are inventoried. Once all the paths are inventoried, GraphUnzip [10] untangles the graph. From the paths of the read in the graph, it deduces which contig to duplicate and which contig to link to improve the contiguity and completeness of the assembly.

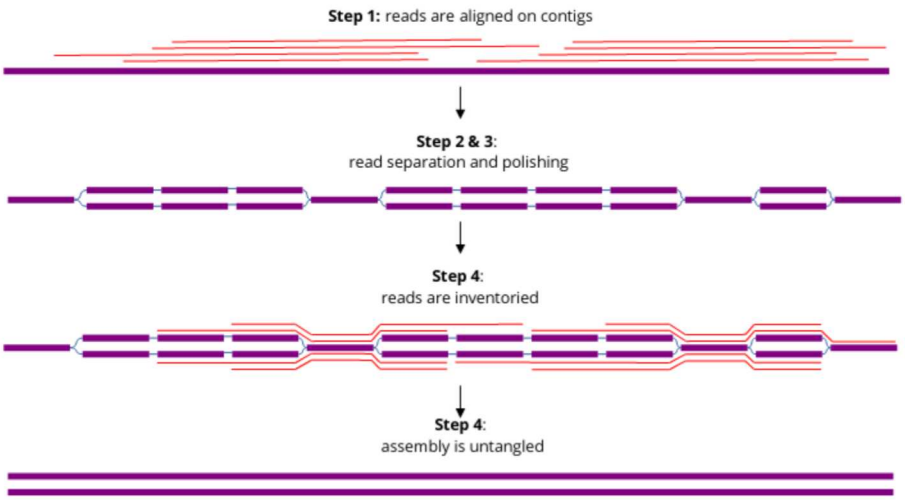


Fig. 2. The HairSplitter pipeline. The purple rectangles represent contigs or subcontigs. The red lines are reads used to build the subcontigs. Only a very small subset of reads is shown here to keep the visualisation readable.

3 Splitting a group of reads in one or more haplotypes

3.1 Detecting variants

To separate the set of reads that align on a contig in several haplotypes, rudimentary variant calling is performed. In this context, we define variants as positions where the different haplotypes are not identical. Once the variants are clearly identified, the reads can be split into the different haplotypes based on these positions, as can be seen in Figure 3. However, due to errors in the reads and in the reference, differences between read and reference do not systematically highlight a variant, as can be seen in Figure 4.

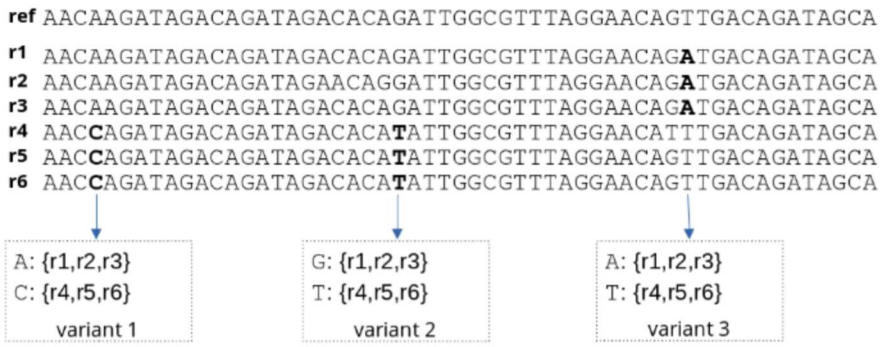


Fig. 3. In this error-free alignment, the variants clearly separate the reads in two haplotypes, one containing r1, r2 and r3 and the other containing r4, r5 and r6.

3.2 Dealing with errors

To distinguish variants from errors, all positions of the alignment are iteratively inspected. For each position, the program inventories the triplet of bases centered there in the reads. A base triplet is defined as the base at the given position flanked by the nucleotide on the left and the nucleotide on the right in the given read. At conserved regions, which make up the majority of the contig, the reference triplet will be the most common, while some residual triplets may be due to sequencing errors or alignment artifacts. At positions of true genomic variants, two triplets (corresponding to the two variants) will be common, with some residual triplets due to sequencing errors or alignment artifacts. *Suspect positions* are defined as positions where the second most abundant triplet is significantly (by a factor s) more abundant than the third most abundant triplet. All variant should generate at least one suspicious positions, therefore only suspect positions are considered for phasing. This will discard most positions where there are only a few random errors. Some positions where there are no true

variants will also be flagged as suspicious, typically positions with alignment artifacts. The selection of suspicious position is illustrated Figure 4.

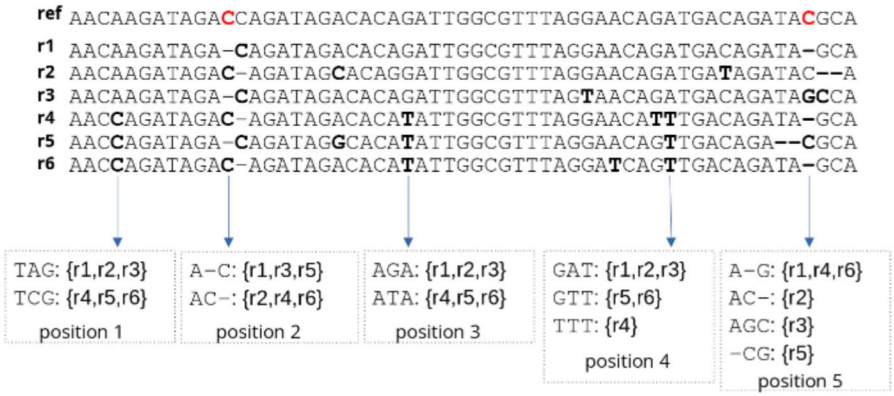


Fig. 4. Selection of suspicious positions. The red bases in the reference represent errors (unknown to HairSplitter). Note that position 2 will be flagged as suspicious because of an alignment artifact. Position 5 will not be considered suspicious because even though there are many different triplets at this position, there is no clear alternative to the triplet 'A-G'.

To distinguish variants from alignment artifacts, HairSplitter implements a method based on the intuition that alignment artifacts are generally randomly distributed on the reads, while genomic variants are not. On the one hand, reads carrying sequencing and alignment errors are not correlated between two positions. On the other hand, reads carrying variants should be strongly correlated between two variants.

Suspicious positions are clustered hierarchically. A cluster is represented by its consensus. Two clusters are merged if more than 90% of each part of each consensus consists of reads from a part of the other consensus. At the end of the process, consensuses consisting of more than p positions are considered solid. We call them the *solid bipartitions* of the reads.

For each suspect position, its correlation with all solid bipartitions is computed by a one-degree-of-freedom chi-square test of independence. If the result is greater than five (strong correlation), the position is marked as *interesting*. The positions that do not correlate well with any solid bipartitions are considered untrustworthy and are discarded. The set of interesting positions corresponds to the set of positions that will be considered as a bi-allelic variants by the algorithm.

This variant calling procedure is quite conservative and may miss existing variants. This is not a problem, as the goal of this step is to confidently identify a set of variants, not to call all variants exhaustively.

3.3 Phasing the reads using the variants

Once a set of variants has been largely extracted from the noise, reads are split into different haplotypes.

A graph is generated for each window of the contig. The reads that cover the window from end to end are the vertices of the graph. Pairwise distances between the reads are calculated as the number of divergent interesting positions divided by the total number of interesting positions overlapped by both reads. Each read is connected to its k nearest neighbors, as shown Figure 5.

The resulting graph forms clusters, grouping reads that share identical variants, corresponding to haplotypes. Empirically, the best algorithm to cluster this graph without knowing a priori the number of clusters seems to be the Chinese Whispers algorithm [11]. However, a limitation of this algorithm is that it is not deterministic, especially when the number of nodes in the graph is small. With low frequency, clusters will be merged or split. To avoid this, HairSplitter exploits the property that if the Chinese Whispers algorithm is initialized with an approximate solution, it will converge to the solution without splitting or merging clusters. The approximate solutions can be found at the interesting positions: each position contains a variant that separates two groups of reads with some

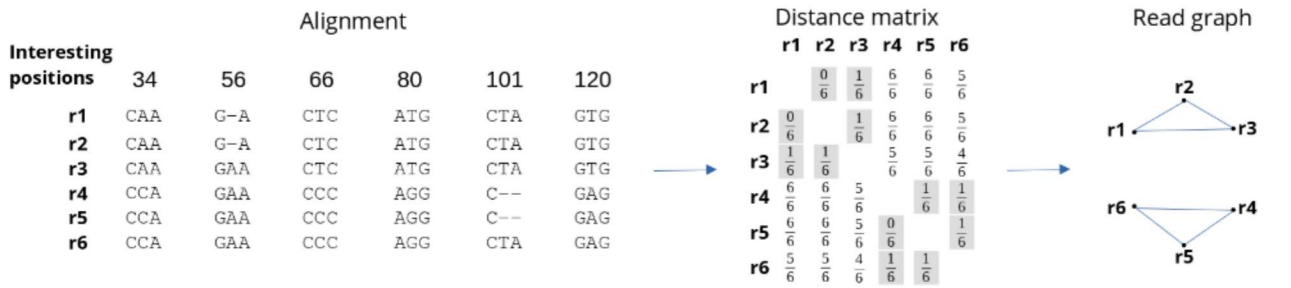


Fig. 5. Generating the read graph from the list of interesting positions with $k = 2$. Light gray squares highlight the k closest neighbors of each read in the distance matrix, which become ones in the adjacency matrix of the read graph.

errors. Running the Chinese whispers algorithm with one position as initialization will cluster the graph into two parts. Each interesting positions will yield a bipartition. All the bipartitions can then be aggregated into a single partition: two reads will be clustered together if and only if they are not separated in any bipartition. This process is illustrated Figure 6.

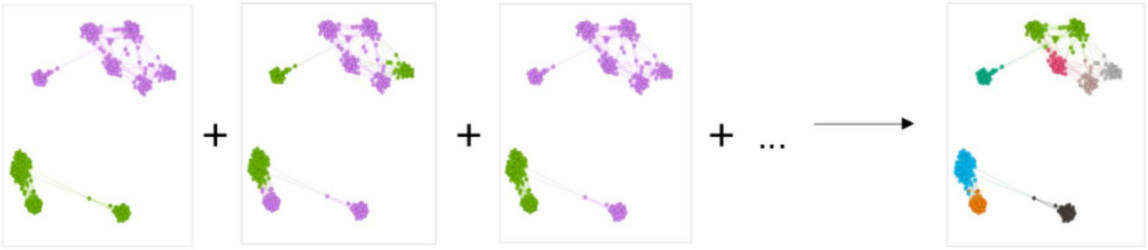


Fig. 6. The read graph is clustered using interesting positions as initialisation, resulting in a series of bipartitions. All these bipartitions can then be aggregated into the final partition.

Each part of the resulting partition corresponds to a haplotype.

4 Results

4.1 Protocol

The protocol is a replica of the protocol proposed in [7].

To systematically test the performance of HairSplitter, we composed a benchmark consisting of different strains of *Escherichia coli*. The reference genomes of the strains were obtained from NCBI. For each strain, we simulated "mediocre" Nanopore sequencing (around 7% error rate), using Badreads [12] with default settings, from the reference genomes.

For each experiment, the sequencing of different strains was concatenated to create a simulation of the sequencing of a metagenomic sample. The reads were then assembled using metaFlye with default settings. Missing strains were then recovered from the assembly using Strainberry and HairSplitter with default settings. For HairSplitter, the parameters are set to $s = 5$, $k = 5$, $p = 5$ and $w = 2000$, but the few tests we ran suggest that the algorithm is not very sensitive to these settings.

4.2 Evaluation metrics

Two metrics were retained to evaluate the quality of the recovered assembly with respect to the known solution genomes.

The first one is the proportion of the 21-mers found in the genomes that are not found in the assembly. This measures how well the different strains are covered. A large number of missing 21-mers indicates that some strains have not been well assembled.

The second metric is the proportion of 21-mers found in the assembly that are found in the genome. This evaluates the accuracy of the assembly. A low number of 21-mers found in the genomes indicates that the assembly contains many errors.

4.3 Influence of strain coverage, divergence and number of strains on strain separation

Divergence A factor that can influence the strain reconstruction is the degree of divergence between the strains. Seven datasets were created, composed of the simulated sequencing of the K12 strain mixed with the simulated sequencing of seven strains having varying degree of divergence with K12. The strains have been chosen to replicate exactly the experiment shown in [7].

Coverage Another crucial factor to reconstruct the strains is the depth at which each strain is covered. To evaluate how this affected the HairSplitter algorithm, tests were carried out on a mixture of the IAI1 and 12009 strains. In a first experiment, the two strains were sequenced at depths ranging from 5x to 50x. In a second experiment, the 12009 strain was sequenced systematically at 50x coverage, while the IAI1 strain was sequenced at depths ranging from 5x to 50x.

Number of strains The authors of Strainberry pointed to the number of strains as a limiting factor in strain reconstruction, with the completeness of the reconstruction decreasing significantly when more than 3 strains were sequenced [7]. Mixtures were made with different numbers of strains. The strains used for the mixtures were 12009, IAI1, F11, S88, Sakai, SE15, *Shigella flexneri*, UMN026, HS and K12. The strains were chosen to cover a wide range of the *Escherichia coli* phylogenetic tree, with some very close strains (such as F11 and S88) and others much more distance strains (such as SE15 and K12).

The results Figure 7 show that HairSplitter and Strainberry recover a very similar amount of k-mers on mixtures of 2 strains. More specifically, both software perform well on pairs of strains with more than 0.3% divergence, as defined by the ANI [13] (Figures 7b), and when the coverage is 20x or more (Figures 7d, 7f). This confirms the results presented in [7]. However, while HairSplitter recovers a similar amount of missing 21-mers compared to Strainberry, it tends to generate much fewer erroneous 21-mers (Figure 7a, 7c and 7e).

The most spectacular result is that even when the mixture contained six or more strains, HairSplitter was able to recover most of the missing 21-mers, producing assemblies with significantly fewer missing 21-mers than metaFlye and even Strainberry assemblies (Figure 7h). For example, in the metaFlye assembly of the mixture of 10 strains, 46% of the 21-mers of the solution were missing. This dropped to 39% when using Strainberry and 16% when using HairSplitter.

4.4 Performance

In all these tests, HairSplitter finished in less than 30 minutes using four threads and less than 5G of RAM. This is similar to Strainberry and small compared to the assembly time, which took at least 5 times longer.

5 Discussion

In this work, we introduced HairSplitter, a new pipeline for performing strain separation on assemblies using only long reads. HairSplitter shows a significant improvement over state-of-the-art methods when the number of strains is high. One of the reasons for this is that, unlike Strainberry, HairSplitter can separate a contig into a variable number of strains. To confirm these results, HairSplitter needs to be benchmarked on real sequencing datasets.

The data we simulated was of low quality to test HairSplitter in the hardest possible case. We expect HairSplitter to perform even better as the quality of the sequencing improves, but this remains to be tested. We also want to see if HairSplitter can improve on the de novo assembly performed by hifiasm in the case of HiFi sequencing.

Another potential application of HairSplitter that deserves investigation is the phasing of polyploid species. Powerful software, such as WhatsHap [14], already exists when the number of haplotypes is known a priori and can be used for polyploid assembly. However, knowing the number of haplotypes in each contig is not necessarily an easy task and using an agnostic approach such as HairSplitter could improve the results.

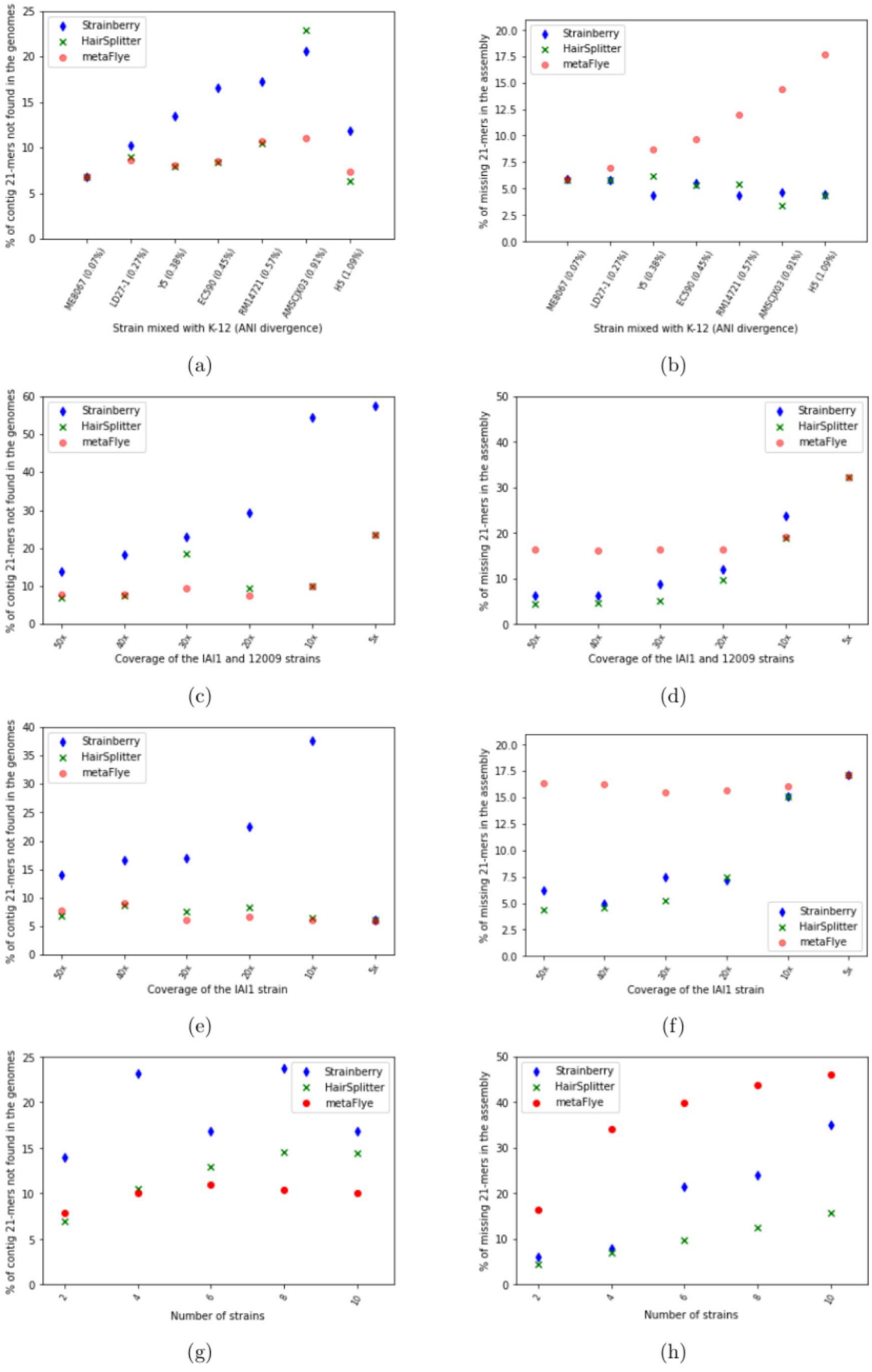


Fig. 7. Evaluation of assemblies obtained using HairSplitter or Strainberry on the metaFlye assembly in different mixes of strains. a and b: mix of K12 strain and another strain at 50x coverage. c and d: mix of the IAI1 and 12009 strains at varying coverage. e and f: mix of the IAI1 and 12009 strains, with 12009 at 50x coverage and IAI1 at varying coverage. g and h: mix of varying number of strains at 50x coverage.

Currently, HairSplitter has two shortcomings that we are trying to fix as a priority. First, it needs at least 20x coverage to distinguish a strain. This is quite high and will undoubtedly be limiting in most real-world applications where rare strains are common. Improving the quality of the data may solve this problem. The second shortcoming is that the contig scaffolding step is still not fully satisfactory and the contiguities of the assemblies obtained is lower than those obtained using Strainberry. This is due to the fact that we are using GraphUnzip slightly outside the use case for which it was designed. We will adapt GraphUnzip to this specific task.

Acknowledgements

We thank the Genouest facility (genouest.org) for providing us the hardware, software and support necessary to run our computations. The software Tablet [15] and Bandage [16] were used to visualize data while developing HairSplitter. For the purpose of Open Access, a CC-BY public copyright licence has been applied by the authors to the present document and will be applied to all subsequent versions up to the Author Accepted Manuscript arising from this submission

References

[1] Olivier Tenaillon, David Skurnik, Bertrand Picard, and Erick Denamur. The population genetics of commensal Escherichia coli. *Nature Reviews Microbiology*, 8(3):207–217, March 2010.

[2] S Hudault. Escherichia coli strains colonising the gastrointestinal tract protect germfree mice against Salmonella typhimurium infection. *Gut*, 49(1):47–55, July 2001.

[3] Christopher Quince, Sergey Nurk, Sebastien Raguideau, Robert James, Orkun S. Soyer, J. Kimberly Summers, Antoine Limasset, A. Murat Eren, Rayan Chikhi, and Aaron E. Darling. Metagenomics Strain Resolution on Assembly Graphs. preprint, Bioinformatics, September 2020.

[4] Xiongbin Kang, Xiao Luo, and Alexander Schönhuth. StrainXpress: strain aware metagenome assembly from short reads. *Nucleic Acids Research*, 50(17):e101–e101, September 2022.

[5] Mikhail Kolmogorov, Derek M. Bickhart, Bahar Behsaz, Alexey Gurevich, Mikhail Rayko, Sung Bong Shin, Kristen Kuhn, Jeffrey Yuan, Evgeny Polevikov, Timothy P. L. Smith, and Pavel A. Pevzner. metaFlye: scalable long-read metagenome assembly using repeat graphs. *Nature Methods*, 17(11):1103–1110, November 2020.

[6] Sergey Koren, Brian P. Walenz, Konstantin Berlin, Jason R. Miller, Nicholas H. Bergman, and Adam M. Phillippy. Canu: scalable and accurate long-read assembly via adaptive k -mer weighting and repeat separation. *Genome Research*, 27(5):722–736, May 2017.

[7] Riccardo Vicedomini, Christopher Quince, Aaron E. Darling, and Rayan Chikhi. Strainberry: automated strain separation in low-complexity metagenomes using long reads. *Nature Communications*, 12(1):4485, July 2021.

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

[9] Li Fang and Kai Wang. Polishing high-quality genome assemblies. *Nature Methods*, 19(6):649–650, June 2022.

[10] Roland Faure, Nadège Guiglielmoni, and Jean-François Flot. GraphUnzip: unzipping assembly graphs with long reads and Hi-C. preprint, Bioinformatics, February 2021.

[11] Chris Biemann. Chinese whispers: An efficient graph clustering algorithm and its application to natural language processing problems. *Proceedings of TextGraphs*, pages 73–80, 07 2006.

[12] Ryan Wick. Badread: simulation of error-prone long reads. *Journal of Open Source Software*, 4(36):1316, April 2019.

[13] Johan Goris, Konstantinos T. Konstantinidis, Joel A. Klappenbach, Tom Coenye, Peter Vandamme, and James M. Tiedje. DNA–DNA hybridization values and their relationship to whole-genome sequence similarities. *International Journal of Systematic and Evolutionary Microbiology*, 57(1):81–91, January 2007.

[14] Sven D. Schrinner, Rebecca Serra Mari, Jana Ebler, Mikko Rautiainen, Lancelot Seillier, Julia J. Reimer, Björn Usadel, Tobias Marschall, and Gunnar W. Klau. Haplotype threading: accurate polyploid phasing from long reads. *Genome Biology*, 21(1):252, September 2020.

[15] I. Milne, G. Stephen, M. Bayer, P. J. A. Cock, L. Pritchard, L. Cardle, P. D. Shaw, and D. Marshall. Using Tablet for visual exploration of second-generation sequencing data. *Briefings in Bioinformatics*, 14(2):193–202, March 2013.

[16] Ryan R. Wick, Mark B. Schultz, Justin Zobel, and Kathryn E. Holt. Bandage: interactive visualization of *de novo* genome assemblies. *Bioinformatics*, 31(20):3350–3352, October 2015.

Peer Community In Mathematical & Computational Biology

RESEARCH ARTICLE

Open Access
Open Peer-Review
Open Data
Open Code

HairSplitter: haplotype assembly from long, noisy reads

Roland Faure [1,2], Dominique Lavenier[1] & Jean-François Flot[2,3]

[1] Univ. Rennes, INRIA RBA, CNRS UMR 6074, Rennes, France
[2] Service Evolution Biologique et Ecologie, Université libre de Bruxelles (ULB), Brussels, Belgium
[3] Interuniversity Institute of Bioinformatics in Brussels – (IB)[2], Brussels, Belgium

Cite as: xxx

Correspondence: roland.faure@irisa.fr

Recommender: FirstName FamilyName

Reviewers: FirstName FamilyName and two anonymous reviewers

This version of the article has not yet been peer-reviewed by *Peer Community In Mathematical and Computational Biology* (https://doi.org/xxx/xxx)

Abstract

Motivation: Long-read assemblers face challenges in discerning closely related viral or bacterial strains, often collapsing similar strains in a single sequence. This limitation has been hampering metagenome analysis, where diverse strains may harbor crucial functional distinctions.

Results: We introduce a novel software, HairSplitter, designed to retrieve strains from a strain-oblivious assembly and long reads. The method uses a custom variant calling process to operate with erroneous long reads and introduces a new read clustering algorithm to recover an a priori unknown number of strains. On noisy long reads, HairSplitter can recover more strains while being faster than state-of-the-art tools, both in the viral and the bacterial case.

Availability: HairSplitter is freely available on GitHub at github.com/RolandFaure/HairSplitter.

Contact: roland.faure@irisa.fr

Keywords: Metagenomes; Metaviromes; Haplotyping; Genome assembly; Strain separation

Introduction

Microbiomes play a crucial roles in many ecosystems, such as soils or human guts, in turn impacting human health (Conlon and Bird, 2014) and soil fertility (Coban et al., 2022). Microbiomes typically contain sets of organisms with highly similar genomes, the sequences of which are called haplotypes (short for "haploid genotypes" (Ceppellini et al., 1967)). Distinguishing these lineages is an important challenge, as small genomic differences between haplotypes can lead to significant phenotypic changes. For instance, some strains of *Escherichia coli* can be pathogenic or commensal while having an Average Nucleotide Identity (ANI) (Konstantinidis and Tiedje, 2005) of more than 98.5% (Frank et al., 2011). A few mutations also became famous for altering significantly the infectiousness of some coronaviruses lineages (Magazine et al., 2022).

De novo sequencing and assembling is a central method to characterize microbial communities. Unlike previous methods, it allows to analyse the composition of a metagenome without culturing the strains, enabling a wide range of analyses (Ward, 2006). While existing genome assemblers proficiently reconstruct genomes of abundant species, they struggle to distinguish viral or bacterial haplotypes. The main difficulty for assemblers lies in the unknown number of haplotypes in a sample and their uneven coverage (Ghurye et al., 2016).

Many tools have been developed to overcome this problem in the context of short-read assemblies, such as OPERA-MS (Bertrand et al., 2019), Constrains (C Luo et al., 2015), STRONG (Quince et al., 2020), StrainXpress (Kang et al., 2022) and VStrains (R Luo and Lin, 2023). However, these methods are not designed for long-read sequencing and do not exploit the long-range information contained in long reads.

Long reads with low error rate, such as PacBio HiFi reads, may in the long term be a solution to finely distinguish strains. Specialized software such as hifiasm (Cheng et al., 2021), metamDBG (Benoit et al., 2024) and stRainy (Kazantseva et al., 2023) have proved successful at distinguishing strains. However, HiFi remains expensive and comes with serious limitations regarding the quantity of DNA needed for sequencing.

Several methods have been implemented to deal with haplotype separation for long reads with high error rates. While the viral and bacterial haplotype assembly problems are identical in their formulation, the characteristics of the input data vary significantly: the genomes are generally much shorter and much more deeply sequenced in the viral case. This has led to the emergence of software specialized in either of the two problems. In the context of bacterial strain separation, Vicedomini et al., 2021 showed that mainstream assemblers such as metaFlye (Kolmogorov et al., 2020) and Canu (Koren et al., 2017) failed to distinguish close bacterial haplotypes and proposed a new tool, called Strainberry, to reconstruct strains. In the context of viral strain separation, Strainline (X Luo et al., 2022) and HaploDMF (Cai et al., 2022) were presented to tackle specifically the viral haplotype reconstruction problem and need very high depth of sequencing to work. The method iGDA (Z Feng et al., 2021) was proposed as a general approach to phase minor variants while handling high error rates and could theoretically assemble both bacterial and viral haplotypes. The main shortcomings of all of these methods is that they struggle to recover haplotypes of low abundance. Additionally, most of these tools are very computationally intensive.

We present HairSplitter, an efficient pipeline for separating haplotypes in the viral and bacterial context using potentially error-prone long reads. HairSplitter first calls variants using a custom process to distinguish actual variants from alignment or sequencing artefacts, clusters the reads into an unspecified number of haplotypes, creates the new separated contigs and finally untangles the assembly graph. HairSplitter can be used for either metaviromes or bacterial metagenomes.

Methods

Overview of the pipeline

The HairSplitter pipeline is depicted on Figure 1 and comprises five steps: 1) correcting the assembly, 2) calling variants on each contig, 3) separating the reads by haplotype on each contig, 4) reassembling the strain-specific contigs and 5) scaffolding.

Assembly correction

Initially, the reads are aligned to the assembly using minigraph (Li et al., 2020, and the assembly is subsequently examined for breakpoints. Breakpoints represent locations in a contig where a significant number of reads stop aligning, typically signalling that all the collapsed strains in the contig do not contain the entire sequence of the contig. These contigs are potential sources of misassemblies, thus HairSplitter breaks the contigs at these breakpoints. Additionally, links are added in the graph between ends of contigs when there is sufficient read support. The process is illustrated in Figure 1a.

The refined assembly resulting from this process is used throughout the subsequent stages of the pipeline.

Mathematical model behind variant calling

To sort reads into haplotypes, the intuitive method of clustering reads based on the similarity of their full sequence proves ineffective due to the dominance of sequencing and alignment errors, obscuring strain differences. HairSplitter first identifies variants positions, pinpointing loci where strains exhibit actual differences. The reads are then separated based only on these loci. We did not find any variant caller suitable for our specific challenge - calling variants with noisy long reads in a metagenomic context including potentially low-abundance strains while maintaining high computational efficiency. Thus, we devised our own variant calling procedure, based on an idea already explored in (Z Feng et al., 2021).

The naivest procedure to identify polymorphic loci consists in going through the pileup of the reads on the assembly and identifying loci where at least a proportion p of reads have an alternative allele. However, this approach falls short when using error-prone reads. For instance, in the case of a strain representing only 1% of the total of the reads, p needs to be less than 0.01 to detect variant positions corresponding to this strain, resulting in the selection of many artefactual positions if the reads have an error rate $> 1\%$.

The key lies in taking several loci into account simultaneously. HairSplitter leverages the assumption that alignment artifacts occur randomly in the pileup, whereas genomic variant are expected to be correlated along the alignment. Consequently, pileups at polymorphic loci exhibit strong correlation, contrary to pileups at non-polymorphic loci. This allows HairSplitter to detect even rare strains, as illustrated below.

Consider a complete pileup of n reads over m positions, which we will model as a matrix of letters. Let us assume that errors occur independently on all reads and at all positions with a probability $\leq e$ and that all errors on a given column are identical (worst-case scenario). We aim to estimate the probability that there exist a reads that share errors at b different loci. In other words, the probability that there exist a submatrix of size $a * b$ containing only errors in the pileup, defined by selecting a rows (reads) and b columns (loci).

There exist $\binom{n}{a}\binom{m}{b}$ submatrices of size $a * b$. Each of these submatrix has probability lower than e^{ab} to contain only errors. Therefore, given that the expectation is linear (DeGroot and Schervish, 2002), the expectation E of the number of submatrices of size $a * b$ containing only errors in the pileup is lower than $\binom{n}{a}\binom{m}{b} * e^{ab}$. Now, to obtain the probability that there exist no submatrix of size $a * b$ containing only errors, we can use Markov's inequality, according to which the probability that a positive random variable be higher than 1 is always smaller than the expectation of this variable (DeGroot and Schervish, 2002). Here, it tells us that the

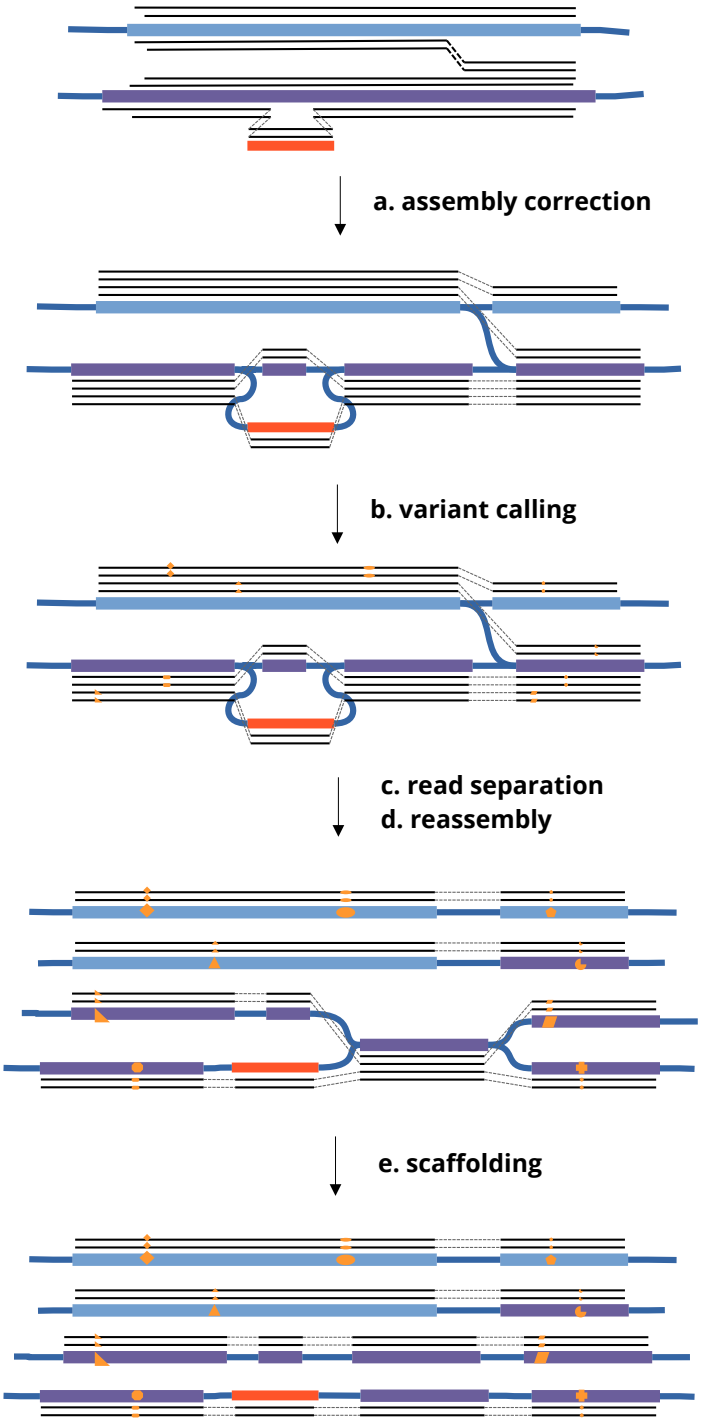


Figure 1. Illustration of the five steps of the HairSplitter pipeline. Colored rectangles represent contigs, thick blue lines are links in the assembly graph and black lines represent the reads aligned on the assembly. Orange shapes on reads and contigs indicate variant positions compared to the original sequence.

Figure 2. In this pileup of reads, does the submatrix of variants highlighted in red vouch for the presence of two strains? The probability that there exist 3 reads having alternative allele at 3 loci if we estimate $e = 0.1$ is less than 0.02: the variants are thus likely not independent and probably underline the presence of at least two different strains.

probability that there exist a submatrix containing only errors is smaller than E. In other terms, the probability that there exist somewhere in the pileup a reads sharing errors at b different loci is lower than $\binom{n}{a}\binom{m}{b} * e^{ab}$.

Now, let us consider a pileup with $n = 1000$ reads across $m = 5000$ positions and $e = 0.1$. The probability that there exist $a = 10$ reads sharing errors at $b = 10$ different loci is lower than $\binom{n}{a}\binom{m}{b} * e^{ab} = 9.10^{-44}$. Therefore, if the error rate is of 10% or less and the pileup indicates 10 reads (1% coverage) sharing an alternative allele at 10 loci (divergence of 0.2%), we can confidently assume that these are not errors, suggesting these reads originate from the same strain, and the loci are polymorphic sites.

Despite its simplified nature, this model underscores the statistical power gained by examining multiple loci simultaneously, enabling the detection of low-abundance, highly similar strains even in the presence of very noisy long reads. The idea behind the model is illustrated in Figure 2.

Variant calling

The approach to identifying polymorphic loci capitalizes on the statistical power underlined above. Specifically, HairSplitter aims to identify clusters of positions featuring alternative alleles on the same reads.

To generate the pileup, all reads are aligned to the assembly using minimap2 (Li, 2018) using default settings. HairSplitter then traverses the pileup, determining, for each position, the majority allele and the main alternative allele. Only positions with a minimum of five reads carrying alternative alleles are considered potential polymorphic sites to ensure statistical robustness (cf. model above). HairSplitter compares each new position to previously observed positions. If the set of reads with alternative alleles at this position and at a previously encountered position share more than 90% reads, the new position is clustered with the old one.

After all positions have been considered, clusters with more than five positions are deemed robust, and the corresponding positions are outputted as polymorphic sites. Once again, the threshold is imposed to avoid the inadvertent selection of artifact-prone positions.

Read binning

The contig is divided into windows with a default size of w (2000 bases by default). Reads are binned by haplotypes sequentially on the windows of a contig. Only reads spanning the entirety of the window are considered for binning. To cluster reads, HairSplitter operates on the premise that reads originating from the same haplotype should be identical at all polymorphic loci. Nevertheless, inherent sequencing and variant-calling errors might introduce unintended discrepancies among reads from a single haplotype. To address this, HairSplitter adopts a three-step strategy.

dataset	species	# strains	strain coverages	ANI divergence	sequencing technology
HBV-2	hepatitis B	2	4000x, 9900x	10%	Nanopore R.9.4.1
Norovirus-7	Norovirus	7	50, 350, 450, 700, 900, 1150, 1400x	1-3.9 %	Nanopore R.9.4.1
V. fluvialis	*Vagococcus fluvialis*	5	90x, 136x, 172x, 182x, 206x	0.01-1.51%	Nanopore R9.4.1
Zymo-GMS Q9	*Escherichia coli*	5	90x, 90x, 90x, 90x, 90x	0.37-1.51%	Nanopore R9.4.1
Zymo-GMS Q20	*Escherichia coli*	5	25x, 25x, 25x, 25x, 25x	0.37-1.51%	Nanopore R10.4.1
Zymo-GMS HiFi	*Escherichia coli*	5	41x, 41x, 41x, 41x, 41x	0.37-1.51%	PacBio HiFi

Table 1. Characteristics of the different datasets used for benchmarking on real data.

Step one is to correct errors at polymorphic loci. HairSplitter corrects the errors at polymorphic loci by performing a k-nearest-neighbour imputation (Fix and Hodges, 1989), with $k = 5$. The distance between two reads is defined as the number of different alleles at polymorphic positions. Each base of the pileup is considered and changed to the most frequent base among the k nearest neighbours on all reads and all positions until convergence.

Step two is to form clusters of reads, clustering reads together if and only if they exhibit no differences at any polymorphic loci.

In the third step, a last check is run to rescue small clusters that can arise from errors in Step 1. HairSplitter constructs a graph linking each read to its k closest neighbours, including links between all pairs of reads differing on one position or less. The graph is then clustered using the Chinese Whispers algorithm (Biemann, 2006), initialising the clustering with the clusters obtained in the second step. The Chinese Whispers algorithm always converge toward a stable solution, i.e. a clustering where all reads are in the same group as at least half of their neighbors. There exist many stable clusterings but the algorithm is likely to converge to a solution close to the initialization: the clusters obtained in the second step are unlikely to be significantly altered.

Reassembly

Across all windows on every contig, the original sequence undergoes repolishing using the haplotype-specific groups of reads previously identified. The polishing can be executed with either Racon (Fang and Wang, 2022) or Medaka (*Medaka* 2018), with the latter being more precise but considerably slower in our experience. By default, HairSplitter uses Medaka exclusively for short genomes ($\leq$ 1 Mb).

The resulting assembly comprises contigs of length w that can easily be stitched into longer contigs. For this purpose, a straightforward algorithm is employed, derived from Faure et al.(Faure et al., 2021) and depicted in Figure 1e. Let us call a contig exhibiting multiple outgoing links with other contigs at one end a “knot”. Knots generally represent collapsed contigs. All reads are initially aligned on the assembly graph. Subsequently, knots are iteratively assessed. If more than three reads traverse a neighbor of the knot (called A), then traverse the knot, and traverse another neighbor at the opposite end of the knot (called B), the knot is duplicated to create a new contig which will have as unique neighbors A and B. The links from A and B to the original knot are deleted, preserving only the links to the copy of the contig. This process is repeated until no further knots can be duplicated.

Results

Datasets

The datasets used in this article are described in Table 1. The accession numbers of the data on public repositories can be found in section* “Reproducibility and data availablility".

Bacterial datasets

We used the Zymobiotics Gut Microbiome Standard (abbreviated to Zymo-GMS) and a *Vagococcus fluvialis* dataset (Rodriguez Jimenez et al., 2022) to compare the performance of different algorithms designed to separate bacterial haplotypes in a metagenomic context. Zymo-GMS is a mixture of bacteria, archaea and yeast, 21 different strains in total, dosed to mimic the composition of the human gut microbiome. These 21 strains include five *Escherichia coli* strains, which we used to evaluate the strain-separation ability of various programs. Three Zymo-GMS sequencing were used, respectively from a Nanopore R9.4.1 run, a Nanopore 10.4.1 run and a PacBio HiFi run. The *Vagococcus fluvialis* dataset consists of a mix of five *Vagococcus fluvialis* strains that were sequenced together using barcoded reads, each barcode corresponding to a strain. We did not use the barcode information for the assemblies, reserving them for validation. Among the five strains, three had an ANI over 99.99%. metaFlye is used to assemble the reads, as it yielded better assemblies compared to Canu according to Vicedomini et al. (Vicedomini et al., 2021).

In addition, we simulated datasets to assess the impact of the number of strains, coverage and divergence on the assemblies. These experiments were directly inspired by the protocol of Vicedomini et al. (Vicedomini et al., 2021). The genomes of ten strains of *Escherichia coli* were downloaded from the SRA, namely 12009 (GCA_000010745.1), IAI1 (GCA_000026265.1), F11 (GCA_018734065.1), S88 (GCA_000026285.2), Sakai (GCA_003028755.1), SE15 (GCA_000010485.1), *Shigella flexneri* (GCF_000006925.2), UMN026 (GCA_000026325.2), HS (GCA_ 000017765.1), and K12 (GCF_009832885.1). These strains were chosen to be representative of the diversity of *E. coli.* We simulated Nanopore sequencing using Badread (R Wick, 2019) with the setting “Nanopore2023" to simulate 50x of R10.4.1 reads. Between 2 and 10 strains were mixed to assess how many strains the software could separate. From the 10-strain mix, the 12009 strain was downsampled to 30x, 20x, 10x and 5x to assess the impact of the coverage on strain separation. Finally, to assess the impact of the divergence of sequences on strain separation, 50x of reads were simulated for strain K12 and for strains of decreasing divergence with K12; assemblies of K12 with each of these strain was evaluated for separation.

Viral datasets

Two datasets were used to benchmark the performance of the programs tested at separating viral haplotypes, a 2-strain hepatitis B Virus (HBV) mix from (McNaughton et al., 2019) and an in-silico mix of the sequencing of seven strains of Norovirus from Cai et al. (Flint et al., 2021). These datasets were directly taken from the paper of HaploDMF (Cai et al., 2022). The reference genomes to run reference-based tools were taken as the reference genome in the GenBank database, GCF_000861825.2 for HBV and MW661279.1 for Norovirus.

Performance evaluation

We used MetaQUAST (Mikheenko et al., 2015) to measure assembly features such as assembly length, NG50, misassemblies, mismatches, indels and completeness. MetaQUAST was run with the –unique-mapping and –reuse-combined-alignments options to prevent a sequence, whether a contig or part of it, from being mapped to multiple distinct reference locations.

To assess if strains are well represented, the most important metric is the completeness of the resulting assembly. We chose to assess MetaQUAST completeness but also 27-mer completeness. MetaQUAST completeness measures the percentage of the solution on which the assembly aligns, while 27-mer completeness measures the percentage of the 27-mers of the solution that are effectively found in the assembly. Collapsed homozygous contigs typically impact negatively MetaQUAST completeness but not 27-mer completeness.

Evaluated software

In addition of HairSplitter, we chose to evaluate the software stRainy (Kazantseva et al., 2023) and Strainberry (Vicedomini et al., 2021), which have been introduced specifically as bacterial strain separation methods,

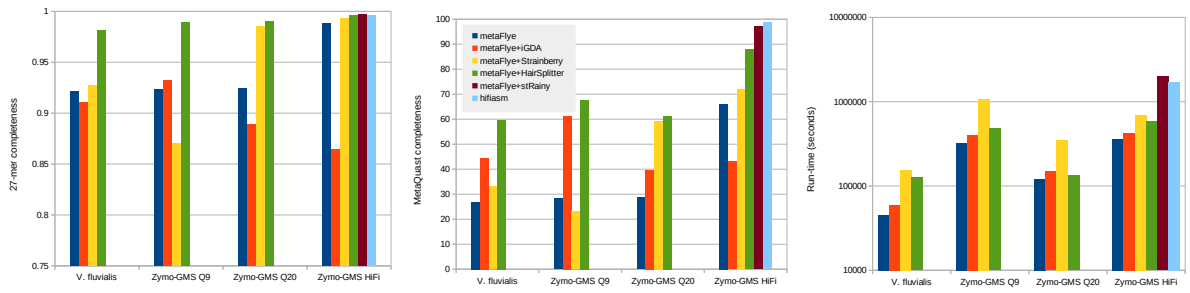


Figure 3. 27-mer completeness, MetaQUAST completeness and run-time of different software on the three Zymo-GMS dataset. The run-times are the run-times of the full assembly pipeline (assembly+strain separation) and are represented in log scale.

hifiasm-meta (X Feng et al., 2022), which is the most popular assembler for direct HiFi assembly, Strainline (X Luo et al., 2022) and HaploDMF (Cai et al., 2022), which have been introduced as viral strain separation methods and finally iGDA (Z Feng et al., 2021), which can perform both.

We have tried using all these software on all datasets. Strainline and HaploDMF failed to run in reasonable time on non-viral datasets and were automatically killed after 15 days of processing. Strainline failed to perform strain separation on the HBV-2 dataset within its allowed RAM limit of 50G, probably because of the high coverage. We tried downsampling the dataset but the problem remained.

The reference-based virus phasing tools were run with the same reference genome as in (Cai et al., 2022), MT622522.1 for hepatitis B and MW661279.1 for Norovirus.

Benchmarking evaluation

Bacterial haplotypes

The benchmark results on the Zymo-GMS and *V. fluvialis* datasets are illustrated in Figure 3 and detailed in Supplementary Table 2. HairSplitter performed better separation of the conspecific strains compared to the original metaFlye assemblies, delivering more comprehensive and accurate assemblies than Strainberry and iGDA. Particularly with Nanopore data, HairSplitter produced the most complete assemblies. On HiFi reads, all assemblies depicted a high duplication ratio but no evidence of erroneous contigs. This could be caused by the presence of small intra-specific variation unrepresented in the reference and would merit further investigation. The direct assembly of reads with hifiasm seems to yield the best results in terms of completeness and contiguity. In several assemblies, 27-mer completeness remains high while MetaQUAST completeness is notably lower. This discrepancy arises when repeated genomic regions are not duplicated to their correct multiplicity. Typically, the three almost identical *V. fluvialis* strains were assembled as one.

The completeness of assemblies in the simulated benchmark is presented in Figure 4, with a detailed evaluation in Supplementary Table 3. The evaluation of iGDA is not depicted because iGDA inexplicably decreased the completeness of the original metaFlye assemblies. Simulations indicated that HairSplitter significantly outperformed Strainberry, particularly in scenarios involving a high number of strains in the metagenome. The relatively high completeness of the 8-strains Strainberry assembly can be attributed to its high duplication ratio. The completeness of HairSplitter assemblies decreased with the depth of coverage and similarity of the strains, especially below 20x coverage and 1% divergence. Interestingly, the decline in MetaQUAST completeness with coverage and divergence was more pronounced than the decline in 27-mer completeness, highlighting HairSplitter's effectiveness in separating divergent regions and its difficulties in duplicating homozygous regions. This corresponds to the results observed in the Zymo-GMS datasets, where many pairwise divergences of strains were $< 1\%$.

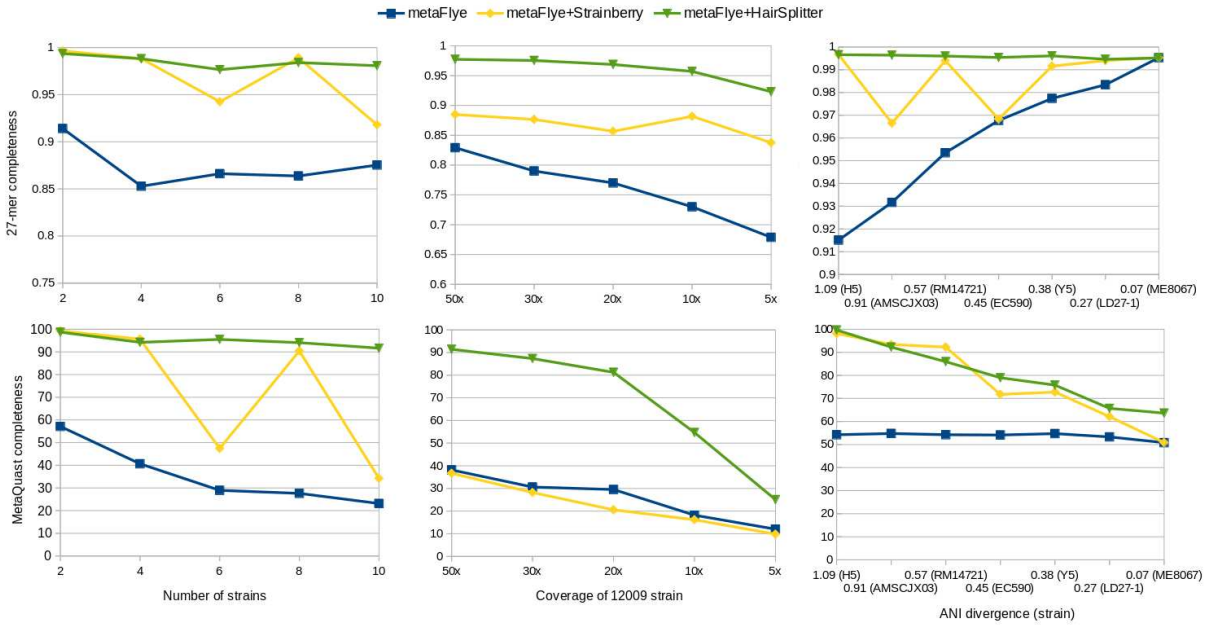


Figure 4. MetaQUAST completeness of assemblies of simulated metagenomes of *E. coli*. On the left, mix of 2 to 10 strains sequenced with 50x coverage were assembled. In the middle, strain 12009 was downsampled in the 10-strains metagenome and completeness of the 12009 strain is measured. On the right, reads of strains of decreasing divergence were mixed with K-12 reads and assembled.

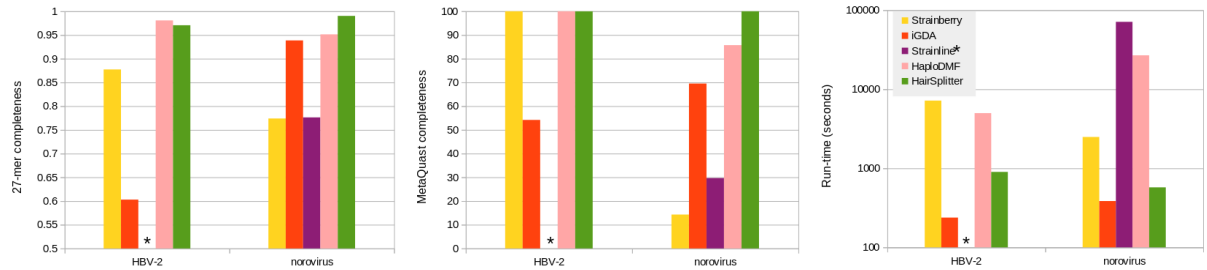


Figure 5. 27-mer completeness, MetaQUAST completeness and run-time of different software on the two viral datasets. Note that the run-time is shown in log scale. The Strainline assembly of HBV-2 is not shown because Strainline could not finish on this dataset.

Viral haplotypes

The completeness results of the benchmark on the viral datasets are depicted Figure 5 and more complete evaluation of assemblies are available in Supplementary Table 5.

HaploDMF and HairSplitter managed to separate completely the HBV strains according to MetaQUAST. iGDA failed to recover the strains, while Strainberry outputted four different haplotypes instead of two (see supplementary Table 5). We checked that HaploDMF and HairSplitter separated the reads adequately, thus the slight differences in 27-mers completeness stem from polishing errors.

HairSplitter stood out as the sole software capable of successfully recovering all seven strains in the Norovirus mix, even capturing the least abundant strain comprising only 1% of the mix. To assess the sensitivity limits of HairSplitter in the viral context, we conducted two additional experiments within the Norovirus mix. In the first experiment, we decreased the relative abundance of the rarest strain to 0.1%, while maintaining 50x coverage by uniformly increasing the coverage of the other strains. Remarkably, HairSplitter still achieved complete recovery (99.99% MetaQUAST completeness) of the rarest strain. The limited amount of data prevented us to further reduce the strain's relative abundance. In the second experiment, we uniformly diminished the coverage of all strains. The rarest strain was entirely recovered (99.99% MetaQUAST completeness) when covered at $\geq$40x, only the most divergent part of the virus was recovered (26.4% MetaQUAST completeness) at

coverage 20x and 30x, and the strain was not recovered at all at 10x coverage. The primary determinant of HairSplitter's sensitivity thus seems to be absolute coverage rather than the strain's relative coverage.

Discussion

In this manuscript, we introduce HairSplitter, a pipeline to assemble haplotypes separately using an input assembly and long reads. The pipeline includes two main novelties, an assembly correction program and a read separation procedure. HairSplitter proved useful when dealing with noisy data ($\geq$ 1% error rate), whereas specialised software such as hifiasm seemed to be better indicated to assemble HiFi reads. We show that HairSplitter can effectively separate several similar strains in both bacterial and viral contexts. Compared to the state of the art, HairSplitter can deal with a high number of strains and with low relative abundances, while maintaining a low computational cost.

HairSplitter faces a significant limitation when strains are highly similar, resulting in two distinct challenges. Firstly, the tool requires reads to span at least five polymorphic loci for effective separation into haplotype groups, posing difficulties in highly similar regions. Secondly, even with optimal read separation, homozygous regions present a challenge, as reads cannot be binned into haplotype groups within these regions. Consequently, full strain recovery necessitates the duplication of homozygous regions to their correct multiplicity. Although an immediate solution to the first problem is not apparent, the results obtained using HiFi data suggest progress could be made concerning homozygous region duplication. Indeed, stRainy achieved a 27-mer completeness comparable to HairSplitter, suggesting that the separation of reads was not significantly finer. However, stRainy outperformed HairSplitter in MetaQUAST completeness, indicating more effective duplication of homozygous regions in the assembly process.

A direction for future work would also be to generalize the assembly correction module. The idea of the module is to make sure all reads align end-to-end onto the assembly graph. We believe such a module could be useful to improve many assemblies. However, the version implemented for now in HairSplitter is very basic and does not perform well in repeated, complicated regions of the graph. A more sophisticated module could involve local reassembly and iterative graph correction.

Since HairSplitter is already successful at separating both bacterial and viral haplotypes, we expect to be able to extend this work naturally towards the phasing of polyploid organisms, including highly heterozygous non-model organisms, which remains an open problem (Guiglielmoni et al., 2021). For this particular case, some extra information could be leveraged to improve the HairSplitter pipeline, such as the fact that all haplotypes are expected to be equally abundant and that the total number of haplotype is usually known.

Reproducibility and data availablility

The HairSplitter code can be found on github at https://github.com/rolandfaure/hairsplitter.

The experiments were run with Flye 2.9.2-b1786, hifiasm HairSplitter v1.6.10, HaploDMF commit a07d082c3, Strainline commit 8d26341, iGDA commit 54ecec9, Strainberry v1.1, stRainy commit 34573cd, hifiasm-meta v0.3-r063.2, minimap2 v2.26-r1175 and Quast v5.2.0.

HBV sequencing reads can be found under accession number ERR3253560 in SRA. The seven Norovirus sets of reads can be found under accession numbers SRR13951181, SRR13951181, SRR13951186, SRR13951185, SRR13951184, SRR13951165 and SRR13951160. The *Vagococcus fluvialis* data are accessible under project PRJNA755170 in SRA. The Zymo-GMS sequencing data can be found under accession numbers SRR17913200, SRR17913199 and SRR13128013.

All the assemblies, simulated data and command lines used are available on Zenodo, DOI 10.5281/zenodo.10495033, https://zenodo.org/records/10495033.

Acknowledgments

We thank Ulysse Faure for his mathematical help. Alexandros Vasilikopoulos, Andrew Woodruff and Alessandro Derzelle tested HairSplitter and kindly helped debugging.

We acknowledge the GenOuest bioinformatics core facility (https://www.genouest.org) for providing the computing infrastructure. The programs Tablet (Milne et al., 2013) and Bandage (RR Wick et al., 2015) were used to visualize data while developing HairSplitter.

For the purpose of Open Access, a CC-BY public copyright licence has been applied by the authors to the present document and will be applied to all subsequent versions up to the Author Accepted Manuscript arising from this submission

Fundings

This work was funded by a Ph.D. AMX grant.

Conflict of interest disclosure

The authors declare that they comply with the PCI rule of having no financial conflicts of interest in relation to the content of the article. The authors declare the following non-financial conflict of interest: Jean-François Flot is a recommender of PCI Genomics.

References

Benoit G, S Raguideau, R James, A Phillippy, R Chikhi, and C Quince (Jan. 2024). High-quality metagenome assembly from long accurate reads with metaMDBG. *Nature Biotechnology*, 1–6. https://doi.org/10.1038/s41587-023-01983-6.

Bertrand D, J Shaw, M Kalathiyappan, AHQ Ng, MS Kumar, C Li, M Dvornicic, JP Soldo, JY Koh, C Tong, OT Ng, T Barkham, B Young, K Marimuthu, KR Chng, M Sikic, and N Nagarajan (Aug. 2019). Hybrid metagenomic assembly enables high-resolution analysis of resistance determinants and mobile elements in human microbiomes. en. *Nature Biotechnology* 37, 937–944. ISSN: 1087-0156, 1546-1696. https://doi.org/10.1038/s41587-019-0191-2.

Biemann C (July 2006). Chinese whispers: An efficient graph clustering algorithm and its application to natural language processing problems. *Proceedings of TextGraphs*, 73–80.

Cai D, J Shang, and Y Sun (Oct. 2022). HaploDMF: viral Haplotype reconstruction from long reads via Deep Matrix Factorization. *Bioinformatics* 38. https://doi.org/10.1093/bioinformatics/btac708.

Ceppellini R, E Curtoni, P Mattiuz, V Miggiano, G Scudeller, and A Serra (1967). Genetics of leukocyte antigens: a family study of segregation and linkage. In: *Report of Histocompatibility testing 1967*. Ed. by Curtoni E.S. Mattiuz P.L. TR.

Cheng H, G Concepcion, X Feng, H Zhang, and H Li (Feb. 2021). Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm. *Nature Methods* 18, 1–6. https://doi.org/10.1038/s41592-020-01056-5.

Coban O, G Deyn, and M Ploeg (Mar. 2022). Soil microbiota as game-changers in restoration of degraded lands. *Science* 375, abe0725. https://doi.org/10.1126/science.abe0725.

Conlon M and A Bird (Dec. 2014). The Impact of Diet and Lifestyle on Gut Microbiota and Human Health. *Nutrients* 7, 17–44. https://doi.org/10.3390/nu7010017.

DeGroot M and M Schervish (Jan. 2002). *Probability and Statistics*. Pearson. ISBN: ISBN 978-0-321-50046-5.

Fang L and K Wang (June 2022). Polishing high-quality genome assemblies. en. *Nature Methods* 19, 649–650. ISSN: 1548-7091, 1548-7105. https://doi.org/10.1038/s41592-022-01515-1.

Faure R, N Guiglielmoni, and JF Flot (Feb. 2021). GraphUnzip: unzipping assembly graphs with long reads and Hi-C. *bioRxiv*. https://doi.org/10.1101/2021.01.29.428779.

Feng X, H Cheng, D Portik, and H Li (June 2022). Metagenome assembly of high-fidelity long reads with hifiasm-meta. *Nature Methods* 19, 1–4. https://doi.org/10.1038/s41592-022-01478-3.

Feng Z, J Clemente, B Wong, and E Schadt (May 2021). Detecting and phasing minor single-nucleotide variants from long-read sequencing data. *Nature Communications* 12, 3032. https://doi.org/10.1038/s41467-021-23289-4.

Fix E and JL Hodges (1989). Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties. *International Statistical Review* 57, 238–247. ISSN: 03067734, 17515823.

Flint A, S Reaume, J Harlow, E Hoover, K Weedmark, and N Nasheri (Sept. 2021). Genomic Analysis of Human Noroviruses Using Combined Illumina-Nanopore Data. *Virus Evolution* 7. https://doi.org/10.1093/ve/veab079.

Frank C, D Werber, JP Cramer, M Askar, M Faber, M an der Heiden, H Bernard, A Fruth, R Prager, A Spode, M Wadl, A Zoufaly, S Jordan, MJ Kemper, P Follin, L Müller, LA King, B Rosner, U Buchholz, K Stark, and G Krause (2011). Epidemic Profile of Shiga-Toxin–Producing *Escherichia coli* O104:H4 Outbreak in Germany. *New England Journal of Medicine* 365, 1771–1780. https://doi.org/10.1056/NEJMoa1106483.

Ghurye J, V Cepeda-Espinoza, and M Pop (Sept. 2016). Metagenomic Assembly: Overview, Challenges and Applications. *The Yale Journal of Biology and Medicine* 89, 353–362.

Guiglielmoni N, A Houtain, A Derzelle, K Doninck, and JF Flot (June 2021). Overcoming uncollapsed haplotypes in long-read assemblies of non-model organisms. *BMC Bioinformatics* 22. https://doi.org/10.1186/s12859-021-04118-3.

Kang X, X Luo, and A Schönhuth (Sept. 2022). StrainXpress: strain aware metagenome assembly from short reads. en. *Nucleic Acids Research* 50, e101–e101. ISSN: 0305-1048, 1362-4962. https://doi.org/10.1093/nar/gkac543.

Kazantseva E, A Donmez, M Pop, and M Kolmogorov (Feb. 2023). *stRainy: assembly-based metagenomic strain phasing using long reads*. en. preprint. Bioinformatics. https://doi.org/10.1101/2023.01.31.526521.

Kolmogorov M, DM Bickhart, B Behsaz, A Gurevich, M Rayko, SB Shin, K Kuhn, J Yuan, E Polevikov, TPL Smith, and PA Pevzner (Nov. 2020). metaFlye: scalable long-read metagenome assembly using repeat graphs. en. *Nature Methods* 17, 1103–1110. ISSN: 1548-7091, 1548-7105. https://doi.org/10.1038/s41592-020-00971-x.

Konstantinidis K and J Tiedje (Mar. 2005). Genomic insights that advance the species definition for prokaryotes. *Proceedings of the National Academy of Sciences of the United States of America* 102, 2567–72. https://doi.org/10.1073/pnas.0409727102.

Koren S, BP Walenz, K Berlin, JR Miller, NH Bergman, and AM Phillippy (May 2017). Canu: scalable and accurate long-read assembly via adaptive *k* -mer weighting and repeat separation. en. *Genome Research* 27, 722–736. ISSN: 1088-9051, 1549-5469. https://doi.org/10.1101/gr.215087.116.

Li H (Sept. 2018). Minimap2: pairwise alignment for nucleotide sequences. en. *Bioinformatics* 34. Ed. by Birol I, 3094–3100. ISSN: 1367-4803, 1367-4811. https://doi.org/10.1093/bioinformatics/bty191.

Li H, X Feng, and C Chu (Oct. 2020). The design and construction of reference pangenome graphs with minigraph. *Genome Biology* 21, 265. https://doi.org/10.1186/s13059-020-02168-z.

Luo C, R Knight, H Siljander, M Knip, R Xavier, and D Gevers (Sept. 2015). ConStrains identifies microbial strains in metagenomic datasets. *Nature biotechnology* 33. https://doi.org/10.1038/nbt.3319.

Luo R and Y Lin (2023). VStrains: De Novo Reconstruction of Viral Strains via Iterative Path Extraction from Assembly Graphs. In: *Research in Computational Molecular Biology*. Ed. by Tang H. Cham: Springer Nature Switzerland, pp. 3–20. ISBN: 978-3-031-29119-7.

382 Luo X, X Kang, and A Schönhuth (Jan. 2022). Strainline: full-length de novo viral haplotype reconstruction from
383 noisy long reads. *Genome Biology* 23. https://doi.org/10.1186/s13059-021-02587-6.
384 Magazine N, T Zhang, Y Wu, M McGee, G Veggiani, and W Huang (Mar. 2022). Mutations and Evolution of the
385 SARS-CoV-2 Spike Protein. *Viruses* 14, 640. https://doi.org/10.3390/v14030640.
386 McNaughton A, H Roberts, D Bonsall, Md Cesare, J Mokaya, S Lumley, T Golubchik, P Piazza, J Martin, C Lara,
387 A Brown, M Ansari, R Bowden, E Barnes, and P Matthews (May 2019). Illumina and Nanopore methods for
388 whole genome sequencing of hepatitis B virus (HBV). *Scientific Reports* 9. https://doi.org/10.1038/s41598-
389 019-43524-9.
390 Medaka (2018). github.com/nanoporetech/medaka.
391 Mikheenko A, V Saveliev, and A Gurevich (Nov. 2015). MetaQUAST: Evaluation of metagenome assemblies.
392 *Bioinformatics* 32, btv697. https://doi.org/10.1093/bioinformatics/btv697.
393 Milne I, G Stephen, M Bayer, PJA Cock, L Pritchard, L Cardle, PD Shaw, and D Marshall (Mar. 2013). Using Tablet
394 for visual exploration of second-generation sequencing data. en. *Briefings in Bioinformatics* 14, 193–202.
395 ISSN: 1467-5463, 1477-4054. https://doi.org/10.1093/bib/bbs012.
396 Quince C, S Nurk, S Raguideau, R James, OS Soyer, JK Summers, A Limasset, AM Eren, R Chikhi, and AE Darling
397 (Sept. 2020). *Metagenomics Strain Resolution on Assembly Graphs*. en. preprint. Bioinformatics. https://doi.
398 org/10.1101/2020.09.06.284828.
399 Rodriguez Jimenez A, N Guiglielmoni, L Goetghebuer, E Dechamps, I George, and JF Flot (Aug. 2022). Com-
400 parative genome analysis of *Vagococcus fluvialis* reveals abundance of mobile genetic elements in sponge-
401 isolated strains. *BMC Genomics* 23. https://doi.org/10.1186/s12864-022-08842-9.
402 Vicedomini R, C Quince, AE Darling, and R Chikhi (July 2021). Strainberry: automated strain separation in low-
403 complexity metagenomes using long reads. en. *Nature Communications* 12, 4485. ISSN: 2041-1723. https:
404 //doi.org/10.1038/s41467-021-24515-9.
405 Ward N (Apr. 2006). New directions and interactions in metagenomics research. *FEMS microbiology ecology* 55,
406 331–8. https://doi.org/10.1111/j.1574-6941.2005.00055.x.
407 Wick R (Apr. 2019). Badread: simulation of error-prone long reads. *Journal of Open Source Software* 4, 1316.
408 ISSN: 2475-9066. https://doi.org/10.21105/joss.01316.
409 Wick RR, MB Schultz, J Zobel, and KE Holt (Oct. 2015). Bandage: interactive visualization of *de novo* genome
410 assemblies. en. *Bioinformatics* 31, 3350–3352. ISSN: 1367-4811, 1367-4803. https://doi.org/10.1093/
411 bioinformatics/btv383.

412

Supplementary material

		Genome Fraction (%)	Duplication ratio	NGA50	#misassemblies	# mismatches per 100 kbp	# indels per 100 kbp
Vagococcus fluvialis	metaFlye	26.90	1.097	-	40	340.14	383.57
	metaFlye + iGDA	44.530	1.151	1313	2	115.57	391.70
	metaFlye + Strainberry	33.112	1.119	-	45	77.71	510.25
	metaFlye + HairSplitter	59.668	1.137	31944	73	31944	355.52
Zymo-GMS Q9*	metaFlye	28.365	1.048	-	66	286.86	38.80
	metaFlye + iGDA	61.293	1.489	12450	21	225.67	41.30
	metaFlye + Strainberry	23.166	1.072	-	45	191.51	51.65
	metaFlye + HairSplitter	67.642	1.131	15357	89	93.44	30.94
Zymo-GMS Q20*	metaFlye	28.742	1.051	-	62	300.25	34.41
	metaFlye + iGDA	39.650	1.118	-	8	181.55	28.23
	metaFlye + Strainberry	59.197	1.138	28421	66	193.55	42.42
	metaFlye + HairSplitter	61.291	1.033	5264	12	34.77	11.97
Zymo-GMS HiFi*	metaFlye	66.064	1.076	79832	39	92.55	6.65
	metaFlye + iGDA	42.996	1.515	17104	15	102.70	9.96
	metaFlye + Strainberry	72.016	1.142	53249	46	57.53	6.92
	metaFlye + HairSplitter	87.906	1.582	45942	56	62.30	10.53
	metaFlye + stRainy	97.078	1.737	41195	47	44.15	12.26
	hifiasm	98.732	1.911	288422	82	30.07	4.99

Table 2. metaQuast metrics of the bacterial assemblies obtained from experimental data.

		Genome Fraction (%)	Duplication ratio	NGA50	#misassemblies	# mismatches per 100 kbp	# indels per 100 kbp
# strains							
2	metaFlye	57.137	1.043	61559	22	216.18	216.53
	metaFlye + Strainberry	99.268	1.074	701492	11	24.83	64.73
	metaFlye + HairSplitter	98.696	1.063	194764	7	11.54	47.26
4	metaFlye	40.666	1.071	-	50	562.91	268.83
	metaFlye + Strainberry	95.631	1.148	251144	39	93.07	81.88
	metaFlye + HairSplitter	94.217	1.077	25068	6	14.21	46.39
6	metaFlye	28.949	1.087	-	63	585.20	264.89
	metaFlye + Strainberry	47.430	1.108	8151	90	289.19	92.23
	metaFlye + HairSplitter	95.505	1.099	37869	35	39.62	46.65
8	metaFlye	27.599	1.051	-	76	527.44	277.85
	metaFlye + Strainberry	90.438	1.533	83755	157	179.31	172.25
	metaFlye + HairSplitter	94.079	1.157	32914	71	48.11	53.62
10	metaFlye	23.130	1.036	-	79	469.27	277.71
	metaFlye + Strainberry	34.207	1.095	-	175	363.06	137.01
	metaFlye + HairSplitter	91.626	1.159	27528	78	51.29	47.10
coverage							
30x*	metaFlye	30.618	1.032	-	1	719.80	55.27
	metaFlye + Strainberry	28.188	1.085	-	2	347.21	127.06
	metaFlye + HairSplitter	87.322	1.134	28641	0	60.09	35.23
20x*	metaFlye	29.522	1.018	-	6	859.11	76.03
	metaFlye + Strainberry	20.562	1.037	-	3	274.86	102.01
	metaFlye + HairSplitter	81.232	1.138	28700	1	94.37	37.13
10x*	metaFlye	18.201	1.005	-	1	498.17	85.10
	metaFlye + Strainberry	16.178	1.007	-	1	347.06	181.70
	metaFlye + HairSplitter	54.665	1.111	11999	5	120.35	53.81
5x*	metaFlye	12.020	1.010	-	2	849.88	201.29
	metaFlye + Strainberry	9.807	1.013	-	2	422.61	273.49
	metaFlye + HairSplitter	25.052	1.042	-	2	311.52	166.26
divergence							
H5	metaFlye	54.246	1.002	19206	22	324.97	29.23
(1.09%)	metaFlye + Strainberry	98.157	1.001	652572	2	324.97	1.35
	metaFlye + HairSplitter	99.666	1.042	196697	2	1.33	7.74
AMSCJX03	metaFlye	54.783	1.001	18675	17	254.48	28.42
(0.91%)	metaFlye + Strainberry	93.390	1.003	279448	3	0.73	1.98
	metaFlye + HairSplitter	92.249	1.003	18001	0	3.09	4.17
RM74721	metaFlye	54.254	1.000	19360	19	132.60	11.73
(0.57%)	metaFlye + Strainberry	92.256	1.006	380826	1	2.97	8.57
	metaFlye + HairSplitter	85.936	1.006	10000	0	3.56	4.98
EC590	metaFlye	54.132	1.000	17337	10	117.79	12.85
(0.45%)	metaFlye + Strainberry	71.749	1.003	156627	10	9.29	1.72
	metaFlye + HairSplitter	78.957	1.005	6001	6	8.48	5.10
Y5	metaFlye	54.736	1.002	22104	21	63.85	9.38
(0.38%)	metaFlye + Strainberry	72.758	1.006	181387	13	8.64	3.28
	metaFlye + HairSplitter	75.774	1.009	6000	1	1.68	1.89
LD27-1	metaFlye	53.295	1.001	19411	8	43.83	5.33
(0.27%)	metaFlye + Strainberry	62.101	1.004	112749	13	7.41	3.53
	metaFlye + HairSplitter	65.674	1.007	4000	0	3.54	2.80
ME8067	metaFlye	50.820	1.000	47356	8	10.29	3.19
(0.07%)	metaFlye + Strainberry	50.820	1.000	47356	8	10.29	3.19
	metaFlye + HairSplitter	63.628	1.002	4000	0	1.57	1.02

Table 3. metaQuast metrics of the bacterial assemblies obtained from simulated Nanopore R10.4.1 data. * The metrics displayed for the downsampled datasets are the metrics computed with respect to the downsampled strain, and not with respect to the complete 10 strains.

		Genome Fraction (%)	Duplication ratio	NGA50	#misassemblies	# mismatches per 100 kbp	# indels per 100 kbp
HBV-2	Strainberry	99.984	2.174	4504	3	881.59	1562.50
	iGDA	54.174	1.001	1081	0	201.15	229.89
	Strainline						
	HaploDMF	99.984	1.000	3207	0	15.58	93.46
	HairSplitter	99.953	1.001	3209	0	46.72	109.02
norovirus	Strainberry	14.283	1.000	-	0	52.97	13.24
	iGDA	69.514	1.548	2838	0	112.55	15.83
	Strainline	29.659	5.787	7541	0	479.44	136.67
	HaploDMF	85.702	1.000	7549	0	165.60	26.50
	HairSplitter	100.000	1.038	7550	0	107.57	35.96

Table 4. metaQuast metrics of the viral assemblies.

strainMiner: Combining Integer Programming and Data Mining Techniques for Strain-level Metagenome Assembly

Roland Faure[1,2*†], Tam Khac Minh Truong[1†], Victor Epain[3], Riccardo Vicedomini[1], Rumen Andonov[1*]

[1]GenScale, Univ. Rennes, Inria RBA, CNRS UMR 6074, Rennes, France.
[2]Service Evolution Biologique et Ecologie, Université libre de Bruxelles (ULB), Brussels, Belgium.
[3]Unaffiliated, independent researcher, Lorient, France.

*Corresponding author(s). E-mail(s): roland.faure@irisa.fr; rumen.andonov@irisa.fr;
†These authors contributed equally to this work.

Abstract

Metagenomic assembly is crucial for understanding microbial communities, but standard tools often struggle to differentiate bacterial strains of the same species, especially with low-accuracy reads from technologies like PacBio CLR and Oxford Nanopore. Current de novo assembly methods typically reconstruct bacterial genomes at the species level but fall short in distinguishing individual strain genomes. Our study presents a novel approach by reformulating the haplotyping problem as a matrix partitioning problem. We address this using Integer Linear Programming (ILP) combined with data mining techniques to improve computational efficiency. We introduce strainMiner, a strain-separation module integrated into an established pipeline to produce strain-separated assemblies. On real and simulated datasets with error rates ranging from 2.5% to 12%, strainMiner compares favorably to state-of-the-art methods in terms of assembly quality and strain reconstruction while significantly reducing computational requirements.

Keywords: Metagenomics, Strain-level assembly, Haplotype phasing, Integer Linear Programming, Hierarchical Cluster Analysis

1 Introduction

Metagenomics is a fairly new research field that consists of the analysis of sequencing data characterizing a mixture of microorganisms within an environment of interest [1]. One of the steps for accomplishing this task is through the precise identification of the organisms that are present in such an environment. This problem often requires reconstructing the genomes of the sequenced species, a problem called *metagenome assembly*. Reconstructing and identifying bacterial genomes within a metagenome from sequencing data is an extremely challenging task due to the need of distinguishing and assembling DNA fragments of distinct microorganisms [2]. Furthermore, genomes may also exhibit widely distinct levels of abundance and relatedness, making it difficult to discern sequence variability from errors [3]. For example, conspecific strains (*i.e.*, strains of the same species) could share sequence identity above 99% and, in practice, are often assembled into species-level consensus sequences which hide strain variability [4]. Being able to precisely identify distinct strains is nevertheless important for studying a microbial environment at a functional level, due to the high phenotypic variability exhibited by conspecific strains [5]. A classical example is *Escherichia coli* which could be found as a probiotic [6] or pathogenic [7] strain.

The challenge posed by the "strain separation" problem, as outlined in [4], arises from two primary factors: (i) the unknown number of strains and (ii) the variable abundance within a sample. Moreover, the precise characterization of what constitutes a "strain" is also not always clear. In this study, we will define a strain as a bacterial haplotype, i.e. a contiguous sequence of nucleotides observed jointly and in sufficient abundance by sequencing reads, in accordance with previous works [4]. Furthermore, we will use the terms strain and haplotype interchangeably.

In the last decade the strain separation problem has been extensively studied, either without (de novo) or with the availability of a reference sequence. Previous works attempted to tackle the de novo problem exploiting data from different sequencing technologies such as short reads [8–11], long reads [4, 12–14], or a combination of the two [15].

The increased accessibility of long-read sequencing (Oxford Nanopore and PacBio) for metagenomic data allows nowadays to accurately reconstruct complete genomes of bacterial species even from complex environments [16], especially using the low-error-rate PacBio HiFi technology [17–19]. At the same time, long reads are able to span far-apart strain-specific variants, offering the possibility to identify and reconstruct bacterial genomes even at the strain level.

Several methods have been recently proposed for the de novo strain-level assembly with long-read metagenomic data, namely Strainberry [4], stRainy [12], Floria [13], and HairSplitter [14]. These approaches take as input a "reference" species-level assembly (*e.g.*, built with a standard metagenome assembly tool) along with a set of long reads. A read alignment against the input assembly is then used to identify single-nucleotide polymorphisms (SNPs) which allow to partition reads likely belonging to the same haplotype. Strain-resolved and unphased sequences are finally represented within a graph in order to output more contiguous strain-resolved sequences.

Strainberry [4] was the first long-read-based tool proposed for the reconstruction of individual strains at the scale of a full metagenome. It exploits HapCUT2 [20] (a

diploid phasing tool based on likelihood optimization through graph-cuts) which is applied iteratively until no more strains need to be separated. While Strainberry does not require long reads from a specific technology, it is mainly limited to low-complexity metagenomes, i.e. containing no more than five conspecific strains.

stRainy [12] constructs a "connection" graph that encodes overlapping reads, sharing and agreeing on SNPs. Then, it recursively clusters reads using a community detection algorithm [21] with increased sensitivity. As opposed to the other approaches, stRainy has been mainly evaluated on long reads with fairly low error rates (i.e., PacBio HiFi, Nanopore R10, simulated reads with error rate up to 3%).

Floria [13] is based on the Minimum Error Correction (MEC) model, typically applied to (genomic) polyploid haplotype phasing. However, it uses an heuristic method on overlapping windows to locally identify strain counts and read partitions. A directed acyclic graph is then constructed from such partitions in order to solve a flow problem whose solution is then used to extract vertex-disjoint paths representing haplotypes.

HairSplitter [14] introduces a novel statistical approach to distinguish sequencing errors from SNPs, making it suitable for all long-read technologies. In order to cluster reads, HairSplitter runs over non-overlapping windows and implements a k-nearest-neighbour algorithm to correct reads at SNP loci. It finally clusters the reads using the Chinese Whispers algorithm [22] in order to identify (and assemble) haplotypes.

The local clustering of reads in strain-specific partitions is at the core of methods for the strain separation problem. In this article, we introduce and study an original mathematical formulation for this specific task. More precisely, we model the problem as a *tiling problem* on a binary matrix defined over a set of SNP positions. The rationale is to identify high (or low) density sub-matrices representing SNPs in which reads share the same nucleotides (within a given tolerance for errors). We formalize this problem as an Integer Linear Programming (ILP) problem and we integrate it within HairSplitter [14], which implements a complete strain-level metagenome assembly pipeline. We named the resulting method strainMiner. This article completes and extends a previously published conference paper [23].

We evaluated strainMiner on mock communities (based on real and simulated data) in which it either improved or compared favorably with respect to competing tools to recover strain-specific sequences from long read sequencing data, while being either an order of magnitude faster or more memory-efficient.

2 Methods

2.1 Pipeline overview

The strainMiner pipeline takes in input a reference sequence (a draft assembly or a reference genome) and a set of long reads. It then mirrors the four main stages of HairSplitter [14] (see Figure 1). These stages include: (i) aligning the reads to the draft assembly or reference genome, (ii) identifying putative SNPs and partitioning the reads, (iii) producing haplotype assemblies from the read partitions, and (iv) enhancing assembly contiguity through scaffolding. Our contribution lies in an original method to tackle the second step of the pipeline, that is separating aligned reads by

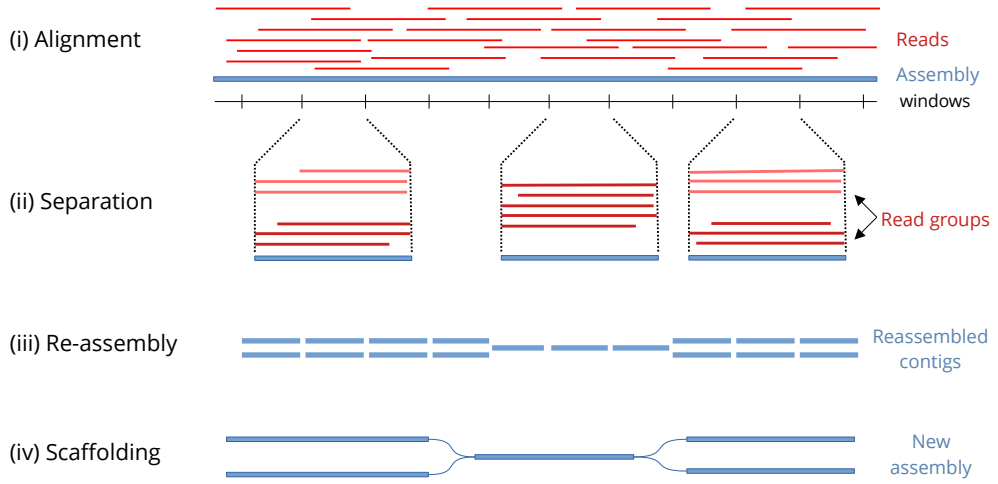


Fig. 1 The strainMiner pipeline [23]: (i) Reads are aligned on the reference or draft assembly, (ii) on each window of the assembly, reads are separated by haplotype of origin - only three windows are shown here, (iii) all groups of reads are locally reassembled and (iv) the locally reassembled contigs are scaffolded to produce longer contigs.

haplotype of origin. The other steps of the pipeline are the same ones implemented in HairSplitter [14] (version 1.6.10).

The problem we are tackling can be defined as follows: given a set of reads aligned to a reference sequence, the goal is to separate the reads into groups based on their haplotype of origin. Ideally, all reads originating from the same strain would be grouped together. However, this level of separation across the entire genome is not always achievable. It is impossible to phase two consecutive variants if they are too far apart to be covered by at least one read. Therefore, our objective is to partition reads locally. Specifically, we consider the input reference in non-overlapping windows of length w, where w should be smaller than the average read length (we set w to 5000 by default).

2.2 Statistical signal

The intuition behind strainMiner is similar to the one behind HairSplitter [14], originally introduced in [24]. The idea is to consider multiple loci simultaneously in order to group reads by their haplotypes of origin. Considering only one locus is not sufficient, as alignment artifacts and error rates can introduce errors at a single locus that cannot be distinguished from alternative alleles, especially when the alternative alleles are rare. On the contrary, by exploiting the correlation between several columns, it is possible to differentiate errors from true polymorphisms.

For instance, consider a hypothetical scenario involving a mixture of two strains, where strain A constitutes 99% of the mix and strain B a mere 1%. Consider also a collection of a thousand reads spanning two polymorphic sites, denoted as a and b.

In an ideal, error-free pileup, conducting a chi-square test for independence with one degree of freedom between the two loci yields a p-value smaller than 10^{-215}.

The presence of errors can greatly decrease the statistical power of detecting correlations between loci. In our simulations, we incorporated random substitution errors with a probability of $p = 0.1$ for all bases across 10,000 simulations. Despite this, the p-value for the correlation between loci a and b remained low, with an average of 10^{-16} and a maximum of 10^{-6} in the worst-case scenario. However, it is crucial to note that thousands of non-polymorphic positions may potentially exhibit correlations with locus a in a single pileup. Furthermore, alignment artifacts can introduce more complex errors with locally higher error rates, which can further decrease the statistical power of detecting correlations.

Including more loci unequivocally eliminates the risk of spurious correlations stemming from artifacts. In this same example, we introduced a third locus, denoted as c, while maintaining the 0.1 error rate. We applied the one-degree of freedom chi-square test to assess the relationship between the three positions. This time, the probability of encountering three non-polymorphic positions with correlations as strong as those observed between a, b, and c by chance was found to be below 10^{-200} in all 10,000 simulations. While this example simplifies the complexities of pileup errors, it emphasizes two fundamental aspects of the method: a) the joint observation of multiple loci significantly enhances the statistical power to distinguish between errors and polymorphism, and b) even low-abundance strains can be reliably identified.

2.3 Reference windows as binary matrices

In each window, strainMiner considers only reads that span at least 60% of the window's length. Subsequently, strainMiner transforms the read alignment pileup into a binary matrix, where each row corresponds to a read, each column corresponds to a position, and cell (i, j) contains the number one if the base at position j of read i matches the dominant base at that position, the number zero if it matches the second most frequent base, and remains empty otherwise. Empty cells can occur if a read does not cover a position or if the base in a read at a given position is not among the two most common bases at that position.

Next, columns are filtered to retain only those where the most common base constitutes at most a proportion p of the aligned reads. By default, strainMiner sets p to 0.95, striking a balance between computational efficiency and precision. However, users aiming to recover low-abundance strains can set p to a higher value.

To populate the empty cells, strainMiner implements the well-known K-nearest-neighbor imputation strategy [25], used for example in [26]. It identifies for each read its "nearest neighbors" which, in this context, are the reads with the smallest Hamming distance. Then, for each empty cell in a row, strainMiner uses a majority vote from the five closest neighbors that have non-empty cells at that position to decide whether the missing value is a 0 or a 1.

The result is a binary matrix $A = (a_{ij})$ of size $|R| \times |L|$, where the set of rows R corresponds to the reads, the set of columns L corresponds to the retained polymorphic loci/positions within the window, and $a_{ij} \in \{0, 1\}$ for all $1 \leq i \leq |R|, 1 \leq j \leq |L|$.

strainMiner aims to identify groups of highly similar columns in this matrix, which statistically can only represent true SNPs if the group is sufficiently large. Reads will then be separated into groups based on their alleles at these positions.

2.4 Definitions and problem formulation

In this section, we translate the statistical signal observed above in a formal problem on the binary matrix.

Quasi-bicluster of ones and zeros

To begin, we first introduce some preliminary definitions. The density $dens(A)$ of a binary matrix A is defined as the total number of ones divided by the total number of elements or, more formally,

$$dens(A) = \frac{\sum_{i \in R} \sum_{j \in L} a_{ij}}{|R| \times |L|}.$$

Furthermore, given a threshold $\gamma \in (0.5, 1]$, a *γ-bicluster of ones* (or *quasi-bicluster of ones*) is any sub-matrix M of A such that $dens(M) \geq \gamma$ holds. When γ is chosen to be close to 1, the corresponding matrix is called *dense*, i.e. a matrix mostly containing ones and tolerating only a small proportion of zeros to account for various sequencing errors. Conversely, a *$(1-\gamma)$-bicluster of zeros* (or *quasi-bicluster of zeros*) is a sparse binary matrix, i.e. characterized by low density (close to 0).

The quasi-bicluster dimensions are constrained by a minimum number of columns (width) and by a minimum number of rows (height), in order to ensure the significance of the statistical signal captured by the quasi-bicluster.

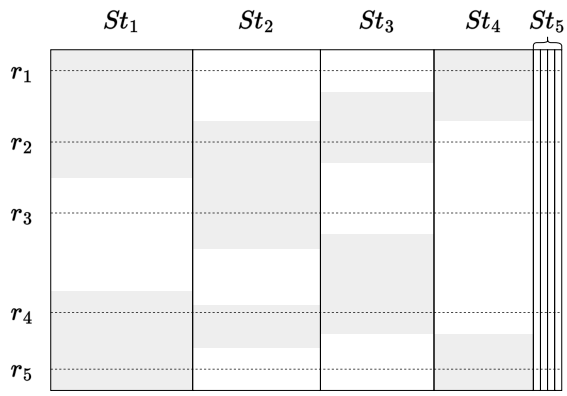
Tiling formulation

We approach the strain separation problem as a specific tiling problem: given the binary matrix A constructed as described in the previous section, we aim to permute and partition its columns into vertical bands, also referred to as *strips* (see Figure 2a). A strip is a group of columns where the rows can be divided into two groups — one with the dominant allele forming a bicluster of ones and the other with alternative alleles forming a bicluster of zeroes. Each strip partitions the reads into a group with the dominant allele and a group with the alternative allele.

By design, a strip groups multiple positions that are highly correlated. Larger strips include a greater number of correlated positions and likely witness the presence of actual SNPs in a multi-haplotype region. On the other hand, narrow strips (i.e., characterized by a number of columns below the width threshold for biclustering) are excluded because they lack sufficient statistical significance to confirm that their columns represent actual SNPs.

Read characteristic vectors

Given a set of strips it is then possible to characterize each read with a binary vector whose size is equal to the number of strips. If the i-th strip of a read is a quasi-bicluster



(a)

	St_1	St_2	St_3	St_4
r_1	1	0	0	1
r_2	1	1	1	0
r_3	0	1	0	1
r_4	1	1	1	0
r_5	1	0	0	1

(b)

Fig. 2 Separation of the domain into four vertical strips. (a) Each strip is a bipartition of rows. Quasi-biclusters of ones (resp. of zeros) are shown in gray (resp. white). Columns in strip 5 (St_5) are not considered because the width of this strip is below the specified threshold. **(b)** Characteristic vectors of the five reads/rows highlighted with a dashed line in the left figure. The comparison of the characteristic vectors results in three strains respectively labelled by a blue square, an orange triangle and a purple circle: the first strain contains r_1 and r_5, the second r_2 and r_4 and the third only r_3.

of ones, the i-th coefficient of this binary vector is set to 1; if not, it is set to 0. This vector is referred to as the *characteristic vector* of the corresponding read. Reads having identical characteristic vectors are grouped together as belonging to the same strain - in biological terms, this corresponds to grouping reads that are identical at all identified polymorphic loci. In practice, and as outlined in section 2.6.2, strains with a small number of reads are not considered, and their reads are possibly re-assigned to other strains.

Figure 2b provides an example of the characteristic vectors for a subset of five reads highlighted in Figure 2a. Among these reads, we observe three strains: r_1 and r_5 compose the first one, r_2 and r_4 the second, and the third strain only contains r_5.

Problem objective

Given three parameters corresponding to the desirable density γ (and sparsity $(1-\gamma)$), and two thresholds to indicate the minimum bicluster width and height, our strategy attempts to cover a maximum width of matrix A with strips.

2.5 A hybrid approach for finding strips

In order to find strips in the matrix, we have developed an Integer Linear Programming approach (denoted here as ILP-QBC), which will be detailed in later. In practice, however, the ILP-QBC does not scale well for large matrices. To overcome this issue, we decided to combine ILP-QBC with a preprocessing step, Hierarchical Cluster Analysis (HCA), a well-known technique frequently used in data mining, to reduce the size of the problem. Figure 3 explains the pipeline.

2.5.1 Strip identification with hierarchical clustering

The search of strips consists of the following two steps:

1. Hierarchical clustering of columns;
2. Checking if groups of columns are strips;

Hierarchical clustering of columns.

We utilize a standard hierarchical clustering to group columns and identify candidate strips. Specifically, we employ the Hamming distance and a complete-linkage strategy. Complete linkage defines the distance between two sub-matrices as the distance between their most distant columns (one from each sub-matrix). Groups of columns are iteratively merged until all groups have more than 35% divergence with all others. Groups of columns that do not satisfy the minimum required number of columns (5 by default) are not further processed by strainMiner.

Checking if groups of columns are strips

For each group of columns identified in the previous step, we apply HCA to partition the reads (rows) into two groups. In the best-case scenario, the group of columns is "unambiguous" and form a strip: the groups form a $(1-\gamma)$-bicluster of zeros and a γ-bicluster of ones. The identified strips are outputted and removed from the matrix, while the remaining "ambiguous" columns are provided as input to the ILP-QBC (Figure 3). The rationale behind this approach is that HCA very efficiently identifies "easy-to-spot" strips, reducing the size of the problem that needs to be solved by ILP-QBC.

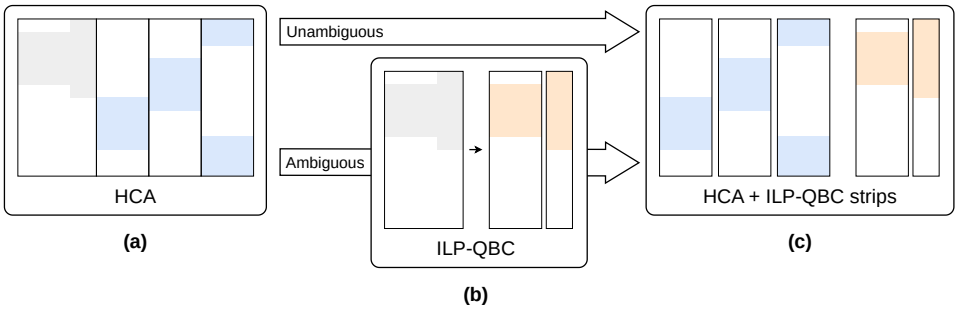


Fig. 3 Hybrid strip search HCA + ILP-QBC. In each of the sub-figure, the colored areas are dense binary sub-matrices. **(a)** The initial matrix with the groups of columns found via HCA. The first one (grey areas) is not a strip, as the bipartitioning does not respect γ. The other three (blue tiles) are strips. **(b)** The remaining matrix is sent to ILP-QBC to find strips respecting γ. In this example, ILP-QBC finds two strips (orange tiles). **(c)** The final set of strips is composed of the HCA-based strips and the ones found by ILP-QBC.

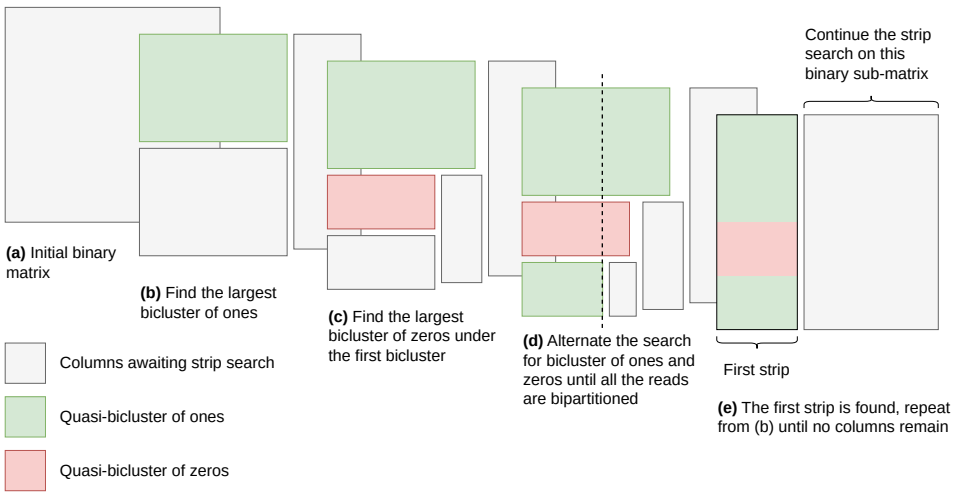


Fig. 4 The ILP-QBC strip iterative search strategy [23].

2.5.2 Strip identification with Integer Linear Programming

Our method, Integer Linear Programming for Quasi-Bicliques identification (ILP-QBC) iteratively identifies strips until all columns of the input matrix have been processed (see Figure 4).

We address the problem of identifying a strip heuristically and iteratively as follows:

1. The largest quasi-bicluster of ones in the matrix is found (Figure 4b);
2. Consider the sub-matrix restricted to the columns of the found quasi-bicluster but consisting of the reads that are not included in it (see the gray sub-matrix highlighted in Figure 4b);. Find the largest quasi-bicluster of zeros (Figure 4c);
3. Restrict the matrix and repeat steps from 2 alternating the search of quasi-bicluster of ones and zeros until no reads remain to be partitioned (Figure 4d).
4. Stop when no further quasi-bicluster can be found (given γ and the minimum height and width)

For the strip under consideration, if the number of remaining reads is below a certain threshold (5 in our case), the strip is considered valid – the remaining reads corresponding to particularly noisy reads. Each remaining read is assigned to the quasi-bicluster of ones if it has a majority of ones; otherwise, it is assigned to the quasi-bicluster of zeros. The columns retained in the last iteration define the width of the strip (Figure 4e). They are removed from the matrix and the search for another strip begins.

If the number of remaining columns is above the threshold, it means that no wide enough strip could be found in the matrix, and the search for strips stops. The remaining columns often correspond to loci that are not polymorphic but particularly prone to sequencing errors (e.g. homopolymers).

To find quasi-bicluster of ones or of zeros, we employ an Integer Linear Programming (ILP) model.

ILP model

Given a binary matrix, the purpose of our ILP model is to select a group of rows and columns that maximizes the count of a specific value (either zeros or ones) respecting the γ parameter. For a given matrix $A \in \mathbb{Z}_2^{|R|\times|L|}$ and a threshold γ, we use binary variables x_{ij}, u_i and v_j to denote the selection (value equals 1) or non-selection (value equals 0) of a cell, row, and column, respectively. The following ILP searches for a quasi-bicluster of ones:

$$\max \sum_{i\in R}\sum_{j\in L} a_{ij}x_{ij} \tag{1}$$

$$x_{ij} \leq u_i, \ \forall i \in R, \forall j \in L \tag{2}$$

$$x_{ij} \leq v_j, \ \forall i \in R, \forall j \in L \tag{3}$$

$$x_{ij} \geq u_i + v_j - 1, \ \forall i \in R, \forall j \in L \tag{4}$$

$$\sum_{i\in R}\sum_{j\in L}(1-a_{ij})x_{ij} \leq (1-\gamma) \times \sum_{i\in R}\sum_{j\in L} x_{ij} \tag{5}$$

$$u_i, v_j \in \{0,1\}, \ x_{ij} \in \{0,1\} \quad \forall i \in R, \ \forall j \in L \tag{6}$$

The function to maximize (1), counts for the number of ones in a sub-matrix determined by the binary variables having value 1. Constraints (2), (3), (4) mean that cell (i, j) is selected into the solution (i.e., $x_{ij} = 1$) if and only if both its corresponding row i and column j are also included into the solution (i.e., $u_i = 1$ and $v_j = 1$).

The coefficient a_{ij} represents the value of the cell at position i and j. It is directly used when searching for occurrences of 1s in the matrix. When the search is for 0s, however, the coefficient is reversed to $(1 - a_{ij})$ as follows:

$$\max \sum_{i\in R}\sum_{j\in L}(1-a_{ij})x_{ij} \tag{7}$$

Constraint (5) ensures that the sub-matrix contains at least a proportion γ of ones. This constraint can also be reversed when necessary to ensure a minimum proportion of zeros:

$$\sum_{i\in R}\sum_{j\in L} a_{ij}x_{ij} \leq (1-\gamma) \times \sum_{i\in R}\sum_{j\in L} x_{ij} \tag{8}$$

Complexity analysis

Biclustering is closely related to bipartite graph partitioning. If we consider reads and positions as vertices of a graph and our binary matrix as the adjacency matrix of this graph, finding a submatrix of ones in the adjacency matrix is equivalent to finding a biclique in a bipartite graph. Finding the maximum biclique is an NP-hard problem [27]. Our problem is even more challenging, as we allow a proportion γ of non-existing links, effectively searching for a maximum quasi-biclique.

2.6 From strips to strain groups

2.6.1 Read characteristic vectors

The final strip set contains the strips generated both from HCA and ILP-QBC. Each strip corresponds to a biclustering of the rows: a row is either labelled one or zero. We associate a binary characteristic vector to each read, where the vector's length corresponds to the number of strips (see Figure 2).

2.6.2 Definition of strain groups

Two reads participate in the same strain group if they have identical characteristic vectors - biologically, this means that these reads are identical at all polymorphic positions. In practice, however, some groups have fewer reads than a specified threshold (5 in our case). These orphan reads generally correspond to noisy reads that have been erroneously grouped in some strips. They are rescued by being reassigned them to bigger groups.

Reassigning orphan reads

To reassign reads, we consider the binary sub-matrix induced by the reads of each big group. The process can be summarized as follows:

1. Compute the representative binary row-vector V_g of each big group g. The j-th column of the row-vector equals to:

$$V_g[j] = \left\lfloor \frac{\sum_{0 \leq i < m} A_g[ij]}{m} \right\rceil$$

   where $A_g \in \mathbb{Z}_2^{m \times |L|}$ is the binary matrix induced by group g with m reads, and $\lfloor x \rceil$ rounds x to its nearest integer.
2. Compute the Hamming distance between all the orphan reads and all the representatives V_g.
3. For each orphan read and its Hamming-distance-closest-group g:
   - if the Hamming distance is less than a given threshold (here 0.1), assign the read to group g;
   - otherwise, do not assign the read to any group.

2.7 Producing a strain-level assembly

At this point, reads are partitioned into strain groups on all windows.

Merging windows

To simplify the assembly process, consecutive identical windows are combined. Specifically, two consecutive windows are deemed identical if they partition the reads into the same groups. In practical terms, two groups are considered identical if more than

70% of the reads from the group in the first window are also found in a corresponding group in the second window, and vice versa (considering only the reads present in both windows). These identical windows are then merged to create longer windows.

Reconstructing the sequences

In each window, a new contig is generated for each group of reads. These contigs are computed using Racon [28] to polish the corresponding (strain-oblivious) reference sequence towards the strain-specific sequence the group of reads represents. Consequently, each original contig from the draft assembly can be divided into multiple windows, and each window can contain several strain-specific contigs (one for each strain). Finally, the resulting contig graph is processed through GraphUnzip [29] to resolve repeats and generate a more contiguous final set of contigs.

3 Results

3.1 Datasets

We decided to evaluate strain-level assembly methods on mock communities based on real and simulated datasets of varying complexity in terms of error rate, number of strains, and level of abundance. This approach allows us to estimate the performance of the methods in a controlled scenario where the sequences of the ideal output are known a priori.

The first dataset we considered is a mixture of five *Vagococcus fluvialis* strains barcoded and sequenced in [30]. These genomes were sequenced with barcodes using a R9.4.1 Nanopore flowcell. By ignoring the barcodes, we obtain a simple mock community containing five different strains of roughly the same abundance. Specifically, this dataset is characterized by three strains whose genomes are almost identical, likely turning the problem into the distinction of three *V. fluvialis* strains, one of which is dominant.

The second dataset is the Zymobiomics gut microbiome standard, a mock community sequenced independently with Nanopore 10.4.1 (error rate of 2.5%) and Nanopore R9.4.1 (error rate of 5%) flowcells. These two samples are available in the European Nucleotide Archive (ENA) with accession numbers `SRR17913199` and `SRR17913200`, respectively. This community consists of 21 genomes of bacteria, archea and yeast, with high variability in terms of abundance. Among the bacteria there are five different strains of *Escherichia coli* characterized by equal abundance, making these samples an ideal dataset for comparing strain separation techniques with two different error rates.

The third type of dataset is simulated and aims to evaluate strainMiner not only with respect to a higher number of strains but also in presence of a strain present at different levels of abundance. More precisely, we adopted a protocol similar to the one outlined in [4] and [14]. We thus simulated a mock community based on 10 strains of *Escherichia coli* to investigate the impact of strain coverage on the ability to recover strain genomes. The *E. coli* strains are the same as those previously employed to evaluate HairSplitter [14] and their complete reference sequences were retrieved from NCBI. More precisely, these include the strains 12009 (GCA_000010745.1), IAI1 (GCA_000026265.1), F11 (GCA_018734065.1), S88

(GCA_000026285.2), Sakai (GCA_003028755.1), SE15 (GCA_000010485.1), UMN026 (GCA_000026325.2), HS (GCA_000017765.1), K12 (GCF_009832885.1), and *Shigella flexneri* (GCF_000006925.2). Then, for each reference sequence, we simulated a 50X depth of coverage of Nanopore reads with 5% error rate. Reads were generated with Badreads [31] using the "nanopore2023" model. All the set of simulated reads were then merged into a single mixture. In order to evaluate the influence of coverage on assembly completeness, we created four additional 10-strain mixures by downsampling exclusively the 12009 strain at 30X, 20X, 10X, and 5X. All simulated reads are available at https://zenodo.org/records/10362565.

3.2 Assembly of the datasets

All datasets were assembled using metaFlye (v2.9.2-b1786) with parameters `--meta` and `--nano-raw`. We chose metaFlye as it is the only long-read metagenome assembler thought to work with Oxford Nanopore reads. The obtained (species-level) assembly was used as input for all the strain-aware assemblers we evaluated. Moreover, an alignment of the reads against the input assembly was produced with Minimap2 [32] (parameter `-x map-ont`) for the software that required it.

We ran the latest available versions of Strainberry (v1.1), Floria (v0.0.1), HairSplitter (v1.9.4), and strainMiner (v1.6.10) with default parameters. Since Strainberry and Floria have been designed to phase no more than 5 strains by default, when running on the 10-strain *E. coli* datasets, we additionally provided the parameters `-n 10` and `-p 10`, respectively, to increase this limit.

It is important to note that Floria does not perform any SNP calling nor does it output a base-level assembly, but rather provides a collection of haplotype-resolved read clusters. SNP calling was carried out using Longshot [33] (v1.0.0). Base-level assemblies of each read cluster were, instead, produced with wtdbg2 [34] (v2.5) following the assembly pipeline suggested in Floria's documentation. The set of assembled haplotypes was finally complemented with the metaFlye contigs that were not phased (i.e., not part of Floria's output).

3.3 Evaluation Metrics

Evaluating metagenome assemblies is a complex task, specifically in a strain-aware context, due to the limited knowledge of the organisms within a metagenomic sample. Standard qualitative metrics should in fact be treated with caution as they might be the result of strain differences rather than errors. For this reason, we assessed the performance of the strain-aware assembly methods on real and simulated mock communities for which we precisely know the sequences of the strains we aim to reconstruct.

For each generated assembly, we computed the following metrics: assembly size, N50, reference fraction percentage, duplication ratio, number of misassemblies, number of mismatches, and number of indels. The assembly size and N50 (*i.e.*, the length such that all contigs of that length or longer cover at least half of the assembly size) provide a quantitative view of the assembly. The other metrics, instead, provide a more qualitative assessment.

Table 1 MetaQUAST metrics on the real and simulated datasets. For each assembly we report the following metrics: assembly size (Total size), N50, reference fraction, duplication ratio (Dup. ratio), number of misasseblies (# Mis.), number of mismatches per 100 kb, and number of indels per 100 kb. The best values among of the four strain-separated assemblies is in bold font.

	Assembler	Total size (Mb)	N50 (kb)	Reference frac. (%)	Dup. ratio	# Mis.	Mismatches /100 kb	Indels /100 kb
V. fluvialis (14 Mb)	metaFlye	4.57	151	26.9	1.036	40	360.12	406.10
	Strainberry	5.71	104	33.1	1.112	45	78.19	513.38
	Floria	8.03	**142**	47.7	1.117	**20**	102.61	636.40
	HairSplitter	**9.34**	74	**58.2**	**1.066**	31	102.49	410.95
	strainMiner	8.84	30	54.5	1.080	48	**50.80**	**340.26**
Zymo Q9 (78 Mb)	metaFlye	65.9	1797	63.0	1.001	136	96.97	58.52
	Strainberry	**79.9**	**224**	61.2	1.170	122	133.31	110.07
	Floria	76.0	64	58.6	1.212	114	230.67	245.47
	HairSplitter	94.6	33	**76.7**	1.179	114	115.38	76.49
	strainMiner	76.7	220	73.4	**1.044**	**95**	**69.90**	**55.46**
Zymo Q20 (78 Mb)	metaFlye	60.4	388	60.2	1.007	142	115.86	76.49
	Strainberry	69.4	**117**	69.4	1.037	141	109.49	67.46
	Floria	**71.0**	59	70.7	1.050	117	80.41	79.81
	HairSplitter	69.2	68	**70.9**	**1.013**	113	69.17	66.36
	strainMiner	69.5	51	70.8	1.022	**109**	**67.98**	**65.14**
E. coli 10 strains (48 Mb)	metaFlye	14.0	70	25.2	1.029	94	477.68	310.42
	Strainberry	19.6	**63**	34.2	**1.082**	175	367.48	138.68
	Floria	35.3	50	60.3	1.155	**147**	171.95	104.12
	HairSplitter	54.8	44	**93.6**	1.182	311	87.12	**52.30**
	strainMiner	**50.1**	56	91.1	1.127	335	**81.73**	72.80

These metrics were obtained with MetaQUAST [35] (v5.2.0) with the option `--unique-mapping` to ensure that each assembled sequence (a contig or part of it) is mapped exclusively to the best location among the reference sequences. This is crucial because MetaQUAST, which relies on sequence alignment, may suffer from sub-optimal mappings on very similar references, such as strains of the same species. For this reason, we complemented MetaQUAST's evaluation metrics by computing the k-mer completeness ($k = 27$) with KAT [36] (v2.4.2). This metric represents the percentage of k-mers in the reference genomes that are also present within an assembly and provides a more accurate view on the strain-specific content that was successfully recovered.

3.4 Assembly evaluation

Table 1 summarizes the MetaQUAST metrics computed for each strain-level assembly tool on each of the evaluated datasets. To manage the size of the table, in the case of the 10-strain *E. coli* datasets, we display only the one in which all strains have the same abundance. The downsampling experiments, however, exhibited similar statistics.

On the *V. fluvialis* dataset, strainMiner is able to yield the lowest amount of mismatches and indels, while HairSplitter is able to achieve the best results in terms

of assembly size, reference fraction, and duplication ratio. strainMiner however is on par with HairSplitter with respect to these metrics. Floria on the other hand is able to provide the most contiguous assembly (N50 equal to 142 kbp) and the lowest number of misassemblies at the expense of a more duplicated and less accurate assembly. While displaying an average performance on most the evaluation metrics, Strainberry was only able to recover the 33% of the reference sequences. The high reference fraction of both strainMiner and HairSplitter, compared to the other tools, is also confirmed by the highest k-mer completeness (Figure 5).

On the Q9 and Q20+ versions of Zymobiomics datasets, results follow the same trend as for the *V. fluvialis* dataset. Specifically, strainMiner generates the most accurate assembly, as witnessed by the lowest number of misassemblies, mismatches, and indels. In terms of contiguity, Strainberry achieves the highest N50 but also at the expense of a high number of misassemblies. There are however some key differences for the assemblies generated with the Q9 and Q20+ Nanopore reads. In the first case, strain-level assemblers are characterized by a lower reference fraction and higher duplication ratio. The only exception is strainMiner, which is able to provide comparable results, thus proving to be more tolerant to lower error rates. As for the *V. fluvialis*, the k-mer completeness is consistent with the highest reference fraction of strainMiner and HairSplitter for both the Q9 and Q20+ datasets (Figure 5).

The simulated 10-strain *E. coli* dataset offered a more challenging scenario due to the presence of a higher number of closely related genomes. Table 1 shows that strainMiner and HairSplitter are the only two tools able to achieve a high reference fraction ($> 90\%$), thus recovering almost completely the 10 different strains. They are also the two tools with the lowest number of mismatches and indels (as for the other datasets). Strainberry and Floria, on the other hand, display a much lower reference coverage (34.2% and 60% respectively) and seem to struggle with high numbers of conspecific strains.

Finally, the right-hand side of Figure 5 shows the k-mer completeness of the 10-strain dataset in which the *E. coli* 12009 strain is characterized by different level of abundance. As expected, strainMiner and HairSplitter exhibit a better ability to retrieve strains with low coverage, even when it is as low as 5X (representing only 1.1% of the total mix), and displayed a k-mer completeness always higher than 0.9. Nevertheless, a positive correlation between the coverage and completeness is noticeable for all the evaluated methods.

Overall, there is no tool that outperforms the others with respect to all evaluation metrics on the different datasets. All the evaluated tools offer different trade-offs between contiguity and accuracy. Floria, for example, is able to achieve higher contiguity and a comparable number of misassemblies at the expense of a lower base-level accuracy and reference coverage. On the other hand, when looking at qualitative metrics, strainMiner generates more accurate assemblies (or comparable with the other competing tools) and a low amount of duplicated sequences, especially with higher error rates.

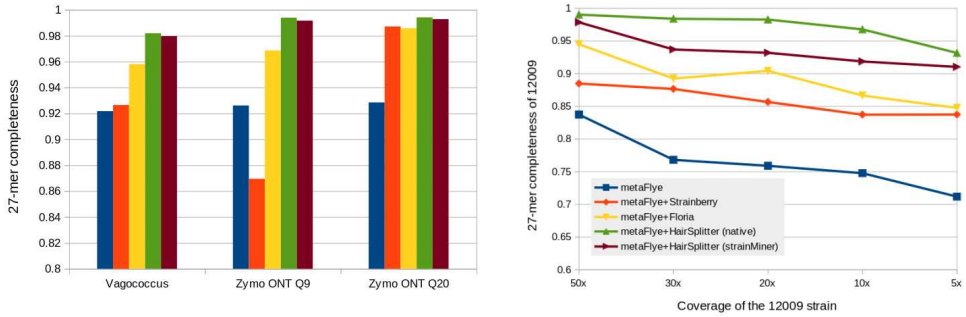


Fig. 5 Assembly k-mer completeness. On the left-hand side, the 27-mer completeness of assemblies obtained from real sequencing data is depicted. The level of completeness in the Zymo communities is based exclusively on the *E. coli* genomes within the samples. On the right-hand side, the 27-mer completeness of the 12009 strain is shown for the 10-strain simulated *E. coli* datasets. The analysis was performed for varying depth of coverage of the 12009 strain (x-axis), while the other nine strains were consistently kept at 50X.

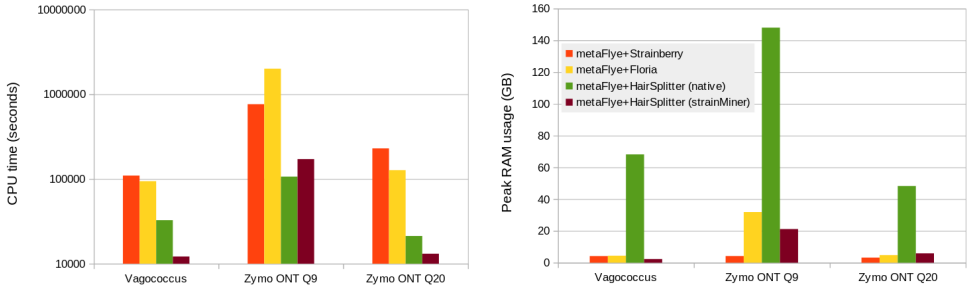


Fig. 6 CPU time and maximum memory usage. Comparison of Strainberry, Floria, HairSplitter, and strainMiner on the three real datasets in terms of CPU time (left) and maximum memory usage (right). The legend applies to both the plots.

3.5 Resource usage

All results were obtained on a server housing 16 Intel Xeon CPUs with four cores each, running at 2.7 GHz. 3.1 TB of RAM was available.

Although strainMiner, like the other competing tools, is trivially parallel, we ran it on a single thread due to limitations imposed by the Gurobi license. For this reason we decided to simply report CPU times.

Across all datasets, strainMiner consistently exhibited a processing speed more than tenfold faster than Strainberry, while its runtime was approximately on par with that of HairSplitter (Figures 6).

On the real sequencing data, strainMiner used between 7 and 30 times less peak memory than HairSplitter (Figure 6).

As a whole, strainMiner significantly diminishes the memory usage of the HairSplitter pipeline without impacting negatively on its speed and arguably improving the quality of the assembly.

4 Discussion

In this work we introduced strainMiner, a method to assemble individual strain genomes from metagenomic sequencing data. Unlike other metagenome assembly methods, strainMiner is based on a novel formulation of the "strain separation" problem as a tiling problem on a binary matrix. We proposed and implemented an Integer Linear Programming (ILP) model in order to efficiently partition such a matrix and to cluster sequences (reads) that likely belong to the same haplotype. The ILP-based formulation allows us to exploit a well-established and highly optimized solver such as Gurobi. This, along with the use of heuristics inspired from data mining, allow strainMiner to require considerably less time and memory compared to other competing software.

In order to assess its capability to distinguish and reconstruct strains, strainMiner is implemented as a fork of HairSplitter (a previously developed tool for strain-level metagenome assembly), from which we replaced the read-clustering step with our approach. On both real and simulated datasets, strainMiner compared favorably to state-of-the-art methods in terms of strain recovery and base-level accuracy. We also showed that strainMiner's output is less affected by reads with high error rates or metagenomes characterized by a high number of distinct strains. At the same time, strainMiner is able to recover strains with a depth of coverage as low as 5X.

Nevertheless, the assemblies obtained with strainMiner had often much lower contiguity compared to the one obtained with HairSplitter. This could be due to the fact that strainMiner is based on the version 1.6.10 of HairSplitter, while in our comparison we used the latest available version (1.9.4). Upgrading strainMiner to this version might further improve output's contiguity. Moreover, it is also possible that the properties of the read clusters computed by strainMiner could be quite different compared to those obtained with HairSplitter. A tailored scaffolding step could thus improve contiguity and would merit further investigation. A possible approach would be to consider overlapping fixed-length windows. As a matter of fact, the use of non-overlapping windows generates clean input matrices for the ILP solver but completely relies on the GraphUnzip module of HairSplitter to improve assembly contiguity. Considering overlapping windows, however, could better take advantage of the co-occurrence of strain-specific nucleotides, allowing to identify longer haplotypes beforehand.

Finally, one additional limitation is the use of the Gurobi solver: an academic license is free but limited to three instances of Gurobi running at the same time. Attempts to use the free CBC solver showed a decrease in performance.

Software availability

strainMiner is open source and available online with the GPL3 licence. The strainMiner used to generate the results is available at https://github.com/rolandfaure/strainminer. A more recent version, under development, is available at https://gitlab.com/haplotype-tiling/strainminer-py.

Acknowledgements

We wish to thank Dominique Lavenier, who formulated the first version of the optimization problem.

We acknowledge the GenOuest bioinformatics core facility https://www.genouest.org for providing the computing infrastructure.

References

[1] Ghurye, J. S., Cepeda-Espinoza, V. & Pop, M. Metagenomic assembly: Overview, challenges and applications. *The Yale journal of biology and medicine* **89**, 353–362 (2016).

[2] Almeida, A. *et al.* A unified catalog of 204,938 reference genomes from the human gut microbiome. *Nature biotechnology* **39**, 105–114 (2021).

[3] Olson, N. D. *et al.* Metagenomic assembly through the lens of validation: recent advances in assessing and improving the quality of genomes assembled from metagenomes. *Briefings in bioinformatics* **20**, 1140–1150 (2019).

[4] Vicedomini, R., Quince, C., Darling, A. E. & Chikhi, R. Strainberry: automated strain separation in low-complexity metagenomes using long reads. *Nature Communications* **12**, 4485 (2021). URL https://www.nature.com/articles/s41467-021-24515-9.

[5] Albanese, D. & Donati, C. Strain profiling and epidemiology of bacterial species from metagenomic sequencing. *Nature communications* **8**, 2260 (2017).

[6] Sonnenborn, U. Escherichia coli strain nissle 1917—from bench to bedside and back: history of a special escherichia coli strain with probiotic properties. *FEMS microbiology letters* **363**, fnw212 (2016).

[7] Frank, C. *et al.* Epidemic profile of shiga-toxin–producing escherichia coli o104:h4 outbreak in germany. *New England Journal of Medicine* **365**, 1771–1780 (2011). URL https://doi.org/10.1056/NEJMoa1106483. PMID: 21696328.

[8] Quince, C. *et al.* DESMAN: a new tool for de novo extraction of strains from metagenomes. *Genome Biology* **18**, 181 (2017). URL http://genomebiology.biomedcentral.com/articles/10.1186/s13059-017-1309-9.

[9] Quince, C. *et al.* Metagenomics Strain Resolution on Assembly Graphs. preprint, Bioinformatics (2020). URL http://biorxiv.org/lookup/doi/10.1101/2020.09.06.284828.

[10] Baaijens, J., Aabidine, A., Rivals, E. & Schönhuth, A. De novo assembly of viral quasispecies using overlap graphs. *Genome Research* (2017).

[11] Kang, X., Luo, X. & Schönhuth, A. StrainXpress: strain aware metagenome assembly from short reads. *Nucleic Acids Research* **50**, e101–e101 (2022). URL https://academic.oup.com/nar/article/50/17/e101/6625806.

[12] Kazantseva, E., Donmez, A., Pop, M. & Kolmogorov, M. stRainy: assembly-based metagenomic strain phasing using long reads. preprint, Bioinformatics (2023). URL http://biorxiv.org/lookup/doi/10.1101/2023.01.31.526521.

[13] Shaw, J., Gounot, J.-S., Chen, H., Nagarajan, N. & Yu, Y. W. Floria: Fast and accurate strain haplotyping in metagenomes. *Bioinformatics* **40**, i30–i38 (2024).

[14] Faure, R., Lavenier, D. & Flot, J.-F. Hairsplitter: haplotype assembly from long, noisy reads. *bioRxiv* (2024). URL https://www.biorxiv.org/content/early/2024/06/14/2024.02.13.580067.

[15] Bertrand, D. *et al.* Hybrid metagenomic assembly enables high-resolution analysis of resistance determinants and mobile elements in human microbiomes. *Nature Biotechnology* **37**, 937–944 (2019). URL http://www.nature.com/articles/s41587-019-0191-2.

[16] Kolmogorov, M. *et al.* metaFlye: scalable long-read metagenome assembly using repeat graphs. *Nature Methods* **17**, 1103–1110 (2020). URL https://www.nature.com/articles/s41592-020-00971-x.

[17] Bickhart, D. M. *et al.* Generating lineage-resolved, complete metagenome-assembled genomes from complex microbial communities. *Nature biotechnology* **40**, 711–719 (2022).

[18] Feng, X., Cheng, H., Portik, D. & Li, H. Metagenome assembly of high-fidelity long reads with hifiasm-meta. *Nature Methods* **19**, 1–4 (2022).

[19] Benoit, G. *et al.* High-quality metagenome assembly from long accurate reads with metamdbg. *Nature Biotechnology* 1–6 (2024).

[20] Bansal, V. Hapcut2: A method for phasing genomes using experimental sequence data. *Methods in molecular biology* **2590**, 139–147 (2022).

[21] Raghavan, U. N., Albert, R. & Kumara, S. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* **76**, 036106 (2007).

[22] Biemann, C. Chinese whispers: An efficient graph clustering algorithm and its application to natural language processing problems. *Proceedings of TextGraphs* 73–80 (2006).

[23] Truong, T. K. M., Faure, R. & Andonov, R. *Assembling close strains in metagenome assemblies using discrete optimization*, 15th International Conference on Bioinformatics Models, Methods and Algorithms, BIOINFORMATICS, February 21-23, 2024, Rome, Italy. URL https://bioinformatics.scitevents.org.

[24] Feng, Z., Clemente, J., Wong, B. & Schadt, E. Detecting and phasing minor single-nucleotide variants from long-read sequencing data. *Nature Communications* **12**, 3032 (2021).

[25] Fix, E. & Hodges, J. L. Discriminatory analysis. nonparametric discrimination: Consistency properties. *International Statistical Review / Revue Internationale de Statistique* **57**, 238–247 (1989). URL http://www.jstor.org/stable/1403797.

[26] Troyanskaya, O. *et al.* Missing value estimation methods for dna microarrays. *Bioinformatics* **17**, 520–525 (2001). URL https://doi.org/10.1093/bioinformatics/17.6.520.

[27] Peeters, R. The maximum edge biclique problem is NP-complete. *Discrete Applied Mathematics* **131**, 651–654 (2003).

[28] Fang, L. & Wang, K. Polishing high-quality genome assemblies. *Nature Methods* **19**, 649–650 (2022). URL https://www.nature.com/articles/s41592-022-01515-1.

[29] Faure, R., Guiglielmoni, N. & Flot, J.-F. Graphunzip: unzipping assembly graphs with long reads and hi-c. *bioRxiv* 2021–01 (2021).

[30] Rodriguez Jimenez, A. *et al.* Comparative genome analysis of vagococcus fluvialis reveals abundance of mobile genetic elements in sponge-isolated strains. *BMC Genomics* **23** (2022).

[31] Wick, R. Badread: simulation of error-prone long reads. *Journal of Open Source Software* **4**, 1316 (2019). URL http://joss.theoj.org/papers/10.21105/joss.01316.

[32] Li, H. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics* **34**, 3094–3100 (2018). URL https://academic.oup.com/bioinformatics/article/34/18/3094/4994778.

[33] Edge, P. & Bansal, V. Longshot enables accurate variant calling in diploid genomes from single-molecule long read sequencing. *Nature communications* **10**, 4660 (2019).

[34] Ruan, J. & Li, H. Fast and accurate long-read assembly with wtdbg2. *Nature methods* **17**, 155–158 (2020).

[35] Mikheenko, A., Saveliev, V. & Gurevich, A. Metaquast: Evaluation of metagenome assemblies. *Bioinformatics* **32**, btv697 (2015).

[36] Mapleson, D., Accinelli, G., Kettleborough, G., Wright, J. & Clavijo, B. Kat: A k-mer analysis toolkit to quality control ngs datasets and genome assemblies. *Bioinformatics (Oxford, England)* **33** (2016).

Conclusion

The three articles highlight the extensive process behind the development of HairSplitter. In my opinion, the main practical contribution of this work is the HairSplitter software itself, while the main theoretical contribution is the statistical test proposed for variant calling in the PCI article. HairSplitter is an efficient tool that significantly improves upon the state of the art. The variant calling procedure is innovative and robust, handling uneven coverage, a high number of strains, and varying error rates effectively. As a software, the strainMiner version of HairSplitter does not offer significant advantages over HairSplitter, especially since I have continuously improved HairSplitter whenever strainMiner outperformed it. Consequently, the results presented in the strainMiner article are already outdated, as the memory footprint of HairSplitter has been considerably reduced since those measurements were taken.

Chapter 3

Assembling high-fidelity reads

Abstract: This chapter focuses on the assembly of high-fidelity long reads (HiFi reads). It introduces a new sketching method derived from a published family of sequence transformations known as Mapping-friendly Sequence Reductions (MSRs). The chapter explores the properties of MSRs as a sketching technique and how they can be parameterized to enhance genome assembly. The main advantage of MSRs sketches in the context of this thesis is their ability to represent the small differences between haplotypes, compared to other sketching methods that involuntarily collapse haplotypes. We present a proof-of-concept assembler called Alice, based on these sketches. Alice is very fast and performs well in distinguishing the haplotypes of five strains of *Escherichia coli* in a sequencing of a mock dataset.

In 2019, scientists from Pacific Biosciences introduced a new sequencing technique with a median error rate of 0.1% [48]. This technique, called High-Fidelity (HiFi) sequencing, did not involve any major changes to the chemistry of existing PacBio sequencing methods. Instead, it combined PacBio sequencing with a technique previously experimented on short reads, called Consensus Circular Sequencing (CCS) [119, 120]. CCS involves circularizing a DNA molecule and sequencing it multiple times, generating several subreads that all sequence the same region. The consensus of all the subreads yields a high-quality read, despite the high error rate in each individual subread. This new sequencing technique quickly gained popularity and became known as PacBio HiFi sequencing.

Nowadays, the accuracy of ONT reads is improving to error rates around and below 1% and the definition of a "high-fidelity" (HiFi) read can be debated. For the purposes of this section, we will align with the PacBio HiFi standard and consider reads to be of high fidelity if they contain significantly less than 1% errors. We will use the term "HiFi" to refer to all types of long reads with this level of precision.

It may seem unnecessary to develop specific assemblers for HiFi reads: since they are shorter and more precise than traditional long reads, previously developed long read assemblers work without modifications. However, while noisy read assemblers tend to collapse highly similar regions to avoid mistaking a sequencing error for a SNP, HiFi-specific assemblers can take advantage of the low error rate to finely distinguish these

regions. The first HiFi-specific assemblers, HiCanu [115] and hifiasm [63], demonstrated improved contiguity and haplotype resolution compared to previous methods. As a result, HiFi sequencing has become the preferred method for producing high-quality genome assemblies [121, 122].

In recent years, numerous HiFi assemblers have been developed [123], but none have fully addressed the challenges of metagenome assembly. We will see that despite the high accuracy of HiFi reads, it remains difficult to distinguish and correctly assemble closely related strains and that, additionally, assemblers have room for improvement regarding computational efficiency.

This chapter will introduce a new approach for HiFi read assembly that aims to address both precision and efficiency issues. The radical novelty of the approach is to use a new sketching method, Mapping-friendly Sequence Reductions (MSRs), to compress sequencing reads and perform efficient assembly without sacrificing accuracy. The first part of the chapter will be a discussion on the properties of MSRs as a sketching method and the second part will present a new assembler, Alice.

3.1 Mapping-friendly Sequence Reductions as a sketching technique for assembly

3.1.1 Objective

Our objective in this section is to propose a new sketching method. Following the definition in [124], a sketch is a "compact data structure that approximates a data set." The challenge in creating a sketch is to maximize the compression of the original data while retaining as much biologically relevant information as possible. In our case, we will demonstrate that Mapping-friendly Sequence Reductions can be used to create reduced sequences that represent longer sequences, specifically HiFi reads, while preserving useful properties for alignment and assembly.

3.1.2 Definition of Mapping-friendly Sequence Reductions

The original motivation for the development of MSRs was to generalize the concept of homopolymer compression. Homopolymer compression involves compressing input reads by deleting consecutive identical bases (CGAATTC $\rightarrow$ CGATC), and then performing

alignment or assembly on the compressed reads. A challenge with this procedure is that the results are also outputted as compressed sequences, which need to be inflated back to obtain the final result (CGATC → CGAATTC), which is non-trivial. Despite this difficulty, the method has gained popularity because long-reads typically contain errors at repeated bases, hence compressed reads have a lower error rate than the original reads. The authors of [125] observed that other compression/decompression procedures could be applied to the reads and set out to generalise homopolymer compression to further improve alignment.

[125] defined Streaming Sequence Reductions as functions that transform a sequence of characters into a new sequence. It is defined by an alphabet (in this case, the DNA alphabet $\{A, C, G, T\}$), an order l (a positive integer), and a transforming function g that maps a sequence of length l, or l-mer, to either a character in the alphabet or a special "empty" character ϵ.

SSRs work by taking the input sequence and breaking it down into successive overlapping l-mers . l-mers are passed in a streaming fashion through the function g. If g returns a character, that character is added to the new sequence. If g returns the empty character ϵ, nothing is added to the new sequence. The pseudocode for this process is provided in Algorithm 1. Homopolymer compression can be seen as a particular SSR with $l = 2$ [1].

Algorithm 1 Streaming Sequence Reductions

```
Function MSR(seq, l, g)
new_seq =""
for i = 0 to len(seq) − l do
   lmer = seq[i : i + l]
   new_char = g(lmer)
   if new_char ≠ ϵ then
      new_seq = new_seq + char
   end if
end for
return new_seq
```

By design, if the length l is not too large, two highly similar sequences will share many l-mers in the same order, resulting in highly similar reduced sequences. Consequently, the reduced versions of two sequences that align have a high probability of aligning as well; we

1. To correctly describe homopolymer compression, a SSR would also need to add the $l-1$ first bases of the original sequence to the beginning of the compressed sequence. The original definition of the SSRs took this into account. However, this adds unwanted complexity in our case and we will work with a simplified definition that does not account for the first $l-1$ bases.

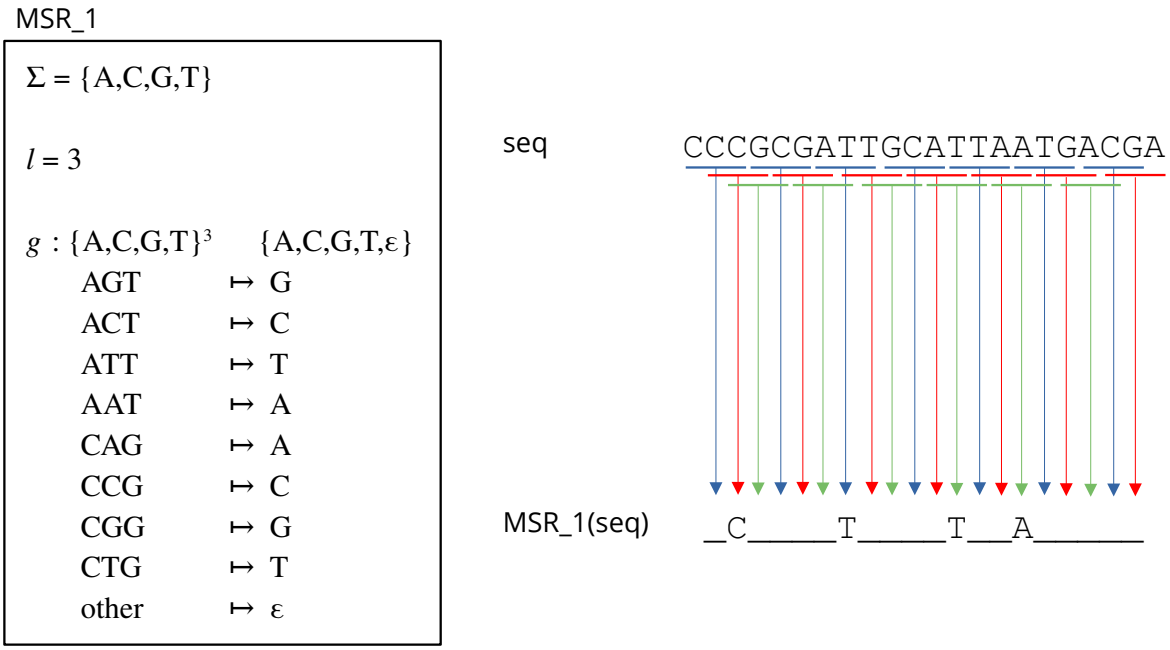


Figure 3.1 – Example of a mapping-friendly sequence reduction named MSR_1. Applied on sequence CCCGCGATTGCATTAATGACGA, MSR_1 yields the reduced sequence CTTA. The colors of the arrows on the right panel are there to help distinguish them and do not carry meaning.

refer to this property of the reduction as mapping-friendliness. To ensure that a Sequence Sketch Reduction (SSR) is mapping-friendly in the context of bioinformatics, where a sequence and its reverse complement must be considered identical, an additional constraint must be imposed on the function g: the image of two reverse complement l-mers must also be reverse complement. These mapping-friendly SSRs are called Mapping-friendly Sequence Reductions (MSRs). We observe that this mapping-friendly property also preserves assembly: the assembly of reduced reads is equivalent to the reduced assembly of the original reads (excluding assembly errors). Reduced reads can thus be used as sketches of their non-reduced counterparts, preserving some alignment properties while being potentially much shorter. Figures 3.1 shows an example of a MSR being applied on a read.

3.1.3 Performing assembly with reduced sequences

One of the most interesting property of MSR sketching is that reads that have been reduced using an MSR can be easily aligned and assembled using traditional assembly methods. The assembly of reduced reads is then equivalent to the reduced assembly. To complete our assembly pipeline, we need an inverse MSR function to convert our assembly of reduced reads into the final assembly. The process is illustrated Figure 3.2. However, multiple sequences can yield the same reduction, making the inverse function not well-defined.

For example, in the case of homopolymer compression, the sequence "ACGT" in the compressed assembly may correspond to either "AAACGT" or "ACCCGT" in the final assembly. The necessary information to accurately decompress the reduced assembly is present in the original sequencing reads. For instance, Shasta [126] solves this problem by keeping track of which reduced sequence corresponds to which inflated sequence throughout the entire assembly process, but this requires significant bookkeeping to generalize to other MSRs. We will propose a simpler method to address this issue.

Our method consists of three main steps to compute the inverse MSR func. First, we create an inventory of k-mers that tile the compressed assembly, using a k-mer size of, for example, k=31. For instance, two 3-mers that tile the sequence "ACCGTT" are "ACC" and "GTT". In the second step, we run the MSR on all original reads again and each time a tiling k-mer is produced, we record the corresponding uncompressed sequence. Finally, the full assembly can be reconstructed by concatenating the uncompressed sequences of the tiling k-mers.

The method we propose guarantees that the resulting assembly is a patchwork of sequencing reads. However, it does not guarantee that the final assembly is accurate. A major difficulty occurs when a tiling k-mer corresponds to several uncompressed sequences. This occurs frequently in homopolymer compression: if both "ACCTG" and "ACCCTG" are observed in the reads, which one should be associated with the reduced "ACTG"? The current solution is to make a majority choice, but this may lead to errors as both sequences may actually exist. An extreme example would be an MSR that only outputs "A"s, resulting in all reads being reduced to stretches of "A"s and the assembly being a long stretch of "A"s. In this case, a tiling k-mer made of a stretch of "A"s would correspond to many different valid uncompressed sequences, making it impossible to accurately inflate the reduced assembly. Therefore, carefully designing the MSR is crucial in order to avoid this problem. We will discuss this in the following section.

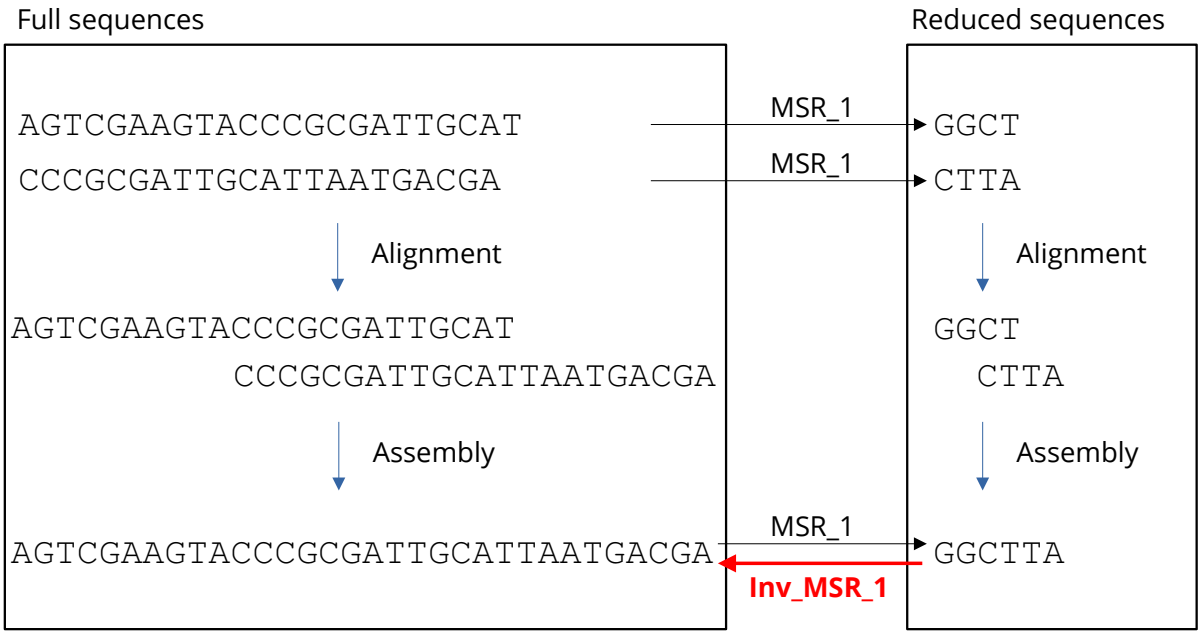


Figure 3.2 – Example showing the mapping-friendly properties of MSR_1 (defined in Figure 3.1). The compression of two overlapping reads yield overlapping reads. The compression of the assembly yields the assembly of the compressed reads. Assembly can be performed by reducing reads, assembling reduced reads and performing the inverse function of the MSR, highlighted in red. The difficulty lies in the fact that the inverse function is not easy to compute.

3.1.4 Designing a good MSR sketch

Portrait of an ideal sketch

The goal of sketches is to represent original data in a smaller space while still retaining enough useful information to accurately perform specific tasks. Since sketches, including MSRs, contain less information than the original reads, they cannot fully represent all the information contained in the original data. As explained above, this translates in our case in difficulties inflating the reduced assembly. The key to a good sketch is to strike a balance between reducing the size of the data and preserving enough relevant information for the task at hand. In our case, the sketches need to carry enough information to 1) perform assembly on the sketches, and 2) recover the original assembly from the result.

The first important factor to consider when creating a MSR is thus the compression factor. In order to significantly increase assembly speed and decrease memory consumption, the MSR must significantly reduce the size of the input reads. The compression factor of an MSR will be defined as the inverse of the proportion of l-mers that are not mapped to the empty character. For example, in Figure 3.1, eight 3-mers out of 64 are not mapped to the empty character, resulting in a compression factor of 64/8=8. This compression factor corresponds to the ratio of sizes between the input sequence and the sketch in the case of a random infinite input sequence. Our goal will be to compress the data as much as possible to speed up the assembly process.

The second factor to consider when designing an MSR is the collision rate of the compression. Collisions occur when two distinct sequences get reduced to the same sketch. The number of collisions between all the sequences represented in the reads should generally be minimised, as collisions make it difficult to accurately inflate the reduced assembly. Yet, in some cases, collisions can be intentionally exploited to improve the quality of the reads. For example, in homopolymer compression, sequences that differ by a homopolymer repeat are purposefully collided because they are suspected to represent the same sequence in the genome. [125] explored all order-2 MSRs and found several that actually improved alignment quality, which can be interpreted as the collisions filtering out sequencing noise and/or difficult regions. The collision rate also impacts the maximum achievable compression factor. An infinite compression factor would result in reads of length 0, which would be extremely efficient to assemble but the resulting assembly of length 0 would be quite challenging to inflate. To sum up, it is important to avoid reducing two sequences that must be distinct in the final assembly to the same sequence.

The final important factor to consider when designing an MSR is the mapping-friendliness property. It is straightforward to see that two identical sub-sequences will be reduced to identical sub-sequences and align if they are long enough. However, achieving mapping-friendliness becomes much more challenging when the original sequences differ. As we will discuss in section 3.1.4, reduced sequences can potentially be more divergent than the original sequences, meaning that reduced sequences may not align as well as the original sequences. This can be a significant obstacle when working with error-prone reads, but can also be viewed as a desirable property for high fidelity reads, as it prevents collapsing haplotypes. Therefore, there is no one-size-fits-all MSR, and an MSR should be designed based on the data it will reduce and how the reduced sequences will be used.

Practical guide to design a MSR

Upper limit on the compression factor c The choice of compression factor is important as it can impact the efficiency of downstream analyses. A higher compression factor can lead to more efficient analyses, but there is a risk of compressing too much and creating collisions between unrelated sequences. For instance, if Illumina reads are reduced by a factor of 100, the resulting reads will be only a few base pairs long and will not contain enough information to accurately map them to a genome or assemble them.

Lower limit on the order l All compression factor are not achievable with a given order. Indeed, an order-1 MSR can only achieve a maximum compression factor of 2 (by deleting all A/T or all C/G). This limit can be generalised: since MSR functions must output at least two different characters, the compression factor is mathematically limited to $\frac{4^l}{2}$ for even l and $\frac{4^l}{4}$ for odd l (to maintain the reverse-complement property). In other words, high compression cannot be achieved with too low l.

Tune the error rate of the sketches The parameters l and c influence the error rate of the sketches. Consider two sequences that differ by n edits. When reducing these sequences, there will be at least n different l-mers fed into the MSR function. If the edits are more than l bases apart, there will be approximately $l \cdot n$ sequence-specific l-mers, grouped into n stretches of l consecutive l-mers. Consequently, on average, a difference between two sequences will result in $l \cdot c$ consecutive differences in their reduced counterparts. This is illustrated in Figure 3.3a. This is a double-edged property.

On one hand, sequencing errors are amplified by MSRs. To illustrate this, we measured

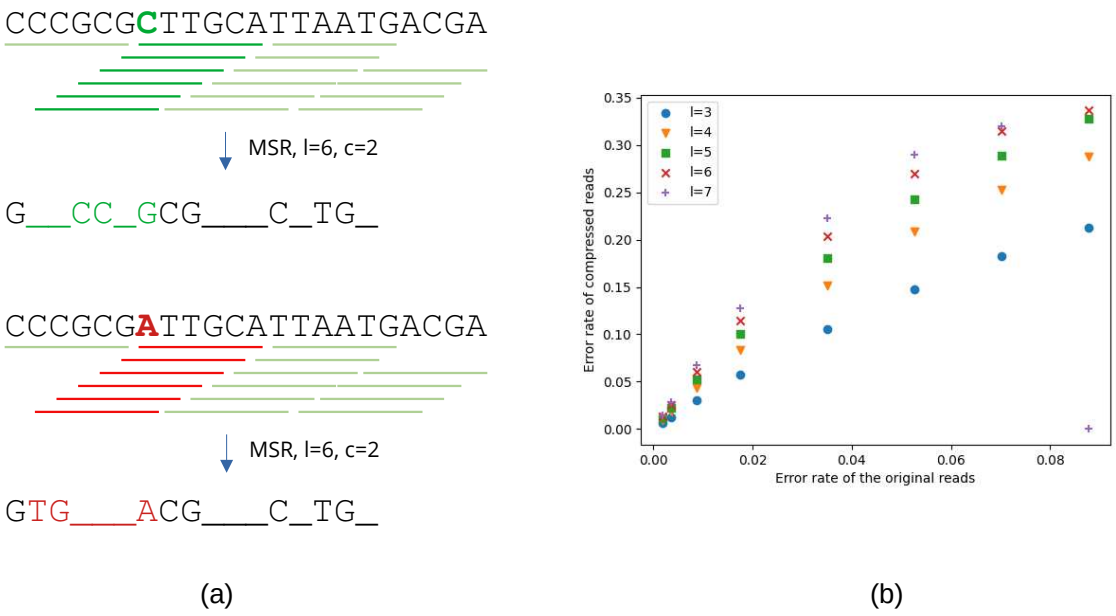


Figure 3.3 – Divergence of reduced sequences compared to divergence of the original sequences. (a) Two sequences differing by one base (out of 24 bases) are reduced by a MSR with an order of 6 and a compression factor of 2. The reduced sequences diverge by 3 bases out of 9, and these bases are grouped. (b) Reads were simulated with different error rates and reduced with $c = 8$ and different l. The reducing function of the MSR was pseudo-random. All the error rates are computed with samtools stats. The error rate of 0 on the bottom right corner of the graph correspond to a situation were correct and erroneous sequences could not be aligned because the error rate was too high. As expected, the error rate increase approximately linearly with l.

the error rates of MSRs of various orders compared to the original reads using samtools [127], as shown in Figure 3.3b. As expected, we observed an increase in the error rate of the reduced reads with increasing l. Decreasing c reduces the size of the sketches and the size of the stretches of errors but increases the density of these error stretches. Therefore, minimizing the error rate of the sketches involves performing minimal compression and using a small l. Homopolymer compression meets these requirements and further reduces the error rate of the sketches by estimating which l-mers are likely to be erroneous. However, it seems difficult to use highly compressive MSRs on erroneous reads, e.g. Nanopore reads.

On the other hand, increasing the divergence between similar sequences can be a desirable property to reduce the collision rate of similar sequences. Genomes often contain repeats, and choosing high l and low c values can ensure that two similar repeats are not reduced to the same sequence. This can be helpful, for example, in distinguishing haplotypes. This is the strategy we implement in our assembler.

Example of the choice of l and c Let us discuss how we chose default parameters for our strain-aware HiFi DBG assembler, described in section 3.2. We set the default compression factor c to 20, to obtain reduced reads with a length of a few hundred bases, enough for performing assembly but much shorter than the original reads. We choose a default value of $l = 101$ to allow us to differentiate between highly similar strains. Assuming an average error rate of 0.1% in the input reads, the resulting reduced reads would have an error rate of less than 10%, with the errors distributed in stretches of $l/c = 5$ bases on average. To perform DBG assembly with a k-mer size of 31, error-free k-mers are required. With these parameters, error-free reduced k-mers correspond to error-free non-reduced sequences of $k * c + l = 731$ bp on average, which are common in reads with an error rate of 0.1%.

Choose the transforming function There exist many transforming functions given an order and a compression factor: should AAC be transformed in T or in C? [125] showed that the choice of the function had a great impact on the downstream analysis. Indeed, not all l-mers are equivalent given a sequencing technology: typically, l-mers containing homopolymers will be more prone to errors. Moreover, all l-mers are not equivalent given a genome. For instance, the 4-mer composition of a sequence can be exploited to separate sequences originating from different species [128].

It is not feasible to explore all possible MSR functions to select the best one, as the number of functions increases combinatorially with l. A more efficient approach would be to use heuristics, such as evolutionary algorithms, to search the vast space of available functions. However, there is a risk that the resulting function may overfit the training dataset (the genome(s) and/or the sequencing technology). In our assembler, we leave this discussion for later and opt for a "random" function based on a pseudo-random hash. Tailoring MSR functions to specific needs is in our opinion a promising lead for future improvement.

3.1.5 Why are MSRs interesting?

An often crucial piece of information to preserve for many applications, e.g. genome assembly, is the structural information of the input sequences, i.e., the order of the sequence. However, many sketching methods achieve high compression by cutting input sequences into k-mers and losing the ordering information [129, 130]. In contrast, Mapping-friendly Sequence Reductions (MSR) is one of the few methods, along with the k-min-mer technique [56] and the seed-chain technique [131], that preserve this structural information.

A second interesting property of MSRs is that they sketch the input sequences quite uniformly. In contrast, k-min-mers and seed-based sketching methods preserve much information at the sites of the sampled k-mers but retain little information elsewhere. This is because each kept k-mer outputs $2 \cdot k$ bits of information, necessitating that the sketch keep few k-mers to remain compact. MSRs, on the other hand, spread the preserved information more evenly along the input sequence, as a kept input k-mer only outputs a base, i.e., 2 bits of information. Therefore, for the same size of the resulting sketch, MSRs conserve many more k-mers, including numerous overlapping k-mers, albeit with much less information for each. In practice, imagine we want to search for a small sequence within the sketches: there is a much higher probability that this sequence will fall between sampled minimizers rather than being reduced to an empty sequence by the MSR. Conversely, if the sequence falls on a minimizer, that minimizer will contain more information to confirm the match, whereas the MSR will have sampled that region less intensively in comparison.

Sketches used in bioinformatics typically require specific software for interpreting them [124]. In contrast, Mapping-friendly Sequence Reductions (MSRs) provide sketches of sequences in the form of shorter sequences. These MSR sketches can be stored using the standard FASTA format and can be aligned, assembled, and indexed much like regular

sequences using existing software. Although this does not guarantee that every software will work seamlessly—for example, we observed that all MSRs are not well-suited for sketching noisy reads—this compatibility is in my opinion one of the most appealing properties of MSR sketches, as they can be integrated into existing software with minimal effort.

We will demonstrate that these three properties of MSRs can be exploited to easily design an HiFi assembler.

3.2 Alice: fast and accurate assembly of high-fidelity reads based on MSR sketching

3.2.1 Introduction

With the rise of high-throughput sequencing, genomic experiments have been producing vast amounts of data, far outpacing the growth of computing power as predicted by Moore's law [132]. It is now common for a single experiment to generate dozens or even hundreds of gigabytes of genomic data. To address this challenge, researchers have been developing efficient algorithms for storing, mapping, comparing and assembling reads and genomes over the past few decades.

Handling such big amount of data involves compressing it to fit within hardware limitation [133]. Several spectacular lossless data compression schemes have been designed to solve bioinformatic problems, e.g. exploiting the repetitions within a genome [134], organising k-mers in a tree [135] or using phylogeny to exploit redundancies between species [136]. However, lossless algorithm remain limited by, precisely, the fact that they are lossless. Lossy compression schemes have the potential to provide much higher compression rates. For example, Bloom filters are extensively used to compress the representation of sets of k-mers, but at the price of only keeping a presence/absence imperfect information [137, 138].

A popular technique to diminish the size of the computations is to sketch (or sample, we will not make a difference here) the input data. This consists in building a reduced, approximate representation of the input data and performing computation on these sketches. The difficulty of this approach is to build a representation on which meaningful computations can be efficiently performed. An example is the Seed-Chain-Extend paradigm, ubiquitous in long-read alignment, where reads are reduced to a small set of k-mers (seeds) which

are then aligned (chained). Another example is Mash [129], which compares datasets by comparing a subset of their kmers.

In the field of genome assembly, sketching has been used to generate all-versus-all read mapping [61] but not until recently and the advent of High Fidelity (HiFi) sequencing has it been decisively introduced in De Bruijn Graphs assemblers. Building on ideas from wtdbg2 [79], shasta [126] and Peregrine [139], mDBG [56] introduced the idea of reducing reads to a small chain of k-mers sampled with a hash function, assembling all chains of sampled k-mers and finally transforming back the final chain of k-mers to a genomic sequence. This approach proved extremely efficient, providing a human assembly in minutes on a personal computer. It was extended as a metagenomic assembler in metaMDBG [73]. However, this sketching approach faces some serious limits.

Sketching strategies generally need to strike a balance between compressing the data and obtaining accurate results. To provide accurate genome assemblies, sketches must represent the small differences between genomic repetitions and slightly different haplotypes. When (meta)mDBG sketch the reads as a chain of k-mer, all diversity that falls between the sampled k-mers is lost. As a consequence, the two mDBG-based assemblers are not able to distinguish similar haplotypes.

In this work, we introduce a new assembler, Alice. The main novelty of this assembler is to employ Mapping-friendly Sequence Reductions (MSR) as a sketching technique for HiFi genome assembly. MSR was originally introduced in [125] to enhance read mapping quality, but its potential as a sketching technique was not explored. The original article examined a limited number of MSRs. In our approach, we use different MSRs to sketch HiFi reads, resulting in a computationally efficient solution that addresses many of the challenges faced by (meta)mDBG. The name of the assembler draws its inspiration from the book Alice in Wonderland [140], in which Alice uses a "drink-me potion" to go through a small door and a "eat-me" cake to inflate back to her original size [2]. By analogy, the small door would represent an assembler of limited capacity, and the potion MSR sketching.

We applied Alice to the sequencing of a mock community containing five strains of *Escherichia coli*. We demonstrate that Alice assembles the reads faster than state-of-the-art assemblers, such as hifiasm_meta [74] and metaMDBG [73], while improving the assembly of the five *E. coli* strains.

2. at least that is her intention, things then go awry for Alice

3.2.2 Methods

The assembly procedure is illustrated Figure 3.4.

Reducing input reads

All reads are initially reduced using a Mapping-friendly Sequence Reduction (MSR) provided by Alice. The MSR allows the user to select the order l (default value of 101) and the compression factor c (default value of 20). The l-mers of the reads are processed through a function g. This function takes an l-mer as input and outputs either a single base, which is appended to the growing reduced read, or an "empty" base ϵ, which is not appended to the growing reduced read.

The function g of the MSR is designed as follows. The l-mer is converted into its canonical form, which is either the original l-mer or its reverse complement if the reverse complement is lexicographically smaller. g then applies to the canonical l-mer a pseudo-random hash function yielding a hash between 0 and 1 [141]. It distinguishes five case:

— if the hash is smaller than $1/2c$ and the original l-mer is canonical, an A is outputted
— if the hash is smaller than $1/2c$ and the original l-mer is not canonical, a T is outputted
— if the hash is between $1/2c$ and $1/c$ and the original l-mer is canonical, a C is outputted
— if the hash is between $1/2c$ and $1/c$ and the original l-mer is not canonical, a G is outputted
— if the hash is between $1/c$ and 1, ϵ is outputted

Assembling reduced reads

Many existing short-read and long-read assemblers were tested to assemble reduced reads, but they did not yield very convincing results, especially to separate haplotypes. We believe this is due to reduced reads having slightly different properties compared to regular sequencing reads of equivalent length. For example, errors tend to cluster when $c \cdot l >> 1$. Most assemblers did not manage to assemble at all the reduced reads.

To address this issue, we developed a simple custom assembler that consists of three steps.

1. Generate an unitig graph with a k-mer length of 31, discarding all k-mers seen only once. This is done with BCALM2 [142] (Figure 3.4a)

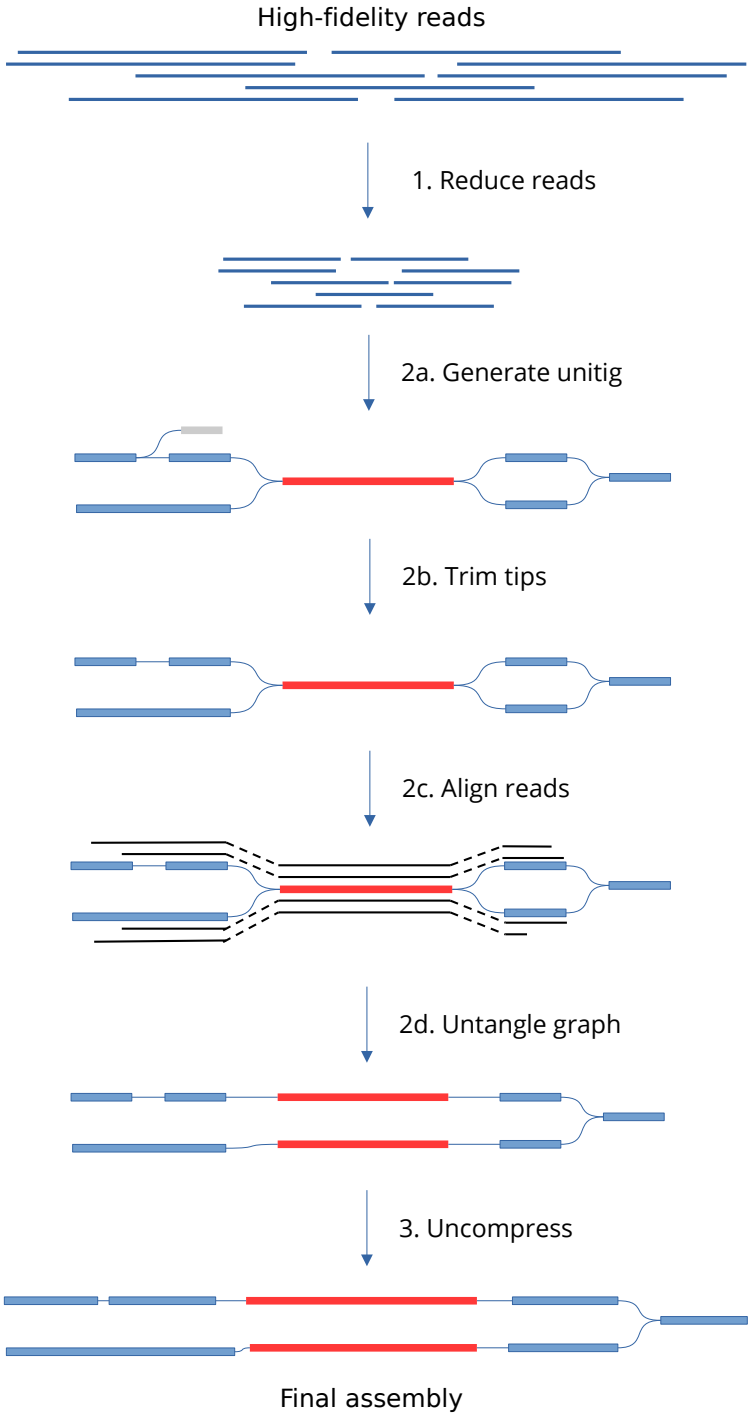


Figure 3.4 – Assembly process of Alice

2. Simplify the graph by removing tips and bubbles composed of k-mers seen fewer than five times, a classic procedure in assemblers, as used for example in [56] (Figure 3.4b)
3. The final step is to untangle the graph to improve contiguity and duplicates unitigs that are present multiple times in the genome. To this end, all reads are aligned on the graph (Figure 3.4ac). All contigs are then iteratively assessed. If the alignment of the reads on the graph suggest that the contig is present in two different paths, the contig is duplicated (Figure 3.4d).

Recovering the uncompressed assembly

The compressed assembly represents the reduced version of the final assembly. Inflating this reduced version back to the full assembly is not straightforward, as the MSR reduction function is not invertible.

Our method involves three steps:

— Create an inventory of k-mers that tile the compressed assembly, using a k-mer size of 31 by default. For example, two 3-mers that tile the sequence "ACCGTT" are "ACC" and "GTT".

— Rerun the MSR on all original reads, and each time a tiling k-mer is produced, record the corresponding uncompressed sequence.

— Reconstruct the full assembly by concatenating the uncompressed sequences of the tiling k-mers.

3.2.3 Results

Zymobiomics Gut Microbiome Standard

We used the Zymobiomics Gut Microbiome Standard to evaluate Alice. This standard is a commercial blend of 19 bacterial strains and two yeast strains, designed to replicate the composition of the gut microbiome. A HiFi sequencing was available on the Sequence Read Archive (SRA) under the accession number SRR13128013. The proportions of each organism in the mix and their genomes are known. We selected this dataset because it includes five strains of *Escherichia coli*, which are difficult to assemble separately.

We compared Alice against metaFlye [75], hifiasm_meta [74] and metaMDBG [73], which are the leading assemblers in the field of metagenomic HiFi assembly. Alice was used with its default parameters, i.e. a compression factor of 20 and an order of 101. In

section 3.2.3, we discuss this choice and test other parameters. We used metaQuast [97] to evaluate the assemblies. The completeness and duplication ratio are shown in Table 3.1, and the contiguities in Table 3.2.

Alice performed assembly in two hours of CPU time, compared to four and a half hour for hifiasm_meta, sixteen hours for metaMDBG and four days for metaFlye. In terms of memory, Alice used a maximum of 3.5G RAM, more than the 2G of metaMDBG but much less than the 102G of hifiasm_meta and the 474G consumed by metaFlye. The goal, using MSR sketches to build a fast and memory-efficient assembler, is thus achieved.

The tested assemblers exhibited different behaviors when assembling the five *E. coli* strains. metaMDBG and metaFlye fully assembled only one strain and partially assembled the other four, as indicated by the low metaQUAST completenesses in Table 3.1. A 27-mer analysis revealed that 20% and 10% of strain-specific *E. coli* 27-mers were missing in the final assemblies, respectively. In contrast, hifiasm_meta and Alice assembled all the strains with high completeness, with only 4.5% and 3% of strain-specific *E. coli* 27-mers missing, respectively. While hifiasm_meta's assembly achieved higher completeness than Alice, it did so at the cost of a high duplication ratio. The relatively lower metaQUAST completeness of the Alice assembly can likely be explained by the "untangling" of the assembly graph, where certain regions need to be duplicated to their correct multiplicity. At this stage, Alice does not duplicate all such regions to their right multiplicity.

All assemblers achieved similar levels of completeness for the other genomes. metaMDBG appeared to be slightly better at recovering low-abundance genomes. Additionally, metaMDBG produced the most contiguous assemblies with fewer over-replicated regions. The lower contiguity and higher duplication ratio of Alice's assemblies can be attributed to the presence of some remaining bubbles in the final assembly graph. It is not clear yet if these bubbles represent actual diversity or simply artefacts.

The final aspect we wanted to verify was whether Alice produced misassemblies, particularly in comparison to hifiasm_meta when assembling the separate *E. coli* strains. metaQUAST detected 30 misassemblies in the Alice assembly of the *E. coli* strains, compared to 81 misassemblies in the hifiasm_meta assembly. However, Alice produced 37 misassemblies in the *B. fragilis* assembly, fewer than the 61 produced by hifiasm_meta but significantly more than metaFlye and metaMDBG, which produced one and four misassemblies, respectively. We hypothesize that this discrepancy is due to diversity present in the sample but not represented in the reference, which could also explain the extremely high duplication ratio of the hifiasm_meta assembly of *B. fragilis* (15). This hypothesis

would need to be confirmed.

In conclusion, Alice was able to produce an assembly of high quality of the Zymobiomics Gut Microbiome Standard in low time and memory. The successful assembly of the five *E. coli* strains proves that MSR can be used as a sketching method capable of distinguishing highly similar sequences.

Genomes	coverage	metaFlye	hifiasm_meta	metaMDBG	Alice
A. muciniphila	133	1.0 / 1.0	1.0 / **1.6**	1.0 / 1.0	1.0 / 1.0
B. fragilis	700	1.0 / 1.0	1.0 / **15**	1.0 / 1.0	1.0 / 1.0
B. adolescentis	750	1.0 / 1.0	1.0 / **2.5**	1.0 / 1.0	1.0 / 1.0
C. albicans	28	**0.72** / 1.0	**0.68** / 1.2	**0.81** / 1.0	**0.65** / 1.2
C. difficile	91	1.0 / 1.0	0.91 / **1.3**	1.0 / 1.0	1.0 / 1.0
C. perfringens	1	**0** / -	**0** / -	**0** / -	**0** / -
E. faecalis	0	**0** / -	**0** / -	**0** / -	**0** / -
E. coli B1109	148	**0.60** / 1.1	0.99 / **2.1**	**0.78** / 1.0	**0.88** / 1.1
E. coli B3008	152	**0.83** / 1.0	1.0 / **1.6**	**0.36** / 1.0	1.0 / 1.0
E. coli B766	140	0.95 / 1.0	0.96 / **1.5**	0.96 / 1.0	0.96 / 1.0
E. coli JM109	152	**0.44** / 1.1	1.0 / **2.1**	**0.38** / 1.0	0.90 / 1.1
E. coli b2207	140	**0.47** / 1.0	0.99 / **2.2**	**0.37** / 1.0	0.92 / **1.5**
F. prausnitzii	1250	1.0 / 1.0	1.0 / **28**	1.0 / 1.0	1.0 / 1.0
F. nucleatum	625	1.0 / 1.0	1.0 / **5.9**	1.0 / 1.0	1.0 / 1.0
L. fermentum	789	1.0 / 1.0	1.0 / **1.4**	1.0 / 1.0	1.0 / 1.0
M. smithii	14	1.0 / 1.0	1.0 / 1.0	1.0 / 1.0	1.0 / 1.0
P. corporis	517	1.0 / 1.0	1.0 / **10**	1.0 / 1.0	1.0 / 1.1
R. hominis	1206	1.0 / 1.0	1.0 / **2.3**	1.0 / 1.0	1.0 / 1.0
S. cerevisiae	27	**0.80** / 1.0	**0.73** / 1.2	**0.89** / 1.0	**0.70** / 1.3
S. enterica	1	**0.02** / 1	**0.12** / 1.0	**0.24** / 1.0	**0.14** / 1.0
V. rogosae	1667	1.0 / 1.0	1.0 / **49**	1.0 / 1.0	1.0 / 1.1

Table 3.1 – Completeness / duplication ratios of the metaFlye, hifiasm_meta, metaMDBG and Alice assemblies with respect to all the genomes of the mix, computed by metaQUAST. Completeness values below 0.9 and duplication values above 1.2 correspond to perfectible assemblies and are highlighted in red.

Influence of different parameters

We experimented with different parameter choices for the compression factor and the order of reduction on the Zymobiomics Gut Microbiome Standard dataset to understand how these parameters influence the final assembly.

Assemblies	metaFlye	hifiasm_meta	metaMDBG	Alice
Akkermansia muciniphila	**2.85**	**2.85**	**2.85**	2.32
Bacteroides fragilis	5.15	**5.63**	4.31	0.57
Bifidobacterium adolescentis	**2.03**	**2.03**	**2.03**	1.59
Candida albicans	0.003	0.003	0.004	0.002
Clostridioides difficile	**4.21**	**4.21**	**4.21**	**4.21**
Clostridium perfringens	-	-	-	-
Enterococcus faecalis	-	-	-	-
Escherichia coli B1109	0.02	0.15	**0.45**	0.09
Escherichia coli B3008	0.02	2.58	-	**4.84**
Escherichia coli B766	-	**0.39**	**0.39**	0.22
Escherichia coli JM109	0.02	**0.18**	-	0.081
Escherichia coli b2207	0.02	**0.23**	-	0.02
Faecalibacterium prausnitzii	**2.9**	0.02	**2.9**	**2.9**
Fusobacterium nucleatum	**2.44**	2.03	**2.44**	**2.44**
Lactobacillus fermentum	**1.91**	1.79	0.17	1.51
Methanobrevibacter smithii	**1.43**	**1.43**	1.21	0.17
Prevotella corporis	**2.50**	2.12	**2.50**	0.21
Roseburia hominis	**3.46**	**3.46**	**3.46**	**3.46**
Saccharomyces cerevisiae	0.05	0.07	**0.11**	0.01
Salmonella enterica	-	-	-	-
Veillonella rogosae	**2.16**	0.03	2.15	0.44

Table 3.2 – NGA50 of the genomes reconstructed by the different assemblers, measured with the metaQUAST. All the figures are in Mbp. Empty cells correspond to genomes that were not at least 50% covered. The best values are highlighted in bold.

We conducted two experiments: one to assess the effects of the order and another to assess the effect of the compression factor. First, we tested compression factors of 100, 50, 20, 10, and 5 with an order of 101. Second, we tested orders of 11, 21, 51, 101, and 201 with a compression factor of 10.

The variation of these parameters primarily impacted the completeness of the resulting assemblies and the run-times of the pipelines, while their accuracy, duplication ratio, and contiguity remained equivalent.

As expected, the run-time increases with the compression factor, as there is more data to assemble. This is illustrated in Figure 3.5.

Compressing the data more also has a positive impact on the completeness and contiguity of the five highly similar *E. coli* strains (Figure 3.5). When investigating the 27-mer

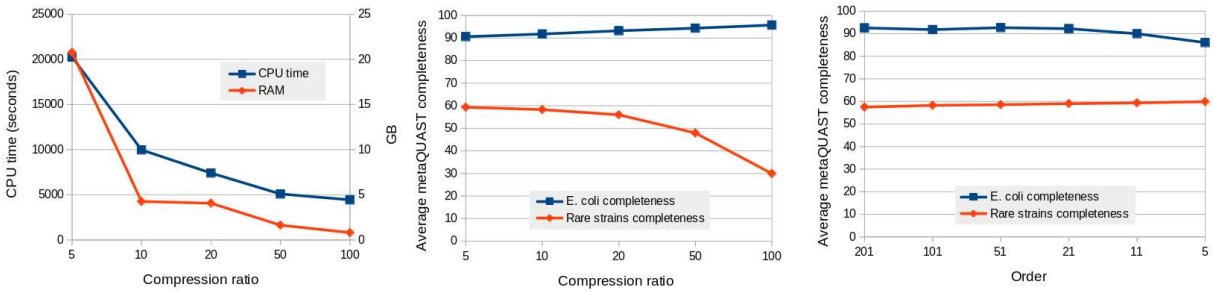


Figure 3.5 – Variation of resource usage and metaQUAST completeness with different compression factor and orders. "Rare strains" refer to *C. albicans*, *S. cerevisiae*, and *S. enterica*. The completeness displayed are arithmetic means of the completeness of the different genomes of the categories.

completeness (not shown), it turns out that all assemblies had a similar amount of missing 27-mers. Hence, the main difference explaining the difference in completeness is that repeated regions are more shrunk when the data is compressed more, which helps to assemble repeated regions closer to their true multiplicity and thereby improving contiguity.

However, the results represented in Figure 3.5 show that compression negatively impacts the completeness of the *C. albicans*, *S. cerevisiae*, and *S. enterica* genomes, which have relatively low coverage (see Figure 3.1 for the coverages). This is because the assembly algorithm requires a sufficient number of error-free 31-mers in the compressed reads to produce a complete assembly. An error-free compressed 31-mer corresponds to an error-free uncompressed sequence of average length $31 * c + l$ without errors. Therefore, as the compression factor c increases, the number of correct 31-mers in the reads decreases. For genomes with low coverage, high compression can result in the loss of important 31-mers, leading to insufficient coverage of some regions, hindering their assembly.

The order was found to have relatively little impact on the resulting assemblies. The only significant effect observed was when the order decreased to 11 and 5, where l/c approached or fell below 1. In these cases, the assembler began collapsing highly similar sequences, leading to a decrease in the completeness of the five *E. coli* strains. Despite the fact that the error rate scales approximately linearly with l, increasing l did not have a significant negative impact on the completeness of the assemblies. This is because the errors in the compressed reads cluster in increasingly large clusters, but the error-free regions between these clusters diminish in size only slowly with l.

3.2.4 Discussion

In this work, we propose accelerating HiFi assembly by using Mapping-friendly Sequence Reductions to sketch input reads and assembling them. We implemented this method in a software named Alice, which we tested on the sequencing of a challenging mock community containing five conspecific strains of *Escherichia coli*. Alice proved to be an order of magnitude faster than metaMDBG and metaFlye, and required an order of magnitude less RAM than hifiasm_meta and metaFlye. Unlike metaMDBG and metaFlye, Alice was capable of assembling the five *E. coli* strains separately. While the Alice assembly exhibited a lower contiguity compared to the hifiasm_meta assembly, it was much less over-replicated. Overall, Alice is already a competitive HiFi assembler on this dataset, though the assembly could still be improved. The next step would be to test Alice further on different datasets.

In terms of implementation, Alice still has room for improvement. Compared to mDBG, it currently takes 10 times longer to reduce reads and inflate the assembly for a comparable task. The assembly procedure between these two steps is relatively basic and could likely be significantly improved. To explore this idea, we tried to use many short-reads assemblers out of the box and, though SPAdes [53] and minia [55] performed reasonably well, none was able to improve Alice.

A very promising, but also vast, avenue to improve the assembler would be to modify the MSR function, which we designed to be pseudo-random. For example, guarantees could be introduced to ensure that at least one base is outputted for all windows of length w. Another idea would be to introduce an equivalence between synonymous codons in the MSR transformation. The authors of [125] have shown that changing the function can significantly improve results when aligning reads reduced with an MSR of order 2, suggesting that the choice of the MSR function has significant impact on the downstream results. The challenge, however, lies in the extremely high number of MSR functions to explore.

In this work, we applied MSR sketching to assembly, but there is potential to extend this technique to other problems. The most immediate application seems to be sequence alignment, although it is unclear how it would integrate with seed-chain-extend methods and how it would handle imperfect matches. Another potential application could be indexing assembled genomes, given their expected low error rate.

We also observe an interesting side result. Efficiently building De Bruijn Graphs for large k is somewhat of an open problem. The authors of [143] note that it takes "pro-

hibitively large time/memory" and propose a complex solution based on building a disjoint graph to efficiently construct De Bruijn Graphs with large k. In the future, building a compressed graph with smaller k and then decompressing it may offer another solution, although the problem does not translate completely trivially.

Chapter 4

Using Hi-C to untangle assembly graphs

Abstract: This chapter focuses on enhancing the contiguity of an assembly using Hi-C data. It introduces the concept of *untangling* an assembly graph with Hi-C or long reads prior to scaffolding. Implemented in a software tool named GraphUnzip, we demonstrate that this strategy improves the Hi-C scaffolded assemblies of both haploid and multiploid genomes. While GraphUnzip is not (yet) applicable to improving metagenomic assemblies with Hi-C, its long-read version is suitable for metagenomes and has been integrated in HairSplitter (Chapter 2).

4.1 Context

Metagenomic Hi-C can be used to separate species within an assembly [95, 96]. However, to the best of my knowledge, there is currently no specialized software aimed at improving the contiguity of metagenomic assemblies with Hi-C. This can be explained by the fact that, with long reads, bacterial genomes generally do not need Hi-C to achieve high contiguity. This chapter will hence not focus on metagenomes but rather on improving the contiguity of diploid and multiploid species with Hi-C.

In practice, I will propose an implementation of an untangling software named GraphUnzip. GraphUnzip is designed to improve the contiguity of multiploid assemblies and is also applicable to haploid assemblies. Towards the end of the chapter, I will explain how part of this work has been extended to metagenomes to improve the contiguity of bacterial assemblies with low-quality ultra-long reads.

4.2 Introduction

The advances of sequencing technologies and of software provide assemblies of unprecedented completeness. Despite these advances, assemblers often fail to produce one

contig per chromosome, even when long or ultra-long reads are available.

The main difficulty stems from the presence of similar regions across eukaryotic genomes. Similar regions include segmental duplications, paralogous genes but also homozygous zones, where the same information is present on several chromosomes. Assemblers will tend to merge all reads coming from these regions together in one contig - when the homozygosity is perfect, even HairSplitter (Chapter 2) cannot separate the haplotypes. Repeated contigs typically overlap two or more contigs at each of their end, thus cannot be merged unambiguously with their neighbors. Many assemblers therefore output assembly graphs that contain the sequences of the contigs as well as all overlaps between the different contigs. The sequences of the chromosomes are unknown paths traversing these graphs.

Chromosome conformation capture approaches, such as Hi-C [69], measure physical interactions in the nucleus of the cell and provide the missing key information to finish assembly: the ability to know which contigs are close to each other on the chromosomes, with a signal that spans even the longest repeated regions. Given a draft assembly and Hi-C sequencing data, scaffolders order and orient all the contigs to obtain chromosome-length scaffolds. Consecutive contigs in a scaffold are separated by stretches of Ns, i.e., unspecified bases. The most widely used Hi-C scaffolders focus on providing haploid genomes and include SALSA2 [144], YaHS [91], pin-hic [145], instaGRAAL [146] and scaffHiC [147]. To reconstruct multiploid genomes, the most popular scaffolders include GreenHill [93] (focused on the diploid case only), AllHiC [92] and HapHiC [94].

All scaffolders today still face two issues: 1) they struggle to place small contigs with few Hi-C contacts, 2) they leave gaps of unknown sequence and unspecified length between stitched contigs. As a result, most chromosome-scale assemblies produced today still contain gaps and unplaced contigs. We propose to exploit the rich information contained in assembly graphs to fill these gaps (literally). So far, only SALSA2 uses the assembly graph, but only to orient contigs, not exploiting the fact that assembly graphs specify the relative position and orientation of all the contigs.

We introduce the concept of *untangling* an assembly graph. Untangling simplifies a graph by generating a new assembly graph. More formally, the set of potential genomes represented by the new assembly graph must be a subset of the potential genomes depicted by the original graph. The goal of untangling, similar to scaffolding, is to enhance the contiguity of the assembly. However, unlike scaffolding, which assumes that any contig could potentially be adjacent to any other contig, untangling only considers linking contigs

that are already connected in the assembly graph. As such, untangling is much easier than scaffolding, as contigs typically have a limited number of neighbors in the graph.

Since each chromosomes is theoretically a single path in the graph, untangling could be sufficient to produce telomere-to-telomere assemblies. Moreover, there is no need to introduce stretches of Ns between linked contigs, as their exact overlap is known in the graph. Nonetheless, scaffolding remains useful in practice, since assembly graphs generally have missing links. Untangling and scaffolding can therefore be complementary: once untangling is finished and the number of contigs much diminished, scaffolding comes in and find the remaining links that were not present in the assembly graph.

We present *GraphUnzip*, a module that untangles an assembly graph. It takes as input an assembly graph and either Hi-C data or ultra-long reads and untangles the assembly graph, yielding an assembly with improved contiguity and completeness. Its naive approach makes no assumption on ploidy, making it fit to untangle haploid and polyploid assemblies as well as genomic repeats.

To prove the usefulness of untangling, we inserted GraphUnzip in four assembly pipelines, including two diploid assemblies. Each time, GraphUnzip provides improvement compared to the pipelines without GraphUnzip. In particular, GraphUnzip rescues many small contigs that do not have sufficient Hi-C signal to be directly scaffolded. For diploid assemblies the value of untangling is striking, as GraphUnzip is able to deduce from the graph which contigs need to be duplicated to be present twice (or more) in the final assembly, unlike scaffolders. We think that untangling may become a standard step in future assembly pipelines.

4.3 Methods

The algorithm behind GraphUnzip is inspired by the one inside Unicycler [148]. It is illustrated figure 4.1.

4.3.1 Input

GraphUnzip takes two inputs: the assembly in GFA format and the Hi-C reads mapped to the assembly.

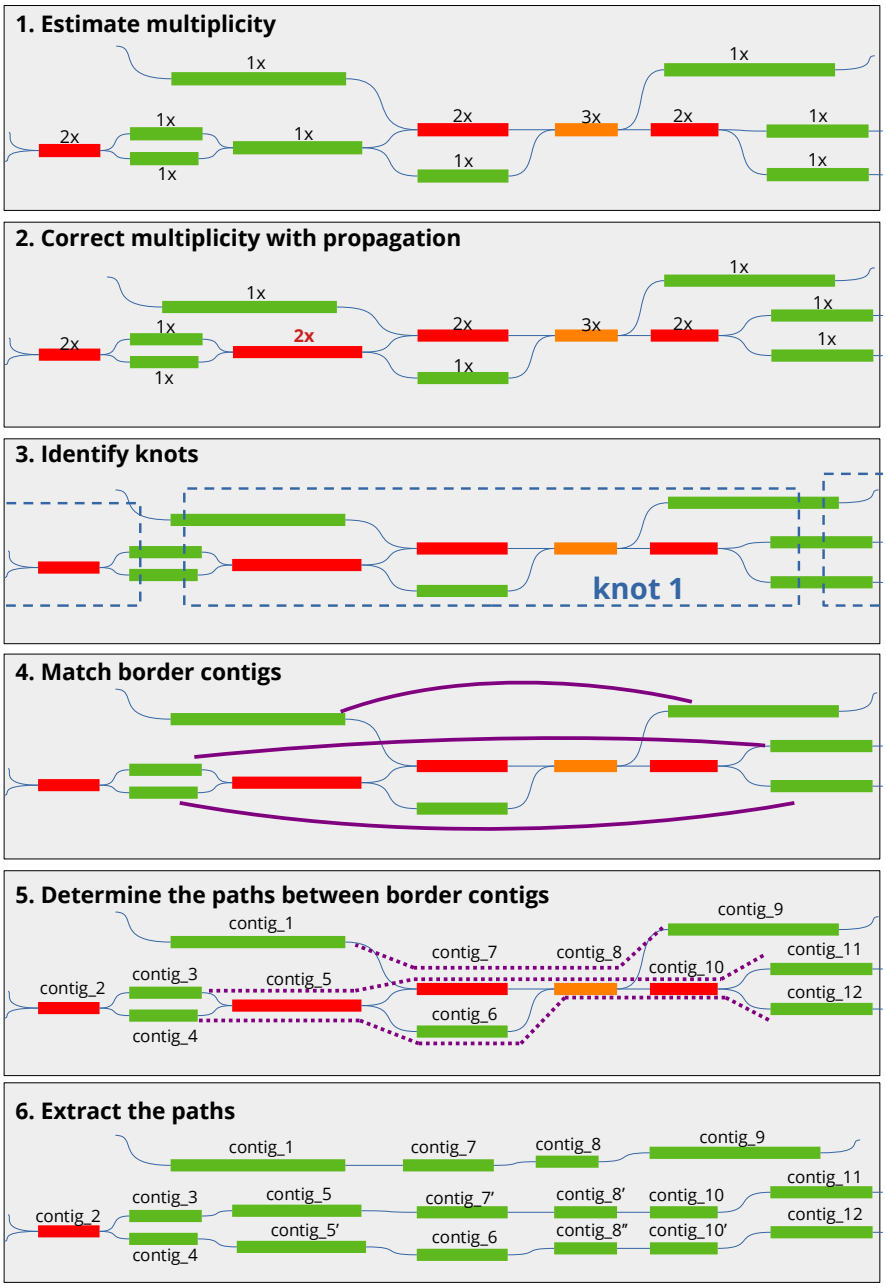


Figure 4.1 – Illustration of the main steps of the GraphUnzip algorithm. **1.** Single-copy contigs are identified based on coverage depth **2.** Multiplicity is propagated to have coherent multiplicities through the graph. **3.** The graph is separated into *knots*, i.e. sub-graphs which only exits are through single-copy contigs, called *border contigs* **4.** In each knot, border contigs are matched pairwise based on their Hi-C interactions. **5.** Paths are drawn between border contigs based on the contigs' coverage and Hi-C contacts. **6.** Paths are extracted from the graph

4.3.2 Multiplicity of contigs

The first step of the algorithm is to determine the multiplicity of each contig of the assembly graph, *i.e.* the number of times the contig is repeated in the genome. Typically, homozygous contigs will have a multiplicity of 2 in an assembly of a diploid genome: they are present two times in the genome, once on each haplotype, but have been assembled in only one copy, as both copies are identical.

GraphUnzip first computes the *haploid coverage*, corresponding to the expected read coverage of contigs that are present only once in the genome. Which contigs are present only once in the genome is generally unknown: as a proxy, GraphUnzip considers haploid all the contigs that have only a single neighboring contig at both their ends. The average read coverage of this list of haploid contigs is considered the haploid coverage. Alternatively, the user can provide the size of the genome, from which the haploid coverage can be inferred.

Knowing the haploid coverage, GraphUnzip gives an estimation of the multiplicity of all contigs of the graph based on their coverage.

Multiplicity is then propagated through the graph: if a contig is the only neighbour of two single-copy contigs, then its multiplicity must be 2. This very simple propagation rule is applied to all parts of the graph. This operation ensures that the multiplicities of all the contigs of the graph are coherent. In some cases, however, it is not possible to find totally coherent multiplicities. In those case, the coverage-based multiplicity is kept.

4.3.3 Knots

The assembly graph is then partitioned in *knots*. A knot is a connected region of the graph delimited by contigs of multiplicity one. Knots contained in other knots are discarded so that the set of all knots effectively partitions the graph. *Border contigs* are defined as the contigs delimiting the knots.

Based on the idea of Unicycler, GraphUnzip connects pair of contigs of multiplicity one. More specifically, it connects pairs of border contigs. In each knot, GraphUnzip finds the pairwise matching of border contigs that maximize the Hi-C interactions. The underlying statistical fact behind that procedure is that two contigs sharing many Hi-C links are probably close on the DNA strand. Sometimes it is impossible to match the haploid contigs, if Hi-C signal is too weak or when there are an uneven number of border contigs in the knot (which should not theoretically happen). In those cases, GraphUnzip

leaves the knot as it is.

4.3.4 Paths

Sometimes, matching contigs are connected by a single unanmbiguous path. Sometimes, matching border contigs may be connected through multiple paths. Finding the right paths in this case is generally a hard task, as there can easily be a huge number of possible paths between two border contigs, especially if there are loops in the knot. GraphUnzip proceeds in two steps to determine the right paths:

— The multiplicity of each *intermediate contig* of the knot (i.e. not a border contig) is already known. Given the multiplicity, the situation of the contig in the knot and its Hi-C contacts, GraphUnzip determines $m(c; b_1 - b_2)$, i.e. the number of copies of contig c that is expected between border contigs b_1 and b_2. For example, the contig_7 of Figure 4.1 has multiplicity two and probably has many Hi-C contacts with border contigs 1, 3, 9 and 11 but relatively few with border contigs 4 and 12. Thus $m(contig\_7; contig\_1 - contig\_9) = 1$, $m(contig\_7; contig\_3 - contig\_11) = 1$ and $m(contig\_7; contig\_4 - contig\_12) = 0$. Note that this is only an estimation.

— For each pair of border contigs, GraphUnzip will explore the possible paths linking them. A simple score s is devised for each path, where $l(c)$ is the length of contig c and $n(c; p)$ the number of copies of contig c that is on path p:

$$s = \sum_c |n(c, p) - m(c; b_1 - b_2)| * l(c)$$

The path minimizing the score is found through a systematic exploration of the paths of the knot.

The assembly is then updated to represent paths between border contigs as a single, unambiguous path in the assembly graph. This involves duplicating some intermediate contigs and changing links in the knot.

When all knots have been looked at, the process of determining knots and finding bridges starts again, as solving some knots may help solve others, until no more knots can be solved.

4.3.5 Output

GraphUnzip outputs an untangled assembly in gfa and/or fasta format, as well as an agp file, in the style of other scaffolders, to indicate what contigs have been duplicated and merged.

4.3.6 Haploid assembly

Many assembly pipelines today aim to provide a haploid assembly. In such cases, duplicating homozygous contigs, as GraphUnzip does, is undesirable. Therefore, GraphUnzip includes a special haploid option for these scenarios. When this option is used, GraphUnzip operates mostly as usual, with an additional step after determining the multiplicity of all contigs to determine the ploidy of the assembly. The ploidy is determined as the highest degree of multiplicity that represents more than 10% of the assembly. For example, if 50% of the assembly has a multiplicity of 4 and 2% has a higher multiplicity, GraphUnzip infers that the ploidy is 4. Border contigs are then selected from contigs with a multiplicity of 4. The paths between border contigs will necessarily be approximate, as several equivalent haplotypes can be used to construct a haploid assembly.

4.4 Results

We introduced GraphUnzip in four assembly processes to show how it improves the quality of the resulting assemblies.

4.4.1 Datasets

Puccina triticina

This diploid fungus (more specifically strain *Pt76*) was sequenced with HiFi (SRR14424683) and Hi-C (SRR14386306) in [149]. This data is particularly interesting because the physical separation of the haplotypes in the nucleus of *Puccina triticina* allowed the authors to provide a high-quality phased assembly. The length of the diploid assembly is approximately 250Mb.

Adineta vaga

We will assemble this allotetrapoloid genome, sequenced with Nanopore (SRR13348928) and Hi-C (SRR13348925) by [150]. [150] also used Illumina sequencing, providing high-quality sequencing reads to compute 31-mer completeness. The haploid size of the reference assembly proposed by the authors in this paper, and confirmed by flow cytometry, is 100Mb.

4.4.2 Protocol

We insert GraphUnzip in two distinct protocols - one for haploid assembly, one for phased assembly - to show that untangling the graph is generally a helpful process. The assembly and scaffolding software were chosen among the most standard in assembly pipelines today, and they were run with the options recommended by their documentation.

For each dataset, we tried to produce a haploid assembly, i.e. with only one out of the haplotypes, and an assembly with the haplotypes fully reconstituted. The *P. triticina* HiFi reads were assembled using hifiasm [63]. The *A. vaga* Nanopore reads were assembled using Flye [78], using the keep-haplotypes option and HairSplitter [110] to recover all haplotypes. After assembly, we introduced the GraphUnzip step with the haploid option in the relevant cases, where the graph was untangled, yielding a new assembly. For haploid assemblies, the assembly was then run through purge_haplotigs [151] and scaffolded using both Salsa2 [144] and YaHS [91]. The diploid *P. triticina* assembly was untangled using either GraphUnzip or the Hi-C option of hifiasm [152] (since they are incompatible) and yielded assemblies that did not need scaffolding. The phased *A. vaga* assembly was scaffolded by HapHiC [94] and Greenhill [93].

4.4.3 Collapsed haploid assembly: results

To assess the quality of the assemblies with or without GraphUnzip, we chose to measure the N50 (a measure of contiguity of the assembly), the number of stretches of N that the scaffolders introduced in the scaffolds and the corresponding "gapless N50", corresponding to the N50 where the scaffolds have been split at each stretch of N.

In addition to these absolute metrics, the scaffolded assembly obtained using GraphUnzip was compared to the assembly using the same protocol but without GraphUnzip. We observed the differences in the order and orientation of the contigs proposed by the two protocols. We observed three scenarios where the two assemblies diverged: most often, a

GraphUnzip scaffold contained an extra contig compared to the other assembly ; more rarely, a GraphUnzip scaffold had a contig missing in a scaffold compared to the other assembly ; even more rarely, a contig was placed completely differently in the two assemblies. These three phenomenons are quantified as "length of missing contigs", "length of wrongly inserted contigs" and "number of misplaced contigs" in Table 4.1. Note that these metrics only compare the two assemblies: no missing contigs just means that an assembly has no missing contigs *compared to the other assembly*.

To know which assembly was correct, the sequencing reads were mapped on the junction of the contigs that were specific to an assembly. For instance, if an assembly proposed the scaffold contig1-contig2 and the other assembly proposed contig1-contig3-contig2, the sequencing reads were mapped on the junctions contig1-contig2, contig1-contig3 and contig3-contig2. If many reads map on junction contig1-contig2 and none map on the other two junctions, it means that the first scaffold is more coherent with sequencing data. The reads systematically supported the junctions proposed by GraphUnzip. Almost every time, no reads mapped on the alternative junctions proposed by the scaffolders alone. In one case, reads mapped on both junctions, thus we left this case out of the analysis (though the GraphUnzip version was the only one coherent with the assembly graph).

The results are summarized in Table 4.1. The results show clearly that inserting GraphUnzip in the pipeline improves the quality of the scaffolding. The most blatant effect is on "missing contigs": these are generally small contigs that have few Hi-C contacts with other contigs. They are thus very hard to place using only Hi-C information. However, they are usually already inserted at a specific place in the graph and thus can be very accurately placed with untangling. We also noted that some of these missing contigs had actually been suppressed while purging the raw assembly, while they had been kept when purging after GraphUnzip incorporated them in larger contigs. Using GraphUnzip also decreases significantly the N gaps in the scaffolds: in many places, the actual overlap between two contigs is actually contained in the graph, and this can be used in place of an arbitrary stretch of Ns.

4.4.4 Diploid assembly: results

Our first concern was to make sure that GraphUnzip proposed a correct phasing. We could check and confirm that the phasing proposed by GraphUnzip was coherent with the reference in the *P. triticina* assembly, i.e. that two contigs linked by GraphUnzip were on

		N50 (Mb)	Number of N gaps	Gapless N50 (Mb)	length of missing contigs	length of wrongly inserted contigs	# of misplaced contigs
P. triticina	Salsa2	4.4	180	1.4	1,793,541	0	0
	GU+Salsa2	7.0	29	6.5	0	0	0
	YaHS	5.3	160	1.3	8,146,570	0	0
	GU+YaHS	7.7	31	6.5	0	0	0
A. vaga	Salsa2	13.2	177	1.5	876,069	0	4
	GU+Salsa2	13.9	98	4.5	0	0	0
	YaHS	15.0	164	1.5	626,952	35,830	1
	GU+YaHS	14.4	78	4.5	0	0	0

Table 4.1 – Comparison of the scaffolded haploid assemblies with and without using GraphUnzip (GU). Note that the missing, wrongly inserted and misplaced contigs are not an absolute evaluation of the assembly, but a comparison between the assemblies using or not GraphUnzip. These columns cannot be used to compare YaHS and Salsa2, for example.

the same haplotype in the reference. This confirms the ability of GraphUnzip to effectively phase the contigs with Hi-C.

Untangling improved the contiguity of both *A. vaga* and *P. triticina* assemblies. For *Adineta vaga*, the N50 rose from 0.37Mb to 1.38Mb, knowing that the chromosomes of *A. vaga* have sizes between 13 and 18Mb. For the *Puccina triticina* assembly, the N50 rose from 848kb to an impressive 5.9Mb, to compare to the size of the chromosomes that ranges from 4 to 10Mb.

By far the most popular pipeline to assembly diploid genomes from HiFi and Hi-C data is to use the Hi-C option of hifiasm [152]. Hifiasm actually untangles the assembly graph "under the hood" using a different algorithm than GraphUnzip. With the high-quality reference of the *P. triticina* genome, we compared the performance of the hifiasm algorithm and the GraphUnzip algorithm using QUAST [102]. Both performed similarly, GraphUnzip yielding a more contiguous and complete but slightly less precise assembly (see table 4.2). However, GraphUnzip has the great advantage of being usable with any assembler and with any kind of long read sequencing data.

We do not have access to a reliably phased reference for *Adineta vaga* and therefore must rely on reference-free methods to evaluate the quality of the scaffolds. The impact of GraphUnzip on the scaffolds of the phased assemblies was significantly greater than its impact on the scaffolds of the haploid assemblies. Consequently, we were unable to

	Genome fraction	N50 (Mb)	misassemblies	#mismatches per 100kbp	#indels per 100kbp
hifiasm (no untangling)	0.952	0.85	1364	16.89	13.76
hifiasm-Hi-C	0.927	3.02	1112	16.63	10.76
hifiasm-GraphUnzip	0.951	5.86	1390	17.42	14.02

Table 4.2 – Comparison of the performance of the untangling algorithm of hifiasm using Hi-C option (hifiasm-Hi-C) and GraphUnzip (hifiasm-GraphUnzip) on the hifiasm assembly of *Puccina triticina*

		N50 (Mb)	Number of N gaps	Gapless N50 (Mb)	Structural misassemblies	31-mer completeness (%)
A. vaga	Original assembly	0.37	0	0.37	164	90.0
	GU only	1.38	0	1.38	232	89.6
	Greenhill	95.7	5929	0.16	462	80.4
	GU+Greenhill	108.0	578	0.94	362	83.6
	HapHiC	18.7	1093	0.37	424	90.0
	GU+HapHiC	30.8	338	1.38	285	90.0

Table 4.3 – Comparison of the scaffolded phased *Adineta vaga* assemblies with and without using GraphUnzip (GU). "Structural misassemblies" are identified as such by Inspector and include missing contigs. Unlike in Table 4.1, the metrics presented here are absolute and can be used to compare Greenhill and HapHiC.

individually verify all phasing differences between the scaffolds with and without GraphUnzip. To address this, we chose to use the tool Inspector [98] to detect "structural misassemblies," which correspond to loci where the alignment of reads indicates a misassembly, including missing contigs and scaffolding errors. Additionally, we estimate the 31-mer completeness as the percentage of 31-mers present more than five times in an Illumina sequencing of the same sample that are present in the final assembly. Results are presented in Table 4.3.

Integrating GraphUnzip in the scaffolding pipeline is clearly beneficial to the quality of the final scaffolds. Final scaffolds have less gaps and structural misassemblies when GraphUnzip has been integrated in the pipeline. The measure of the N50 must be taken with scepticism: the 12 chromosomes of the 200Mb genome of *Adineta* have relatively balanced length, meaning that the true N50 of the genome is expected around 30Mb - the scaffolds proposed by Greenhill are artificially long, probably because several chromosomes have been erroneously merged. The plain Greenhill scaffolds are significantly less complete than its GraphUnzip+Greenhill counterpart because Greenhill struggled to place small

contigs and eventually discarded them, while GraphUnzip integrated them into longer contigs using the assembly graph.

4.4.5 Performance

Overall, GraphUnzip adds negligible extra time in the global assembly pipeline.

The untangling steps of GraphUnzip are extremely fast (less than 1 minutes on all the examples presented here). The bottleneck in terms of time is mapping the Hi-C reads to the assembly, an operation that must in any case be performed for the scaffolding step. By changing the graph, GraphUnzip changes the names of the contigs, but there is no need to run the alignment procedure again, the original alignment file is edited by GraphUnzip to produce an alignment file corresponding to the untangled assembly. There is a little overhead in editing the bam file but it is orders of magnitude faster than mapping the reads.

4.5 Untangling the graph with long reads

In some cases, long reads are available to help to improve the contiguity of the assembly graphs. When the reads have been used to produce the assembly using an OLC assembler, all of the information contained in the reads generally has already been exploited by the assembler. However, when the reads have been assembled following a DBG paradigm, or when the assembly was produced using shorter and more precise reads, long reads can span several contigs and provide useful information to improve the contiguity of the assembly graph.

The GraphUnzip algorithm can be readily adapted to untangle the graph using long reads. First, reads are aligned to the assembly graph minigraph [106] or GraphAligner [153]. Then, haploid contigs and knots are identified in the graph following the same procedure as with Hi-C data. The algorithm finds the best paths to link the haploid contigs. When using long reads, this process is actually simpler compared to the Hi-C procedure, as the long reads directly align along the path between two haploid contigs. Once the paths have been identified, they are extracted, similar to the Hi-C procedure.

The information contained in long reads is much more local compared to the information contained in Hi-C data and can only be effectively used to resolve “small” knots, where the distance between haploid contigs does not exceed the length of the reads.

4.6 Extending to metagenomes

The challenge of extending the GraphUnzip procedure to metagenomes lies in determining the multiplicity of all contigs. Since genomes have uneven coverage, utilizing coverage information becomes much more complex. Further work is needed to deduce the multiplicity of all contigs from the assembly graph, coverage information, Hi-C data, and potentially user-provided information.

However, we have found a workaround when GraphUnzip uses long reads to untangle the graph. In this case, determining the multiplicity of all contigs is not necessary; only haploid contigs need to be identified. As a substitute for haploid contigs, GraphUnzip identifies *pseudo-haploid* contigs. For each contig, GraphUnzip examines the paths on the graph of all contigs aligning there. If all the paths are consensual (i.e., they can all be seen as extracts of one "consensus path" on the graph), the contig is labeled as pseudo-haploid. GraphUnzip then proceeds normally. These contigs are considered "pseudo-haploid" because there is no guarantee that they are truly haploid, but a path can still be safely extracted between two pseudo-haploid contigs.

This untangling has been implemented in the Alice assembler, to untangle an unitig graph with full-length reads and in HairSplitter to improve the contiguity of the HairSplitter assembly once new contigs have been reconstructed.

4.7 Conclusion

We introduce a novel concept called "untangling an assembly graph," which involves simplifying an assembly graph using long-range data to enhance contiguity. Unlike traditional scaffolding, untangling leverages the assembly graph extensively but cannot recover links missed by the assembler. We argue that the optimal strategy combines both untangling and scaffolding. We present a method to untangle assembly graphs using Hi-C or long reads, implemented in the software GraphUnzip. In the case of haploid and phased assemblies of *Puccina triticina* and *Adineta vaga*, untangling graphs before scaffolding results in more accurate and contiguous scaffolds with fewer gaps compared to plain scaffolding. This improvement was observed across the four scaffolders tested, leading us to hypothesize that all Hi-C scaffolders would benefit from being combined with GraphUnzip.

The strategy presented in this work involves distinctly separating the untangling and

scaffolding procedures. This approach allows us to provide software that can be easily integrated with all Hi-C scaffolders and will directly benefit from advancements in scaffolding techniques. However, there is no compelling reason to believe that separating untangling and scaffolding is the optimal strategy for achieving contiguous assemblies. In fact, the missing links in the assembly graph can lead to errors in untangling, which could be mitigated by modifying the assembly graph. We propose that the best strategy is to develop strongly graph-guided scaffolders, which we leave for future work.

4.8 Availability

GraphUnzip is freely available on github, at github.com/nadegeguiglielmoni/graphunzip.

Chapter 5

Conclusion

5.1 Practical contribution

Over the course of this Ph.D., I have endeavoured to propose solutions to bioinformatics challenges that stood in the way of multi-haplotype assembly. Though the initial goal of the Ph.D. was to improve the assembly of polyploid non-model genomes, over time the focus of the Ph.D. broadened to include metagenomics. I have focused on what I believe will be the key techniques for genome assembly in the coming years, namely affordable long reads sequencing such as Nanopore, high-fidelity sequencing and Hi-C to finish assemblies. I present Figure 5.1 a graphical summary of how my different software can be integrated in an assembly pipeline.

The tools proposed in this manuscript all improve the assembly of multiple haplotypes. GenomeTailor and GraphUnzip focus on improving the large-scale quality of assemblies by reducing structural errors and ensuring that all genomes are represented as paths through the assembly graph with few missing contigs and phasing errors. In contrast, HairSplitter and Alice concentrate on base-level accuracy by recovering SNPs and small indels that may have been lost during the assembly process. Since the speed of assembly is rarely the bottleneck in sequencing pipelines, I have not emphasized the speed of these tools (except for Alice). Nevertheless, I have endeavored to make all these tools efficient, and they do not significantly increase the time required for assembly compared to equivalent software.

Moreover, the development of haplotype assembly methods led to unexpected applications. While designed to improve the quality of multi-haplotype assemblies, GenomeTailor and GraphUnzip also enhanced the quality of haploid assemblies—respectively for *Debaryomyces hansenii* and for *Puccina triticina* and *Adineta vaga*. Additionally, HairSplitter proved useful in an unexpected context: amplicon sequencing. Amplicons are markers (typically genes, such as 16S for bacteria or 28S for eukaryotes) found in all species of a kingdom and amplified by PCR to detect and distinguish species in a sample. However, highly similar species have similar genes, making them challenging to differentiate with

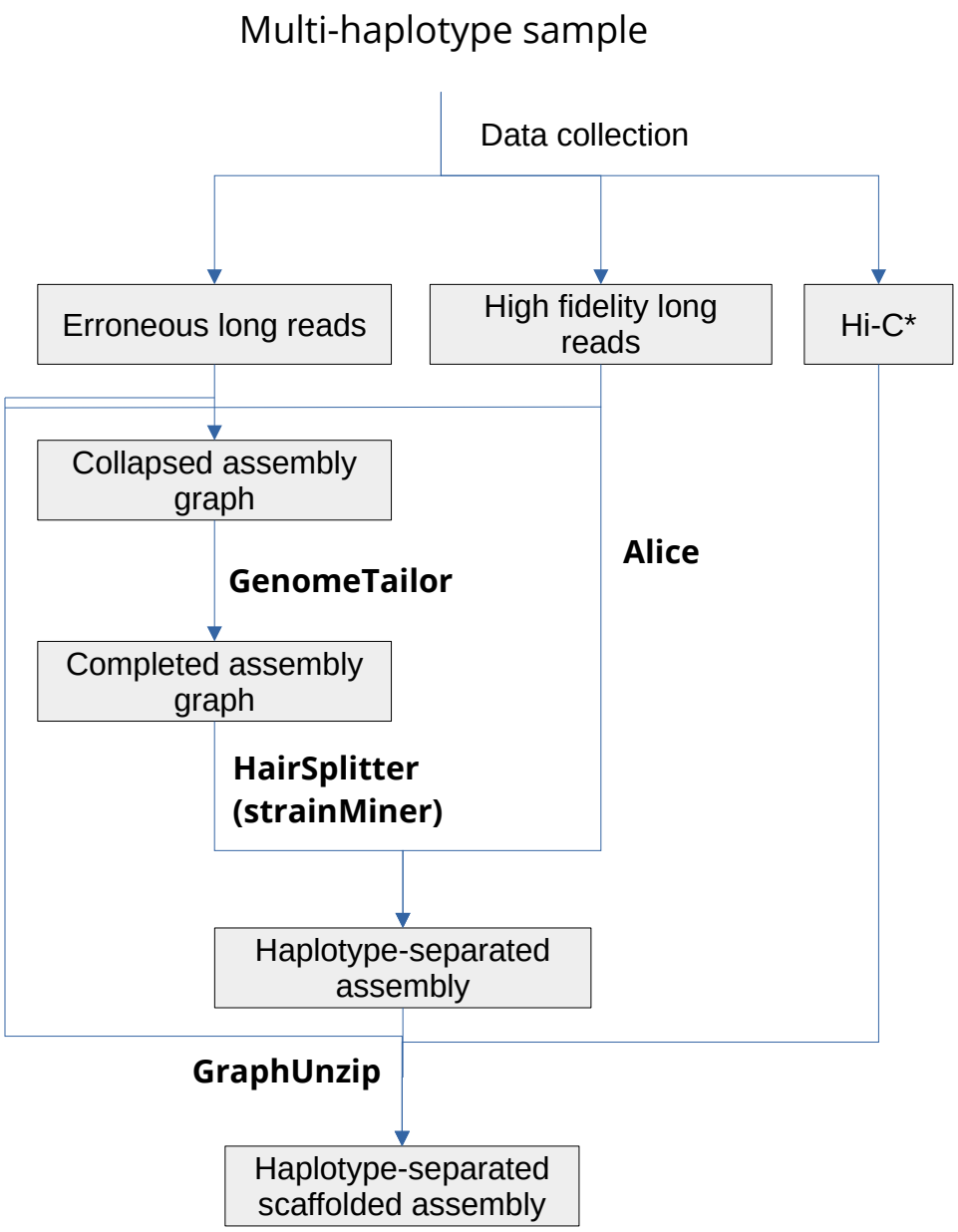


Figure 5.1 – Illustration of the organisation of the different software I introduced in this Ph.D. thesis, with their respective name in bold font. *Hi-C can only be exploited by GraphUnzip in the context in single-genome assemblies

Nanopore sequencing. To address this, amplicons are often sequenced using the more expensive and labor-intensive Sanger sequencing. We successfully applied HairSplitter to Nanopore-sequenced *Stylophora* amplicons and demonstrated (by comparing to Sanger sequencing) that HairSplitter reliably recovered the amplicon diversity.

5.2 Methodological contributions and perspectives

I hope to have convincingly demonstrated that all my software improves upon the current state of the art on some aspects. Still, none of them definitively solve all problems of genome assembly. Assemblies produced by GenomeTailor, HairSplitter, Alice, and GraphUnzip may still contain errors and be fragmented and incomplete. However, they are less incomplete, less incorrect and less fragmented that state-of-the art assemblies. I anticipate (and hope) that most of these tools will be surpassed in the coming years. Nevertheless, this thesis introduces several new methods that I hope will inspire future work, which I will now list in the order of their appearance in the manuscript.

Completing assembly graphs GenomeTailor introduces the concept of completing an assembly graph, which consists in adding links and creating new contigs to try to align all reads from end-to-end on the assembly graph. Even with the crude strategy implemented in GenomeTailor, this procedure significantly improves the quality of (multi-haplotype) assemblies. I hope that in the future, more sophisticated programs will be developed to correct and complete assembly graphs more effectively than GenomeTailor.

Metagenomic variant calling HairSplitter proposes a new statistical test for variant calling in the challenging context of metagenomic assembly. The test is robust to unbalanced coverages and arbitrary numbers of strains. While the intuition behind this test - testing correlation across multiple loci - was already presented in [85], the mathematical formulation proposed here is new and much easier to use compared to the one in [85]. In the strainMiner paper, this test is used to transform the metagenomic variant calling problem into a well-defined optimisation problem. These results can easily be used as a basis for future work, even if the error rate of the reads decrease significantly.

MSR sketching My personal favorite idea presented in this manuscript is the use of Mapping-friendly Sequence Reductions (MSRs) as a sketching technique. MSRs generate

sketches in the form of short sequences, which are easy to manipulate. These sketches have unique properties that make them particularly well-suited for haplotype-aware assembly. Predicting the potential applications of MSR sketching is challenging, as I have explored only a small fraction of the wide range of existing MSRs. Beyond the presented application in assembly and its natural extension to alignment, I envision MSRs being used to sketch large collections of assembled genomes. Such collections could, for example, enhance taxonomic assignment, which is currently limited by the size of databases — the largest Kraken2 database is 684 GB [135].

Graph untangling The main methodological contribution of GraphUnzip is its attempt to fully exploit assembly graphs during the scaffolding process. GraphUnzip can be conveniently integrated with any existing scaffolder. While I am not sure that all future scaffolders will adopt the two-step strategy of untangling followed by scaffolding, I believe that they will make much greater use of the information contained in assembly graphs than their current counterparts, especially when scaffolding multi-haplotype assemblies.

5.3 Future applications

Assembling genomes is rarely the final objective of a study, but merely an important step to answer a biological question.

Many studies today are limited by the production of haploid genomes. For example, Nature published in 2020 a study revealing structural rearrangements between several wheat strains and their impact on a disease resistance [154]. In this study, the assemblies produced and compared by the authors were all collapsed version of the hexaploid wheat genomes. As such, the differences identified by the authors between the strains could also have been present, undetected, between the haplotypes of a single strain [1]. In the future, haplotype-separated assemblies will allow finer comparisons between genomes.

A very promising avenue for understanding microbiomes today is the deduction of metabolisms and community-scale functional networks directly from the metagenome. This field of research is very active, with practical applications such as understanding biochemical cycles [155], diabetes [156], and nutrition [157]. However, current limitations in metagenome assemblies prevent existing software from accounting for conspecific strains,

1. In practice, haplotypes are probably highly similar, as the strains are highly inbred. I am certainly not questioning the scientific validity of the study.

even though the coexistence of such strains likely influences the behavior of microbiomes [158]. The advent of haplotype-specific assemblies will likely enhance the ability of these tools to understand the dynamics of microbiomes.

Improving the scaffolding of diploid and polyploid genomes is necessary to improve our understanding of the evolution of large genomes, which is tightly linked with heterozygosity. For example, a 2013 study of the asexual animal *Adineta vaga* published in Nature [159] observed that "allelic regions [were] rearranged and sometimes even found on the same chromosome", which led the authors to believe that "such structure [did] not allow meiotic pairing". While the question of meiotic pairing in *A. vaga* remains open, further study by the same group revealed that this observation was the result of scaffolding errors [160]. Current research on *A. vaga* includes GraphUnzip to provide a new telomere-to-telomere phased assembly which will hopefully help solve the mystery shrouding the asexual evolution of this species.

BIBLIOGRAPHY

1. Magner, L. N., *A History of the Life Sciences, Revised and Expanded* (CRC Press, 2002).
2. Lane, N., The unseen World: Reflections on Leeuwenhoek (1677) 'Concerning little animals', *Philosophical Transactions of The Royal Society B Biological Sciences* **370** (Apr. 2015).
3. Pasteur, L., Mémoire sur la fermentation appelée lactique, *Ann. Chim. Phys. Ser.* **52,** 404–418 (Jan. 1858).
4. Koch, R., Verfahren zur Untersuchung, zum Konservieren und Photographieren der Bakterien, *Beiträge zur Biologie der Pflanzen* **2,** 399–434 (1877).
5. Gradmann, C. & Koch, R., Die Ätiologie Der Tuberkulose (1882), *Robert Koch: Zentrale Texte,* 113–131 (2018).
6. Madigan, M. & Martinko, J., *Brock Biology of Microorganisms (13th ed.)* (ed Education, P.) ISBN: 978-0-321-73551-5 (2006).
7. Hiltner, L., Die Keimungsverhältnisse der Leguminosensamen und ihre Beeinflussung durch Organismenwirkung, *Arb Biol Abt Land u Forstw K Gsndhtsamt* (eds P, P. & (Eds.), S. J.) 1–545 (1902).
8. Berg, G. *et al.,* Microbiome definition re-visited: old concepts and new challenges, *Microbiome* **8** (Dec. 2020).
9. Wade, W., Unculturable bacteria - The uncharacterized organisms that cause oral infections, *Journal of the Royal Society of Medicine* **95,** 81–3 (Mar. 2002).
10. Yen, S. & Johnson, J., Metagenomics: a path to understanding the gut microbiome, *Mammalian Genome* **32** (Aug. 2021).
11. Wilson, D., Binford, L. & Hickson, S., The gut microbiome and mental health, *Journal of Holistic Nursing* **42,** 79–87 (1 2023).
12. Birnbaum, C. & Trevathan-Tackett, S., Aiding coastal wetland restoration via the belowground soil microbiome: an overview, *Restoration Ecology* **31** (7 2022).

13. Jiao, S., Chen, W. & Wei, G., Core microbiota drive functional stability of soil microbiome in reforestation ecosystems, *Global Change Biology* **28,** 1038–1047 (3 2021).

14. Jacoby, R. P., Peukert, M., Succurro, A., Kopřivová, A. & Kopriva, S., The role of soil microorganisms in plant mineral nutrition—current knowledge and future directions, *Frontiers in Plant Science* **8** (2017).

15. Dick, G., The microbiomes of deep-sea hydrothermal vents: distributed globally, shaped locally, *Nature Reviews Microbiology* **17** (Mar. 2019).

16. `https://en.wikipedia.org/wiki/DNA`.

17. Crick, F., On Protein Synthesis, *Symposia of the Society for Experimental Biology, Number XII: The Biological Replication of Macromolecules* (ed Press, C. U.) 138–163 (1958).

18. Pinkard, O., McFarland, S., Sweet, T. & Coller, J., Quantitative tRNA-sequencing uncovers metazoan tissue-specific tRNA regulation, *Nature Communications* **11,** 4104 (Aug. 2020).

19. Lau, N., Lim, L., Weinstein, E. & Bartel, D., Lau, N. C., Lim, L. P., Weinstein, E. G. , Bartel, D. P. An abundant class of tiny RNAs with probable regulatory roles in C. elegans. Science 294, 858-862, *Science (New York, N.Y.)* **294,** 858–62 (Nov. 2001).

20. Yang, J. & Hansen, A., Enhancer selectivity in space and time: from enhancer–promoter interactions to promoter activation, *Nature Reviews Molecular Cell Biology* **25** (Feb. 2024).

21. Dupont, C., Armant, D. & Brenner, C., Epigenetics: Definition, Mechanisms and Clinical Perspective, *Seminars in reproductive medicine* **27,** 351–7 (Oct. 2009).

22. Schörner, M., Beyer, S., Southall, J., Cogdell, R. & Köhler, J., Conformational Memory of a Protein Revealed by Single-Molecule Spectroscopy, *The Journal of Physical Chemistry B* **119,** 150929213049000 (Sept. 2015).

23. Callaway, E., Million-year-old mammoth genomes shatter record for oldest ancient DNA, *Nature* **590** (Feb. 2021).

24. Watson, J. D. & Crick, F. H., *The structure of DNA in Cold Spring Harbor symposia on quantitative biology* **18** (1953), 123–131.

25. Belcour, A. *et al.,* Metage2Metabo, microbiota-scale metabolic complementarity for the identication of key species, *eLife* **9** (Dec. 2020).

26. Pavlopoulos, G. *et al.,* Unraveling the functional dark matter through global metagenomics, *Nature* **622** (Oct. 2023).

27. Dairawan, M. & Shetty, P. J., The evolution of DNA extraction methods, *Am. J. Biomed. Sci. Res* **8,** 39–45 (2020).

28. Dahm, R., Discovering DNA: Friedrich Miescher and the early years of nucleic acid research, *Human genetics* **122,** 565–81 (Feb. 2008).

29. Wu, R., Nucleotide sequence analysis of DNA. I. Partial sequence of the cohesive ends of bacteriophage λ and 186 DNA, *Journal of molecular biology* **51,** 501–21 (Sept. 1970).

30. Sanger, F., Nicklen, S. & Coulson, A., DNA Sequencing With Chain Terminating Inhibitors, *Proceedings of the National Academy of Sciences of the United States of America* **74,** 5463–7 (Jan. 1978).

31. Shendure, J. *et al.,* DNA sequencing at 40: Past, present and future, *Nature* **550** (Oct. 2017).

32. Fiers, W. *et al.,* Complete nucleotide sequence of bacteriophage MS2 RNA: primary and secondary structure of the replicase gene, *Nature* **260,** 500–7 (May 1976).

33. Fleischmann, R. *et al.,* Whole-Genome Random Sequencing and Assembly of Haemophilus Influenzae Rd, *Science (New York, N.Y.)* **269,** 496–512 (Aug. 1995).

34. Venter, J. C. *et al.,* The Sequence of the Human Genome, *Science* **291,** 1304–1351 (2001).

35. `https://www.genome.gov/about-genomics/educational-resources/fact-sheets/human-genome-project`.

36. International Human Genome Sequencing Consortium and others, Initial sequencing and analysis of the human genome, *Nature* **409,** 860–921 (2001).

37. Green, P., Against a whole-genome shotgun, *Genome Research* **7,** 410–417 (1997).

38. Venter, J. C. *et al.,* Shotgun sequencing of the human genome, *Science* **280,** 1540–1542 (1998).

39. Lancet, D., Consortium, H. G. S., *et al.,* Initial sequencing and analysis of the human genome, *Nature* **409,** 860–921 (2001).

40. Venter, J. C. *et al.,* The sequence of the human genome, *science* **291,** 1304–1351 (2001).

41. Margulies, M. *et al.,* Genome sequencing in microfabricated high-density picolitre reactors, *Nature* **437,** 376–380, ISSN: 00280836 (2005).

42. Rothberg, J. M. *et al.,* An integrated semiconductor device enabling non-optical genome sequencing, *Nature* **475,** 348–352, ISSN: 00280836 (2011).

43. McKernan, K. J. *et al.,* Sequence and structural variation in a human genome uncovered by short-read, massively parallel ligation sequencing using two-base encoding, *Genome Research* **19,** 1527–1541, ISSN: 10889051 (2009).

44. Bentley, D. R. *et al.,* Accurate whole human genome sequencing using reversible terminator chemistry, *Nature* **456,** 53–59, ISSN: 00280836 (2008).

45. Vollger, M. *et al.,* Segmental duplications and their variation in a complete human genome, *Science (New York, N.Y.)* **376,** eabj6965 (Apr. 2022).

46. MacKenzie, M. & Argyropoulos, C., An Introduction to Nanopore Sequencing: Past, Present, and Future Considerations, *Micromachines* **14,** 459 (Feb. 2023).

47. Hu, T., Chitnis, N., Monos, D. & Dinh, A., Next-generation sequencing technologies: An overview, *Human Immunology* **82** (Mar. 2021).

48. Wenger, A. M. *et al.,* Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome, *Nature biotechnology* **37,** 1155–1162 (2019).

49. Stoeck, T., Katzenmeier, S., Breiner, H.-W. & Rubel, V., Nanopore duplex sequencing as an alternative to Illumina MiSeq sequencing for eDNA-based biomonitoring of coastal aquaculture impacts, *Metabarcoding and Metagenomics* **8** (May 2024).

50. Sutton, Granger and White, Owen and Adams, Mark D. and Kerlavage, Anthony R., TIGR Assembler: A new Tool for Assembling Large Shotgun Projects, *Genome Science and Technology* **1,** 9–19 (1995).

51. Flye Sainte-Marie, C., 48, *L'Intermédiaire des Mathématiciens* **1,** 107–110 (1894).

52. De Bruijn, N. G., A combinatorial problem, *Koninklijke Nederlandse Akademie v. Wetenschappen* **49** (1946).

53. Bankevich, A. *et al.,* SPAdes: A new genome assembly algorithm and its applications to single-cell sequencing, *Journal of Computational Biology* **19,** 455–477, ISSN: 10665277 (2012).

54. Li, D., Liu, C.-M., Luo, R., Sadakane, K. & Lam, T.-W., MEGAHIT: An ultra-fast single-node solution for large and complex metagenomics assembly via succinct de Bruijn graph, *Bioinformatics (Oxford, England)* **31** (Sept. 2014).

55. Chikhi, R. & Rizk, G., Space-efficient and exact de Bruijn graph representation based on a Bloom filter, *Algorithms for Molecular Biology logo Algorithms for Molecular Biology* **8** (2013).

56. Ekim, B., Berger, B. & Chikhi, R., Minimizer-space de Bruijn graphs: Whole-genome assembly of long reads in minutes on a personal computer, *Cell Systems* **12** (Sept. 2021).

57. Staden, R., A strategy of DNA sequencing employing computer programs, *Nucleic Acids Research* **6,** 2601–2610 (1979).

58. Guiglielmoni, N., *Improving genome assemblies of non-model non-vertebrate animals with long reads and Hi-C* Available at `https://difusion.ulb.ac.be/vufind/Record/ULB-DIPOT:oai:dipot.ulb.ac.be:2013/331242/Holdings`, PhD thesis (Université Libre de Bruxelles, Brussels, Belgium, June 2021).

59. Denisov, G. *et al.,* Consensus generation and variant detection by Celera Assembler, *Bioinformatics* **24,** 1035–1040, ISSN: 13674803 (2008).

60. Vaser, R. & Šikić, M., Time-and memory-efficient genome assembly with Raven, *Nature Computational Science* **1,** 332–336 (2021).

61. Li, H., Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences, *Bioinformatics* **32,** 2103–2110 (2016).

62. Koren, S. *et al.,* Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation, *Genome Research* **25,** 1–11, eprint: `071282` (2017).

63. Cheng, H., Concepcion, G. T., Feng, X., Zhang, H. & Li, H., Haplotype-resolved *de novo* assembly using phased assembly graphs with hifiasm, *Nature Methods,* 1–6 (2021).

64. Wick, R. R., Judd, L. M. & Holt, K. E., Assembling the perfect bacterial genome using Oxford Nanopore and Illumina sequencing, *PLOS Computational Biology* **19,** e1010905 (2023).

65. Makałowski, W., The human genome structure and organization. *Acta Biochimica Polonica* **48,** 587–598 (2001).

66. Ceppellini, R. *et al., Genetics of leukocyte antigens: a family study of segregation and linkage. in Report of Histocompatibility testing 1967* (ed Curtoni E.S. Mattiuz P.L., T. R.) (1967).

67. Rice, E. S. & Green, R. E., New approaches for genome assembly and scaffolding, *Annual review of animal biosciences* **7,** 17–40 (2019).

68. Boetzer, M. & Pirovano, W., SSPACE-LongRead: scaffolding bacterial draft genomes using long read sequence information, *BMC bioinformatics* **15,** 1–9 (2014).

69. Lieberman-Aiden, E. *et al.,* Comprehensive mapping of long-range interactions reveals folding principles of the human genome, *science* **326,** 289–293 (2009).

70. Burton, J. N. *et al.,* Chromosome-scale scaffolding of de novo genome assemblies based on chromatin interactions, *Nature biotechnology* **31,** 1119–1125 (2013).

71. Rautiainen, M. *et al.,* Telomere-to-telomere assembly of diploid chromosomes with Verkko, *Nature Biotechnology* **41,** 1474–1482 (2023).

72. Nurk, S., Meleshko, D., Korobeynikov, A. & Pevzner, P., MetaSPAdes: A new versatile metagenomic assembler, *Genome Research* **27,** gr.213959.116 (Mar. 2017).

73. Benoit, G. *et al.,* High-quality metagenome assembly from long accurate reads with metaMDBG, *Nature Biotechnology,* 1–6 (Jan. 2024).

74. Feng, X., Cheng, H., Portik, D. & Li, H., Metagenome assembly of high-fidelity long reads with hifiasm-meta, *Nature Methods* **19,** 1–4 (June 2022).

75. Kolmogorov, M. *et al.,* metaFlye: scalable long-read metagenome assembly using repeat graphs, *Nature Methods* **17,** 1–8 (Nov. 2020).

76. Friedmann, H., Escherich and Escherichia, *EcoSal Plus* **6,** 1–34 (May 2014).

77. Méric, G., Hitchings, M., Pascoe, B. & Sheppard, S., From Escherich to the Escherichia coli genome, *The Lancet Infectious Diseases* **16,** 634–636 (June 2016).

78. Kolmogorov, M., Yuan, J., Lin, Y. & Pevzner, P. A., Assembly of long, error-prone reads using repeat graphs, *Nature Biotechnology* **37,** 540–546 (2019).

79. Ruan, J. & Li, H., Fast and accurate long-read assembly with wtdbg2, en, *Nature Methods* **17,** 155–158, ISSN: 1548-7105 (2020).

80. Edge, P., Bafna, V. & Bansal, V., HapCUT2: robust and accurate haplotype assembly for diverse sequencing technologies, *Genome research* **27,** 801–812 (2017).

81. Martin, M. *et al.,* WhatsHap: fast and accurate read-based phasing, *BioRxiv,* 085050 (2016).

82. Schrinner, S. D. *et al.,* Haplotype threading: accurate polyploid phasing from long reads, *Genome biology* **21,** 1–22 (2020).

83. Luo, X., Kang, X. & Schönhuth, A., Phasebook: haplotype-aware de novo assembly of diploid genomes from long reads, *Genome biology* **22,** 1–26 (2021).

84. Vicedomini, R., Quince, C., Darling, A. E. & Chikhi, R., Strainberry: automated strain separation in low-complexity metagenomes using long reads, en, *Nature Communications* **12,** 4485, ISSN: 2041-1723 (July 2021).

85. Feng, Z., Clemente, J., Wong, B. & Schadt, E., Detecting and phasing minor single-nucleotide variants from long-read sequencing data, *Nature Communications* **12,** 3032 (May 2021).

86. Kazantseva, E., Donmez, A., Pop, M. & Kolmogorov, M., *stRainy: assembly-based metagenomic strain phasing using long reads* en, preprint (Bioinformatics, Feb. 2023).

87. Luo, X., Kang, X. & Schönhuth, A., Strainline: full-length de novo viral haplotype reconstruction from noisy long reads, *Genome Biology* **23** (Jan. 2022).

88. Cai, D., Shang, J. & Sun, Y., HaploDMF: viral Haplotype reconstruction from long reads via Deep Matrix Factorization, *Bioinformatics* **38** (Oct. 2022).

89. Ghurye, J., Pop, M., Koren, S., Bickhart, D. & Chin, C.-S., Scaffolding of long read assemblies using long range contact information, *BMC genomics* **18,** 1–11 (2017).

90. Dudchenko, O. *et al.,* De novo assembly of the Aedes aegypti genome using Hi-C yields chromosome-length scaffolds, *Science* **356,** 92–95 (2017).

91. Zhou, C., McCarthy, S. A. & Durbin, R., YaHS: yet another Hi-C scaffolding tool, *Bioinformatics* **39,** btac808 (2023).

92. Zhang, X., Zhang, S., Zhao, Q., Ming, R. & Tang, H., Assembly of allele-aware, chromosomal-scale autopolyploid genomes based on Hi-C data, *Nature plants* **5,** 833–845 (2019).

93. Ouchi, S., Kajitani, R. & Itoh, T., GreenHill: a de novo chromosome-level scaffolding and phasing tool using Hi-C, *Genome Biology* **24,** 162 (2023).

94. Zeng, X. *et al.,* Chromosome-level scaffolding of haplotype-resolved assemblies using Hi-C data without reference genomes, *Nature Plants,* 1–17 (2024).

95. Baudry, L., Foutel-Rodier, T., Thierry, A., Koszul, R. & Marbouty, M., MetaTOR: a computational pipeline to recover high-quality metagenomic bins from mammalian gut proximity-ligation (meta3C) libraries, *Frontiers in genetics* **10,** 753 (2019).

96. Du, Y. & Sun, F., MetaCC allows scalable and integrative analyses of both long-read and short-read metagenomic Hi-C data, *Nature Communications* **14,** 6231 (2023).

97. Mikheenko, A., Saveliev, V. & Gurevich, A., MetaQUAST: evaluation of metagenome assemblies, *Bioinformatics* **32,** 1088–1090 (2016).

98. Chen, Y., Zhang, Y., Wang, A. Y., Gao, M. & Chong, Z., Accurate long-read de novo assembly evaluation with Inspector, *Genome Biology* **22,** 1–21 (2021).

99. Li, K., Xu, P., Wang, J., Yi, X. & Jiao, Y., Identification of errors in draft genome assemblies at single-nucleotide resolution for quality assessment and improvement, *Nature Communications* **14,** 6556 (2023).

100. Zhang, Y., Lu, H.-W. & Ruan, J., GAEP: a comprehensive genome assembly evaluating pipeline, *Journal of Genetics and Genomics* **50,** 747–754 (2023).

101. Madrigal, G., Minhas, B. F. & Catchen, J., Klumpy: A tool to evaluate the integrity of long-read genome assemblies and illusive sequence motifs, *Molecular Ecology Resources,* e13982 (2024).

102. Gurevich, A., Saveliev, V., Vyahhi, N. & Tesler, G., QUAST: Quality assessment tool for genome assemblies, *Bioinformatics (Oxford, England)* **29** (Feb. 2013).

103. Rhie, A., Walenz, B. P., Koren, S. & Phillippy, A. M., Merqury: reference-free quality, completeness, and phasing assessment for genome assemblies, *Genome biology* **21,** 1–27 (2020).

104. Mapleson, D., Accinelli, G., Kettleborough, G., Wright, J. & Clavijo, B., KAT: A K-mer Analysis Toolkit to quality control NGS datasets and genome assemblies, *Bioinformatics (Oxford, England)* **33** (Oct. 2016).

105. Asalone, K. C. *et al.,* Regional sequence expansion or collapse in heterozygous genome assemblies, *PLoS computational biology* **16,** e1008104 (2020).

106. Li, H., Feng, X. & Chu, C., The design and construction of reference pangenome graphs with minigraph, *Genome biology* **21,** 1–19 (2020).

107. Eizenga, J. M., Lorig-Roach, R., Meredith, M. M. & Paten, B., *Walk-preserving transformation of overlapped sequence graphs into blunt sequence graphs with GetBlunted in Conference on Computability in Europe* (2021), 169–177.

108. Garrison, E. & Guarracino, A., Unbiased pangenome graphs, *Bioinformatics* **39,** btac743 (2023).

109. Vaser, R., Sović, I., Nagarajan, N. & Šikić, M., Fast and accurate de novo genome assembly from long uncorrected reads, *Genome research* **27,** 737–746 (2017).

110. Faure, R., Lavenier, D. & Flot, J.-F., HairSplitter: haplotype assembly from long, noisy reads, *bioRxiv,* 2024–02 (2024).

111. Seemann, T. & Booth, T., Barrnap: basic rapid ribosomal RNA predictor, *GitHub repository* (2018).

112. Long, E. O. & Dawid, I. B., Repeated genes in eukaryotes, *Annual review of biochemistry* **49,** 727–764 (1980).

113. Wick, R. R., Schultz, M. B., Zobel, J. & Holt, K. E., Bandage: interactive visualization of de novo genome assemblies, *Bioinformatics* **31,** 3350–3352 (2015).

114. Hickey, G. *et al.,* Pangenome graph construction from genome alignments with Minigraph-Cactus, *Nature biotechnology* **42,** 663–673 (2024).

115. Nurk, S. *et al.,* HiCanu: accurate assembly of segmental duplications, satellites, and allelic variants from high-fidelity long reads, *Genome research* **30,** 1291–1305 (2020).

116. Luo, X., Kang, X. & Schönhuth, A., VeChat: correcting errors in long reads using variation graphs, *Nature communications* **13,** 6657 (2022).

117. Faure, R., Flot, J.-F. & Lavenier, D., *HairSplitter: separating strains in metagenome assemblies with long reads in Proc. JOBIM* (2023).

118. Truong, T. K. M., Faure, R. & Andonov, R., *Assembling close strains in metagenome assemblies using discrete optimization in Proceedings of the 15th International Conference on Bioinformatics Models, Methods and Algorithms* (Feb. 2024), `https://bioinformatics.scitevents.org`.

119. Travers, K. J., Chin, C.-S., Rank, D. R., Eid, J. S. & Turner, S. W., A flexible and efficient template format for circular consensus sequencing and SNP detection, *Nucleic acids research* **38,** e159–e159 (2010).

120. Loomis, E. W. *et al.,* Sequencing the unsequenceable: expanded CGG-repeat alleles of the fragile X gene, *Genome research* **23,** 121–128 (2013).

121. Li, H. & Durbin, R., Genome assembly in the telomere-to-telomere era, *Nature Reviews Genetics,* 1–13 (2024).

122. Logsdon, G. A., Vollger, M. R. & Eichler, E. E., Long-read human genome sequencing and its applications, *Nature Reviews Genetics* **21,** 597–614 (2020).

123. Yu, W. *et al.,* Comprehensive Assessment of Eleven de novo HiFi Assemblers on Complex Eukaryotic Genomes and Metagenomes, *bioRxiv,* 2023–06 (2023).

124. Rowe, W. P., When the levee breaks: a practical guide to sketching algorithms for processing the flood of genomic data, *Genome biology* **20,** 1–12 (2019).

125. Blassel, L., Medvedev, P. & Chikhi, R., Mapping-friendly sequence reductions: Going beyond homopolymer compression, *Iscience* **25** (2022).

126. Shafin, K. *et al.,* Nanopore sequencing and the Shasta toolkit enable efficient de novo assembly of eleven human genomes, *Nature biotechnology* **38,** 1044–1053 (2020).

127. Li, H. *et al.,* The sequence alignment/map format and SAMtools, *bioinformatics* **25,** 2078–2079 (2009).

128. Teeling, H., Waldmann, J., Lombardot, T., Bauer, M. & Glöckner, F. O., TETRA: a web-service and a stand-alone program for the analysis and comparison of tetranucleotide usage patterns in DNA sequences, *BMC bioinformatics* **5,** 1–7 (2004).

129. Ondov, B. D. *et al.,* Mash: fast genome and metagenome distance estimation using MinHash, *Genome biology* **17,** 1–14 (2016).

130. Bradley, P., Den Bakker, H. C., Rocha, E. P., McVean, G. & Iqbal, Z., Ultrafast search of all deposited bacterial and viral genomic data, *Nature biotechnology* **37,** 152–159 (2019).

131. Li, H., Minimap2: pairwise alignment for nucleotide sequences, *Bioinformatics* **34,** 3094–3100 (2018).

132. ENA, *ENA statistics* `https://www.ebi.ac.uk/ena/browser/about/statistics` (2024).

133. Loh, P.-R., Baym, M. & Berger, B., Compressive genomics, *Nature biotechnology* **30,** 627–630 (2012).

134. Burrows, M. & Wheeler, D., A Block-Sorting Lossless Data Compression Algorithm, *Digital Systems Research Center Research Reports* **1** (July 1995).

135. Wood, D. E. & Salzberg, S. L., Kraken: ultrafast metagenomic sequence classification using exact alignments, *Genome biology* **15,** 1–12 (2014).

136. Břinda, K. *et al.,* Efficient and Robust Search of Microbial Genomes via Phylogenetic Compression, *bioRxiv,* 2023–04 (2023).

137. Chikhi, R., Holub, J. & Medvedev, P., Data structures to represent a set of k-long DNA sequences, *ACM Computing Surveys (CSUR)* **54,** 1–22 (2021).

138. Salikhov, K., Sacomoto, G. & Kucherov, G., Using cascading Bloom filters to improve the memory usage for de Brujin graphs, *Algorithms for Molecular Biology* **9,** 1–10 (2014).

139. Chin, C.-S. & Khalak, A., Human genome assembly in 100 minutes, *BioRxiv,* 705616 (2019).

140. Carroll, L., *Alice's Adventures in Wonderland* (Macmillan and Co., London, 1865).

141. Kazemi, P. *et al.,* ntHash2: recursive spaced seed hashing for nucleotide sequences, *Bioinformatics* **38,** 4812–4813 (2022).

142. Chikhi, R., Limasset, A. & Medvedev, P., Compacting de Bruijn graphs from sequencing data quickly and in low memory, *Bioinformatics* **32,** i201–i208 (2016).

143. Bankevich, A., Bzikadze, A. V., Kolmogorov, M., Antipov, D. & Pevzner, P. A., Multiplex de Bruijn graphs enable genome assembly from long, high-fidelity reads, *Nature biotechnology* **40,** 1075–1081 (2022).

144. Ghurye, J. *et al.,* Integrating Hi-C links with assembly graphs for chromosome-scale assembly, en, *PLOS Computational Biology* **15,** Number: 8 Publisher: Public Library of Science, e1007273, ISSN: 1553-7358, https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1007273 (2021) (Aug. 2019).

145. Guan, D., McCarthy, S., Ning, Z., Wang, G. & Wang, Y., Efficient iterative Hi-C scaffolder based on N-best neighbors, *BMC Bioinformatics* **22** (Nov. 2021).

146. Baudry, L. *et al.,* instaGRAAL: chromosome-level quality scaffolding of genomes using a proximity ligation-based scaffolder, *Genome Biology* **21,** 1–22 (2020).

147. Sur, A., Noble, W. S. & Myler, P. J., *A benchmark of Hi-C scaffolders using reference genomes and de novo assemblies* en, preprint (Genomics, Apr. 2022), http://biorxiv.org/lookup/doi/10.1101/2022.04.20.488415 (2022).

148. Wick, R. R., Judd, L. M., Gorrie, C. L. & Holt, K. E., Unicycler: Resolving bacterial genome assemblies from short and long sequencing reads, en, *PLOS Computational Biology* **13,** Publisher: Public Library of Science, e1005595, ISSN: 1553-7358, https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1005595 (2021) (June 2017).

149. Duan, H. *et al.,* Physical separation of haplotypes in dikaryons allows benchmarking of phasing accuracy in Nanopore and HiFi assemblies with Hi-C data, *Genome Biology* **23,** 84, ISSN: 1474-760X, https://doi.org/10.1186/s13059-022-02658-2 (2022) (Mar. 2022).

150. Simion, P. *et al.,* Chromosome-level genome assembly reveals homologous chromosomes and recombination in asexual rotifer Adineta vaga, *Science Advances* **7,** eabg4216, ISSN: 2375-2548, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8494291/ (2022).

151. Roach, M., Schmidt, S. & Borneman, A., Purge Haplotigs: Allelic contig reassignment for third-gen diploid genome assemblies, *BMC Bioinformatics* **19** (Nov. 2018).

152. Cheng, H. *et al.,* Haplotype-resolved assembly of diploid genomes without parental data, en, *Nature Biotechnology* **40,** Number: 9 Publisher: Nature Publishing Group, 1332–1335, ISSN: 1546-1696, https://www.nature.com/articles/s41587-022-01261-x (2022) (Sept. 2022).

153. Rautiainen, M. & Marschall, T., GraphAligner: rapid and versatile sequence-to-graph alignment, *Genome biology* **21,** 253 (2020).

154. Walkowiak, S. *et al.,* Multiple wheat genomes reveal global variation in modern breeding, *Nature* **588,** 277–283 (2020).

155. Zhou, Z. *et al.,* METABOLIC: high-throughput profiling of microbial genomes for functional traits, metabolism, biogeochemistry, and community-scale functional networks, *Microbiome* **10,** 33 (2022).

156. Belcour, A. *et al.,* Metage2Metabo, microbiota-scale metabolic complementarity for the identification of key species, *Elife* **9,** e61968 (2020).

157. Zorrilla, F., Buric, F., Patil, K. R. & Zelezniak, A., metaGEM: reconstruction of genome scale metabolic models directly from metagenomes, *Nucleic Acids Research* **49,** e126–e126 (2021).

158. Zhu, A., Sunagawa, S., Mende, D. R. & Bork, P., Inter-individual differences in the gene content of human gut bacterial species, *Genome biology* **16,** 1–13 (2015).

159. Flot, J.-F. *et al.,* Genomic evidence for ameiotic evolution in the bdelloid rotifer Adineta vaga, *Nature* **500,** 453–457 (2013).

160. Simion, P. *et al.,* Chromosome-level genome assembly reveals homologous chromosomes and recombination in asexual rotifer Adineta vaga, *Science Advances* **7,** eabg4216 (2021).

Résumé en français

Note to non-French-speaking readers: This 5-page summary of the thesis in French is a requisite for French universities. It does not contain any original information.

Introduction

Si l'étude des grands organismes eucaryotes est historiquement bien documentée, on ne peut en dire autant de celle des microorganismes, qui représentent pourtant l'écrasante majorité de la biodiversité. Leur petite taille et leur diversité rend leur observation et leur compréhension complexe et, des avis subjectifs que j'ai pu recueillir, ennuyeuse. Je m'en étouffe d'indignation. Historiquement, des microorganismes ont été observés pour la première fois en 1676. Au XIXème siècle, on découvre leur rôle dans la fermentation et les maladies. Au XXème siècle, on découvre que les microorganismes forment des écosystèmes invisibles. Grace aux techniques de génomique du XXIème siècle, on a découvert que ces écosystèmes, qu'on appelle microbiomes, ont en fait une importance primordiale sur notre santé, à travers notre intestin. Depuis, les découvertes s'accumulent. Par exemple, les microbiomes joueraient un rôle central dans les cycles biogéochimiques, dans la croissance des cultures et dans certaines chaînes alimentaires. L'étendue des découvertes dans à venir dans ce domaine est par essence difficile à estimer, mais à mon avis immense. L'objectif de ma thèse est d'améliorer les méthodes informatiques utilisées en génomique et de fournir aux biologistes de nouveaux outils permettant l'étude des microbiomes.

Le séquençage de l'ADN est la technique centrale de l'étude des microbiomes. Le premier séquençage a été publié en 1970 et en 1977 Frederick Sanger publie la première méthode utilisée à grande échelle, ce qui lui rapportera un deuxième prix Nobel. La méthode proposée par Sanger est chère pour les longs génomes - la première bactérie n'est séquencée qu'en 1995 et le premier génome humain en 2001, pour la modique somme de 3 milliards d'euros. Depuis, des séquenceurs de deuxième puis de troisième génération sont apparus, ce qui a permis de diviser par plus d'un million le prix du séquençage.

Les protocoles de séquençage actuels produisent de nombreux fragments d'ADN, appelés "lectures" et qui sont des extraits des génomes de l'échantillon. Dans cette thèse,

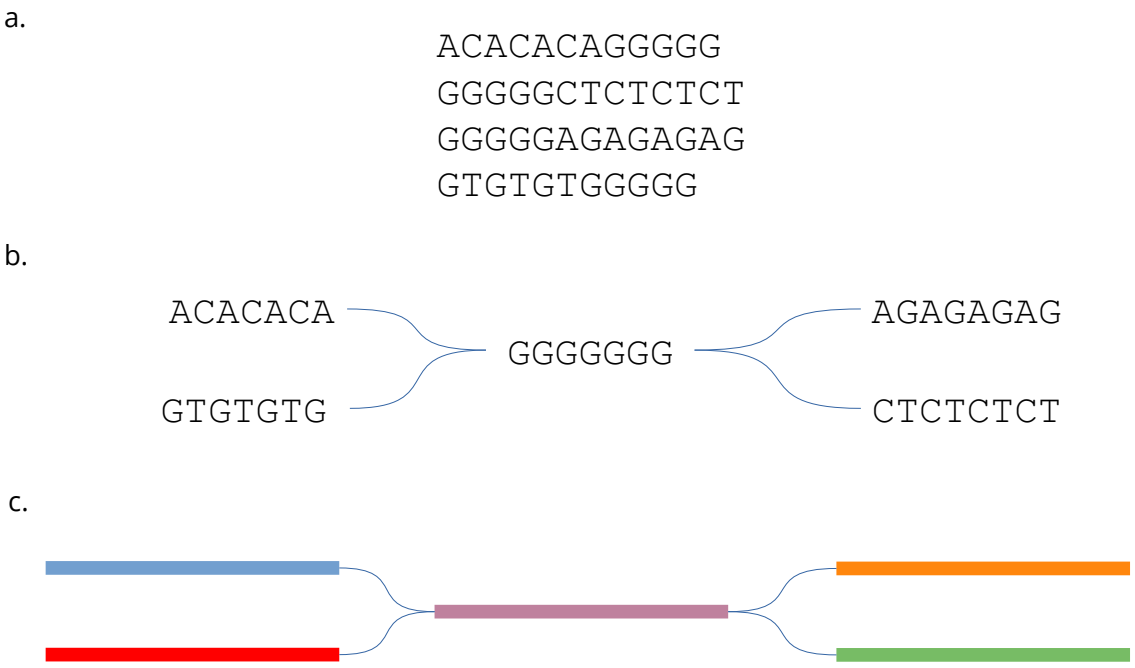


Figure 5.2 – Graphes d'assemblages. a) Lectures b) Un graphe d'assemblage qui représente plusieurs génomes. Les génomes pourraient être {`ACACACAGGGGGGGAGAGAGAG`, `GTGTGTGGGGGGGGCTCTCTCT`} ou {`ACACACAGGGGGGGCTCTCTCT`, `GTGTGTGGGGGGGGAGAGAGAG`} ou les deux à la fois. c) Une convention est de remplacer des séquences par des rectangles de couleur.

je vais me concentrer sur la technologie des "lectures longues", qui produit des lectures de quelques milliers de bases. "Longues" est un qualificatif tout relatif, sachant que les chromosomes humains mesurent plusieurs centaines de millions de bases. Ces lectures ont donc besoin d'être assemblées par des logiciels appelés assembleurs pour reconstruire les génomes entiers. Les assembleurs produisent des graphes d'assemblage, comme illustré Figure 5.2. Ces graphes sont composés de contigs, qui sont des séquences, et d'arêtes entre ces contigs. Chaque chromosome des génomes peut être représenté par un chemin dans ce graphe.

Un problème majeur des assemblages d'aujourd'hui est la présence de plusieurs séquences très proches, ou haplotypes, dans un génome. Typiquement, plusieurs bactéries très semblables peuvent cohabiter dans un microbiome mais avoir des fonctions très différentes. Bien qu'il soit important de réussir à les distinguer, la tâche est rendue ardue par le taux d'erreurs parfois élevé des lectures. Les assembleurs confondent prennent les différences

les souches bactériennes pour des erreurs de séquençage et produisent un assemblage qui ne représente le génome que d'une des souches, ce qui ne peut que laisser un goût amer dans la bouche du bioinformaticien consciencieux, sans même parler du goût laissé dans la bouche du biologiste minutieux. Cette thèse s'attaque à ce problème.

Complétion et correction des graphes d'assemblage

La notion de complétude d'un assemblage est centrale dans l'évaluation de sa qualité – cela mesure si le génome a été entièrement reconstitué par l'assembleur. Je propose de généraliser les définitions habituelles de la complétude et de définir un graphe d'assemblage complet si tous les chromosomes sont représentés par un chemin dans le graphe d'assemblage. Malheureusement, en pratique, les graphes d'assemblages sont rarement complets. En particulier, quand plusieurs haplotypes sont présents, les assembleurs ont tendance à faire de nombreuses erreurs. Voir Figure 5.3 pour un exemple de graphe d'assemblage complet et incomplet.

Cette thèse introduit une nouvelle méthode, implémentée dans le logiciel GenomeTailor, qui permet de corriger et de compléter les graphes d'assemblage. Nous montrons sur plusieurs exemples que GenomeTailor améliore la qualité des assemblages. Si le coeur de notre réflexion porte sur les assemblages compliqués de souches bactériennes proches, nous avons également réussi à améliorer un assemblage de levure et obtenu un assemblage de *Debaryomyces hansenii* d'une qualité exceptionnelle.

Assemblage à partir de lectures bruitées

Le sujet d'origine, et sans doute le plus gros travail de cette thèse, a porté sur l'obtention d'un assemblage phasé à partir de lectures bruitées. La stratégie adoptée a été de partir d'un premier assemblage et d'essayer d'en récupérer les différents haplotypes. Dans une première étape, le graphe d'assemblage est corrigé avec GenomeTailor, ce qui permet de représenter les différences structurales entre les haplotypes. En revanche, les petites différences entres les haplotypes (par exemples les SNPs, des différences d'une seule base) sont beaucoup plus difficiles à détecter. En effet, il est facile de constater qu'une lecture diffère de l'assemblage en un point. Il est par contre beaucoup plus dur de dire si cette différence est due à une erreur de séquençage ou à la présence de deux souches proches qui ne diffèrent que d'une base.

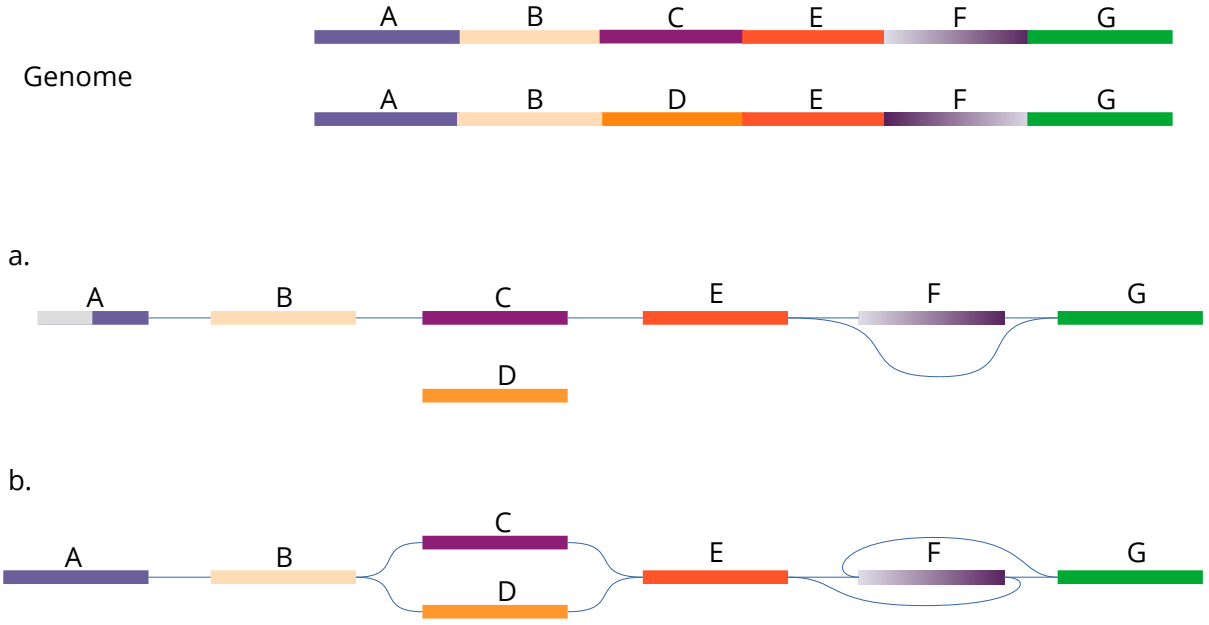


Figure 5.3 – Deux graphes d'assemblage différents visant à représenter le même génome. Le génome est composé de deux haplotypes, avec deux régions hétérozygotes (C et D) et une région inversée, F. L'assemblage *a* contient plusieurs problèmes. Tout d'abord, le contig A est incorrect, car la séquence (grise et bleue) n'est pas présente dans le génome d'origine. Bien que les autres contigs soient corrects, le graphe est incomplet : aucun chemin continu à travers le graphe ne peut inclure le contig D ou représenter correctement l'inversion de F. En revanche, *b* est un graphe d'assemblage complet

La thèse propose une nouvelle méthode statistique pour différencier les erreurs de séquençage des différences entre les souches bactériennes. L'astuce principale consiste à considérer simultanément plusieurs positions proches dans l'assemblage. Les erreurs de séquençage ne sont pas corrélées le long d'un assemblage. La détection d'une corrélation le long de l'assemblage est alors une signature forte de la présence de plusieurs souches. Nous proposons un test statistique simple et robuste pour tester cette corrélation.

Nous avons implémenté un logiciel, HairSplitter, qui exploite ce test statistique pour produire un assemblage phasé en détectant les souches perdues lors de l'assemblage. HairSplitter aligne les lectures sur l'assemblage de départ, détecte les éventuelles souches perdues, identifie les lectures qui proviennent de ces souches et reconstruit un assemblage complet. Nous avons testé HairSplitter sur des jeux de données réels et simulés, et montrons que ce logiciel améliore significativement l'état de l'art dans le cas où de nombreuses souches sont présentes dans un microbiome, tout en prenant moins de temps.

Finalement, nous avons décrit le problème de séparation des haplotypes comme un problème d'optimisation. L'implémentation de HairSplitter de manière modulaire permet de facilement intégrer des nouvelles idées d'optimisation dans le logiciel.

Obtenir un assemblage phasé à partir de lectures précises

Le taux d'erreurs de séquençage est en train de fortement diminuer. Dans les dernières années, il est apparu dans le paysage des technologies de séquençages des séquenceurs "haute fidélité" (HiFi), qui produisent des lectures dont le taux d'erreur ne dépasse pas le demi pourcent. Bien que ces lectures restent chères et compliquées à obtenir, il est possible qu'elles deviennent hégémoniques dans les années à venir.

Le faible taux d'erreurs des lectures simplifie considérablement les problèmes d'assemblage, et notamment l'assemblage de plusieurs haplotypes. Malheureusement, les assembleurs spécifiques developpés au cours des dernières années ne parviennent pas à assembler des mélanges de souches de manière satisfaisante. Nous proposons ici une nouvelle méthode d'assemblage, qui consiste à faire des "sketchs" des lectures en se basant sur la technique des "Mapping-friendly Sequence Reductions" (MSRs). Dit autrement, les lectures sont représentées par de plus courtes séquences qui sont assemblées. L'assemblage final est ensuite déduit de l'assemblage réduit. Cette technique a le double avantage d'être très efficace computationnellement parlant et de pouvoir produire des assemblages de mélange

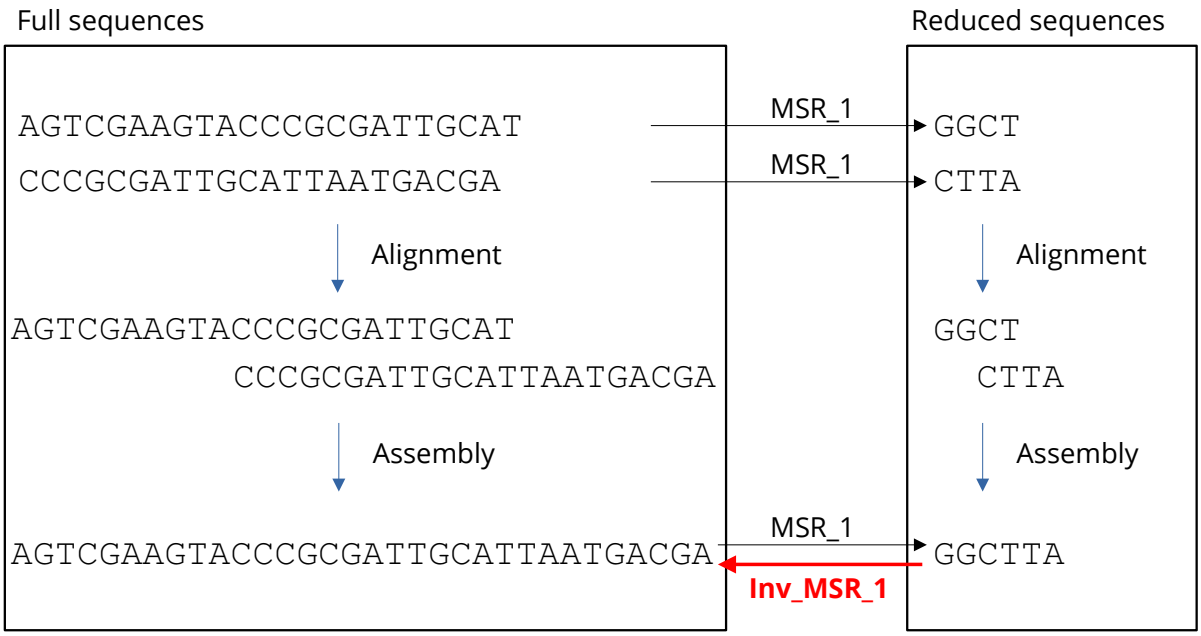


Figure 5.4 – Un exemple d'assemblage qui utilise un sketch nommé MSR_1. Appliqué sur CCCGCGATTGCATTAATGACGA, MSR_1 donne la séquence réduite CTTA. Les séquences réduites peuvent être assemblées puis décompressées à l'aide de la fonction Inv_MSR_1 pour obtenir l'assemblage désiré.

de souches de haute qualité. L'assembleur est décrit Figure 5.4.

Amélioration du scaffolding des génomes

Le dernier travail de cette thèse a porté sur le scaffolding de génomes. Quel que soit l'assembleur, les lectures de séquençage ne comportent généralement pas suffisament d'information pour reconstruire entièrement les génomes - c'est pourquoi les assembleurs ne reconstituent que des graphes (voir Figure 5.2). Le but d'un scaffoldeur est d'utiliser des technologies auxiliaires, typiquement celle appelée Hi-C, pour améliorer la contiguité d'un assemblage, c'est à dire augmenter la longueurs des contigs, idéalement jusqu'à n'avoir qu'un contig par chromosome.

Si de nombreux scaffoldeurs existent pour les génomes haploïdes et multiploïdes, ils exploitent très peu l'information représentée par les arêtes des graphes d'assemblage. Nous proposons une nouvelle méthode pour scaffolder des assemblages, qui consiste d'abord à

démêler le graphe d'assemblage puis à faire un scaffoldage classique. Cette méthode permet d'exploiter pleinement les graphes d'assemblage. Implémenté dans le logiciel GraphUnzip, nous avons largement amélioré les scaffoldages d'assemblages haploïdes et multiploïdes. GraphUnzip est compatible avec tous les scaffoldeurs actuels et pourrait y être aisément intégré.

COLLEGE DOCTORAL BRETAGNE
MATHS, TELECOMS
INFORMATIQUE, SIGNAL
SYSTEMES, ELECTRONIQUE

Université de Rennes

Titre : Assemblage d'haplotypes à partir de lectures longues

Mot clés : Assemblage de génome, metagénomique, haplotypage, séquençage de troisième génération (TGS)

Résumé : Cette thèse propose des solutions pour améliorer l'assemblage des génomes à partir de lectures de séquençage de troisième génération (lectures longues). Plus précisément, elle se concentre sur l'amélioration de l'assemblage des (méta)génomes contenant plusieurs haplotypes, comme des génomes polyploïdes ou des souches bactériennes proches. Les assembleurs actuels ont du mal à séparer les haplotypes très similaires, et fusionnent généralement des (parties d')haplotypes, ce qui entraîne la perte de polymorphismes et d'hétérozygotie dans l'assemblage final. Ce travail présente une série de méthodes et de logiciels pour obtenir des assemblages contenant des haplotypes bien séparés. Plus précisément, GenomeTailor et HairSplitter transforment un assemblage obtenu avec des lectures longues erronées en un assemblage phasé, améliorant considérablement l'état de l'art lorsque de nombreuses souches sont présentes. Le logiciel Alice propose une nouvelle méthode, basée sur des nouveaux sketchs "MSR", pour assembler efficacement plusieurs haplotypes séquencés avec des lectures de haute fidélité. Enfin, cette thèse propose une nouvelle stratégie de scaffolding Hi-C basée sur le démêlage des graphes d'assemblage qui améliore considérablement les assemblages finaux, en particulier lorsque plusieurs haplotypes sont présents.

Title: Haplotype assembly from long reads

Keywords: Genome assembly, metagenomics, haplotyping, Third Generation Sequencing (TGS)

Abstract: This thesis presents solutions to improve genome assembly from third-generation sequencing reads, with a specific focus on improving the assembly of (meta)genomes containing multiple haplotypes, such as polyploid genomes or close bacterial strains. Current assemblers struggle to separate highly similar haplotypes, often collapsing all or parts of the haplotypes into one, thereby discarding polymorphisms and heterozygosity. This work introduces a series of methods and software tools to achieve haplotype-separated assemblies. Specifically, GenomeTailor and HairSplitter transform a collapsed assembly obtained with erroneous long reads into a phased assembly, significantly improving on the state of the art when numerous strains are present. The software Alice introduces a new method based on the new "MSR" sketching technique for efficiently assembling multiple haplotypes sequenced with high-fidelity reads. Additionally, this thesis proposes a new Hi-C scaffolding strategy that involves untangling assembly graphs which significantly improves final assemblies, particularly when several haplotypes are present.